MPI^{GL} : un projecte d'utilització del Programari Lliure en la visualització gràfica en xarxes de càlcul científic

Autor: Alan Ward Koeck

Professor responsable: Miguel Martín Mateo

Curs: 2008-2009, S1

Aquesta memòria es presenta el desembre del 2008, en compliment del Projecte Final del ***Màster en Programari Lliure*** *en la seva especialització* ***d'Administració de xarxes i sistemes operatius****.*

Copyright © 2008 Alan Ward Koeck.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation. It contains no Invariant Sections. A copy of the license is included in the section entitled "GNU Free Documentation License".

Es permet la lliure còpia, distribució i modificació d'aquest document segons els termes de la GNU Free Documentation License, Versió 1.2 o qualsevol versió posterior publicada per la Free Software Foundation. No conté seccions Invariants. S'inclou una còpia de la llicència en la secció sota el títol "GNU Free Documentation License".

Resum

GNU/Linux és un dels sistemes operatius que més es fan servir pel càlcul científic en paral·lel, que sigui per construir clústers de càlcul dedicats (clústers Beowulf o Grid computing), o per emprar cicles de CPU d'ordenadors que també fan altres tasques (Clusters of Workstations, COW). En aquests casos, la seva estabilitat com sistema permet càlculs d'una certa durada, que pot anar fins a dies o setmanes. Algunes aplicacions poden ser problemes de mecànica de fluids, càlculs d'estructures i altres.

Però no és fàcil administrar aquest tipus de conjunt de càlcul, ja que el problema tractat sol ser complex. Així, sovint no disposem d'eines per seguir l'estat d'avançament del procés.

La proposta d'aquest projecte consisteix en la construcció d'un projecte de codi obert mitjançant el qual els nodus de càlcul poguessin emprar un llenguatge de tipus OpenGL per enviar manaments gràfics a un únic nodus gràfic (amb la seva pantalla), sigui a través d'una connexió TCP, sigui enviant missatges MPI.

La primera etapa prevista serà l'estudi teòric del problema, per determinar la càrrega suplementària sobre la xarxa, l'aparició de possibles colls de botella que poguessin alentir el procés de càlcul, i sobretot l'estudi de la seva escalabilitat segons el nombre de nodus de càlcul.

Després es passarà a dissenyar i desenvolupar l'aplicació en llenguatge C++, amb la forma corrent d'un projecte de PL, i alliberat sota llicència GPL. Donades les seves característiques (gairebé de biblioteca de rutines), es preveu la necessitat de generar una documentació comprensiva de l'API de programació.

Finalment, caldrà discutir els resultats del sistema a partir de dos o tres exemples reals (però simplificats) del seu funcionament, escollits de manera a tractar diferents escales de granularitat del problema. Així mateix, s'acabarà indicant algunes direccions en què queden possibilitats de desenvolupament posterior.

4

蝃蝀在東
莫之敢指

Hi ha un arc de Sant Martí a l'est,
I ningú gosa esmentar-lo.
Shijing, Poemes de Yong

Índex

1 Presentació del projecte **7**
1.1 Situació i problemàtica 7
1.1.1 Descripció de la situació inicial 7
1.1.2 Problemàtica 7
1.2 Funcionalitat demanada 8
1.3 Estructura prevista de la solució 10

2 Anàlisi **11**
2.1 Presentació de la problemàtica 11
2.2 Aplicació de la Llei d'Amdahl 12
2.3 Situació i mesura de colls de botella que limitin l'escalabilitat 13
2.4 Estat actual de la tecnologia 17
2.4.1 Infraestructura de xarxa 17
2.4.2 Connexió de xarxa del nodus gràfic 17
2.4.3 Tractament per part de la unitat central 18
2.4.4 Bus de connexió 19
2.4.5 Capacitat de tractament de la tarja gràfica 20
2.5 Resum de l'estat actual dels colls de botella 21

3 Disseny del sistema **23**
3.1 Disseny general 23
3.1.1 Requeriments dels usuaris 23
3.1.2 Estructura modular 23
3.1.3 Mòduls d'entrada 24
3.1.4 Mòdul d'emmagatzematge 25
3.1.5 Mòdul de representació 25
3.2 Implementació 27
3.3 Llenguatge de descripció de les instruccions gràfiques . 28

4 Resultats de la implementació del sistema **33**
4.1 Implementació 33
4.2 Dificultats en la implementació 39

4.2.1 Funcions *callback* 40
4.2.2 Accés concurrent a recursos 41

5 Anàlisi d'alguns casos pràctics d'aplicació 43
5.1 Simulació de la col·lisió entre galàxies 43
5.1.1 Descripció 43
5.1.2 Càlcul del volum de dades 44
5.1.3 Implementació real del cas 46
5.2 Modelització de la transmissió de calor en un sòlid . . . 48
5.2.1 Descripció 48
5.2.2 Càlcul del volum de dades 50
5.2.3 Implementació real del cas 51
5.3 *Raytracing* d'anells d'Einstein 53
5.3.1 Descripció 53
5.3.2 Càlcul del volum de dades 55
5.3.3 Implementació 55

6 Posada a disposició de la comunitat en un entorn de Programari Lliure 57

7 Conclusions i reflexions suplementàries 59

A Ús del Programari Lliure en aquest projecte 65

B Glossari 67

C GNU Free Documentation License 69

Capítol 1

Presentació del projecte

1.1 Situació i problemàtica

1.1.1 Descripció de la situació inicial

El present projecte parteix d'una necessitat que ha sorgit en un grup de recerca d'una universitat catalana. En aquest grup de recerca, un dels alumnes que en breu[1] es vol integrar un programa de recerca vol dur a terme un tesi de doctorat en el camp de la Dinàmica de Fluids assistida per Ordinador (Computational Fluid Dynamics o CFD). Per aquest motiu, requereix l'ús de recursos de càlcul en la mateixa universitat, entre altres el *cluster* gestionat per aquest grup de recerca.

Aquest *cluster* és constituït per un total de 8 nodus de càlcul, cadascun amb 4 nuclis, el que porta a un total de 32 nuclis de càlcul disponibles. Es preveu que una part important dels codis CFD emprats seran o bé originaris del projecte OpenFOAM [12], o bé codis MPI [8] escrit pel propi investigador.

La submissió de seqüències de càlcul es farà de manera no presencial, a través de la infraestructura de xarxa pública d'Internet i una connexió segura SSH. El temps de càlcul pot variar, però es preveuen algunes seqüències que requereixin durades de diverses hores i fins a dies sencers.

1.1.2 Problemàtica

Naturalment, aquest *cluster* no és dedicat exclusivament a les finalitats pròpies de cap investigador concret, el que implica una gestió del recurs limitant el temps d'execució per cada usuari, i a la vegada optimitzant l'ús que en fa durant el temps que se li assigni.

[1] Durant el mes de gener 2008.

Per aquest motiu, es vol buscar la possibilitat de monitoritzar el procés de càlcul durant la seva execució. Idealment, seria convenient disposar d'una pantalla gràfica, amb capacitats 3D, per poder visualitzar la formació de la solució en els nodus de càlcul. D'aquesta manera, l'investigador podrà:

1. Seguir l'estat d'avançament dels càlculs en pantalla.
2. Identificar els càlculs que presentin importants desviacions respecte al model teòric. Així es podran detectar errors de representació o de solució dels sistemes d'equacions sense haver d'esperar l'acabament del procés de càlcul complet, agilitzant el procés de depuració que acompanya la creació de qualsevol programari.
3. Detectar situacions en què fallades d'una part del maquinari puguin afectar l'equilibri del conjunt, fent inviable la continuació del càlcul. Aquesta mena de situació es pot produir molt fàcilment durant l'ús de la computació paral·lela per la resolució de problemes de dinàmica de fluids, problemes coneguts com *fine-grained* i en el qual la producció de cada nodus de càlcul és imprescindible per poder continuar les etapes successives de càlcul en tots els altres nodus.

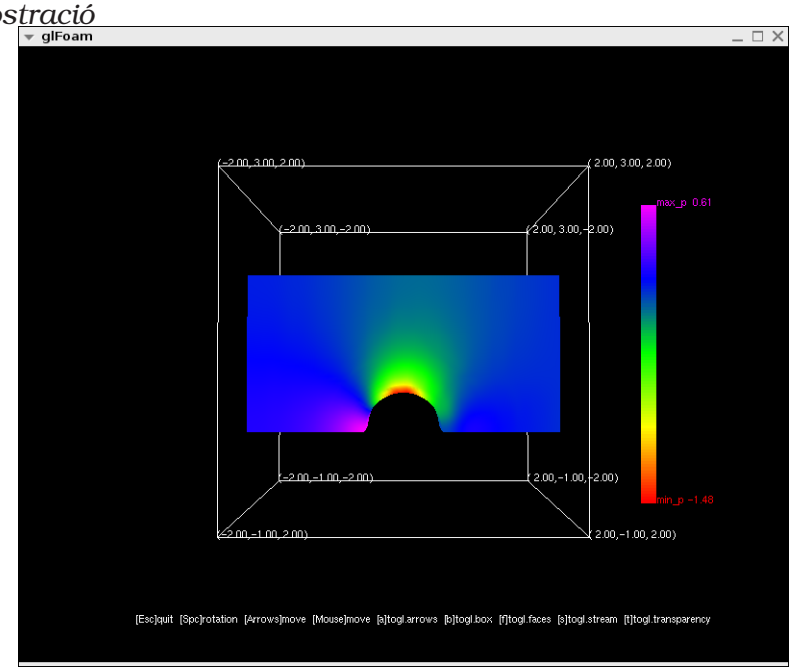
1.2 Funcionalitat demanada

Donat el fet que en aquest grup de recerca es fan investigacions en una sèrie de camps relacionats, i que en cada cas se sol desenvolupar el programari adient de manera interna, es demana que el programari desenvolupat en aquest projecte sigui com més flexible millor. Amb això en vista, s'han determinat les característiques desitjables del mateix:

- El programari desenvolupat s'haurà de poder emprar a partir de diverses aplicacions ja existents, o en desenvolupament. Encara que la programació es faci en aquest grup de recerca sobretot en llenguatge C/C++, ha de ser possible emprar el programari a partir d'aplicacions en altres llenguatges[2].
- Cal que permeti a cada nodus de càlcul presentar elements gràfics (punts, línies, polígons i texts) en una mateixa pantalla remota. En principi, no es preveu una separació de la zona

[2]Concretament, ja existeix una base de codi escrit en Java.

Figura 1.1: Possible aspecte gràfic del producte final. *Versió de demostració*



controlada per cada nodus; serà doncs la responsabilitat del programador que l'empri determinar aquest aspecte.

- El resultat final ha de ser a la vegada atractiu visualment, i fàcil de manipulació perquè l'usuari final es pugui "desplaçar"en l'entorn gràfic generat i examinar tots els aspectes de la informació gràfica generada. En aquest sentit, s'ha creat una versió de demostració de l'aspecte gràfic del programari, validat per l'usuari final i presentat en la Figura 1.1.

- Un punt important és que s'haurà de validar que el programari no representi una sobrecàrrega excessiva sobre la xarxa, tant a l'interior del *cluster* (a on ja existeix el tràfic de xarxa propi de la computació), com en l'enllaç extern entre el *cluster* i el terminal de l'investigador. Si existeixen limitacions en aquest nivell, convé que es presentin formes de calcular-les de manera que l'investigador en pugui tenir compte a l'hora de dissenyar les seves aplicacions.

1.3 Estructura prevista de la solució

A partir de la funcionalitat demanada, una primera anàlisi ha determinat de dissenyar la solució amb una arquitectura *client-servidor*. D'aquesta manera:

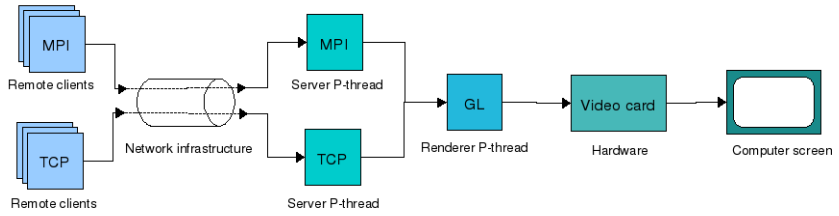
1. El programari existent es pot adaptar sense modificacions profundes de la seva estructura.

2. Es podrà emprar fàcilment tant a través de MPI com una versió adaptada d'HTTP. Així s'assegura que pugui proporcionar servei tant a aplicacions MPI que s'executin directament a l'interior del *cluster*, com també a altres aplicacions menys distribuïdes, sovint programades en Java, i que hi podran accedir a través de la interfície HTTP.

3. El servidor gràfic es podrà escriure en qualsevol llenguatge, de manera independent a les aplicacions clientes que l'hagin d'emprar (Figura 1.2).

Per fer que el servidor gràfic pugui ser a la vegada *lleuger* en termes d'ús dels recursos del sistema, però a la vegada fàcilment *portable* a diferents plataformes i sistemes operatius, s'ha determinat que es programarà exclusivament en llenguatge C++, emprant recursos àmpliament disponibles com les biblioteques **p-thread**[3] [14] per gestionar els diferents serveis oberts, i **Mesa 3D**[4] [17] per la representació gràfica.

Figura 1.2: Estructura prevista del programari



[3]Threads natives segons la norma POSIX.
[4]Versió de codi obert del llenguatge gràfic també obert, OpenGL.

Capítol 2

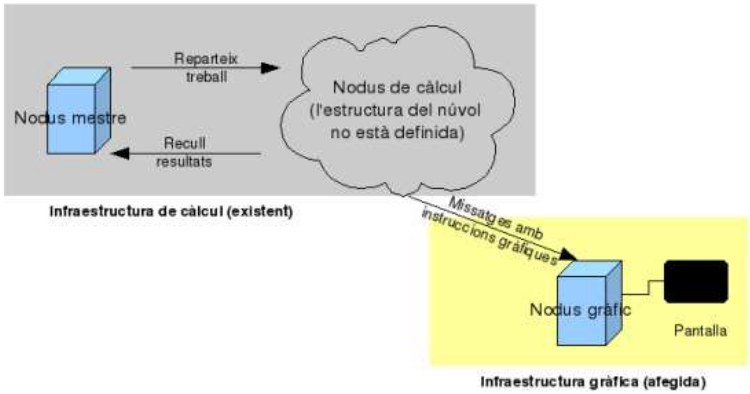
Anàlisi

2.1 Presentació de la problemàtica

En aquest document es presenta l'anàlisi i un inici de disseny de la solució d'un problema en què un cert nombre de nodus de càlcul, durant el curs del processament paral·lel d'un problema, empren els serveis d'un únic nodus gràfic. En aquest, els altres nodus poden representar objectes tridimensionals com vèrtexs, línies, polígons i texts. La problemàtica de base se situa al nivell de *l'escalabitat* de la solució: o sigui, fins a quin punt una solució satisfactòria per un cert nombre N de nodus de càlcul es pugui estendre a un nombre major, $N' = 2N, 3N, \cdots$.

Per la present discussió, suposarem un conjunt de nodus de càlcul homogenis, d'idèntica potència de càlcul per la computació de referència. Si no fos el cas, no afectaria el resultat del nostre raonament, apareixent tan sols en alguns detalls de les equacions.

Figura 2.1: Situació del conjunt de nodus de càlcul, amb un nodus gràfic afegit.



2.2 Aplicació de la Llei d'Amdahl

En una computació paral·lela, en poden distingir:

- t_S el temps de processament seqüencial: aquest seria el temps de processament per la part del càlcul que no es pot processar en paral·lel, i que doncs caldrà executar en un únic nodus de càlcul. En la majoria d'instal·lacions actuals, té lloc en el nodus principal (nodus *mestre*). Per exemple, pot incloure la preparació de les dades a enviar als nodus esclaus, i la recepció dels seus resultats.

- t_P el temps de processament de la part del càlcul que es pugui dur a terme de manera paral·lela, en el supòsit de disposar **d'un únic nodus** de càlcul.

- t_{Total}, el temps total de processament o *wall-clock time*. Aquest és el factor que es desitja minimitzar de cara a l'usuari.

Si disposem de N nodus de càlcul, s'en pot derivar la llei d'Amdahl [1]:

$$t_{Total} = t_S + \frac{t_P}{N}$$

Asimptòticament, el millor temps de càlcul que podríem esperar (a condició de poder suposar la possibilitat d'augmentar el nombre de nodus a l'infinit, tot i mantenint sempre les mateixes condicions de càlcul entre els nodus) seria $T_{Total} = t_S$.

Si en aquest projecte afegim un nou nodus, un nodus gràfic que donarà servei a la pantalla de visualització, es poden produir dues demores suplementàries:

1. ${t_S}'$, un augment del temps de processament seqüencial;

2. ${\tau_P}'$, l'augment del temps de processament paral·lel *en cada nodus de càlcul*.

En aquest cas, la nostre formula del temps esdevé

$$t_{Total} = t_S + {t_S}' + \frac{t_P + N \cdot \tau_P}{N}$$

$$t_{Total} = t_S + {t_S}' + \tau_P + \frac{t_P}{N}$$

En resum, ens podem esperar a un temps suplementari per completar el càlcul de l'ordre de ${t_S}' + \tau_P$.

Analitzant els dos factors, podem arribar a dues observacions:

- El terme $t_S{}'$ consistiria en el temps necessari perquè el nodus mestre inicialitzés el nodus gràfic, per exemple enviant-li instruccions pels elements gràfics fixos de l'escena (eixos de coordenades, etiquetes, etc.). Encara que ens convingui limitar aquest temps, tampoc ha de resultar excessiu comparat a la durada total del càlcul científic[1].

- El terme τ_P és el que representa la sobrecàrrega de feina per cada nodus de càlcul, deguda a la necessitat de generar i enviar els elements gràfics. Ens convé que sigui reduït en proporció, el que es pot aconseguir fàcilment augmentant l'interval d'enviament de noves informacions gràfiques a uns quants segons o fins a minuts.

A través d'aquesta discussió inicial, hem vist que afegir un nodus gràfic no ha de resultar un inconvenient per la durada de la computació científica. Existeix, però, un element que no hi hem tingut en compte, i que és el mateix nodus gràfic i el treball que s'hi haurà d'executar.

Es preveu que sigui en aquest nivell que es puguin produir els majors *colls de botella* en l'execució del sistema, ja que finalment estem tractant d'un únic nodus gràfic que haurà de donar servei a un conjunt de nodus de càlcul que en algunes instal·lacions es pot comptar en milers.

Per aquest motiu, resulta imprescindible tenir en compte aquesta càrrega i dimensionar correctament la quantitat i freqüència d'enviament de noves informacions gràfiques per part dels nodus de càlcul.

Com se sol recordar en la llista de correu Beowulf [3], *VMMV*[2]; o sigui que cada investigador haurà de calcular la càrrega màxima que pot enviar al nodus gràfic en funció de les seves condicions concretes de infraestructura material i de xarxa, tipus de càlcul i resultats gràfics desitjats. En la secció següent analitzem els factors a tenim en compte per aquest càlcul.

2.3 Situació i mesura de colls de botella que limitin l'escalabilitat

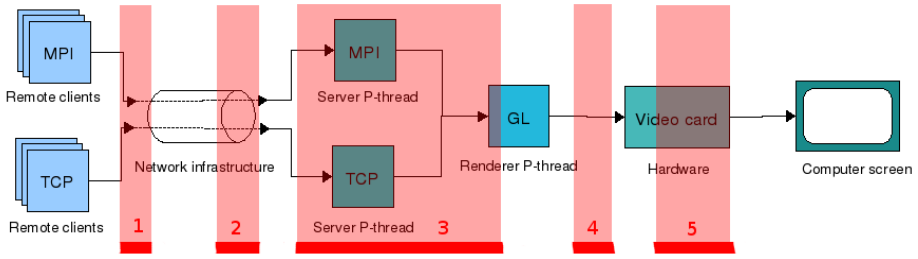
La principal diferència amb el tractament clàssic d'aquest tipus d'aplicació és la presència d'un únic nodus gràfic. En aquest context, es podria formar un *coll de botella* (veure Figura 2.2):

[1] Donat l'estat actual de la tecnologia, s'espera un temps de l'ordre del mil·lisegon per aquesta inicialització.

[2] "Your Milage May Vary"

1. En algun lloc de la infraestructura de xarxa.
2. A la connexió de xarxa a l'entrada del nodus gràfic.
3. Durant el tractament i transmissió per part de la unitat central, entre l'arribada de la xarxa i la tarja gràfica.
4. En el bus de connexió entre el processador central i la tarja gràfica.
5. Degut a una capacitat de tractament insuficient per part de la tarja gràfica.

Figura 2.2: Possibles colls de botella en l'estructura de l'aplicació.



En el moment de determinar quin d'aquests punts pot resultar un factor que limiti les possibilitats del conjunt de l'aplicació, existeixen dos problemes. Un és la falta d'una mètrica comuna, ja que les amplades de banda en cada punt se solen mesurar emprant unitats diferents. Si es pot aconseguir unificar aquestes unitats, es podran a la vegada determinar els punts més susceptibles de limitar el flux de dades, i a més es tindran elements per resoldre el segon problema que és la variabilitat dels dispositius físics emprats segons la instal·lació (i el seu pressupost de *hardware*).

Per proposar una unitat de mesura, una possibilitat és considerar el resultat final que desitja l'usuari: poder representar elements gràfics en una escena 3D. S'haurà de transmetre la descripció d'aquests elements del nodus de càlcul, en el nostre cas emprant un protocol de comunicació derivat del llenguatge OpenGL [17].

En primera aproximació, les instruccions GL es poden tractar com cadenes de caràcters. Un exemple en seria

$$glVertex3f\ 0.44\ 0.52\ 0.63$$

per representar un punt a les coordenades $[0.44, 0.52, 0.63]$ de l'espai[3]. Aquesta representació textual pot semblar poc eficient, però de fet no ho és gaire més que una de codificada emprant el binari.

[3]Les unitats són arbitràries, de la representació 3D.

Considerem la següent codificació binària:

- Un primer camp de tipus *byte*, amb un codi numèric per representar l'operació GL a realitzar (per exemple, 1 = glVertex3f, 2 = glBeginPoints, etc.).
- Tres altres camps de tipus punt flotant, amb les dades de l'operació [4]. Emprant *double* sobre 64-bits, cada un d'aquests requeriria 8 bytes.

En aquesta codificació binària ja prou compacte, emprarìem un total de 25 bytes per cada instrucció.

Amb una codificació textual - i limitant la precisió dels camps amb punt flotant - ens en podem sortir amb 40 a 50 caràcters, el que no representa un sobrecost massa elevat i presenta l'avantatge de ser fàcilment codificable tant pel nodus de càlcul com interpretable pel nodus gràfic. Hi ha doncs menys necessitat de processament en els dos extrems de la cadena de transmissió entre nodus de càlcul i nodus gràfic.

Emprant un format obert, ens assegurem a més la independència de la plataforma i *byte-order*, i facilitem l'ús del sistema descrit per aplicacions en què els nodus de càlcul siguin heterogènies.

Després d'aquestes consideracions, s'ha decidit emprar per cada instrucció OpenGL una codificació textual (cadena de caràcters), limitat a $q_{ins} = 50bytes$ de llargada.

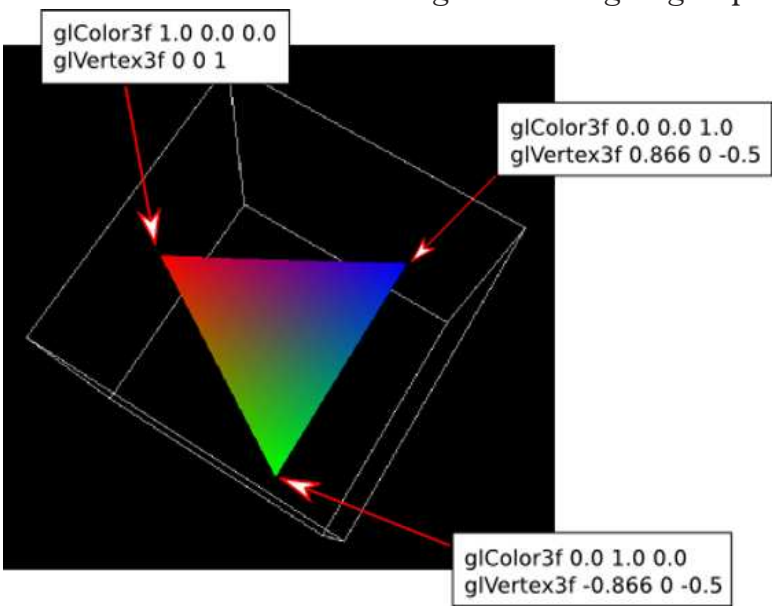
Però, quantes instruccions necessitarem per efectuar un treball útil? La resposta depèn molt del tipus de seqüència d'instruccions que es doni. Alguns exemples:

- En un primer cas (minimalista), per dibuixar una seqüència de p punts (vèrtex) tots del mateix color, necessitaríem una instrucció per cada punt, una instrucció per l'especificació de color inicial, i dues instruccions per englobar la seqüències (instruccions BEGIN i END). En total, haurem pres $p + 3$ instruccions.
- Per dibuixar una seqüència de p rectes, també caldrà una instrucció d'especificació de color, dues de seqüència, i a més dues instruccions per cada línia (definint els seus extrems com vèrtex). Això ens porta a un total de $2p + 3$ instruccions.
- Finalment, per en un cas més complex en què volem dibuixar una seqüència de p polígons, cada un amb c costats, i si a

[4] Poden ser quatre camps si desitgem donar suport a un canal α per a la transparència

més volem que cada vèrtex del polígon tingui un color diferent, haurem d'emprar un total de $c \cdot p$ instruccions de color, $c \cdot p$ instruccions per les coordenades dels vèrtex definint els polígons, a més de les dues instruccions de seqüència. En total, haurem necessitat $2 \cdot c \cdot p + 2$ instruccions (veure Figura 2.3).

Figura 2.3: Construcció d'un triangle amb llenguatge OpenGL.



Malauradament, sembla que aquest últim cas hagi de ser el més freqüent, a mesura que els usuaris vulguin emprar el potencial d'aquesta eina. Podem, però, suposar que el tipus de polígon més freqüent serà el triangle, i doncs que caldran unes 8 instruccions per cada triangle representat individualment[5]: $q_{tri} = 8q_{ins} \approx 400bytes$.

Val a dir que aquesta estimació és prou conservadora, i que els valors reals siguin inferiors segons l'aplicació concreta tractada.

Amb aquesta estimació, podrem convertir les unitats acostumades de velocitats de transferència en una unitat homogènia de cara a la seva utilitat per l'usuari: $triangles/s$.

[5]Es podran fer alguns estalvis en cas d'enviar més d'un triangle en una mateixa seqüència, *back-to-back*.

2.4 Estat actual de la tecnologia

2.4.1 Infraestructura de xarxa

Els diferents tipus d'infraestructura de xarxa que s'empren en l'actualitat no són tan nombrosos com fa uns quants anys. Avui en dia, es poden resumir en

- Les diferents variants d'Ethernet, sobretot Gigabit (1 Gbit/s) però amb també una part creixent de 10-Gbase-X a 10 Gbit/s (i l'esperança de l'arribada del 100-Gbase-X).
- Altres sistemes d'interconnexió amb tarificació bastant superior, acompanyat també per latències molt més favorables que els derivats d'Ethernet. Per motius de cost i d'eficiència, encara se solen tornar a aprofitar materials més antics de tipus Myrinet, si bé el material nou d'aquest tipus sigui una de les variants d'Infiniband.

Sempre que la infraestructura de xarxa (switchs i cablejat) estigui correctament dimensionada, podem proposar la següent comparativa (Taula 2.1).

Taula 2.1: Comparativa de velocitat de la infraestructura de xarxa.

Tecnologia	Ampla de Banda (Gbit/s)	Velocitat útil (triangles/s)
Gigabit 1-Gbase-X	1,00	250.000
Gigabit 10-Gbase-X	10,00	2.500.000
Myrinet 1	1,00	250.000
Myrinet 2000	2,00	500.000
Infiniband 1X	4,00	1.000.000
Infiniband DDR 4X	16,000	4.000.000

Així, la velocitat característica en aquest punt pot variar considerablement en funció de la tecnologia emprada. Per la present discussió, prendrem un valor conservador de 250.000 triangles/s.

2.4.2 Connexió de xarxa del nodus gràfic

A nivell del dispositiu físic d'entrada al nodus gràfic, poden existir limitacions de velocitat diferents de les de la xarxa en general. Trobarem dos casos possibles:

1. La xarxa està sota dimensionada respecte als dispositius de connexió de xarxa de cadascun dels nodus. Aquest cas, prou

freqüent, es podria exemplificar posant el cas d'un switch amb un total de 600 Mbit/s de velocitat de *backplane*, i al que es connecten 16 nodus de càlcul amb targes FastEthernet a 100 Mbit/s. Si mai ens trobem en la situació en què es formen 8 parelles de nodus que intercanviïn dades a 100 Mbit/s entre els nodus de cada parella, el *backplane* del switch no podrà proporcionar el total de 800 Mbit/s d'amplada de banda necessària. En aquest cas, el factor limitant seria la infraestructura de xarxa.

2. La tarja de xarxa del nodus gràfic està sota dimensionada respecte a l'amplada de banda de la xarxa en general. Com que en general els dispositius de xarxa de cada família solen ser compatibles físicament i a nivell de protocols amb els anteriors dispositius de la mateixa tecnologia, és relativament fàcil veure situacions en què l'operador no s'ha fixat en la velocitat de la tarja de xarxa. Un exemple en seria en el moment de tornar a aprofitar un material antic, massa desfasat per ser un nodus de càlcul. És força probable que acabem connectant un nodus gràfic amb tarja FastEthernet a 100 Mbit/s amb un switch Gigabit.

Un altre punt al que convindria portar atenció a aquest nivell és la necessitat que la connexió interna entre la tarja de xarxa i la placa mare del nodus gràfic pugui seguir el ritme imposat per la xarxa, el que podria no ser el cas en el supòsit d'un bus PCI i una tecnologia de xarxa ràpida[6].

La discussió de les velocitats en $triangles/s$ seria el mateix que en el cas anterior, prenent aquí també un valor conservador de 250.000 triangles/s.

2.4.3 Tractament per part de la unitat central

La unitat central del nodus gràfic s'haurà d'encarregar de transformar totes les informacions que arribin en instruccions gràfiques[7] apropiades per enviar a la tarja gràfica.

Es fa difícil estimar la càrrega de feina involucrada en el tractament de cada instrucció gràfica. Dependrà en part del format de codificació de la informació (si s'ha de fer un *parsing*, etc.), i en part de l'eficiència del codi de la llibreria gràfica. En tot cas, és raonable suposar que aquesta llibreria gràfica serà prou optimitzada[8], i

[6]Aquesta discussió no és rellevant únicament pel nodus gràfic, sinó que també es pot aplicar als mateixos nodus de càlcul.

[7]Per exemple, de la llibreria Mesa 3D.

[8]Amb l'objectiu de la seva utilització per programar jocs d'ordinador.

doncs que la major part de la càrrega sobre el CPU serà pel tractament i emmagatzematge de les instruccions a la seva entrada.

Entre lectura de l'*input* en el *buffer* de la xarxa, còpia a la RAM, tornar a llegir la RAM pel seu enviament a la tarja gràfica, sembla que cada un dels 400 bytes de cada instrucció s'hagi de tractar al menys 3, i potser fins a 4 vegades. Per fer-ne una estimació sempre pessimista, suposarem que la càrrega sigui de l'ordre de les $4 \cdot 400 = 1600$ operacions de tractament pur de les cadenes de caràcters de cada instrucció. Si a més afegim unes operacions pel seu tractament condicional en funció del tipus d'instrucció de la que es tracta i per la seva identificació (a quin nodus de càlcul pertany aquesta instrucció), proposem un còmput global molt aproximat d'unes 2000 operacions de CPU per cada instrucció.

Encara que la mètrica *BogoMips* dels nuclis de Linux no sigui una mètrica perfecte, ens permetrà donar una estimació de treball útil del CPU del nodus gràfic a partir de la seva potència de càlcul aproximada (Taula 2.2).

Taula 2.2: Comparativa de potències de càlcul de diferents CPUs.

CPU	Potència (BogoMips)	Velocitat útil (triangles/s)
Intel Mobile Celeron (en un portàtil)	3400	1.700.000
AMD Athlon @3.4 GHz	6800	3.400.000
Intel Pentium Dual Core @2.0 GHz	8000	4.000.000
Intel Xeon Quad Core @3.0 GHz	24000	12.000.000

Aquí també veiem que la potència de processament útil en la unitat que ens interessa, triangles/s, pot variar considerablement en funció de l'equipament del que disposem. En tot cas, emprant el material actual menys potent, encara ens podem assegurar de l'ordre de 1.700.000 triangles/s.

2.4.4 Bus de connexió

El bus de connexió entre la placa mare del nodus de càlcul i la seva tarja gràfica és un punt en què s'ha portat l'atenció dels fabricants de material informàtic d'ús domèstic des de fa més de 10 anys[9]. Per aquest motiu, la progressió entre el bus PCI avui en dia pràcticament obsolet per la connexió de targes gràfiques, i els més recents AGP i PCI Express ha estat notable. La desglossem en la Taula 2.3.

[9]Concretament, des de que la velocitat de càlcul dels PC s'ha fet comparable a la de les consoles de joc.

Taula 2.3: Comparativa de velocitat de busos de connexió de la tarja gràfica.

Tecnologia	Ampla de Banda (Gbit/s)	Velocitat útil (triangles/s)
PCI 32 bit, 33 MHz	1,067	250.000
AGP 4x	8,533	2.130.000
PCI Express 2(x8)	32,000	8.000.000

A partir d'aquestes dades, la conclusió és prou clara: a menys de disposar d'un bus de connexió amb la tarja gràfica de tecnologia antiga com el PCI, és probable que sigui diverses vegades més ràpida que la infraestructura de xarxa, de l'ordre al menys de 2.130.000 triangles/s.

Aquest punt tan sols serà preocupant en el cas d'estar tornant a aprofitar material antic.

2.4.5 Capacitat de tractament de la tarja gràfica

Existeix una competència important entre les cases principals que fabriquen els processadors gràfics com el de la Figura 2.4, el que fa que les seves principals característiques puguin ser prou comparables.

Figura 2.4: Fotografia d'un processador gràfic. Font: Wikipedia/-Berkut.

En el moment d'escriure aquesta memòria, els tres processadors

més alts de gama tenien les característiques de la Taula 2.4.

Taula 2.4: Comparativa de velocitat de tres models de tarja gràfica alt de gama.

Model	Memòria (MByte)	Ampla de Banda E/S (Gbit/s)	Velocitat (10^9 pix./s)
nVidia GeForce 9800 GX2	1024	128	12,2
nVidia GeForce 8800 Ultra	768	103,7	14,7
ATI Radeon HD 4870	1024	115,2	12,0

Com s'aprecia, aquesta vegada la unitat de mesura ja és més difícil de convertir en $triangles/s$, ja que a un mateix triangle tridimensional pot correspondre un nombre variable de píxels, segons l'aspecte sota el qual s'observa en pantalla[10].

Per prendre un valor, farem la suposició que en una pantalla de superfície $1200 \cdot 900 = 1.080.000 pixels$ hi podrem trobar típicament de l'ordre de 1.000 triangles. Així, treballarem amb el valor aproximat de $1.000 pixels/triangle$.

Això ens dona la Taula 2.6 comparativa amb valors útils per l'usuari.

Taula 2.5: Comparativa de velocitats útils de tres models de tarja gràfica alt de gama.

Model	Velocitat teòrica (10^9 pix./s)	Velocitat útil (triangles/s)
nVidia GeForce 9800 GX2	12,2	12.200.000
nVidia GeForce 8800 Ultra	14,7	14.700.000
ATI Radeon HD 4870	12,0	12.000.000

Aquí també veiem que emprant una tarja gràfica moderna, ens podem esperar a velocitats de tractament de l'ordre de 12.000.000 triangles/s. És clar, aquest valor s'hauria de revisar a la baixa si la tarja gràfica no fos d'última generació, però en tot cas la totalitat de les targes gràfiques presents en el mercat al moment d'escriure aquestes línies sobrepassen les 1.000.000 triangles/s.

2.5 Resum de l'estat actual dels colls de botella

A partir de les dades obtingudes (i resumides a la Taula 2.6), podem arribar a unes conclusions sobre el punt més probable d'aparició

[10]Entre 0 píxels si s'està observant completament de costat, fins a aproximadament 1 Megapíxel si cobreix una extensió important de la pantalla.

d'un coll de botella: es tracta sense dubte de la infraestructura de xarxa en general, o bé del punt de connexió del nodus gràfic amb la xarxa.

S'indiquen aquests resultats sempre amb el *caveat* del cas en què s'empren altres materials, moment en què convé que l'investigador que vulgui emprar el sistema gràfic descrit en aquestes pàgines faci la seva pròpia determinació dels punts a vigilar en el seu sistema.

Taula 2.6: Velocitats típiques en diferents punts del sistema.

Punt	Velocitat útil típica (triangles/s)
Infraestructura de xarxa	250.000
Connexió de xarxa del nodus gràfic	250.000
Tractament per la unitat central	1.700.000
Bus de connexió	2.130.000
Tractament per la tarja gràfica	12.000.000

Capítol 3

Disseny del sistema

3.1 Disseny general

3.1.1 Requeriments dels usuaris

Un sistema d'aquestes característiques ha de poder prendre el seu lloc en dos contexts diferents:

1. El d'un cluster de càlcul, de tipus Beowulf cluster. En aquests, els nodus de càlcul estan connectats de manera propera a través d'una infraestructura de xarxa ràpida, el que permet emprar un dels dos paradigmes de programació més comuns en aquest tipus d'entorn: MPI (Message Passing Interface, [8]) i PVM (Parallel Virtual Machine, [18]).

2. El d'un conjunt de càlcul més distribuït, com podria ser un Grid o encara un COW[1], situació en què nodus de càlcul més distribuïts estan connectats emprant una infraestructura de xarxa en principi heterogènia, amb velocitats inferiors i emprant tècniques d'interconnexió variables.

En el cas d'un cluster, el llenguatge de càlcul sol ser C, C++ o bé Fortran. El cas d'un Grid pot ser molt més variat, incloent-hi opcions menys comunes com Java (Parallel Java, [10]) o alguns llenguatges de programació interpretats com Python (iPython, [16]) i Ruby (MPIRuby, [2]).

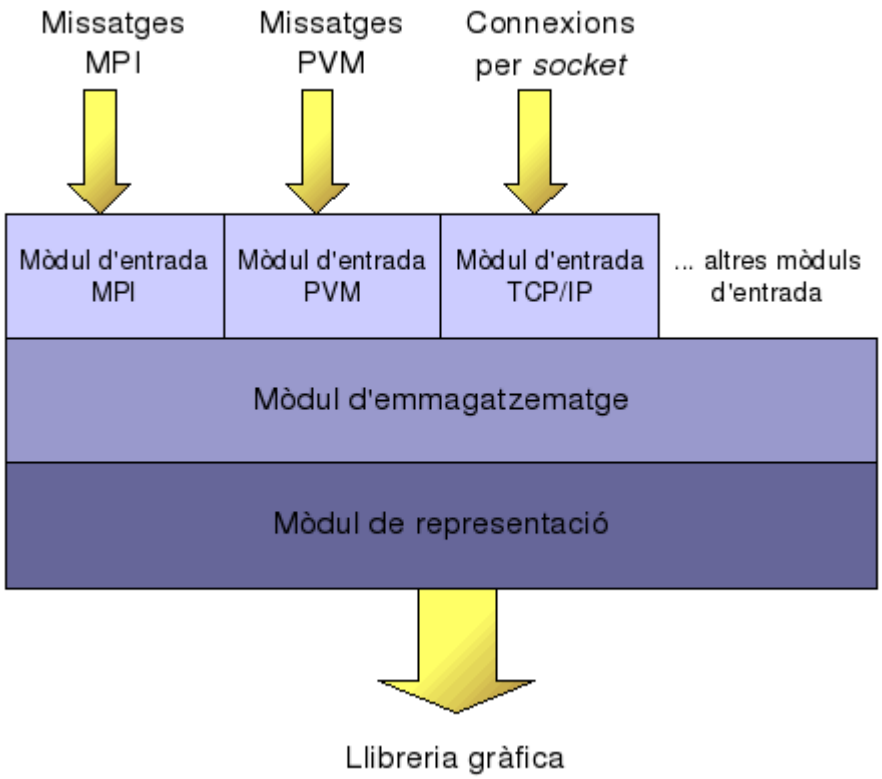
3.1.2 Estructura modular

Per permetre l'ús en situacions diverses de l'aplicació objecte d'aquest estudi, s'imposa una estructura modular i flexible. Així, s'ha separat en tres mòduls (Figura 3.1).

[1] Cluster Of Workstations

- Els mòduls d'entrada, consistent en elements que puguin acceptar respectivament (a) missatges MPI, i (b) missatges de text pla enviats a través d'una simple connexió per Socket TCP/IP.
- El mòdul d'emmagatzematge, que conté les llistes d'objectes gràfics a representar en tot moment, identificat a partir de la cadena d'identificació del nodus de càlcul que els hagi enviat.
- El mòdul de representació, màquina de *rendering* gràfic que tradueix les dades emmagatzemades en instruccions de la llibreria gràfica de baix nivell.

Figura 3.1: Disseny general del sistema.



3.1.3 Mòduls d'entrada

Inicialment, es preveu implementar tan sols els mòduls d'entrada MPI per nodus de càlcul connectats mitjançant aquest protocol, i TCP/IP per altres nodus que estiguin connectats amb una connexió de xarxa qualsevol.

Posteriorment, es podran implementar mòduls per PVM o altres sistemes d'interconnexió[2].

3.1.4 Mòdul d'emmagatzematge

Pel que fa al mòdul d'emmagatzematge, es preveu que aquest sigui un punt "calent"del sistema. De la seva eficiència dependrà la velocitat general del sistema, ja que els nodus de càlcul estaran contínuament enviant noves dades a incloure en les seqüències d'instruccions gràfiques a través dels mòduls d'entrada, i el mòdul de representació estarà també contínuament llegint-hi les dades per generar la sortida en pantalla.

Així, caldrà que l'estructura reservada a les dades sigui eficient a la vegada

1. En termes de l'espai necessari per cada assignació.
2. En termes del temps necessari per assignar i alliberar l'espai de memòria corresponent.
3. Pels processos d'afegit i lectura de les dades.

Aquests aspectes gairebé obliguen a emprar sistemes dinàmics enllaçats amb punters, evitant així el sobredimensionament associat a estructures estàtiques com taules. Notem que en principi es podria saber quants elements gràfics seran enviats per cada nodus de càlcul, però generar aquesta informació suposaria un esforç suplementari pel programador de l'aplicació de càlcul, el que es vol evitar.

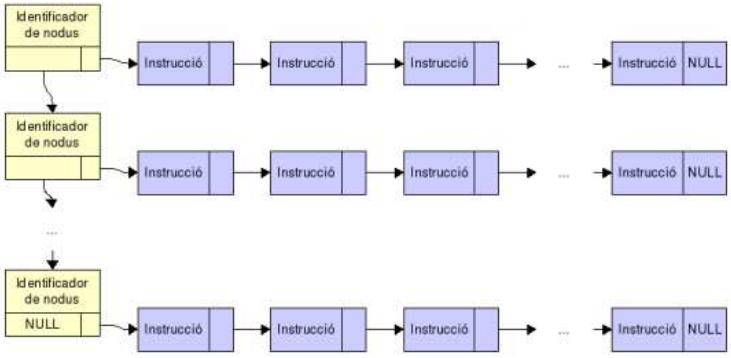
Com que les lectures i afegits d'instruccions tindran sempre lloc de manera seqüencial, es proposa l'ús de llistes enllaçades de manera unidireccional, per ser les més simples i suficients pels nostres propòsits (Figura 3.2). Caldrà, però, afegir alguna indicació suplementària per evitar que el mòdul de representació torni a llegir les dades de manera innecessària, si no s'han vist modificades mentre tant.

3.1.5 Mòdul de representació

Aquest mòdul és el que ha de gestionar la pantalla 3D. Implica la gestió de l'entorn OpenGL, generant la generació de crides a la llibreria gràfica de baix nivall a partir de noves llistes d'instruccions,

[2]Com per exemple mòduls amb base al paradigme de memòria compartida de tipus NUMA. Veure [11].

Figura 3.2: Una llista d'instruccions gràfiques per cada nodus de càlcul.



o de llistes d'instruccions que s'hagin vist modificades pels nodus de càlcul corresponents.

Per simplificar aquesta gestió, resulta apropiat l'ús de llistes d'instruccions precompilades en OpenGL. Així, es pot programar una seqüència de primitives gràfiques en la tarja gràfica en una única operació. No caldrà tornar-les a modificar a cada cegada que es canvii la reprentació de l'escena gràfica, per exemple mitjançant una rotació.

També caldrà que aquest mòdul gestioni el conjunt de les operacions d'interacció entre l'usuari i l'escena gràfica. Aquestes operacions haurien d'incloure al menys:

- La rotació manual, sobre dos eixos (vertical i horitzontal). Aquesta operació es mana amb el ratolí.
- La rotació automàtica.
- El desplaçament del centre d'interés.
- El zoom (apropar-se i allunyar-se).
- Canviar opcions com la transparència (veure objectes gràfics a través d'altres objectes, o no).
- Fer aparèixer els elements gràfics com la referència dels eixos, l'escala associat als colors, i altres.
- Sortir de l'aplicació.

3.2 Implementació

Existeixen diferents possibilitats d'implementació. De fet, hi han molts llenguatges de programació que tenen accés no només a llibreries de rutines de comunicació amb protocols MPI, PVM o directament a través de TCP/IP sockets, sinó que a més comencen a existir moltes possibilitats d'accés a llibreries de tipus gràfic que implementin OpenGL.

En el context d'un Màster en Programari Lliure , és clar que triarem en primera instància les opcions més obertes possibles. Val a dir, però, que aquestes també poden ser entre les més completes i a la vegada més optimitzades, ja que un col·lectiu prou ampli de programadors hi ha treballat. Concretament, les nostres escollides seran:

Sistema operatiu En principi, no es vol limitar a cap sistema operatiu en concret. Ara bé, l'escollida dels altres elements de programari aconsellen l'ús d'un sistema operatiu que sigui compatible amb els estàndards POSIX. Així, ens podrà convenir tant opcions estrictament del Programari Lliure com diferents versions de Linux, BSD o Hurd, però també opcions propietàries com Solaris o altres versions comercials de Unix.

Implementació MPI La implementació oberta OpenMPI és el que emprarem en el nostre cas per fer la implementació, encara que es podria convertir a Mpich[3] [9] sense massa esforç.

Implementació OpenGL La implementació més usada d'aquest protocol gràfic i que a la vegada és multiplataforma i de codi obert és Mesa 3D [13].

Llenguatge de programació Aquest punt podria ser motiu de debat. Ara bé, considerant que l'estructura modular es presta a una programació orientada a objectes d'una banda, i de l'altra que l'ús de les diverses llibreries és facilitada a partir d'un llenguatge de programació de la família del C, s'ha decidit fer una primera implementació del programari en llenguatge C++.

Emprant el clàssic compilador GNU g++, s'obté a la vegada un codi raonablement optimitzat i la possibilitat de portar el sistema a les moltes plataformes per les que existeix aquest compilador. Mac OS-X en seria un exemple.

Finalment, també cal tractar el tema de la concurrència. Sobretot els mòduls d'entrada han de tenir un alt nivell de multiprocessa-

[3] Mpich també és lliure.

ment, per donar servei a cada un dels clients sense frenar el client. A la vegada, el mòdul de representació haurà de poder dur a terme la seva activitat seguint les entrades de l'usuari (tecles, moviments del ratolí) gairebé en temps real. Sinó, si aquest tingués una impressió de demora a cada monipulació la seva *user experience*[4] no seria molt bona.

Per aquest motiu, es proposa:

- Executar el mòdul de representació dins el camí d'execució principal de l'aplicació.
- Executar un *POSIX thread* (p-thread) [14] independent per cada mòdul d'entrada.

  Aquesta solució és més "lleugera"en termes de recursos temporals i de processament que l'alternativa de generar processos independents mitjançant el manament *fork()*. Al mateix temps, presenta l'avantatge de permetre compartir espais de memòria entre els threads, comunicant així directament les dades entre mòduls.

  Encara que la possibilitat de memòria compartida (*shared memory* o *shmem*) també existeix entre processos, el seu funcionament és molt més feixuc.

3.3 Llenguatge de descripció de les instruccions gràfiques

Com s'ha comentat anteriorment, s'ha pres la decisió de codificar les instruccions gràfiques mitjançant cadenes de caràcters, per ser la solució més flexible i accessible a una àmplia gama de clients possibles.

El llenguatge gràfic OpenGL presenta una llista prou llarga de primitives gràfiques. En la implementació Mesa 3D es troben repartides entre les sub-llibreries:

- **gl.h**, que és la llibreria principal i que conté la definició de les estructures de dades i constants així com la definició de les primitives gràfiques segons l'estàndard OpenGL versió 1.1 (creació de vèrtex, segments, ...).
- **glu.h**, amb primitives gràfiques més potents (creació de corbes i polígons, ...).

[4]Per emprar una terminologia de la casa Apple.

- **glut.h**, que conté sobretot primitives d'interacció amb l'usuari (definició de tecles, respostes a l'input d'usuari). Aquestes primitives seran útils per la programació del mòdul de representació.

De totes aquestes primitives, tan sols una selecció relativament estreta ja dona prous recursos a l'investigador científic per programar la seva escena gràfica. En tot cas, per fer una primera implementació, se n'han seleccionat les següents i que formarien l'API[5] gràfica de cara a l'usuari final.

---

glBeginPoints Començar una seqüència que conté vèrtex. A partir d'aquest manament, s'espera una sèrie de manaments de definició de color o de vèrtex. Cada manament de definició de color s'aplica a tots els vèrtex següents, fins a finals de seqüència o bé un nou manament de definició de color.

---

glBeginLines Començar una seqüència de segments. El seu funcionament és com l'anterior, però aquesta vegada cada parell de vèrtex consecutius es consideren els extrems d'un segment. En conseqüència, cal que el nombre de vèrtex de la seqüència sigui divisible per 2.

---

glBeginTriangles Començar una seqüència de triangles. En aquest cas, cada triplet de vèrtex es considera la definició d'un triangle. Cal que el nombre de vèrtex de la seqüència sigui divisible per 3.

---

glEnd Acabar una seqüència de qualsevol tipus.

---

glColor3f Definir el color actual, a través de 3 números amb punt flotant. Aquests tres números seran compresos en $[0..1]$, i representaran els valors respectius dels canals RGB (vermell, verd i blau).

---

glVertex3f Emplaçar un vèrtex, definit a través de 3 números amb punt flotant. Aquests números corresponen a les seves coordenades XYZ. Cal tenir en compte que el referencial és "de mà esquerra": els eixos X i Z formen el pla horitzontal, mentre que Y és l'eix vertical.

---

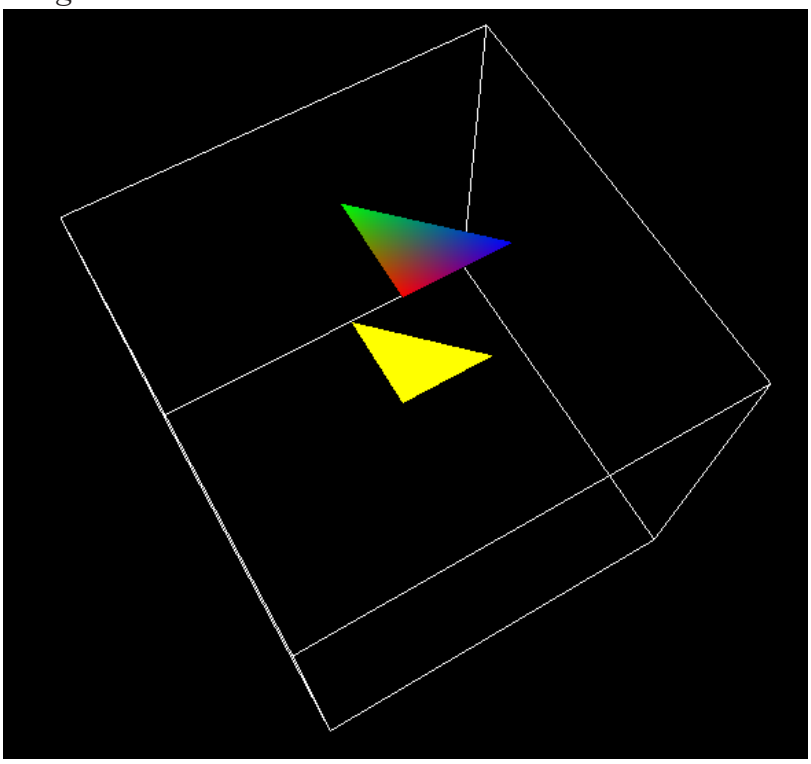
glClear Esborrar la llista d'instruccions gràfiques corresponents a aquest nodus de càlcul.

---

[5]Advanced Programmer's Interface.

Queda clar que aquesta selecció inicial de possibilitats es pot estendre fàcilment per incloure noves primitives, per exemple pel tractament de texts, de la transparència, i altres possibilitats.

Figura 3.3: Resultat de l'enviament de la seqüència de creació de dos triangles.



Per exemple, la seqüència següent definiria dos triangles sobreposats, el primer d'un sol color (groc), i el segon amb un degradat triple cap a cada un dels seus vèrtex (veure Figura 3.3).

```
glBeginTriangles
glColor3f 1.0 1.0 0.0
glVertex3f 0.0 0.0 0.0
glVertex3f 0.5 0.0 0.0
glVertex3f 0.0 0.0 0.5

glColor3f 1.0 0.0 0.0
glVertex3f 0.0 1.0 0.0
glColor3f 0.0 1.0 0.0
glVertex3f 0.5 1.0 0.0
glColor3f 0.0 0.0 1.0
glVertex3f 0.0 1.0 0.5
```

glEnd

Capítol 4

Resultats de la implementació del sistema

4.1 Implementació

El programari MPI^{GL} s'ha implementat en la seva totalitat en llenguatge C++. Degut a la seva estructura modular, s'ha dissenyat emprant un estil orientat a objectes, i reservant una classe per a cada funcionalitat:

Launcher Aquesta classe és definida en el fitxer principal *mpiGL.C*. Correspon a la mateixa l'aplicació gràfica, i la seva funcionalitat principal és:

1. Interpretar les diferents opcions de línia de manaments, i que controlen els serveis que ofereix l'aplicació.
2. Engegar el mòdul corresponent a cada un dels serveis, a més del repositori central de dades.

Aquest funcionament resulta prou flexible perquè la introducció de nous mòduls requereixi la modificació tan sols d'aquest *llençador* per fer-los accessibles als usuaris.

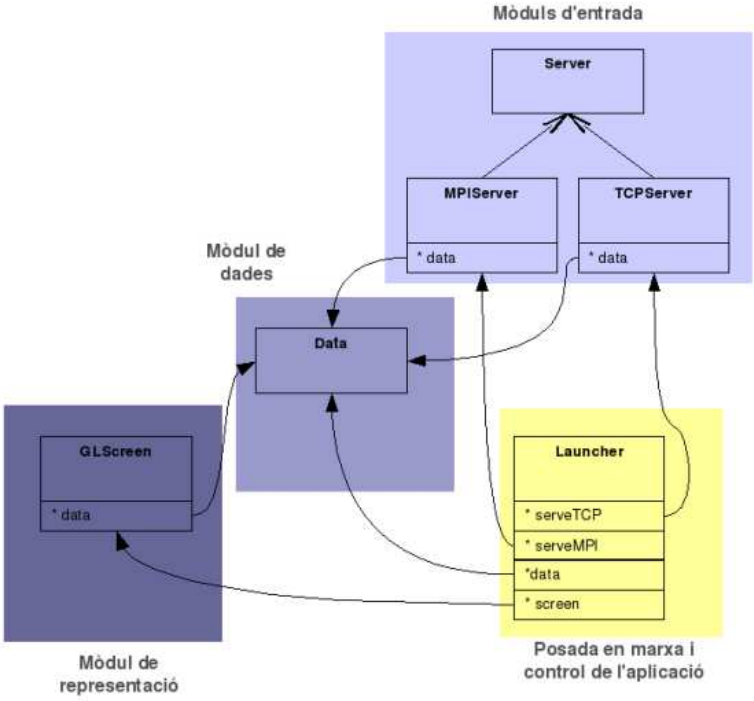
Tan sols proposa dos mètodes: el constructor per engegar el conjunt de sistema

```
Launcher::Launcher(int argc, char *argv[])
```

i una funció de terminació per assegurar el correcta tancament del conjunt de serveis tant gràfics com de xarxa

```
void Launcher::shutdown()
```

Figura 4.1: Diagrama de classes de l'aplicació.



Server Aquest objecte abstracte correspon al prototip del mòdul d'entrada. De moment, no s'ha previst cap mètode abstracte en particular que s'hagués d'implementar en tots els seus descendents, ja que aquests tenen característiques i maneres de funcionar prou distints. Així, la seva declaració és simplement

```
class Server {};
```

MPIServer Aquest descendent de *Server* és el mòdul que acceptarà els missatges amb instruccions gràfiques des dels nodus de càlcul connectats mitjançant el protocol MPI.

Donada la naturalesa asíncrona dels missatges MPI, es pot proporcionar aquesta funcionalitat mitjançant un únic *thread*. Una vegada establerta, aquest camí d'execució ha d'esperar de manera infinita que arribi algun missatge MPI, tractar-lo, i continuar amb el pròxim. La cua d'entrada de missatges assegura la sincronització temporal dels accessos a les dades.

Donada la independència del seu funcionament respecte a la resta de l'aplicació, els seus principals mètodes són en primer lloc el constructor

```
MPIServer::MPIServer(Data *d, bool b_v)
```

en què s'observa el pas d'un punter sobre l'objecte principal contenidor de dades (primer paràmetre). Caldrà doncs fer la inicialització d'aquest objecte de dades en primer lloc, **abans** de la creació del servei MPI.

Per altra banda, el segon paràmetre, booleà, modula en seu funcionament, indicant si volem que sigui més (true) o menys (false) verbós en termes de missatges emesos al terminal de control.

Aquest constructor té la responsabilitat de la creació del *p-thread* corresponent, mitjançant una funció *callback* sobre la única instància existent d'aquesta classe d'objecte.

També inclou el mètode de terminació

```
void MPIServer::shutdown()
```

amb el tancament correcte del servei i destrucció del *thread* corresponent.

TCPServer Aquest és el servidor accessible directament a través d'una connexió TCP/IP. El seu funcionament és anàleg al de *MPIServer*: un únic *thread* dóna servei a diverses connexions TCP/IP de manera successiva.

Val la pena detallar aquest funcionament. Existeixen diferents maneres de concebre un servidor TCP/IP perquè pugui donar servei a una sèrie de clients de manera concurrent, o gairebé concurrent. Com que es tracta, en general, de d'ordinadors monoprocessadors, sempre existirà un elements de seqüenciació necessària, però que degut a la curta durada de processament de cada petició donarà una impressió de simultaneïtat. Són:

1. Un únic camí d'execució, que tracta successivament cada una de les connexions entrants. És la solució més simple de programar, encara que presenta l'inconvenient d'una possible espera per algun client, si el seu intent de connexió arriba al servidor de manera concurrent a d'altres connexions entrants.
2. Un camí d'execució principal, que deriva un *thread* separat per cada connexió. Aquest mecanisme és emprat molt sovint en llenguatges interpretats com el Java, i presenta a la vegada l'avantatge de ser un mecanisme relativament "leuger"en termes de recursos de memòria necessaris pel

seu funcionament[1], i que separa l'execució dels diferents camins que donin serveis als clients individuals. D'aquesta manera, cada client té la impressió de disposar de la totalitat del potencial de resposta del servidor.

3. La clonació de processos, generant un nou procés complet per cada connexió entrant mitjançant una crida fork() . Aquest és el sistema més "pesant"en termes de requeriments de memòria i de processament, motiu pel qual no se sol emprar molt sovint avui en dia[2].

Encara que existeixi avui en dia un ampli debat sobre la conveniència d'un o altre mètode, es pot dir que el consens seria dir que, a mesura que cada connexió individual requereix més tractament (i temps de processament), més convé passar de la primera solució a la segona, o fins i tot a la tercera. Així:

- Reservarem la clonació de processos a casos en què el client requereix una connexió llarga, persistent, i que demani molts recursos de processament. Un cas en podria ser una connexió a un servidor d'aplicacions que demani un càlcul d'una certa envergadura.
- La creació de *threads* pot convenir a casos de tipus general, en què possiblement caldrà que el client esperi durant la connexió perquè el servidor accedeixi a dades ubicades en el sistema de discs. Uns exemples en podrien ser servidors HTTP, de fitxers, etc.
- Finalment, el tractament directe de totes les connexions per part d'un únic camí d'execució pot ser interessant en el cas en què cada connexió provoca una quantitat relativament reduïda de feina, que no impliqui accessos a disc pel servidor, i en el qual un procés no interactiu (i doncs de tipus *batch* no requereix que el servidor esperi cap client individual provocant així demores pels altres clients en espera d'obtenir una connexió.

En aquest cas concret, ens ha semblat que estem en aquest últim cas, i doncs hem decidit tractar totes les connexions TCP/IP entrants a través d'un únic *thread*.

[1]Amb un *thread*, no es duplica tot el context d'execució, contràriament a la clonació de processos.

[2]Aquest comentari s'hauria de modular en el cas dels *microkernels*, situació en la qual generar un nou procés pot ser menys consumidor de recursos que en un nucli monolític com el de Linux.

Un motiu suplementari per aquesta decisió és la possibilitat d'adaptar l'esquema de *MPIServer* de manera molt directa al tractament del TCP/IP.

Així mateix, els mètodes proposats per *TCPServer* són anàlegs als anteriors, consistent en un constructor

```
TCPServer::TCPServer(Data *d, bool b_v, int _tcp_port)
```

en què, a més d'un punter sobre el mòdul de dades i la indicació de verbositat, a més s'adjunta el port TCP a emprar per escoltar les connexions entrants, i

```
void TCPServer::shutdown()
```

per tancar correctament el servidor, alliberant el port TCP per usos ulteriors.

Data Aquest és l'objecte que conté les dades corresponents a les instruccions gràfiques emesos per cada client o nodus de càlcul. També s'ha aprofitat per desar algunes altres informacions que calia compartir entre mòduls d'entrada i de sortida.

Per seguir millor aquesta discussió, pot ser d'interès pel lector consultar la Figura 3.2 a la pagina 26.

En aquesta implementació, el seu mètode constructor té una interfície més senzilla, ja que es crear un conjunt de dades inicialment buit:

```
Data::Data()
```

Cada una de les connexions entrants, sigui a través de MPI, sigui a través de TCP, té una identificació en forma de cadena de caràcters. Aquest serà el que farà servir el client per identificar-se, per més facilitat pel programador de l'aplicació que faci servir el client. D'altra banda, cal poder fer la relació entre aquesta identificació i la posició de la llista d'instruccions gràfiques corresponents. Per això, s'ha dotat del mètode

```
int Data::get_connection(char *name)
```

que torna el valor constant pre-definit NOTFOUND (actualment definit en mpiGL.H amb el valor -1) si no existeix una cadena d'instruccions amb aquella identificació. En aquell cas, es pot crear emprant

```
int Data::create_connection(char *name)
```

que retorna l'índex numèric corresponent a la posició d'aquesta cadena d'instruccions en el sí de la base de dades.

Com que periòdicament, pot ser interessant pel client emetre la instrucció *glClear* per eliminar les instruccions contingudes en una antiga llista, i crear-ne una de nova, s'ha implementat el mètode

```
void Data::clear_list(int index)
```

que emprant l'índex de la llista d'instruccions per identificar-la n'elimina totalment el seu contingut.

Per afegir noves instruccions a una llista, els mòduls d'entrada poden emprar el mètode

```
void Data::add_line (int index, char *s)
```

en què s és la cadena de caràcters que conté la nova instrucció a afegir a la llista. Val a dir que aquesta inserció s'efectua sempre en posició terminal; o sigui que les instruccions presents en una llista s'hi trobaran en l'ordre cronològic en què es varen afegir.

Quan un nodus de càlcul, client d'un dels mòduls d'entrada, dóna per completa la seva llista d'instruccions mitjançant la instrucció gràfica *glEnd*, cal enviar aquesta informació cap a la llibreria gràfica de baix nivell en el mòdul de representació gràfica. Això es fa mitjançant el procés de *compilació* de la llista d'instruccions, amb el mètode

```
void Data::compile_list(int index)
```

S'ha escollit aquest funcionament amb dos temps perquè així no cal que existeixi una interacció contínua entre el mòdul de dades i el de representació; tan sols es fa el pas de dades entre l'un i l'altre quan el client completa la introducció d'una seqüència completa d'instruccions.

GLScreen Aquesta última classe correspon al mòdul de representació gràfica. Com a tal, és la responsable de la gestió directe de la llibreria de rutines gràfica OpenGL.

Per la seva creació, la constructora

```
GLScreen::GLScreen(Launcher *l, Data *d,
                   bool b_v,
                   int _screen_width, int _screen_height)
```

necessita el pas com a paràmetres d'un punter sobre el conjunt de dades, però també d'un altre sobre el mecanisme de posada en marxa del programa, de tipus (Launcher *). El motiu és que en el moment en què l'usuari vol tancar l'aplicació, aquest ordre (tecla Escape) és rebut per una instància d'aquesta classe *GLScreen*. Aquest haurà de posar en marxa el procediment de tancament dels servidors i del conjunt de l'aplicació a través de l'execució del mètode *shutdown* del Launcher.

Els altres mètodes d'aquesta classe *GLScreen* fan totes referència a diferents funcions gràfiques, com la creació del conjunt de l'escena, el marc (*box*) cúbic pel conjunt d'elements gràfics, la presentació de les indicacions textuals en pantalla (*hint*), etc.

Val a dir que, degut al fet que disposem de totes les informacions sobre les instruccions gràfiques en una instància de la classe Data, es pot reemplaçar a voluntat aquesta classe de representació gràfica GLScreen *per altres mòduls frontals. A mode d'exemple, descrivim a continuació dues possibilitats que podrien ser interessants desenvolupar en futures versions d'aquest programari.*

AutoCAD_ Export Aquest mòdul també es connectaria a la instància de l'objecte Data per tal de fer-ne una exportació en un fitxer a disc en format AutoCAD[3], programat per exemple per fer-ne l'exportació a intervals regulars durant l'execució del conjunt de càlcul. Així, l'investigador acabaria amb un registre puntual (*snapshot*) de l'estat d'avançament del seu càlcul.

SVG_ HTTP_ Server La concepció d'aquest mòdul de representació seria una mica diferent de l'anterior: en aquest cas seria un servidor HTTP seria accessible a través d'un navegador d'Internet capacitat per representar imatges en format *image/svg+xml*[4] Així producció del servidor no seria un fitxer de text codificat en HTML, sinó la construcció d'una representació bidimensional, creada de manera dinàmica a partir de les dades de la instància de Data.

4.2 Dificultats en la implementació

Durant la implementació d'aquesta aplicació, varen sortir diferents dificultats, relacionades sobretot amb el seu caràcter d'aplicació amb més d'un camí d'execució.

[3]Es pot obrir aquest format emprant el Programari Lliure QCad.
[4]Com, per exemple, Mozilla Firefox.

4.2.1 Funcions *callback*

La primera dificultat fa referència al funcionament de les pantalles gràfiques que emprin OpenGL i llibreries com Mesa 3D. En aquests, cal establir una comunicació entre el nucli de l'aplicació i el subsistema gràfic:

- En el sentit programa principal → sistema gràfic, la comunicació s'efectua executant crides a funcions de la llibreria gràfica. Seguint el costum de les llibreries d'extensió en el món Unix, els noms de les funcions gràfiques van prefixades amb el nom de la llibreria en qüestió. Així, per exemple, el programa principal pot executar la funció *glutPostResdisplay*[5] per demanar al sistema gràfic refrescar la seva pantalla.

- En el sentit sistema gràfic → programa principal, la situació és més complexa ja que en un principi el sistema gràfic no està concebut per treballar amb cap programa en concret, sinó amb un enfocament més genèric. Per aquest motiu, no sap directament què fer per tractar els diversos esdeveniments d'usuari.

  La manera d'implementar aquestes accions és l'ús de *callback functions*. Aquestes són funcions pròpies del programa principal, i que al moment de posar en marxa el sistema gràfic se li senyalen a través de la seva adreça en la memòria (estructura de punter). Per exemple, per indicar al sistema gràfic quina funció emprar per gestionar les accions del ratolí, en el nostre cas s'ha connectat mitjançant:

  ```
  glutMouseFunc(&gl_mouse_button);
  ```

  Així, la funció *gl_ mouse_ button* del programa principal es cridarà a cada vegada que la finestra OpenGL detecti que l'usuari ha pitjat un botó, i serà la seva responsabilitat tractar aquesta acció.

Aquest funcionament s'ha concebut per un entorn de programació no orientat a objectes, del llenguatge C. La seva conversió a la POO emprant C++ ha posar algun problema, degut al caràcter dinàmic de la instanciació dels objectes: no es pot indicar a la llibreria gràfica que ha d'emprar una funció callback que faci part d'una instància d'un objecte ja que aquest no es pot acabar d'ubicar en la memòria durant la compilació de l'aplicació.

La solució que s'ha proposat a aquest problema en el context d'aquesta aplicació és l'ús d'identificadors d'objectes *estàtics*. Així, una

[5]Pertany a la llibreria GLUT.

vegada creada una instància de la classe GLScreen, es pot "penjar" al punter estàtic declarat per

```
static GLScreen *current_screen;
```

Totes les rutines gràfiques, siguin o no de la llibreria gràfica OpenGL, podran ubicar i accedir a aquesta adreça de memòria en temps d'execució.

L'inconvenient d'aquest sistema és que tan sols podrà existir una única instància de la classe OpenGL: només podrem disposar d'una única finestra gràfica. Aquest esquema es podria millorar creant una taula de punters estàtics per poder gestionar més d'una instància de GLScreen a la vegada, si en el futur se'n veiés la necessitat.

Es va presentar un problema anàleg pel que fa als *threads* corresponents als servidors, i es va resoldre de la mateixa manera.

4.2.2 Accés concurrent a recursos

Existeixen diferents moments en què dos o més camins d'execució podrien potencialment voler accedir als mateixos recursos, un al menys dels quals voldria modificar el recurs. Concretament, es tracta de la gestió de les llistes d'instruccions gràfiques en la classe Data, al que han d'accedir:

1. El mòdul d'entrada MPI, en lectura però també en escriptura (modificació). Aquest mòdul pot afegir noves llistes (quan es connecti per primera vegada un client), esborrar instruccions d'una llista (quan un client executi *glClear*) o afegir-ne de noves.

2. El mòdul d'entrada TCP (i futurs mòduls d'entrada), amb les mateixes condicions.

3. El mòdul de representació GLScreen, tan sols en lectura. També serà el cas de futurs mòduls de sortida.

Aquesta situació presenta un perill potencial per la coherència de les dades. La seva resolució depenia de la correcte identificació de la localització dels possibles conflictes en el codi, i l'aplicació d'una tècnica d'accés exclusiu. En el nostre cas, ens hem decantat per l'ús dels *mutex*[6], degut a la seva senzillesa d'ús i inclusió en la llibreria *p-thread* del compilador gcc.

S'ha creat una taula d'estructures de tipus *pthread_ mutex_ t*, una per cada connexió entrant, en la classe Data:

[6] MUTual EXclusion.

A cada vegada que s'accedeix a una de les llistes d'accés en concret, es "clava" el *mutex* abans de la secció de codi que llegeixi o modifiqui la llista, i es tornar a "desclavar" a la sortida de la secció perillosa. Per exemple, si la variable *index* conté el número de la llista en qüestió, podem trobar en Data:

```
pthread_mutex_lock(&conn_mutex[index]);

        // accedir a la llista conn[index] ...

pthread_mutex_unlock(&conn_mutex[index]);
```

Pel que fa a l'estructura del conjunt de les connexions, s'ha implementat un *mutex* únic per tota l'aplicació, *list_ mutex*. Aquest serveix per evitar que dos mòduls d'entrada[7] intentin crear una nova connexió emprant un mateix índex.

[7] Per exemple, els servidors MPI i TCP.

Capítol 5

Anàlisi d'alguns casos pràctics d'aplicació

5.1 Simulació de la col·lisió entre galàxies

5.1.1 Descripció

El problema de la col·lisió entre galàxies es pot tractar com un problema de dinàmica, en què cal projectar les posicions $\vec{P}$ (3 coordenades) i velocitats $\vec{V}$ (3 coordenades) d'un nombre d'estrelles al llarg d'un eix temporal. Es contemplen els efectes de la gravitació, i possiblement també l'existència de col·lisions entre estrelles o fins i tot l'evolució pròpia d'estrelles individuals en el model (2 a 3 coordenades més) que poden variar en funció de les trajectòries de les galàxies [20]. Una extensió d'aquest problema, a més gran escala, és la modelització de la formació de les primeres estructures de l'univers [4].

En primera aproximació, pot semblar un problema molt adient per la computació paral·lela, ja que es poden repartir les estelles entre els nodus, per exemple en proporció de la potència de càlcul de cada un[1]. A la pràctica, però, forma part dels problemes dits *fine-grained*, ja que la posició de cada estrella a cada etapa depèn de la de totes les altres estrelles. Així, cal repartir la nova informació generada per cada nodus a tots els altres nodus a finals de cada etapa del càlcul, segons l'algoritme aproximat següent:

```
* per a cada nodus
  * per a cada estrella
    * calcular els efectes gravitacionals d'altres estrelles
    * calcular la trajectoria pel proxim interval dt
```

[1]També existeixin altres maneres de repartir les unitats de treball, basades en la mètrica de l'espai estudiat. Veure [21].

```
  * posar al dia posicio i velocitat
 * per a cada estrella
  * calcular altres efectes (fusio, intercanvi de temperatura, magnet
  * posar al dia les dades de l'estrella segons convingui
 * intercanviar dades generades amb altres nodus
```

En la present anàlisi, prendrem les dades següents:

- N nombre de nodus de càlcul
- S nombre d'estrelles assignada a cada nodus
- D nombre total d'estrelles o *data set size*: $D = S \cdot N$
- Q volum de dades (en bytes) de nova informació per cada estrella i per iteració del càlcul

5.1.2 Càlcul del volum de dades

Existeixen dos tipus de càlcul a fer en aquest cas.

Un és del volum de dades a intercanviar a cada iteració entre nodus de càlcul. Aquesta haurà de contemplar tota la informació que ha variat per cada estrella, i doncs en cas d'emprar nombres amb punt flotant sobre 64-bits, es pot estimar Q com

$$Q \approx 8 \cdot 9 = 72 bytes$$

A cada iteració, cada nodus haurà de transmetre de les seves S estrelles, o sigui una càrrega sobre la xarxa de $S \cdot Q$. Aquí se'ns poden presentar tres possibilitats; en les dues primeres s'enviaran les dades físiques al nodus de càlcul (que haurà de preparar la transformació a dades gràfiques), mentre que en la tercera els nodus de càlcul prepararan ells mateixos les dades gràfiques i les enviaran al nodus gràfic:

Cas (a): transmissió punt a multipunt eficient En aquest cas, cada nodus transmetrà tan sols una única vegada les seves dades, a través d'un missatge de multidifusió que serà rebut per tots els altres nodus a la vegada. El volum global de dades enviades per la xarxa serà:

$$Vol = N \cdot S \cdot Q$$

el que és proporcional a

$$O(N \cdot S) = O(D)$$

La presència d'un nodus gràfic, frontal MPI^{GL} al sistema no hi canvia res - a condició que aquest pugui captar i interpretar directament els missatges originaris dels altres nodus. En aquest supòsit, el nodus gràfic podrà capturar totes les dades sense afegir més càrrega a la xarxa.

Cal remarcar que no és possible fer-ho directament amb la implementació escollida per MPI^{GL} , sinó que s'hauria de modificar l'aplicació creant un nou mòdul d'entrada específic per aquest problema.

Cas (b): transmissió punt a punt, o multipunt poc eficient Aquest cas es pot produir si estem en presència d'una xarxa heterogènia, o de nodus que no participin en un mateix segment d'adreces IPv4 o IPv6. Podria ser el cas d'un *grid* o bé *COW*, més rarament el d'un cluster de càlcul dedicat. Donades aquestes suposicions, és possible arribar al cas pitjor en termes d'eficiència, que seria la necessitat de transmetre les dades de cada nodus als $(N-1)$ altres nodus de manera individual.

El volum global de dades enviades per la xarxa seria aleshores:

$$Vol = N \cdot (N-1) \cdot S \cdot Q$$

o sigui proporcional a

$$O(N \cdot (N-1) \cdot S = O((N-1) \cdot D)$$

En aquest cas, afegir un nodus gràfic MPI^{GL} suplementari sí que tindria un efecte, la d'augmentar el nombre de transmissions a partir de cada nodus de $(N-1)$ a N, i el volum global de dades a transmetre seria de l'ordre de

$$O(N \cdot D)$$

En aquest cas també caldria escriure un mòdul d'entrada específic per MPI^{GL} , que converteixi les dades de posició en instruccions gràfiques que després es podrien injectar en el mòdul Data.

Cas (c): transmissió tan sols d'indicacions GL En aquest últim supòsit, els nodus de càlcul no enviaran les dades físiques al nodus gràfic, sinó tan sols les instruccions de representació gràfica. Aquest funcionament té alguns avantatges:

1. En els dos primers supòsits (a) i (b), el nodus gràfic és el que hauria de fer una sèrie de càlculs per transformar

coordenades físiques a gràfiques, el que podria suposar un coll de botella en el cas d'escalar les dimensions del *problem set*.

2. Si és el nodus de càlcul que fa la transformació a dades gràfiques, podrà seleccionar tan sols una part de les dades (per exemple, tan sols la posició). També pot efectuar una tria sobre les dades, enviant-ne només una part (per exemple, una sola estrella per representar un conjunt de vàries).
3. Simplificar la programació del nodus gràfic, ja que no ha de tenir coneixement precís del problema tractat - pot ser més genèric.

Per contra, té l'inconvenient:

1. De posar més càrrega sobre els nodus de càlcul, evidentment en detriment dels recursos disponibles pel seu funcionament previst. Val a dir que és una sobrecàrrega lleugera, relativament al pes del càlcul físic a efectuar per cada nodus.

Amb una codificació textual - i limitant la precisió dels camps amb punt flotant - ens en podem sortir amb 40 a 50 caràcters, el que no representa un sobrecost massa elevat i presenta l'avantatge de ser fàcilment codificable tant pel nodus de càlcul com intel·ligible pel nodus gràfic.

En tots els casos, el nombre de bytes rebut per cada estrella serà $Q \in [25, 50]$, i a cada iteració el nodus gràfic haurà de rebre

$$Vol = N \cdot S \cdot Q$$

o sigui un factor d'escala proporcional al nombres d'estrelles, $O(D)$.

5.1.3 Implementació real del cas

Per fer-ne la implementació, s'ha escollit la seva programació en llenguatge C, emprant el protocol de comunicació MPI. Cada nodus de càlcul enviarà doncs instruccions gràfics GL directament a l'aplicació a través del mòdul d'entrada *MPIServer*.

Per major simplificació, s'ha escollit representar cada galàxia com un conjunt d'estrelles individuals, i cada estrella com un únic punt (glVertex). S'ha assignat un color concret a cada galàxia, respectivament el vermell per la galàxia A fix en l'espai, i blau per la galàxia B, mòbil i que s'aproxima gradualment a la galàxia A.

Per poder distingir més fàcilment la provinença de cada estrella individual, s'ha codificat la seva posició inicial mitjançant un degradat; les estrelles que inicialment es trobin a proximitat del centre de cada galàxia tenen colors més clars, mentre que les estrelles que hagin començat a la perifèria d'una de les galàxies tenen un color més profund.

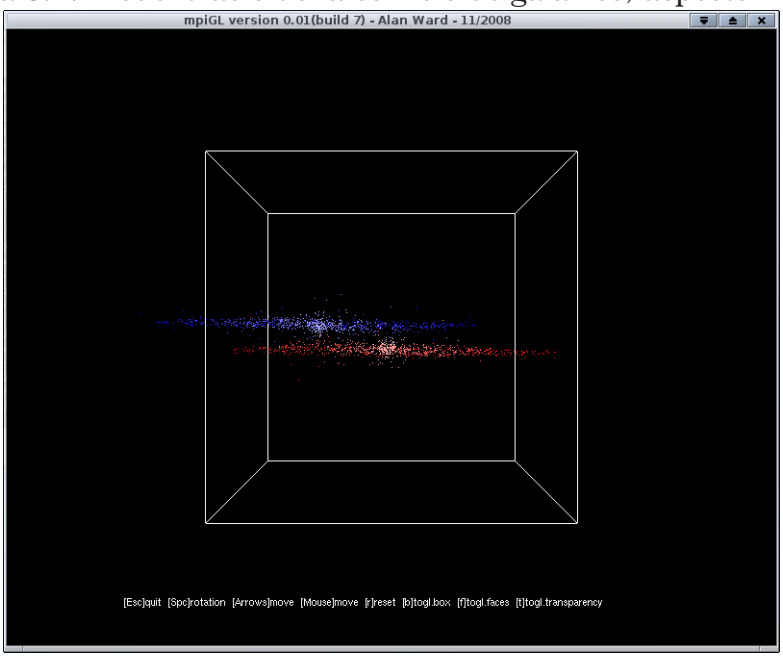
El procés de càlcul consisteix en iterar les dues etapes

1. Calcular la suma de les forces gravitacionals exercides sobre cada estrella. En el context d'aquesta aplicació de demostració, s'han negligit tots els altres efectes: col·lisions, interacció magnètica, etc.

2. A partir de l'acceleració calculada per cada estrella, posar al dia la seva velocitat i posició mitjançant la tècnica *d'interpolació d'Euler*. Encara que aquesta no sigui molt precisa[2], és suficient per les necessitats d'un programa de demostració.

S'ha començat per validar el model, emprant una única galàxia i verificant que les estrelles mantenien trajectòries aproximadament circulars durant un cert nombre d'iteracions.

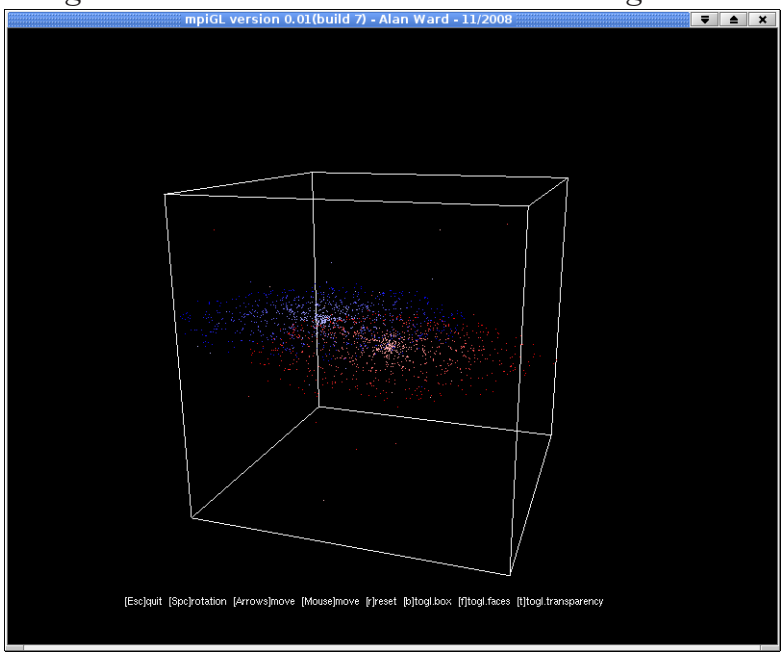
Figura 5.1: Modelització de la col·lisió de galàxies, aspecte frontal.



[2]En un cas més desenvolupat, s'hauria de preferir alguna variant de la *interpolació Runge-Kutta*.

Finalment, l'execució de la simulació ha donat lloc als resultats gràfics següents (Figures 5.1 i 5.2). Corresponen a dues captures de pantalla (instantànies), però cal tenir en compte que l'investigador podia girar lliurement l'escena per poder examinar aspectes individuals de la simulació.

Figura 5.2: Modelització de la col·lisió de galàxies.



En les dues figures, ja es comença a apreciar el fenomen d'ejecció d'algunes estrelles degut a les complexes interaccions gravitacionals entre les dues galàxies.

5.2 Modelització de la transmissió de calor en un sòlid

5.2.1 Descripció

El problema de la transmissió de calor és un exemple típic de l'aplicació del càlcul informàtic a la modelització de fenòmens de transport[3]. Avui en dia s'empren conjunts de càlcul en paral·lel tant per dissenyar estructures amb petites (vàlvules, canalitzacions, motors) i grans dimensions (turbines de generació d'electricitat, aeronaus),

[3]Val a dir que encara que en un problema de transport de calor en un objecte fix sembli en primera aproximació que no existeix cap fluid, en el tractament del flux de calor apareixen el mateix tipus d'equacions.

com també per estudiar els efectes de construccions humanes sobre el medi ambient.

Encara que la majoria dels codis de Computational Fluid Dynamics (CFD) són de caire comercial, també en el món del Programari Lliure s'han començat a proposar algunes alternatives obertes (veure [12] i [19]). Aquestes presenten no només l'avantatge de la seva absència de cost directe[4], sinó a més les característiques desitjables de *verificabilitat* i *adaptabilitat* a la situació exacte a la que s'enfronta l'investigador.

En aquest cas, ens centrarem en una aplicació simple, que seria l'estudi d'un dissipador tèrmic destinat a un processador.

Aquest tipus de giny es presenta en la forma d'un objecte compacte, generalment amb aletes, i que avui en dia es pot construir emprant una combinació de metalls:

- **Coure**, generalment en contacte directe amb el CPU, per la seva alta capacitat de transport de calor. La conductivitat tèrmica és $k_{Cu} = 401W \cdot m^{-1} \cdot K^{-1}$

- **Alumini**, més barat i més lleuger que el coure, però amb una capacitat de transport de calor inferior: la seva conductivitat tèrmica és $k_{Al} = 237W \cdot m^{-1} \cdot K^{-1}$.

- **Acer**, pel seu baix preu i resistència física, en aplicacions menys crítiques. La seva conductivitat tèrmica és molt més baixa: $k_{Fe} = 80.4W \cdot m^{-1} \cdot K^{-1}$

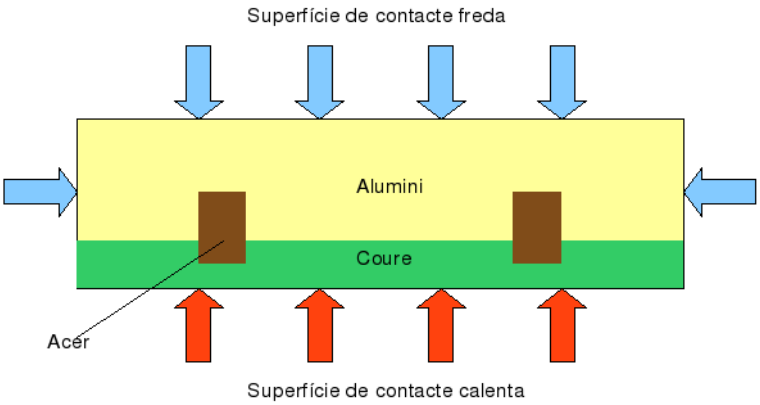
En el nostre cas, ens centrarem amb una estructura simplificada (que es presenta en tall en la Figura 5.3), en què una base de coure està en contacte amb el processador (superfície de contacte calenta), mentre que una capa superior en alumini està en contacte amb l'aire fred tant en la seva part superior com els laterals.

Cal notar la inclusió de peces d'acer entre l'alumini i el coure, i que serveixen tant per mantenir la seva cohesió física com per millorar el contacte tèrmic entre les dues parts del dissipador. L'efecte d'aquest tipus d'inclusió serà d'especial interès pràctic per l'enginyer que estudii la fabricació del dissipador, però a la vegada es troben emplaçats *a l'interior* de l'estructura.

Així, podrem tenir ocasió d'emprar les capacitats de transparència de MPI^{GL} .

[4]Els codis comercials per CFD solen tenir preus molt elevats.

Figura 5.3: Estructura d'un dissipador bimetàl·lic simplificat.



5.2.2 Càlcul del volum de dades

El volum de dades generat depèn en primer lloc de la graella emprada per *discretitzar* el domini. Com que el volum escollit té una geometria simple, en el nostre cas es podrà emprar una simple graella cúbica, subdividint les tres dimensions principals en respectivament n_x, n_y i n_z intervals.

La temperatura es calcularà per cada punt d'encreuament de la graella, o sigui que es treballarà amb un nombre total de $N = n_x \cdot n_y \cdot n_z$ de punts, i per cada punt amb les dades de posició (3 quantitats) i temperatura (1 quantitat).

Encara que no calgui intercanviar en cada moment les dades de posició (fixes), sinó tan sols la temperatura (variable), si la graella presenta una resolució important el nombre de punts N pot arribar a ser molt elevat. Aquesta situació és comuna a un gran nombre de problemes de càlcul de dinàmica de fluids.

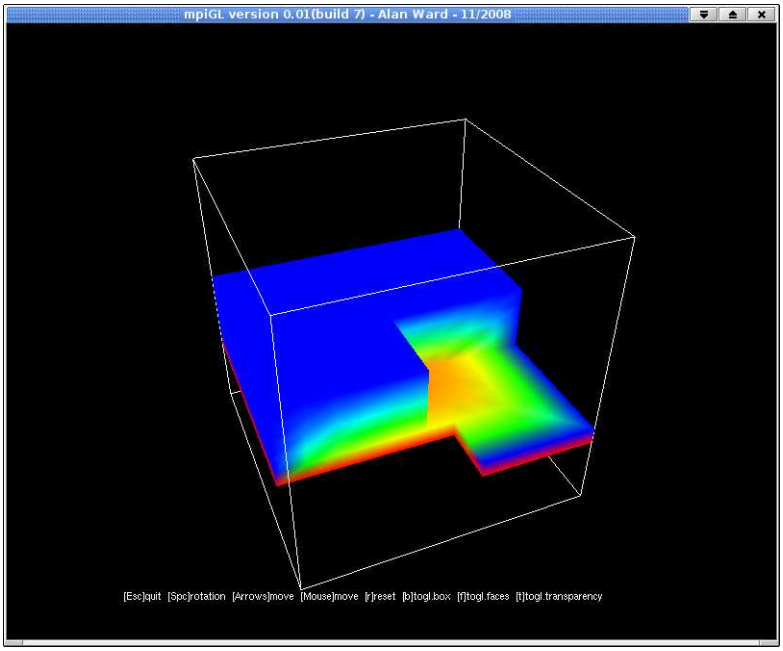
Pel que fa a la seva representació gràfica, però, hi ha una opció que permet reduir una mica la quantitat de dades i que consisteix en enviar al nodus gràfic tan sols la part de dades gràfiques que seria necessària per la seva representació gràfica. Així, en el nostre cas s'ha escollit enviar tan sols les informacions de temperatura corresponents a diferents talls de l'objecte, obviant les dades de temperatura que es trobin entremig dels talls representants, i doncs no es consideren interessants per a la representació. Queda clar, però, que es mantenen aquestes informacions en el sí dels nodus de càlcul.

Les informacions enviades correspondran a camps de temperatura, que es podran codificar fàcilment mitjançant una gama de colors. Concretament, es dividirà la superfície de cada tall en una

sèrie de triangles, enviant per cada triangle sis instruccions que corresponen a les posicions i color dels seus vèrtexs.

Disposant d'una graella de $N = n_x \cdot n_y \cdot n_z$ cubs originals, aquests cubs presenten $2 \cdot (n_x \cdot n_y + n_y \cdot n_z + n_z \cdot n_x)$ cares a l'exterior, i comptant amb la necessitat d'emprar dos triangles per representar cada cara quadrada, això ens porta a un total de $4 \cdot (n_x \cdot n_y + n_y \cdot n_z + n_z \cdot n_x)$ triangles.

Figura 5.4: Representació del camp de temperatures en el dissipador.



5.2.3 Implementació real del cas

Per la implementació real del present cas, s'ha escollit programar el cas emprant el llenguatge de màquina virtual Java, que es connectarà a l'aplicació gràfica mitjançant el mòdul d'entrada *TCPServer*. Així es vol demostrar les possibilitats d'ús emprant altres tecnologies pels nodus de càlcul que no siguin els llenguatges compilats C/C++ i el protocol de comunicació MPI.

L'equació principal de transport de la calor mitjançant conducció es pot escriure com

$$P = \frac{\delta E}{\delta t} = k \cdot A \cdot \Delta T$$

, en què P és la potència tèrmica, o sigui la quantitat d'energia tèrmica E transmesa per unitat de temps. Com veiem, és proporci-

onal a la vegada a la conductivitat k del medi, de l'àrea A a través de la qual flueix, i al diferencial de temperatures ΔT entre els dos extrems de la zona estudiada.

Com que, una vegada que el dissipador ha arribat a l'equilibri tèrmic, cada unitat individual de l'espai ha de mantenir el balanç entre entrades i sortides d'energia[5], això ens porta a resoldre el problema de Dirichlet sobre el domini. Com que existeix una relació de proporcionalitat entre l'energia tèrmica continguda en una unitat de metall i la seva temperatura, el sistema resultant es pot escriure com:

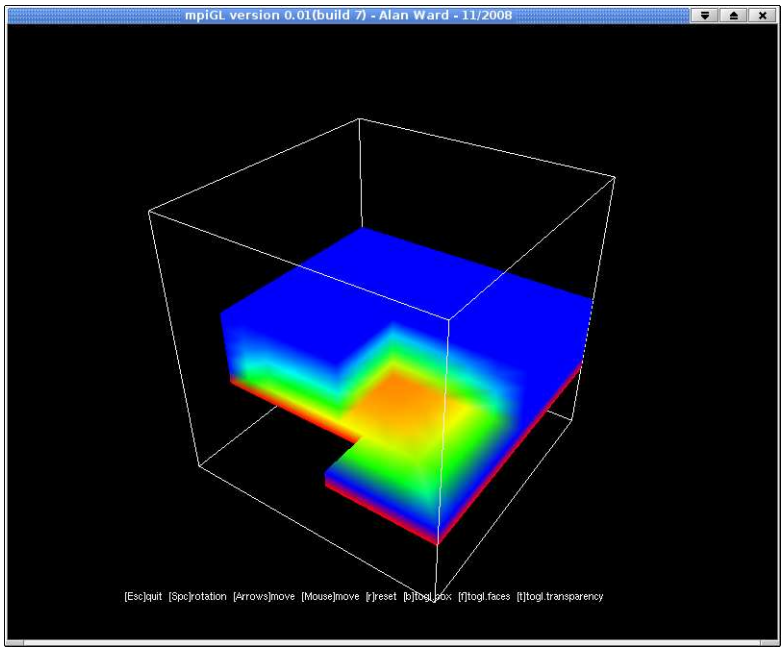
$$\nabla T = 0 \qquad \text{a l'interior del domini}$$

$$T = f(\vec{V}) \qquad \text{a les vores del domini}$$

i en què f és una funció que ens dona la temperatura segons la posició (zones de contacte freda i calenta).

El programa en Java efectua la resolució del problema mitjançant aproximacions successives, i aleshores envia les dades gràfiques al programa principal mitjançant la connexió TCP al port 12345.

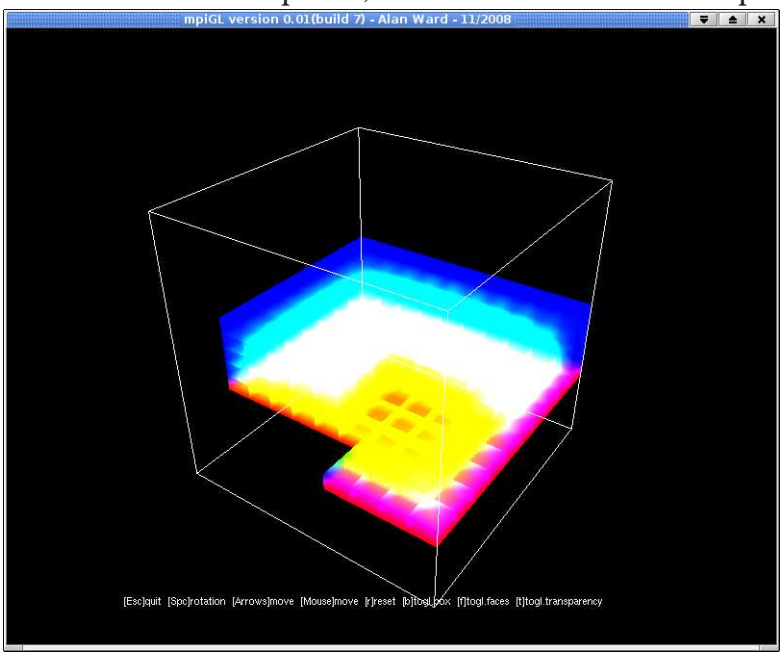
Figura 5.5: Una altra aspecte del camp de temperatures en el dissipador.



El resultat es pot observar en les Figures 5.4, 5.5 i 5.6. En aquestes, s'ha marcat com "no representables" diversos volums del

[5]Sinó, s'escalfaria o es refredaria de manera infinita, el que no seria possible.

Figura 5.6: Vista del dissipador, amb activació de la transparència.



dissipador, per poder visualitzar més bé el camp de temperatures a l'interior del dispositiu. És clar, encara es mantenen aquests volums pel que són els càlculs de temperatura.

El codi de colors va del vermell en la part en contacte amb el CPU (més calent) progressivament al groc, verd, blau clar, i finalment blau fosc (més fred) de la part en contacte amb l'aire ambient.

En la figura fig:dissipador3, s'ha activat l'opció de transparència del mòdul de representació *GLScreen*. Així, es pot veure la distribució de temperatures interna a través de les capes exteriors.

És interessant poder confirmar de manera gràfica el resultat al que es podia arribar mitjançant raonament: que la presència de l'acer, metall menys conductor de la calor, fa pujar de manera significativa la temperatura al cor del dispositiu de dissipació. Així, el gradient de calor hi serà menor, i també el flux d'energia dissipada. Això no és ben bé el que es desitja en un bon dissipador de CPU...

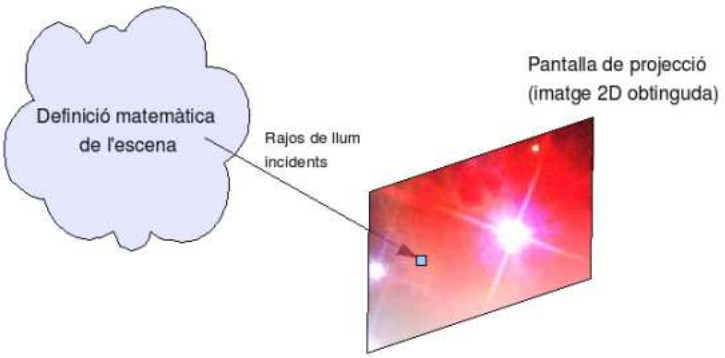
5.3 *Raytracing* d'anells d'Einstein

5.3.1 Descripció

La tècnica del *raytracing* és emprada en aplicacions de creació d'escenes virtuals tridimensionals, tant fixes com vídeo. En resum de

manera breu, es tracta de calcular el valor de cada pixel de la imatge projectada sobre una pantalla amb dues dimensions, a través de l'estudi de la trajectòria dels rajos de llum incidents en aquesta pantalla (Figura 5.7).

Figura 5.7: Esquema del *raytracing*.



Així es poden crear escenes que incorporin diferents efectes òptics (lents, difuminació), i respectar les propietats òptiques dels materials modelitzats per produir resultats força properes a la realitat, per exemple amb el conegut programa de *raytracing* Povray [5].

En un context científic, aquestes tècniques es poden emprar per la modelització de la forma observada dels anells d'Einstein. Aquests anells (predits per Einstein [6] i observats des del 2005 [7]) es formen quan una galàxia llunyana i un objecte massiva són gairebé alineades respecte a la Terra; en aquestes circumstàncies el camp gravitatori de l'objecte intermedi corba la trajectòria dels fotons originaris de la galàxia llunyana, actuant com a *lent gravitacional*. La manera amb què es modifica la trajectòria dels fotons i la forma final de la imatge observada ens pot aportar informacions no només sobre la galàxia llunyana, sinó també sobre les característiques de l'objecte massiu intermedi. S'ha proposat [15] que aquests objectes massius concentrin no només matèria visible, sinó també l'anomenada *matèria fosca*, constitutiva de tres quartes parts de la matèria de l'univers i que també participaria en l'efecte de lent gravitacional.

A partir de diferents distribucions hipotètiques per la matèria fosca, el rol de l'eina informàtic serà predir la forma dels anells d'Einstein resultant, podent així determinar de manera experimental quina de les distribucions proposades s'aproxima millor a la realitat.

Des del punt de vista computacional, es tracta d'un problema

típicament *large-grained*: el valor de cada pixel no depèn del dels altres pixels, sinó tan sols de les dades de definició del problema i de la seva direcció d'observació associada. Així, es poden repartir els pixels a calcular entre els nodus de càlcul, sense gairebé cap comunicació entre nodus [6].

Afegir un nodus gràfic MPI^{GL} al sistema de càlcul, que presenti en pantalla els valors dels pixels a mesura que es vagin calculant presenta dos avantatges:

1. Es pot observar la formació d'una imatge parcial a mesura que es va generant, per exemple, una primera escollida aleatòria de píxels. Així, l'investigador pot formar una idea del resultat prou ràpidament, i descartar els càlculs que s'allunyin visiblement del resultat observat abans de completar-se del tot.

2. Es pot afegir una informació tridimensional sobre els camins recorreguts per tots els rajos incidents, o una selecció d'entre ells. D'aquesta manera i contràriament al procés acostumat de *raytracing* en què la trajectòria de fotons individuals no sol tenir major importància, l'investigador podrà obtenir una informació gràfica suplementària sobre la forma amb què l'objecte massiu intercepta i deforma la llum.

5.3.2 Càlcul del volum de dades

El sobrecost de tràfic de xarxa associat a la presència del nodus gràfic és de l'ordre del nombre de píxels, o sigui

$$Vol = Q \cdot amplada \cdot alcada$$

de la imatge resultant del procés de càlcul.

Per cada pixel, la quantitat de dades transmesa des del nodus de càlcul encarregat d'aquell pixel es pot estimar entre 25 bytes (per formar un únic pixel mitjançant una instrucció GL), a diversos centenars si es vol seguir la seva trajectòria a través de l'espai.

5.3.3 Implementació

No s'ha implementat aquesta aplicació en el marc del present projecte, degut tan a la falta d'espai com al nivell de complexitat del càlcul de les deformacions de l'espai-temps.

[6] Tan sols cal que el nodus mestre comuniqui prèviament la definició de *l'escena* als nodus de càlcul, i reculli els valors generats per cada pixel a finals del procés de càlcul

Capítol 6

Posada a disposició de la comunitat en un entorn de Programari Lliure

En el context d'un Màster d'aquestes característiques, va semblar important respectar els principis de base de la comunitat del Programari Lliure . En aquest sentit, es va prendre la decisió emprar les eines posades a disposició per altres membres de la comunitat per tal de compartir el codi generat.

Aquest fet de compartir el codi respon a diversos objectius:

1. El primer és que l'aplicació producte d'aquest projecte final ha estat desenvolupat de manera individual per un únic analista/programador. És previsible que, en el futur, altres programadors hi podran aportar contribucions importants tant a nivell de la seva concepció com programació.

2. Un segon objectiu existeix a nivell de les aplicacions científiques de la plataforma desenvolupada. L'autor de l'aplicació (i d'aquest informe) té una experiència en un cert nombre restringit de camps de la ciència moderna; és doncs força possible que d'altres persones puguin fer aportacions interessants des d'altres àmbits que l'autor original desconeix totalment.

3. Finalment, com que aquest projecte s'ha pogut dur a terme gràcies exclusivament a productes fruits de la col·laboració de programadors en el context del Programari Lliure , és coherent fer-hi un retorn a la comunitat que, per modest que sigui, comporta en sí el testimoniatge del reconeixement que els usuaris d'aquest tipus de producte pot tenir a un col·lectiu de persones desinteressades.

El mecanisme escollit per posar el contingut d'aquest projecte a disposició de la comunitat ha estat la coneguda plataforma *Sourceforge*. La seva escollida respon a diversos criteris.

El primer i més important és a nivell de la seva reputació. Es coneix àmpliament aquesta plataforma dins el món del Programari Lliure , de manera que pot ser un dels primers punts en què una persona que necessiti una aplicació o llibreria de rutines amb les característiques d'aquesta començarà la seva cerca. Aquest factor és important perquè autors originals i usuaris del programari es pugui trobar i conèixer els projectes respectius.

Un segon criteri va ser el nombre de serveis diferents que aporta la plataforma *Sourceforge*, i que si bé no sempre semblen útils a un projecte modest com el present, podrien servir en cas d'expansió del seus nombre d'usuaris. Una llista no-exhaustiva podria incloure no només les facilitats per compartir fitxers i documentació, sinó a més els recursos de comunicació (correu, llistes de discussió) i "publicitat" – en el sentit de posar el producte final de la nostra recerca en coneixement d'altres persones.

L'identificador escollit va ser *mpigl*, i doncs l'adreça d'aquest projecte a Sourceforge és:

https://sourceforge.net/projects/mpigl/

Figura 6.1: Plana principal del projecte a Sourceforge.net .

Capítol 7

Conclusions i reflexions suplementàries

En l'anàlisi prèvia a la realització d'aquest projecte, hem pogut veure:

1. La necessitat de reduir el temps d'inicialització del nodus gràfic per part del nodus mestre, minimitzant el nombre d'elements gràfics comuns de l'escena (*decoració*).

2. Que es pot reduir l'efecte del treball suplementari que implica l'adjunció del nodus gràfic, regulant la freqüència de refresc dels elements gràfics per part dels nodus de càlcul.

3. La necessitat d'una mètrica comuna entre les diferents parts de la cadena de transmissió de les instruccions gràfiques entre els nodus de càlcul i el nodus gràfic. En el nostre cas, hem adoptat una unitat *ad hoc* i enfocat des del punt de vista de l'usuari, el nombre de $triangles/s$.

4. Que els punts més dèbils pel que fa a la possible aparició de colls de botella en el sistema seran probablement a nivell de la infraestructura de xarxa o connexió a la xarxa del nodus gràfic.

A partir d'aquesta anàlisi, s'ha pogut dissenyar una solució modular en llenguatge C++ i emprant llibreries existents del món del Programari Lliure . Emprant els POSIX thread per implementar la concurrència a nivell dels mòduls d'entrada, a través d'un llenguatge d'instruccions gràfiques inspirada en OpenGL però simplificada per les necessitats de l'usuari científic, permet de reduir considerablement el tràfic per la xarxa simplificant així la gestió del coll de botella principal identificat en el sistema.

A la vegada, aquest llenguatge basat en cadenes de caràcters és prou flexible per permetre la seva utilització per part de nodus de càlcul programats amb diferents llenguatges de programació i infraestructures de xarxa.

S'ha pogut verificar parcialment la seva idoneïtat per aplicacions de computació científica reals, a través de la construcció de dos prototips i la validació de les seves capacitats sobre una configuració material típica[1]. S'han escollit els prototips de manera a poder demostrar:

1. El funcionament de les diferents entitats gràfiques (vèrtexs, línies i triangles), i les instruccions gràfiques corresponents.

2. La correcta sincronització dels accessos al conjunt de dades per part dels diferents *thread*, emprant el mecanisme de *mutex*.

3. Els dos mecanismes de comunicació implementats entre nodus de càlcul i l'aplicació gràfica (MPI i TCP/IP). Al mateix temps s'ha pogut validar l'ús de programes sobre els nodus de càlcul escrits en llenguatges (C i Java) diferents del llenguatge de programació de l'aplicació gràfica (C++).

També a través dels dos prototips, s'ha pogut constatar la possibilitat d'una regulació el volum de dades tractades pel nodus gràfic, si fos necessària, ja a nivell dels nodus de càlcul que poden enviar una informació més o menys detallada i així limitar el risc de saturació de la xarxa arribant al nodus gràfic.

Tot i això, existeixen dos punts sobre els que valdria la pena fer un treball complementari.

El primer seria la mesura de la capacitat de l'aplicació a suportar càrregues importants en condicions reals. Encara que, com s'ha s'ha dit, *Your Milage May Vary* i és responsabilitat de cada investigador que usi el sistema verificar el seu comportament pel seu propi cas, podria ser d'utilitat la publicació d'una sèrie de proves fetes amb materials diferents[2] i en condicions real de treball: càrrega induïda en els sistemes i infraestructures de xarxa per la presència d'altres usuaris, panes del material o del programari, etc.

En segon lloc, en la majoria d'instal·lacions importants (i també és el cas del clúster DCPS Grid a la UOC), se sol controlar el funcionament del conjunt de càlcul a distància. Encara que la pantalla gràfica d'aquesta aplicació pugui passar fàcilment a través d'una

[1]Concretament: un portàtil amb processador Celeron Mobile a 1,7 GHz per l'aplicació gràfica, i un conjunt heterogeni d'ordinadors fixos com a nodus de càlcul Tots executaven una versió o altre de la distribució Ubuntu.

[2]Dels que actualment no disposa l'autor.

connexió SSH[3], la seva reactivitat a entrades del ratolí no és òptima en aquestes condicions.

Enviar tan sols les instruccions gràfiques en comptes del conjunt de píxels que forma la pantalla gràfica podria ser una solució en aquest cas, però aleshores caldria trobar la manera de connectar un mòdul de dades ubicat en un ordinador (el nodus mestre del conjunt de càlcul) a un mòdul de representació ubicat físicament en un altre ordinador remot (l'ordinador personal de l'investigador). A més, podrien sorgir qüestions relacionades amb la travessada de diferents *firewalls*, tant del costat del clúster com del de l'investigador.

Es tracta, en tot cas, d'aspectes complementaris que es podran tractar progressivament mentre es continua desenvolupant aquest projecte del Programari Lliure en el sí de la comunitat. Es pot veure com una aplicació concreta de les tècniques de compartició d'informacions, programari i idees que formen el dorsal d'aquest món del Programari Lliure .

[3]Emprant l'opció -X .

Bibliografia

[1] G. Amdahl. Validity of the single processor approach to achieving large-scale computing capabilities. In *AFIPS Conference*, pages 483–485, 1967.

[2] Christopher C. Aycock. Mpi ruby with remote memory access. *International Symposium on High-Performance Distributed Computing*, 0:280–281, 2004.

[3] Beowulf mailing list. URL: http://www.beowulf.org (Última consulta: 31/10/2008).

[4] E. Bertschinger. Simulations of structure formation in the universe. *Annual Review of Astronomy and Astrophysics*, pages 599–654, 1998.

[5] D. Buck. Povray original creation message, 1991. URL: http://www.povray.org/documentation/view/3.6.1/8/ (Última consulta: 26/09/2008).

[6] A. Einstein. Lens-like action of a star by the deviation of light in the gravitational field. *Science, New Series*, 84(2188):506–507, 1936.

[7] R.A. Cabanac et al. Discovery of a high-redshift einstein ring. *Astron. Atrophys.*, pages L21–L25, 2005.

[8] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*. Message Passing Interface Forum, 1994. URL (versió 1.0): http://www.mpi-forum.org/docs/mpi-10.ps (Última consulta: 26/09/2008).

[9] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, 1999.

[10] Alan Kaminsky. Parallel java: A unified api for shared memory and cluster parallel programming in 100 percent java. *International Parallel and Distributed Processing Symposium*, 0:231, 2007.

[11] Kai Li and Paul Hudak. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems*, 7, 1989.

[12] OpenCFD Ltd. Openfoam: The open source cfd toolbox. URL: http://www.opencfd.co.uk/openfoam/ (Última consulta: 20/3/2008).

[13] The mesa 3d graphics library. URL: http://www.mesa3d.org (Última consulta: 04/11/2008).

[14] Frank Mueller. Pthreads library interface. Technical report, Florida State University, 1993.

[15] NASA. Biggest 'zoom lens' in space takes hubble deeper into the universe, news release number stscl-2003-01, 2003. URL: http://hubblesite.org/newscenter/archive/releases/2003/01/text/ (Última consulta: 12/10/2008).

[16] Fernando Pérez and Brian E. Granger. Ipython: A system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, 2007.

[17] M. Segal and K. Akeley. *The design of the OpenGL graphics interface*. Silicon Graphics Computer Systems, 1994.

[18] V.S. Sunderam. Pvm: A framework for parallel distributed computing. *Concurrency: Practice and Experience*, 2(4):315–339, 1990.

[19] S. Turek and Chr. Beker. Featflow: finite element software for the incompressible navier-stokes equations. URL: http://www.iwr.uni-heidelberg.de/ featflow/ (Última consulta: 25/11/2008).

[20] J.F. Wallin and B.V. Stuart. Mass transfer and loss in interacting disk galaxies. *Astrophysical Journal*, 399(1):29–37, 1992.

[21] M.S. Warren and J.K. Salmon. Astrophysical n-body simulations using hierarchical tree data structures. In *Supercomputing '92*, 1992.

Apèndix A

Ús del Programari Lliure en aquest projecte

Degut a les característiques d'aquest projecte, s'ha pres especial cura en emprar exclusivament aplicacions del Programari Lliure durant la seva realització - gairebé de manera religiosa! Algunes de les eines emprades varen ser:

- El nucli de Linux (www.kernel.org).
- Les altres eines del projecte GNU que fan possible un ús pràctic del nucli (www.gnu.org).
- El compilador del llenguatge C/C++ del projecte GNU, *gcc*.
- La llibreria de rutines MPI i les altres eines de gestió del MPI del projecte OpenMPI (www.openmpi.org).
- El gestor de despatx Gnome (www.gnome.org)
- El llenguatge TEXi el conjunt suplementari de macros LATEXper la posada en pàgina del present informe.
- Diversos editors de text com *kate* i *gedit* per la redacció pròpiament dita.
- *aspell* per la correcció ortogràfica.
- Gimp pel tractament de les imatges (www.gimp.org).
- Inkscape per la realització de gràfics vectorials (www.inkscape.org).
- *amarok* i *vlc* per l'ambientació sonora (www.videolan.org).
- La distribució Ubuntu per lliurar i administrar tot el programari de forma raonablement simple (www.ubuntu.com)

... i possiblement alguns altres que m'he descuidat.

Aquesta llarga llista posa en evidència les possibilitats reals del Programari Lliure en el món de la producció científica actual.

Apèndix B

Glossari

Beowulf cluster Un clúster de càlcul dedicat. Als seus inicis, s'emprava material físic de tipus COTS (*Common, Off-The Shelf* - d'ús corrent en altres àmbits de la informàtica) de manera corrent. Avui en dia, una tendència important empra components de relativa alta gama com poden ser targes de xarxa Infiniband.

CFD *Computational Fluid Dynamics*, o sigui la resolució per ordinador de problemes de dinàmica de fluids. Generalment es procedeix a través la discretització del domini estudiat, transformant les equacions de dinàmica en un sistema d'equacions a resoldre mitjançant tècniques computacionals.

COW *Cluster of Workstations*, en què un conjunt d'ordenadors normalment emprats per altres finalitats (ofimàtica, laboratori informàtic per a la docència) s'usen puntualment per efectuar càlculs científics, per exemple durant les nits o caps de setmana.

Fine-grained Dit d'un càlcul paral·lel en què els diferents elements de càlcul tenen un alt grau de relació entre ells, el que dificulta la seva repartició entre els nodus de càlcul ja que cal un elevat nivell de comunicació entre els nodus.

Grid computing Sistema de computació en què els elements físics que efectuïn el càlcul es poden trobar a grans distàncies els uns dels altres, connectats (per exemple) mitjançant Internet. Un exemple en seria el projecte Seti@home.

Large-grained Per oposició a *fine-grained*, càlcul en què els diferents elements de càlcul tenen poca relació entre ells. Es considera que aquest tipus de càlcul és més fàcil de paral·lelitzar,

ja que el nivell de comunicació entre nodus de càlcul pot ser relativament baix.

Llei d'Amdahl Llei proposat per Amdahl ([1]), en el que es contrasta la proporció del temps de càlcul passat en processos que es puguin paral·lelitzar, contra el temps passat en processos essencialment seqüencials. Si bé la primera part potencialment es pot accelerar afegint recursos de càlcul, la segona no. Així, aquesta proporció ens limita l'acceleració potencial d'un sistema de càlcul paral·lel. Per exemple, una aplicació directe d'aquesta llei és el fet que afegint un segon nucli a un processador mononucli mai podrà doblar la seva velocitat de càlcul: a afectes pràctics, se sol obtenir una millora d'entre 10 i 50%, segons el valor de l'esmentada proporció en el treball concret a efectuar.

Llei de Moore Llei empírica segons la qual la complexitat (en nombre de transistors) dels dispositius electrònics es pugui doblar cada 18 mesos. A vegades s'interpreta com el doblement periòdic de la potència de càlcul, encara que aquest enfocament pugui semblar simplista ja que l'eficiència de les arquitectures i dels algoritmes també influeix fortament en el temps de tractament computacional. Un altre factor que es té en compte avui en dia és el límit físic a la reducció de mida dels transistors, que a la vegada limita la seva densitat en la superfície dels *wafers* de silici.

run Seqüencia completa de computació, que inclou la distribució del programari específic a tots els nodus de càlcul, la distribució del conjunt de dades sobre el que treballar, l'execució del procés de càlcul, i la recollida dels seus resultats.

Apèndix C

GNU Free Documentation License

Version 1.2, November 2002
Copyright © 2000,2001,2002 Free Software Foundation, Inc.

51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not

Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However,

you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you

release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old

one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with . . . Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.