Títol: TUTORIAL DE RAY TRACING

Volum: 37,5
Alumne: JAVIER MAZAIRA FONT

Director/Ponent: PERE PAU VÁZQUEZ ALCOBER
Departament: LSI
Data: 18/12/2008
Índex

Introducció
  Objectius 7
  Motivacions 7
  Estructura de les memòries 7

Gràfics per computador
  Definició 8
  Passeig per la història 8
  OpenGL 14
    Definició 14
    Funcionament 14
  Ray-tracing 16
    Definició 16
    Funcionament 16
    Variants 18

Comparació d'OpenGL vers Ray tracing 21

Planificació
  Introducció documental 22
  Planificació temporal 22
  Pressupost 23

Ray Lab
  Introducció documental 24
  Introducció 24
  Estructura 27
  Disseny 28

Visualitzador OpenGL
  Introducció 29
  Estructura 29
  Disseny 30

Carregar i desar informació 30
  Gestió de 3ds 30
  Gestió de XML 32
  Navegació entre objectes/llums 33

Selecció per ratoli 33
Selecció per combo box 34
Fleta Guia 36
  Gestió de materials 38
Tutor de ray tracing índex

Gestió de llums 39
Llençar rajuos 39
Ampliacions futures 41
Animador de rajuos
  Introducció 42
  Estructura 42
  Disseny 43
  Sistema d’interseccions 43
    Creació del M.D.E. (model de divisió espaial) 43
    Càlcul de les interseccions 44
    Càlcul de les dades associades a les interseccions 48
    Traçat de rajuos 51
    Visualització de rajuos 52
    Càlcul del color 53
    Algorismes de traçat 55
  Ray tracing 55
  Distributed ray tracing 57
  Ampliacions futures 59
Visor Ray Tracing
  Introducció 60
  Estructura 60
  Disseny 61
    Sistema de pintat 61
    Sistema de refresc 62
    Gestió d’imatges 63
    Ampliacions futures 64
Anàlisi del desenvolupament
  Introducció documental 65
  Conclusions 66
Bibliografia
  Pàgines web 67
  Libres 67
Glossari
Introducció

1. Objectius

L'objectiu principal del projecte consisteix en realitzar un tutor 3D de l'algorisme de Ray tracing, esdevenint una eina per la comprensió satisfactoria d'aquesta tècnica de síntesi d'imatges per computador. Concretament, es vol dotar a la comunitat d'una aplicació que permeti veure de manera acurada els comportaments del raigs de llum que permeten la síntesi d'imatges en el ray tracing.

2. Motivacions

Els motius pels quals s'ha optat per realitzar aquest projecte serien: introduir una nova eina en l'entorn educatiu, aprofundir en la síntesi d'imatges per ordinador tant a nivell del paradigma d'OpenGL com el de traçat de raigs, elaborar una aplicació software en la seva totalitat, és a dir, des de la planificació inicial, fins l'entrega al client i, en definitiva, aplicar els coneixements adquirit al llarg de la carrera.

3. Estructura de les memòries

El document adjunt s'estructura de la següent manera: primer de tot es presenta una contextualització de l'àmbit de treball, parllant sobre la història dels gràfics en tres dimensions i comentant els tipus de pintat que es veuran en l'aplicació vinculada al projecte.

Tot seguit es mostra una planificació prèvia a la realització del projecte, descriuint una estimació temporal expressada en un diagrama de Gantt i la consequent estimació econòmica del desenvolupament plantejat.

A partir d'aquí s'inicia una explicació detallada dels mòduls que component l'aplicació, arrencant amb una descripció a nivell general, per posteriorment anar aprofundint en els diferents mòduls, donant-ne una descripció que en justifique la seva existència, fent-ne una breu especificació en UML, plantejant alguns temes sobre la implementació i suggerint possibles millores a realitzar en posteriors versions.

Finalment, es destina un darrer punt per analitzar el desenvolupament del projecte, comentant les desviacions més significatives i intentant extreure conclusions per evitar errors començats, com per aprofitar els punts positius que hagin sorgit.

No es pot arribar a l'alba sinó és pel camí de la nit
Khial Gibran (1883-1931) - novelista i poeta llibanes
1. Definició

Ès el camp de la informàtica que engloba el conjunt de tècniques que permeten la generació d’imatges des d’un punt de vista matemàtico-computacional. Aquesta branca es pot dividir en subcampaments d’estudi tals com: modelat o la visualització.

2. Passeig per la història

És considera que la història dels gràfics per computador començà l’any 1945, al Massachusetts Institute of Technology, amb la fabricació del computador Whirlwind. Aquest aparell tenia com a objectiu oferir un entorn “programable” de simulació de vol i va ser el primer ordinador capaç d’operar a temps real usant un visor de vídeo com a mètode de sortida d’informació.

Durant la dècada dels 60 la indústria centrada en els gràfics per computador va començar a créixer: IBM va fabricar el primer computador comercial amb capacitats gràfiques; l’IBM 2250, van aparèixer els primers videojocs, programes senzills que permetien interacció gràfica amb l’usuari (OXO, Spacewar o el famós “tennis for two” entre altres), van dissenyar-se part dels algorismes de tractament d’imatges encara avui vigents i, fins i tot, es desenvolupà el primer dispositiu de síntesi d’imatges tridimensionals que, conegut com the Sword of Damocles (1966), permetia generar la sensació de volum gràcies a estar format per un casc amb dos visors independents que mostraven un mateix model des de punts de vista diferents.

Figura 1: Demostració de Sketchpad a mans de Sutherland (imatge extreta de: www.wikipedia.org)

Paral·lelament a tot això, un dels estudiants del mateix MIT, Ivan Sutherland, va desenvolupar en el decurs de la seva tesi doctoral (1963) l’aplicació Sketchpad (figura 1), que va ser el primer programa de dibuix per a ordinador i la primera interfici d’usuari gràfica. Sketchpad permetia, amb l’ús d’un llapis òptic, dibuixar objectes damunt una pantalla CRT i un cop plasmats en ella, desar-los en memòria, aplicar transformacions geomètriques, carregar informació anteriormen dibuixada, etc.

---

1 OXO: Versió electrònica del tres en ratlla inventat el 1952 per A. S. Douglas

2 Spacewar: Joc d’ordinador on dues naus s’enfronten dins el camp gravitatori d’una estrella
Amb aquest programa, Ivan Sutherland va introduir una sèrie de conceptes encara emprats avui en dia, entre ells cal destacar el sistema per crear cosos geomètrics donant només el lloc en l’espai i el tipus i el fet de treballar amb objectes enllà d’imatges. Amb el primer d’aquests mètodes l’usuari es lliurava de fer dibuixos amb rectes i angles perfectes, doncs especificant unes dades mínimes, l’aplicació realitzava el dibuix reper, per entendre-ho millor, el concepte estava en només indicar que, per exemple, es volia fer un quadrat de certes característiques en un punt determinat, en comptes d’haver de traçar les línies manualment, reduint les imprecisacions del dibuix manual.

L’altre concepte destacable es basava en aplicar transformacions a nivell d’objecte, així donada una escena formada per un vehicle, l’usuari podia modificar la mida dels “pneumàtics” sense afectar la resta del xàssis, en oposició a un sistema sense cap mena de vinculació a objectes, on el programa no podria diferenciar els elements (una imatge s’interpreta com una estructura monolítica, en oposició a un conjunt d’objectes en una superfície) i, per tant, les transformacions s’aplicarien sempre a la totalitat dibuixada.

Sketchpad va marcar els inicis del DAC (disseny assistit per computador) que continuaria prosperant a mans de IBM i General Motors, introduint noves tècniques com el “clipping” (funció per indicar zones de l’escena que no desitgem que siguin dibuixades), rotacions o zoom.

A finals de dècada (1968) Sutherland es va unir a un programa de desenvolupament de la Universitat de Utah, a partir d’aleshores aquesta univeritat començà a despuntar com a centre neuràlgic de tot allò referent a gràfics per computador; sorgint de les seves aules els fundadors d’Adobe Systems, Silicon Graphics, Pixar o Atari i desenvolupant-se nous algorismes com: antialiasing, z-buffer, mapat de textures,... Potser un dels algorismes més notables de l’època fou el d’ocultació de superfícies (Lawrence Roberts), que dins el visionat 3d determina, per a una escena i punt de vista concret, quins objectes oculten a quins altres. En la mateixa dècada, A. Appel mostrà una nova tècnica per generar imatges tridimensionals: el ray casting. La base d’aquest algorisme radicava en traçar raigs des del punt de vista de l’observador per valorar l’objecte més proper amb el que intersectaven i extreure el color en base al primer punt d’intersecció.

A principis dels setanta els gràfics es van introduir en el món televisiu, la tendència del moment era presentar imatges cada cop més “foto-realistes” a base d’augmentar la geometria dels objectes, provocant una complexitat creixent que desembocà en escenes que cada cop tardaven més en generar-se.

Davant la problemàtica computacional que es dibuixava en l’horitzó, Henri Gouraud presentà un sistema innovador per a l’acolorat de superfícies (figura 2). El principi d’aquesta metodologia radicava en deixar de pintar els polígons de forma homogènia en base a tenir un color definit per cara, a fer-lo en base a tenir colors per vèrtex, calculant el color dels punts de la cara per interpolació lineal de

3 Antialiasing: Tècnica que minimitza els efectes de distorsió que es donen en imatges quan hi ha salts bruscos de color
4 Foto-realisme: Creació d’imatges que resultin indistingibles de fotos de la realitat
les dades a nivell de vèrtex, donant una sensació de suavitzat de la superfície que permetia obtenir efectes visuals iguals amb menor poligonatge.

Figura 2: Esfera amb flat shading i amb pintat de Gouraud (imatge extreta de: www.howstuffworks.com)

El pintat de Gouraud aviat va veure's potenciat al convinar-se amb noves tècniques, així doncs, l'any 1973 Edwin Catmull de la Universitat de Utah presentà en la seva tesi el mapat de textures (figura 3), una tècnica que permetia assignar una imatge a la superfície d'un polígon, aconseguint una millora espectacular en el pintat d'objectes. També en la seva tesi va presentar l'algorisme de z-buffer, encarregat de gestionar les coordenades de profunditat de les imatges en els gràfics en tres dimensions i esdevenir d'aquesta manera una solució per al problema de visibilitat, que és el problema de decidir quines parts dels elements d'una escena són visibles i quines ocultes (per estar darrera de quelcom respecte l'observador).

Figura 3: Vehicle amb mapat de textures i sense cap tipus de textura

En una altra direcció, el camí cap al foto-realisme va avivar-se amb les aportacions del matemàtic francès Benoit Mandelbrot i el descobriment dels fractals (estructures recursives que es manifesten en la natura com arbres o muntanyes) junt amb com aplicar-los en la generació d'imatges per ordinador.

La dècada dels vuitanta arrencà amb la producció de Tron, una pel·lícula produïda per Walt Disney on s'empraren prop de 32 minuts d'animació per ordinador, establint un punt d'unió entre els gràfics per computador i el cinema. A partir d'aquest moment, la indústria cinematogràfica esdevindrà un dels principals motors per al desenvolupament dels gràfics per ordinador en l'anhel de presentar efectes especials cada cop més espectaculars. Per altra banda, fou també a inicis de dècada quan IBM introduí en el mercat el primer ordinador personal, un dispositiu que suposava la introducció a tots els nivells dels computadors i, conseqüentment, va suposar un espectacular creixement de tots els sectors de la indústria. Encara que IBM havia realitzat un pas considerable amb el PC, el pas més transcendental el donà Apple, l'any 1984, amb el Macintosh: el primer ordinador personal comercial amb interfície gràfica.
Durant el mateix any s'obrí una nova via en la síntesi d'imatges per ordinador, el programa de gràfics per computador de la universitat de Cornell engendrà un nou mètode per a generar imatges realistes centrats en el comportament de la llum: la radiositat\(^5\). Abans que sorgís aquest algorisme, les escenes només disposaven de llum directa, és a dir, tot allò que no estava en trajectòria directa d'un focus de llum quedava en la foscor total, deixant de banda l'aspecte difós i la llum reflectida per superfícies que caracteritza el món real.

Passats dos anys, Pixar tornà a fer girar la roda amb la creació del RiSpec, una API\(^6\) centrada en oferir un estàndard per descriure escenes tridimensionals (objectes, fonts de llum, càmeres, efectes atmosfèrics,...), en forma de llenguatge d'alt nivell capaç d'especificar un gran conjunt de primitives geomètriques.

Les operacions gràfiques resultaven cada cop més complexes i la CPU (unitat de processament central) es ressentia cada cop més d'aquests càlculs addicionals, la resposta a nivell hardware la va introduir Commodore Amiga amb la invenció del blitter (block transfer), un xip co-processor que es dedicava a transferir els mapes de bits que codificaven les imatges a memòria RAM, alleugerint la càrrega de treball de la CPU.

El darrer fet remarcable dels vuitanta fou la formulació de l'algorisme de ray tracing (figura 4) per Turner Whitted, una versió basada en els mateixos principis que el ray casting presentat durant els seixanta, però amb un major potencial per generar imatges foto-realistes gràcies a que introduïen el càlcul de la llum indirecta que sorgeix dels fenòmens de reflexió i transmissió de la llum.

Els noranta van començar amb el naixement de Windows 3.0 (figura 5), el sistema operatiu de Microsoft que introduïa com a principal novetat una interfície gràfica en la línia del Machintosh d'Apple. El fet que Windows cada cop donés més pes als gràfics per computador i que es situés com el sistema operatiu més utilitzat va originar l'aparició de hardware especialitzat. Els blitters dels 80 començaven a quedar obsolets i, per tal d'encarar les necessitats creixents, va sorger una nova generació de xips gràfics que acabaren derivant en un processador a banda del principal la GPU.

---

\(^5\) Radiositat: Algorisme d'il·luminació global que només té present l'aspecte difós

\(^6\) API: Interfície de programació d'aplicacions oferta per un sistema operatiu o llibreria que descriu un conjunt de serveis
El maquinari especialitzat cada cop ajudava més a agilitzar els càlculs en la síntesi d'imatges, però per contra, cada cop hi havia més divertat d'aquest nou hardware especialitzat, provocant que realitzar un software que fos compatible amb un ventall raonable de targetes gràfiques resultés molt costós. Amb l'objectiu d'homogeneïtzar el sector i oferir un estàndard que permetés crear software compatible amb qualsevol peça hardware, comencen a sortir les primeres APIs entre elles PHIGS o IRIS GL\(^7\), però el punt d'inflexió el generà OpenGL.

Desenvolupat originalment per Silicon Graphics Inc. durant el 1992, OpenGL (Open Graphics Library) era i és una especificació estàndard que defineix una API multiplataforma capaç d'escrivir aplicacions que produeixin gràfics tan en dos, com en tres dimensions.

Durant el 1995 es van produir dos grans revolucions, per una banda el cinema assolí una nova cota en emetre Toy Story, que encara que no mostrés els efectes més revolucionaris, va ser la primera pel·lícula íntegrament feta amb imatges generades per ordinador. Per altra banda, s'obrí un nou camí que encara avui en dia es recorre: la introducció de gràfics 3D en el món de les videoconsoles. Aquests dispositius van començar a guanyar un pes considerable, convertint-se en un dels principals sectors en impulsar la fita del foto-realisme, incrementant la recerca i la demanda de nou hardware gràfic.

---

\(^7\) IRISGL: API creada per Silicon Graphics que és considerada el pare de l'actual OpenGL.
PlayStation, Sega Saturn i Nintendo 64 captiven l'audiència amb imatges interactives en entorns tridimensionals i, dins aquest nou horitzó, destacava el joc "Mario 64" (figura 6), que permetia una interacció completament tridimensional. Les consoles seguirien expandint el seu potencial fins a l'actualitat, on l'actual generació ens mostra imatges espectaculars gràcies a l'ús de les tècniques més innovadores i el potent hardware que hi ha al darrera, provocant que a vegades costi distingir realitat de ficció.

L'any noranta cinc encara reservava secrets i és que durant aquest any Microsoft va mostrar al públic la primera versió de Direct3D, un motor gràfic per a Windows 95 que es situava com a possible rival d'OpenGL i que, actualment, és el pilar central per qualsevol aplicació que desitgi aprofitar al màxim el hardware gràfic en entorns Windows.

Els xips gràfics que havien sortit a la llum a principis dels noranta progressaven a mans de 3dfx i el seu xip Voodoo. La nissaga d'aquest xip va anar fent prosperar i evolucionar el hardware encarregat de la gestió dels gràfics, marcant un llistó que seguirien companyies com ATI o NVIDIA que actualment es reparteixen el mercat amb els seus respectius xips Raedon i GeForce.

De mitjans dels noranta a l'actualitat el software i el hardware gràfic no ha deixat de progressar; cada pocs anys surt una nova versió de Direct3D o OpenGL que porta noves eines per crear entorns cada cop més espectaculars, les targetes gràfiques són cada cop més potents i més flexibles a ser programades, permeten esprèmer les seves capacitats més enllà del que es podia pensar. Cinema, medicina, videojocs, televisió i molts altres sectors s'aproximen del que poden oferir les imatges que generen els ordinadors i sembla que, avui per avui, les possibilitats que ofereix aquesta branca de la informàtica tingui com a única limitació la imaginació humana.

Figura 7: Gears of war (2006) per XBOX360 (copyright: Gears of War EpicGames - Microsoft Corporation)
3. **OpenGL**

3.1. **Definició**

OpenGL és una interfície software per als gràfics per computador. Aquesta interfície consta de prop de 700 comandes diferenciacies que permeten especificar el conjunt d'objectes i operacions necessàries per tal de produir aplicacions amb interacció tridimensional.

Desenvolupat per Silicon Graphics (1992) es defineix com una API descrita en C que té implementacions en múltiples llenguatges, multi-plataforma i plena d'extensions i afegits sobre les funcionalitats bàsiques que fan que es mantingui vigent en l'actualitat.

3.2. **Funcionament**

OpenGL és una màquina d'estats que en conjunt són coneguts com el **pipeline** (figura 8).

---

**Figura 8:** Representació esquemàtica del pipeline d'OpenGL.

L'entrada del pipeline són primitives bàsiques: línies, punts o polígons que circulen per les diferents unitats funcionals fins, en la majoria dels casos, expressar-se en forma de pixels. Per definició les primitives s'emmagatzemen en forma de vèrtex (Vertex Data) i des d'allà poden ésser emprades en les display list (conjunt d'ordres d'OpenGL agrupades amb l'objectiu de resultar optimitzades) o en generar nova informació a través d'aplicar operacions vàries en els vèrtex que les defineixen.

Totes les primitives estan constituides en darrera instància per vèrtexs, quan entren en el pipeline es treballa només amb els seus punts ja que per exemple, si sempre venen els 3 vèrtexs d'un triangle en ordre, no ens cal altra informació per reconstruir-lo a través d'ells, en alguns casos però, pot ésser que només amb els punts no tinguem prou informació com per reconstruir a posteriori la primitiva, és en
la unitat d’avaluació on, si procedeix, s’evaluen les funcions polinomials que defineixen certes primitives d’entrada com les superfícies NURBS\(^8\), aproximacions a corbes i geometria de la superfície, per tal de generar informació suplementària com les normals per superfície, les coordenades de textura, colors, etc.

Amb la feina feta per la unitat d’avaluació, la informació es dirigeix a la unitat d’operacions a vèrtex, allà els punts que arriben expressats en el sistema de coordenades del model (x,y,z,w), a través d’un seguit de transformacions matríciais, esdevenen punts en coordenades de clipping, és a dir, la geometria que surt d’aquest procés compleix que \(-w < x < w\) i \(-w < y < w\). A més a més, en aquesta unitat també es realitza el càlcul de certes característiques, sempre que estiguin activades, com la il·luminació.

Figura 9: Exemple de clipping d’un polígon qualsevol (imatge extreta de www.edndoc.esri.com)

Els vèrtexs transformats passen a la unitat d’ensamblat de primitives on, donat el “clipping” anterior que ha descartat i generat nous punts, les primitives són reconstruïdes novelment amb els vèrtex modificats, el color calculat, profunditat i, a vegades, els valors de coordenades de textura.

Aquesta gestió de retallar la primitiva és important perquè encara que virtualment tinguem un món ple de geometria, per pantalla només en mostrem una part i per tant, tot allò que no es veu queda tallat i descartat. L’acció de tallar la geometria que no apareixerà per pantalla i reconstruir la que finalment es veurà és coneix amb el nom de clipping (figura 9).

Fins al moment, hem passat del món de l’objecte a com es veuria l’escena si la nostra pantalla fos una finestra oberta a aquest espai virtual, per fer-ho s’ha treballat amb les unitats elementals de la geometria (vèrtexs), transformant-les fins quedar redefinides en noves primitives tals que només estan formades per allò que des de la nostra finestra es pot veure. Tota aquesta geometria arriba a la unitat de rasterització on es traduirà en forma de fragments, on un fragment es pot definir com el conjunt d’atributs necessaris per actualitzar el valor d’un pixel del framebuffer.

Abans que els fragments es plasmin en el framebuffer, entren a la unitat d’operacions per fragment on s’apliquen tot un seguit de característiques com textures, efectes de transparència (blending), boira… sempre que aquestes estiguin actives.

Finalment cada fragment tractat es tradueix en un pixel que està situat en un lloc i amb un color dins el framebuffer. Aquest element que ja ha anat sortint anteriorment és una matriu on cada cel la acaba codificant la informació dels pixels que es dibuixaran en el dispositiu gràfic final (pantalla).

\(^8\) NURBS: model matemàtic emprat per representar i generar corbes
El camí comentat fins ara és el que realitza la geometria, però hi ha un camí paral·lel que el realitzen els píxels. La font de píxels és una part del pipeline que pots alimentar-se directament de bitmaps o textures i, eventualment, de dades del framebuffer, els píxels d'aquesta font primer son usats en la unitat d'operacions a píxel on es poden veure afectats per transformacions vèries. A partir d'aquí poden anar directament a la rasterització o passar prèviament per l'ensamblat de textures, unitat que serveix per accelerar les gestions amb textures quan l'aplicació utilitza un gran nombre d'elles.

4. Ray-tracing

4.1. Definició

Ray tracing és un algorisme per a la generació d'imatges tridimensionals. Ideat per Turner Whitted l'any 1980, parteix d'estendre anteriors mètodes emprats per al càlcul de superfícies visibles (ray casting) o per simular trajectòries balístiques i de partícules nuclears (Goldstein i Nagel, 1971). La mecànica bàsica de l'algorisme és llançar un raig per píxel des del punt de vista de l'observador i extreure el color a través d'estudiar les diferents interseccions que experimenta el raig amb els objectes que conformen l'escena.

4.2. Funcionament

L'algorisme de traçat de rajos dibuixa una escena a partir de llançar un raig per cada píxel, per a fer-ho, parteix d'una posició anterior a la finestra de visió que és on es trobaria l'observador i, fixat un punt, defineix rajos tals que la direcció d'aquests s'expressa amb el punt on es troba observador com a origen i el punt situat al píxel actual com a destí.

Per a cada un dels rajos "primaris" es mira la intersecció més propera que fan amb algun element de l'escena, un cop localitzada, el color del raig i per extensió del píxel associat a ell queda definit com la suma de tres factors: l'aportació local, la reflectida i la transmesa (figura 10).

![Figura 10: Representació dels 3 tipus de raig possibles donada una intersecció](image)

La contribució local es calcula a través de localitzar les llums que incideixen en el punt d'intersecció, és a dir, aquelles tals que entre llum i intersecció no hi ha cap objecte en mig que pugui deixar el punt a l'ombra. Per cada una de les fonts de llums que compleixen aquesta propietat s'avalua la seva
contribució al color final, essent calculada en base a les característiques pròpies de la font i de l'objecte il·luminat.

Quan la superfície d'intersecció és reflectora, part del color final s'expressa per la contribució reflectida. Per calcular-la es llena un nou raig “reflectit” que pren la trajectòria en base al raig “primari” i a la normal del punt d'intersecció. De manera anàloga, si l'objecte és transmissor, generar un raig cap a l'interior d'aquest en base a la llei de Snell. (figura 11)

![Diagrama de la llei de Snell](image)

**Figura 11:** Exemple de refracció d'una ona

La llei de Snell descriu el fenomen de la refracció, el qual es dóna quan una ona incideix sobre la superfície de separació entre dos medis (amb índex de refracció diferents), transmetent una part de la seva energia al segon medi i canviant la seva direcció de propagació en conseqüència. Més concretament, aquesta llei diu que si \( n_1 \) i \( n_2 \) són els índex de refracció dels dos medis i si \( a_1 \) i \( a_2 \) són els angles que formen el raig incident i el refractat respecte una línia perpendicular a la superfície, és complex que \( n_1 \sin a_1 = n_2 \sin a_2 \).

El raig reflectit i transmès actuaran com el raig primari, de tal manera que se'n calcularà la intersecció més propera i d'ella se'n derivaran nous raigs reflectit i transmès, definit-se el mètode de càlcul del color com un sistema recursiu que dóna lloc a l'anomenat arbre de raigs traçats (figura 12).

![Diagrama de l'arbre de raigs traçats](image)

**Figura 12:** Arbre de raigs traçats

La recursivitat plantejada durarà fins que el raig no intersecti amb cap element de l'escena (prendrà per color el del fons de l'escenari), la seva aportació al color final sigui inapreciable o topí amb una font de llum. Per altra banda, cal tenir present que cada un dels tres factors afecta en diferent mesura al color final del raig i per extensió del píxel, així doncs, la majoria del color vindrà definit per la llum.
directa (contribució local), mentre que el raig reflectit i transmès, a mesura que vagin intersectant, aniran afectant en menor grau al color definitiu.

A continuació es presenta una escena per tal d'exemplificar el comportament del ray tracing:

Figura 13: Exemple de traçat de rajos

En l'exemple proposat es pot observar el comportament d'un dels molts rajos que surten des de l'observador cap a l'entorn a visualitzar: El raig es propaga fins que intersecciona amb un objecte (intersecció A), un cop allà es calcula la contribució local traçant línies cap a les diferents fonts de llum (raig ombra) per tal de comprovar si hi ha algun element entre la font de llum i el punt d'intersecció, en aquest cas concret, el primer punt d'intersecció està a l'ombra de totes dues fonts ja que per una banda es veu tapada pel “núvol” i per l'altra per la pròpia esfera a la que pertany el punt. De totes maneres el punt actual tindrà color diferent del negre absolut i això és gràcies al color aportat pels rajos reflectit i transmès respectivament.

El primer reflectit surt de l'escena completament, així que el seu color queda definit en la seva totalitat com el “color de fons”, per altra, el raig transmès arriba a interseccionar amb l'interior de l'objecte (intersecció B), generant un nou parell de rajos reflectit i transmès que en definiran el seu color.

4.3. Variants

Existeixen altres versions de l'algorisme de traçat de rajos que també formaran part de l'àmbit d'estudi, aquests són: ray casting, path tracing, distributed ray tracing i two-pass ray tracing.

El ray casting és l'antecessor del ray tracing, dissenyat l'any 1968 per A. Appel divergeix en el traçat de rajos clàssic en el fet que només tracta els rajos primaris, deixant de banda tots els fenòmens lligats a reflexió i transmissió. El punt fort d'aquest algorisme radica en que la seva simplicitat permet que s'executi més ràpid que altres de la seva espècie, però per contra, el resultat és més pobre.
El **path tracing** es planteja l'any 1986 en un article de James Kajiya com una adaptació dels principis físics que es remunten de les ecuacions d'electromagnetisme de Maxwell. El mètode parteix del concepte del "passeig aleatori", una formalització matemàtica d'una trajectòria definida a traves de prendre successius passos en direccions aleatòries, en concret, el que es fa és llençar "n" raigs per píxel, on "n" sol rondar un valor de 40 i per cada raig, quan es produceix una intersecció, enllà de generar el parell reflectit i transmès, es calcula un únic raig de manera aleatòria (segons la distribució de probabilitat associada al material de l'objecte intersecat).

El punts forts de l'algorisme són que simplifica l'estructura de raigs exponencial per una de lineal (veure figura 14), al mateix temps, al traçar múltiples raigs per píxel suavitza el color final d'aquest amb un mostreig més ric a l'hora de definir-ne el color i, finalment, amb l'aleatorietat s'aconsegueix que els "n" raigs tendeixin a produir resultats diferenciats, donant un aspecte més variat en la llum difosa ja que deixarà de ser uniforme i un espectre especular un xic desviat. El principal problema de l'algorisme és que a vegades pot donar-se una excessiva aleatorietat que repercutixi negativament al resultat final.

![Figura 14: Arbre de raigs traçats del path ray tracing i del distributed ray tracing](image)

El **distributed ray tracing** va ser desenvolupat l'any 1984 per R. L. Cook, T. Porter i L. Carpenter amb l'objectiu principal d'afegir al ray tracing la suavització dels fenòmens de la llum. La idea de l'algorisme parteix que en el **ray tracing clàssic** els mostrejos per avaluar els fenòmens de la llum es realitzen amb un sol raig, produint un resultat binari: o es dona el fenomen pertinent o no es dona, així per exemple, en el cas del càlcul d'ombres el mostreig es realitza traçant una recta des del punt d'intersecció a cada font de llum per veure si aquest està a l'ombra (hi ha alguna intersecció entre punt i llum) o no, generant com a conseqüència que un punt no pot estar parcialment iluminat per una font concreta. Paral·lelament, per als fenòmens de reflexió i transmissió també es traça un únic raig, produint novament un efecte de manca de realisme al tenir reflexions i transmissió completes o nul·les en un punt.

Amb l'objectiu d'eliminar aquests efectes, el **distributed ray tracing** pren múltiples mostres per punt, així doncs, a l'hora d'avaluar ombres, transmissions i reflexions es tracen "n" raigs segons una certa
distribució de probabilitat, obtenint un resultat més suavitzat gràcies a un mostreig més ric que dóna lloc a resultats parciaus (veure figura 14). La principal pega d’aquest algorisme és l’augment del cost computacional que implica tractar “n” raigs enfront d’un per avaluar la casuística dels fenòmens de la llum.

Encara que el distributed i el path ray tracing aporten un millora a l’algorisme base al introduir el concepte de mostreig múltiple, tenen una mancança important en determinades escenes, un exemple d’això seria la composició que es pot veure en la figura 15, en ella hi ha un focus de llum que il·lumina un mirall, aquest reflecteix la llum sobre una paret provocant que es vegi el lluminada encara que no està directament afectada pel focus. En els algorismes anteriors, al traçar-se raigs des de l’observador quan aquests impactessin amb la paret, al no reflectir-se en ella, no arribarien a representar l’efecte del reflex de la llum en el mirall, donant un resultat erroni.

Figura 15: Exemple d’escena problemàtica per a certs algorismes de ray tracing

Davant la problemàtica plantejada neix el two-pass ray tracing o bi-directional path tracing un algorisme definit com una combinació del path tracing i el light tracing (versió del path tracing on els raigs es llacen en direcció oposada, és a dir; de la llum a l’escena) que parteix de la base de traçar raigs des dels dos elements clau: observador i fonts de llum. Així doncs, aquest algorisme consta de dos passos diferencials: el primer pas té com a objectiu dipositar energia en les superfícies difoses de l’escena, per tal de dur-ho a terme, des de cada font de llum es llacen “n” raigs cap a l’escenari seguint les reflexions especulars fins topar amb una superfície difosa, en altres paraules, el raig seguirà rebutjant fins que el punt d’intersecció tingui propietats difoses i un cop intersectat el punt difós, s’emmagatzemarà l’energia dipositada sobre la superfície.

Finalitzat el primer pas, el segon pas es tracta d’executar un path ray tracing (traçar raigs des de l’observador ), però amb el matís que quan un raig incideix en una superfície difosa, s’empra el color calculat en el primer pas de l’algorisme. Amb aquest afgit, en l’escena conflictiva de la figura 15, el primer pas de l’algorisme llançaria raigs des del focus de llum, en impactar en el mirall, com aquest és especular provocaria el traçat de nous raigs que arribarien a la paret i carregarien d’energia els punts tocats d’aquesta. Ja en el segon pas de l’algorisme, el raigs de l’observador anirien intersectant amb la paret però, a diferència d’altres algorismes de traçat de raigs, en els punts on el mirall hagué reflectit
el focus de llum el color difós es veuria modificat per aquesta energia desada, produint una paret fosca excepte en els punts pertinents que es veurien il·luminats de manera tènue.

5. **Comparació d’OpenGL vers Ray tracing**

OpenGL i Ray tracing són dos paradigmes que en darrera instància permeten visualitzar entorns tridimensionals, ara bé, els principis dels que parteixen provoquen propietats finals diferents en termes de qualitat i temps de generació d’imatge, ja que per una banda OpenGL utilitza un model d’il·luminació local i empíric, on un model local implica que el color en un punt no té present la resta d’elements de l’escena ja que només té en compte on es troba cada focus de llum, el material de l’objecte actual i la posició de l’observador per calcular el color d’un pixel i on empíric fa referència al fet que s’utilitza una fórmula aproximada que dóna resultats acceptables per al càlcul del color. Per altra banda Ray tracing utilitza un model d’il·luminació global i físic, on un model global comporta que el color en un punt té present la resta d’elements de l’escena, el medi on està l’entorn (aire, aigua, espai ...), on es troben les fonts de llum i el material de l’objecte i on físic fa referència a que el càlcul del color és seguint una base física.

Com a conseqüència del model d’il·luminació de partida, el principal aspecte d’OpenGL és poder mostrar entorns tridimensionals de gran complexitat en temps real donada la seva simplicitat, per contra, les imatges que mostra en principi són molt sensibles perquè només tenen present la il·luminació directa (model d’il·luminació local), que és aquella que arriba procedent d’una font de llum. Per altra banda, el ray tracing mostra imatges amb un gran realisme perquè simula el comportament de la llum de forma més fidedigna, tenint present no només la il·luminació directa, sinó també la indirecta (model d’il·luminació global), que és aquella que arriba per reflexió o transmissió en altres objectes. Conseqüentment, el traçat de raigs genera de forma automàtica efectes com ombres, miralls i qualsevol altre tipus de reflexió, transparències, etc.

Davant aquest panorama ray tracing es presenta com un mètode millor a OpenGL i similars pel que fa al realisme de la imatge final, el problema és que mostrar tant de realisme suposa un cost computacional elevat que no permet la renderització a temps real. Apart, els objectes de gruix proper a zero com “rectes” no són visibles en ray tracing, ja que només allò que és intersectat per un raig pot ésser visionat.

S’ha anat desenvolupant certes tècniques que poden treballar sobre OpenGL com els **shadows maps** o les textures que intenten simular aquells efectes que sorgeixen quan s’empra un sistema d’il·luminació més ric o un millor modelat. Encara que utilitzar aquests mètodes augmenta la complexitat de càlcul, segueix mantenint-se el renderitzat a temps real, deixant una qualitat d’imatge acceptable en termes de foto-realisme.

Així doncs, encara que ray tracing és capaç de generar un foto-realisme superior al d’un OpenGL ampliat amb certes afeigits, aquest segon és l’opció escollida per qualsevol aplicació que desitgi una interacció a temps real (actualment ja existeix interacció a temps real amb ray tracing, però sol requerir molta potència i normalment sol donar-se en escenes de reduïda complexitat).
1. Introducció documental

L'apartat següent documenta la planificació prèvia al projecte, expressant les fites temporals que s'esperen assolir inicialment i generant un esbós dels costos de la realització completa recollits en un pressupost.

2. Planificació temporal

El projecte es planteja per a una durada d'uns 6 mesos. Inicialment es realitzarà una etapa de planificació on es pretén estudiar l'àmbit de recerca en el que es mourà el projecte, plantejar l'estructura de l'aplicació i especificar els requisits que ha de satisfacer. Posteriorment es realitzarà una fase d'implementació de l'aplicació vinculada al projecte i, finalment, es realitzaran els test de proves finals, retocs i ampliacions opcionals i formalització dels darrers tràmits per a la correcta entrega.

La primera etapa arrencarà amb la concepció de les bases del projecte, definint les fites més destacables que es pretenen assolir. Un cop posades en comú amb el director, es profunditzarà en els objectius més específics de l'estructura i es plantejarà una planificació global del projecte per atendre les necessitats definides. En paral·lel, s'estudiarà l'àmbit del projecte per tal de preparar-se per les properes etapes, s'ampliaran les bases plantejades amb la nova informació generada amb l'estudi i s'arrencarà el redactat de les memòries, documentant els aspectes que introdueixen el context en el que es mourà el projecte.

Acotats els elements introductoriis, s'iniciarà l'aplicació associada al projecte. Aquesta aplicació constarà de tres parts diferenciades: un visualitzador d'escenes en OpenGL, un visor d'animacions de rajos, encarregat de mostrar com funciona internament cada algorisme de traçat de rajos i, finalment, el visor de Ray tracing, que mostrarà el resultat de les interaccions amb la resta d'elements.

La planificació per a cada una de les parts serà la mateixa, inicialment es dissenyarà l'estructura UML per tenir una visió aproximada dels elements que constituiran aquella secció d'aplicació. Posteriorment es realitzaran els storyboards, que pretendran especificar la interacció gràfica que trobarà l'usuari final i, per acabar, s'elaborarà una especificació del codi més significatiu.
A mesura que les planificacions estiguin dissenyades es combinarà la feina de disseny amb la d’implementació per tal de diversificar el treball i aprofitar possibles feedbacks que es generin en el procés. Per acabar el projecte, es destinarà un temps per a proves de cohesió de tots els elements, verificant el correcte funcionament dels mòduls per separat i funcionant com un conjunt, apart es finalitzarà la redacció de la memòria, es realitzarà el redactat d’un manual que introduirà i a l’usuari a l’aplicació i, en la messura que sigui possible, s’estudiarà afegir alguna millora a la base creada.

3. Pressupost

El cost del projecte queda especificat en funció de tres perfils: cap de projecte, analista i programador. Per a la planificació anteriorment presentada es treballa a 4h/dia i a un cost de 45, 32 i 23 €/h depenent del perfil. A partir d’aquí, les hores de concepció de les bases i planificació corren a càrrec del cap de projecte, l’estudi sobre el ray tracing, el plantejament de les especificacions i els test de cohesió seran responsabilitat de l’analista i, conseqüentment, la feina d’implementació i la generació de la documentació es comptabilitzarà com a feina de programador.

Sota aquestes bases, el cost final del projecte és de **14620€** tal com queda desglosat en la taula inferior:

<table>
<thead>
<tr>
<th></th>
<th>Hores</th>
<th>€/hora</th>
<th>Total (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cap de projecte</strong></td>
<td>92</td>
<td>45</td>
<td>4140</td>
</tr>
<tr>
<td><strong>Analista</strong></td>
<td>120</td>
<td>32</td>
<td>3840</td>
</tr>
<tr>
<td><strong>Programador</strong></td>
<td>280</td>
<td>23</td>
<td>6640</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td><strong>14620</strong></td>
</tr>
</tbody>
</table>

El geni comença les grans obres, però només el treball les acaba

Joseph Joubert (1754-1824)
1. Introducció documental

L’apartat següent documenta l’especificació del programa vinculat al projecte i part de la seva implementació separada per mòduls conceptuels. Per a cada un dels mòduls primer de tot es descriu la raó de la seva existència en el punt d’introducció i ja posteriorment es cita l’estructura aproximada en UML que presentarà.

En el punt de disseny es comenten alguns dels punts conflicts en termes d’implementació, raonant les eleccions que s’han considerat, les dificultats localitzades i adjuntant pseudocodi per plasmar de manera més precisa la realització final.

Ja per acabar, s’adjunta un punt d’ampliacions que pretén mostrar quins camins d’evolució podria prendre l’aplicació, cal tenir present que és un projecte tancat amb uns objectius i temps fixe i aquest apartat mostra elements addicionals que podrien afegir-se per abarcar nous objectius que es podrien traduir en versions posteriors del software.

2. Introducció

L’objectiu primordial del projecte és desenvolupar un software que ajudi a la comprensió de l’algorisme de ray tracing i les seves variants, per tal d’acomplir aquesta fita es planteja una aplicació amb el següent funcionament: l’usuari navega en una escena tridimensional sota tecnologia OpenGL, estableix el punt de vista amb el que es desitja treballar, escull entre executar un traçat de raux complet o, amb la intenció de comprendre l’algorisme de visualització, prèmer punts de la pantalla de manera que per cada d’ells es llenci un raig.

Arribats aquí, l’acció es trasllada a un altre entorn on es presenta una animació dels raux acotats per l’usuari, podent observar com van impactant amb l’escena, calculant el seu color, descomposant-se en reflectit i transmès, etc. El darrer component és un visor de ray tracing que permet visualitzar el resultat final conseqüència de la resta d’accions realitzades en el visor d’OpenGL i l’animador de raux.

En la imatge adjunta (veure figura 16) és veu la interfície que gestiona el conjunt de funcionalitats explicats, en línies generals la pantalla consta de 4 subfinestres: visor d’OpenGL, llençador de raux, animador de raux i visor de Ray tracing.

La idea és que cada una de les 4 subfinestres inicials es gestionable via la barra d’eines superior, la barra d’eines pròpia, la interacció a través de ratolí o teclat i, finalment, fent ús de la zona d’informació ampliada, que es destina per a mostrar informació tal com les característiques de l’objecte seleccionat en el visor OpenGL o les propietats actuals de l’algorisme de traçat de raux usat.
Figura 16: Entorn de l’aplicació comentat amb un model carregat

Concretant les funcionalitats, en la zona superior esquerra de la imatge hi ha el visor d’OpenGL per gestionar elements, en ell es pot navegar pel model carregat utilitzant el ratolí per fer rotacions i zoom i ajudant-se de les tecles de direcció del teclat per al pan. A través de la zona d’informació ampliada es pot modificar les propietats òptiques de les fonts de llum i objectes que siguin seleccionats realitzant doble-click o bé fent ús dels combo box pertinents d’aquesta mateixa zona.

A la dreta es troba el visor d’OpenGL per llençar raigs, a diferència del visor anterior, en aquest segon no es pot navegar pel model (el punt de vista d’aquesta subfinestra el fixa el visor d’OpenGL per gestionar elements a través d’un botó de la seva barra d’eines) i només està permès marcar els punts de destí de fins a quatre raigs de llum experimentals prement la zona amb el ratolí. Lògicament es permet esborrar i modificar l’elecció dels quatre raigs a través dels botons facilitats en la barra de la subfinestra.

En la zona inferior esquerra s’ubica l’animador de raigs, al igual que el primer visor té una navegació en el model tridimensional a través del ratolí i el teclat, però no permet seleccionar cap element de l’entorn, enlloc d’això mostra els raigs de llum que poden ser reproduïts en diferents moments del temps amb el botons de la seva barra d’eines.

Finalment a la dreta de l’animador hi ha el visor de ray tracing que no permet cap tipus d’interacció directa ja que només mostra imatges renderitzades en base als criteris acotats en la resta de l’aplicació.
Figura 17: Entorn de l'aplicació amb un model carregat

El resultat final es pot apreciar de manera més significativa en la figura 170 en que es veu en plena acció les 4 subfinestres de l'entorn. En un altre ordre de coses, les 4 subfinestres interactuen entre elles passant-se informació (veure figura 18), de manera resumida, el visor d'OpenGL encarregat de seleccionar elements marca les propietats òptiques dels materials i les fonts de llum de l'escena que veuen la resta de visors, apart a través d'un botó accessible en la seva barra d'eines marca el punt de vista des d'on es veurà l'escena en el visor de ray tracing i en l'encarregat de llençar raigs. Per altra banda, el visor d'OpenGL que llença raigs per cada punt que es marca en ell comunica aquesta informació a l'animador de raigs per convertir-lo en un punt de destí d'un nou raig.

Figura 18: Diagrama d'interacció entre els visors
3. Estructura

Cada un dels diferents visors que conformen l’aplicació tenen un conjunt d’elements en comú que es plasmen dins l’escena base. Tal i com es veu en el diagrama adjunt, aquesta escena está formada per un conjunt de llums, un seguit d’objectes i un llistat de materials.

![Diagrama UML](image)

*Diagrama UML 1:* Diagrama de classes en comú de tots els visors

Analitzant el diagrama, la classe escena té com a atribut més important un model geomètric codificat com una llista d’objectes, un seguit de llums per il·luminar-lo i materials per matizar el seu acolorat, apart té un centre de coordenades i informació sobre els elements seleccionats, paral·lelament les principals operacions aplicables en una escena són carregar-ne una de feta anteriorment des de memòria, desar l’actual o dibuixar-la per pantalla.

Una escena té fins a 9 llums, cada llum té un identificador, una posició en l’espai i un seguit de propietats òptiques com les intensitats ambient, difosa i especular que indiquen com il·luminen els objectes des de qualsevol lloc (ambient), quan la direcció a la font de llum i el pla del punt compleixen cert angle (difosa) i quan a més a més aquest angle és el mateix que l’angle amb l’observador (especular). Com a principals operacions les llums poden encendre’s o apagar-se (interruptor) i dibuixar-se en pantalla.

Per altra banda, una escena té un model geomètric expressat en un llistat d’objectes, cada un d’ells té un nom que pot ser definit per l’usuari, informació de la seva geometria expressada a través de triangles formats per vèrtexs amb les seves normals i coordenades de textura (indiquen com serà col·locada la textura sobre el triangle) i com a principal aplicació un objecte pot ser dibuixat. Finalment, les propietats òptiques d’un objecte queden definides en base a un material dels oferts per l’aplicació.
Tutor de ray tracing: Ray Lab

Un material és el recull de propietats òptiques aplicables a un objecte, concretament un material està format per un nom, definible per l’usuari, un seguit de propietats d’actuació vers la llum ambient, difosa i especular (constants d’emissivitat) i, finalment, aspectes de comportament per als algorismes de traçat de rajos (Kt, Kr i coeficient de refracció).

Apart d’això, els materials es divideixen entre els que es relacionen directament amb l’escena són els aomenats “pre-definit”, invariables i que poden ser utilitzats per més d’un objecte, per altra banda, hi ha materials definits pel propi usuari que no estan vinculats directament a l’escena, sinó a cada objecte en particular, així doncs, cada objecte té un material pre-definit i un generat per l’usuari. Per a major enriquiment, un material pot ser una textura, oferint un resultat final més proper a la realitat i més intuitiu per a l’experimentació.

4. Disseny

En línies generals s’empraran les següents tecnologies per a l’aplicació: C++, OpenGL i Qt.

Donada que és una aplicació que executa algorismes de cost elevat com el traçat de rajos, interesa implementar en un llenguatge que tingui un bon rendiment, és per aquesta raó que s’ha optat pel C++, a part que al estar orientat a objectes facilita la comprensió i gestió del codi d’una aplicació extensa. Altres alternatives descartades eren el C, però la seva falta d’orientació a objectes fa que resulti menys agradable com a entorn de treball o Java que si bé és cert que es caracteritza per ser un entorn amigable i també amb orientació a objectes, és dels 3 llenguatges citats és més ineficient.

Per a la navegació prèvia a la visualització amb traçat de rajos s’ha escollit OpenGL, bàsicament per ser un dels entorns més coneguts d’aquest àmbit, per ésser obert i multiplataforma.

Finalment, per a tota la gestió de modificar valors, mostrar informació, etc. s’utilitzarà Qt, una biblioteca multiplataforma per a desenvolupar interfícies gràfiques que utilitza C++ com a llenguatge núvol i que aporta tot un seguit d’elements com “combo box”, “scrolls”, “quadres de text”... que resulten indispensables per a elaborar una interfície adequada a les necessitats planteades per a aquesta l’aplicació i que fan més seductora aquesta alternativa en contra d’altres com SDL que si bé és més eficient, suposaria un esforç previ molt gran per tal de crear un entorn amb tots aquests “elements”, desviant-se dels objectius del projecte.
Visualitzador OpenGL

1. Introducció

El visualitzador d'OpenGL té com a objectiu oferir un entorn que permeti a l'usuari definir l'escena que vol mostrar amb traçat de raigs, entenent una escena com un model geomètric i les dades que en vinculen la manera en que és mostrat: il·luminació, punt de vista, materials,... Més concretament, aquest mòdul permet navegar pel model tridimensional, escollint el punt de vista d'estudi i les propietats dels elements que conformen l'escena. Referent a aquest darrer aspecte, la idea és poder decidir les propietats òptiques dels diferents objectes com de les fonts de llum, podent avaluar el comportament dels raigs envers situacions diferenciades.

2. Estructura

L'estructura d'aquest mòdul està conformada per 3 grans elements: una escena i dos visors. L'escena exposada és l'element genèric que utilitzen els diferents blocs de l'aplicació i que s'ha comentat a anterioritat (veure diagrama UML 1), per altra banda, els dos visors es complementen en l'objectiu d'oferir un entorn per preparar el context del traçat de raigs.

![Diagrama UML 2: Diagrama de classes dels visors de selecció d'elements i llançament de raigs]

Analitzant el diagrama, la classe “visorOpenGL” té com a principals operacions dibuixar l'escena i “seleccionar” els diferents elements que la conformen per tal de modificar-ne les seves propietats òptiques, amb l'objectiu de facilitar aquesta tasca, el visor disposa d'una ajuda en forma de fletxa
tridimensional dibuixada en el marge inferior dret que apunta a l'objecte / llum actual per tal d'identificar-lo de manera immediata. Aquesta fletxa queda definida per un objectiu on assenyalar; unes coordenades on dibuixar-se i les dades de la geometria que la conformen. Paral·lelament, el “visorOpenGLRajos” es centra en “lleçar” rajos damunt l’escena, aquest rajos són empleats posteriorment per la resta de visors en l’àmbit de recrear imatges via ray tracing. Per tant, la idea és que primer definim les característiques de cada un dels elements de l’escena i a continuació situem una càmera adequada des d’on anem a traçar rajos de llum.

Finalment cada visor té una càmera, aquesta classe queda definida per un punt on es troba l’observador, un objectiu a enfocar, un angle de visió, un plans de retallat de la imatge (near i far) que indiquen a partir d’on i fins a on es veuen elements i informació sobre les rotacions aplicades respecte la col·locació inicial. Com a principals operacions la càmera pot moure’s, dibuixar-se i reiniciar els seus valors.

D’aquesta manera les funcionalitats de la subfinaestra d’OpenGL encarregada de seleccionar elements són:
- Navegació interactiva a través del ratolí i del teclat (rotació, zoom i pan).
- Selecció d’elements de l’entorn tridimensional: objectes i fonts de llum
- Manipulació de les propietats òptiques dels diferents elements de l’entorn
- Fixar punt de vista per al traçat de rajos.

Per altra banda, les funcionalitats de la subfinaestra d’OpenGL encarregada de llançar rajos són:
- Gestionar els punts de destí dels rajos de llum arbitraris

3. Disseny

3.1. Carregar i desar informació

El visor d’OpenGL treballa amb dos tipus de fitxers: el model tridimensional i el descriptor d’escenes. El primer tipus ve donat de forma externa, doncs es fa ús del format “3ds” per a rebre la codificació de les maltes tridimensionals, la raó d'escollir aquest format és perquè és un format molt estès i codifica tota la informació sobre la geometria (veure apartat de gestió de 3ds) que pugui usar aquesta aplicació.

3.1.1. Gestió de 3ds

El tractament dels documents en “3ds” es dóna a partir de convertir aquest format a un de genèric, adequant-se completament a les necessitats de l’aplicació i deixant la porta oberta a extensions que aportin l’acceptació d’altres documents 3d.

El format genèric plantejat es troba expressat en el diagrama UML i on el model 3dStudio amb el que l’aplicació estiguí treballant queda definit com una llista d’objectes independents dins una escena. En concret el model 3ds original defineix els objectes i el seu material inicial, per a fer-ho es desa el llistat de vèrtex i les seves normals, el llistat de triangles i, en cas d’estar definides, les coordenades de textura com a dades del format genèric. Apart, es defineixen dos materials per objecte: el d’usuari que sol iniciar-se amb el material del model 3dStudio (si el model no té material s’assigna un per defecte),
Tutor de ray tracing: Visor d'OpenGL

i que es modificable completament i el material actual, que indica el material que s'està utilitzant en el pintat de l'objecte, prenen per valor les dades del material d'usuari o les d'un material pre-definit en l'aplicació. Apart de tractar amb dos tipus de materials, el format genèric dissenyat descarta tota la resta d'informació del "3ds" com il·luminació, textures associades, esquelets... i per contra afgeix un sistema d'il·luminació i de gestió de textures propi.

En línies generals, les dades d'un model 3ds es troben estructurades en forma d'arbre, on cada node pot contenir informació diversa (fons de llum, malles de triangles, dades definides per l'usuari...), però per al context de l'aplicació, l'única informació rellevant és la d'aquells nodes\(^9\) que defineixen objectes 3d. Un fitxer de 3dStudio pot contenir diversos objectes que es poden trobar desats en l'arbre de dades de formes diferents, això és així perquè no hi ha cap estàndard que delimiti com desar la geometria i, per tant, cada aplicació que opera amb models 3ds utilitza el seu propi criteri per tal de guardar la geometria de la manera més idònia.

La idea final és que donat un fitxer 3ds es vol obtenir un llistat d'objectes als que poder assignar propietats òptiques diferenciades, però per això cal seguir un criteri de classificació de la informació, així per exemple, es pot decidir que cada node que conté informació d'un objecte tridimensional acabi sent un objecte en l'aplicació o bé que donat un node amb geometria, tots els seus fills s'interpreten com parts d'aquesta (per exemple un node cos, pot tenir com a fills cames, braços i cap) i per tant, tot el subarbre s'interpreta com un sol objecte final, davant un ventall de situacions tan diverses es plantegen dues alternatives: aplicar un algorisme d'inundació o aplicar un criteri arbitrari.

![Figura 19: Dofí de 3d studio junt amb un zoom de la secció del seu ull.](image)

Un algorisme d'inundació és basa en seleccionar un vèrtex a l'atzar i començar a marcar tots aquells que són accessibles a partir d'un camí qualsevol des d'aquest vèrtex d'origen, un cop hem marcat tots els accessibles, si encara hi ha que no han estat visitats vol dir que pertanyen a un altre objecte o component connexa. El problema d'aquesta solució és que resulta costosa ja que primer cal un pre-tractament per extreure els vèrtexs de l'estructura de dades del fitxer 3ds i un cop fet això cal iterar sobre aquests múltiples vegades fins tenir-los tots classificats, a part no sempre és un resultat satisfactori, un exemple d'això el trobem a la figura 19, on l'ull del dofí no té cap vèrtex en contacte

---

\(^9\) Els nodes que contenen objectes 3d són aquells que el seu tipus val "LIB3DS\_OBJECT\_NODE"
amb la resta del cos (destacat amb la zona pintada de vermell) i, per tant, davant una inundació resultarien objectes diferents quan conceptualment resul ten ésser un de sol.

Per altra banda, aplicant un criteri arbitrari també poden sorgir fenòmens com els que veiem en inundació, però per contra el sistema serà menys costós de dur a terme i més eficient. Un criteri raonable pot ser el fet de suposar que els fills que contenen informació sobre models tridimensionals que pengen de l’arrel principal són, cadascun dels, components connexes diferents i els nodes amb geometria dels seus sub-arbres són parts d’aquestes components, com resulta ésser un braç del cos d’una persona. Lògicament, aquelles escenes que no estiguin modelades amb aquest esquema veuran un desglossament en objectes imprevist, però el tractament de totes les peculiaritats de com cada aplicació desa 3ds no és objectiu d’estudi d’aquest projecte.

De manera més específica, les principals dades de la conversió a un format genèric són els vèrtexs, normals i triangles del model, que són desats en objectes diferents en base al criteri anteriament comentat. A part d’aquesta informació, també són recollides les coordenades de textura, però a diferència de les anteriors dades, com no sempre venen donades, per aquells models on manquin es planteja un mapat automàtic (la textura s’aplica sobre els polígons seguint un criteri que no requereix tenir coordenades de textura definides), en concret l’ús de “sphere mapping” (mapat que assigna punts de la imatge sobre els polígons com si estiguessin en una esfera), que tendeix a donar bons resultats exceptuant que davant escenes mòbils les textures semblen lliurar damunt la superfície.

Finalment, en els models de 3ds el material ve definit a nivell de vèrtex, però guardar tota aquesta informació crearia una disparitat enorme entre els materials carregats del model i els que aporta l’aplicació, de cara a donar un entorn homogeni, la solució passa per només seleccionar el primer material definit en algun vèrtex, quedant en tot l’objecte un material homogeni, resultant indistingible l’ús d’un material predefinid per l’aplicació a un del fitxer 3ds.

### 3.1.2 Gestió de XML

El descriptor d’escenes és l’altre tipus de fitxers amb els que treballa l’aplicació, la seva funcionalitat és oferir un suport on poder-hi desar el conjunt de característiques aplicades a l’escenari per part de l’usuari (el nom dels objectes i la informació dels materials associats a ells i les dades de cada una de les fonts de llum), amb motiu de codificar aquesta informació s’utilitza el format “xml” que destaca per ser un estàndard de codificació d’estructures d’informació senzill i lleigible.

L’estructura implementada per codificar l’escena es presenta a continuació, en ella es pot veure que una escena té el nom d’un model 3ds associat, un llistat d’objectes que es poden extreure d’aquest model i un seguit de llums, on per cada un objecte i llum se’n especificuen les característiques asignables a través de l’aplicació.
Per tal de poder tractar aquesta informació es fa ús de DOM, un dels dos parsers d'xml que facilita Qt, la idea bàsica és que DOM interpreta els fitxers xml com a arbres, d'aquesta manera, per llegir l'escena senzillament identifiquem el node arrel com a "RayLabScene" i es viatjarà a través dels diferents fills que conformen objectes, materials i llums.

El motiu per escollir DOM enlloc de SAX, l'altre parser inclòs en Qt, és perquè l'aplicació no només ha de llegir xml, si no també modificar-ne el contingut, és en aquest segon aspecte on DOM és més adequat. L'escriptura es realitza de la mateixa manera que la lectura, viatjant per l'arbre associat al xml, però aquest cop modificant-ne el valor.

3.2. Navegació entre objectes/llums

Els dos elements que canvien el comportament de l'algortisme de ray tracing són les fonts de llum i els objectes de l'escena, per tal de modificar-ne les propietats associades a cada d'ells i poder experimentar amb les diferents situacions, es presenten diferents sistemes d'interacció:

3.2.1. Selecció per rotei

La primera proposta es tracta d'una navegació en l'espai tridimensional, és a dir, per viatjar entre els elements l'usuari canvia el punt punt de vista des d'on veu l'entorn i selecciona l'element pertinent a través del cursor del rotei, d'aquesta manera hi haurà suficient amb fer un doble click en l'objecte o llum que desitgem tractar per tal de tenir accessibles les seves dades.
De cara a la implementació, aquesta primera proposta empra els mecanismes oferts per OpenGL en termes de selecció d'objectes, com es pot veure en l'esquema presentat a continuació, la idea del mecanisme emprat està en crear una piràmide de visió (volum de visualització d'una càmera en un punt determinat) que parteix de les coordenades del ratolí i dibuixar l'escena en mode "selecció", que és una modalitat que permet registrear en una estructura de dades els objectes que queden dins la piràmide de visió, un cop fet això, es recuperen els elements de l'estructura i s'extreu aquell que està més a prop de la càmera (z menor) i, finalment, es gestiona la selecció/deselecció dels objectes en base aquest criteri (veure figura 20).

```c
//Entrem en mode de selecció
glRenderMode(GL_SELECT);

//Projectem la càmera en les coordenades del cursor
glMatrixMode(GL_PROJECTION);
LoadIdentity();
glGetIntegerv(GL_VIEWPORT, matriz);
 gluPickMatrix(ratoliX, ratoliY, dimensioPiramideX, dimensioPiramideY, matriz);
 gluPerspective(fov, aspecte, plaAnterior, plaPosterior);

//
escena.dibuir();

// Tornem al mode de pintat normal
int objectesPintats = glRenderMode(GL_RENDER);
camera.reprojectar();
// Cerquem l'objecte de menor z
for(int i = 0; i < objectesSeleccionades; i++)
 {
   if(zActual < zMinima) seleccionat = objecte[actual];
   ...
 }
// Gestionem la selecció
(...)
escena.dibuir();
```

3.2.2. Selecció per combo box
La segona proposta es basa en una navegació a nivell de desplegables proporcionats per la interfície gràfica, així doncs, la idea és codificar tots els objectes i llums en llistats de l'interfície, en concret un cop
tenim la visió ampliada d'objectes a la zona esquerra de l'aplicació, existeix un combo box o desplegable (veure figura 21) amb tots els noms d'objectes de l'escena, d'aquesta manera, es pot canviar d'objecte seleccionant el seu nom de la llista, evitant cercar-lo dins l'escena amb el cursor. Aquest sistema és completament anàleg a les fonts de llum que estan contingudes en l'escenari tridimensional.

Tal i com es mostra en la figura 21.a, inicialment res es troba seleccionat i la zona d'informació ampliada de la dreta del tot de l'aplicació es troba sense cap informació en concret. A partir d'aquest moment es pot clicar damunt d'un objecte o font de llum o aprofitar el combo box on estan continguts tots els objectes o fonts de llum.

**Figura 21:** Efectes de la selecció d'objectes (A - no seleccionat; B - seleccionat)

A partir d'aquí passem a la situació de la figura 21.b, on es carreguen totes les dades de l'element seleccionat, és a dir: nom de l'objecte, nom del material assignat, coeficient de refacció, constants de reflectecció i transmissió, color del material i textura d'aquest i l'objecte passa a pintar-se amb un altre color per poder ser reconegut visualment (adoptant sempre una tonalitat verdosa).

Es pot donar el cas que, en seleccionar un objecte aquest no es vei inicialment en la projecció actual, davant aquest panorama la primera solució plantejable és la de situar la càmera encarada a l'objecte, però aquesta alternativa té dos problemes en termes d'orientació i visibilitat.

El primer problema, el de l'orientació, fa referència a que fer un moviment brusc de càmera sol desorientar a l'usuari, un exemple es pot donar en una escena on tenim un seguit d'habitacions i, trobant-nos en una d'elles, la càmera salta de cop a una altra habitació provocant una desorientació immediata per part de l'usuari.

El segon problema, el de la visibilitat, fa referència a que quan resituem la càmera és molt difícil establir un criteri de a quina distància i com s'ha de trobar respecte l'objecte seleccionat, ja que és factible que per a un criteri donat, sempre pot existir un objecte que es trobi entre mig de la càmera i l'objecte...
seleccionat. En la figura 22 es mostra com la bola vermella en estar rodejada per diferents boles de mida molt superior i en diferents flancs resulta invisible a diverses col·locacions de la càmara.

![Figura 22: Exemple de problemes amb la recol·locació de la càmara](image)

La solució que es planteja implementar és que la càmara no es mogui, ja que en lloc d’això s’introdueix una ajuda, activable per part de l’usuari, que en forma de fletxa indica cap a on es troba l’element seleccionat respecte la posició actual, permetent a l’usuari comprendre la col·locació concreta del nou membre seleccionat respecte la situació en la que es trobava abans de la selecció.

### 3.2.3. Fletxa Guia

Un cop seleccionat un element pot perdre’s la referència d’on es troba, per tal de poder-lo localitzar en tot moment s’ha ideat el sistema d’una fletxa guia que apunta cap a l’objecte de la selecció actual. A molt alt nivell, la base d’aquest mecanisme seria primer de tot montar la geometria de la fletxa base, després calcular-ne l’orientació en base a on es troba l’element seleccionat i finalment pintar la fletxa en la zona pertinent de la finestra. Referent al segon punt, es pot apreciar en la figura 23 com es calcula l’angle d’orientació de la fletxa a través de l’arctangent dels costats del triangle format amb les coordenades de l’objectiu de la càmara actual (centre de visió) i el centre de l’element seleccionat, lògicament com la càmara de l’escena pot rotar lliurement, el que es fa és treballar amb coordenades de finestra, d’aquesta manera a través d’usar el mètode `gluProject` sobre el centre de coordenades de l’objecte seleccionat en memòria i tenir presents que el centre de la finestra és constant (amplada/2, alçada/2).

![Figura 23: Esquema que mostra com es calcula la rotació de la fletxa d’ajuda](image)

En el moment de la implementació, cal tenir present que l’arctangent ve donada en el domini de \([-0.5\pi\text{ a } 0.5\pi]\) així doncs, per satisfacer la totalitat del gir es divideix la casuística en dos problemes:
quan l'objecte seleccionat rau per sota el centre de la finestra respecte la coordinada “y” i quan no. L'únic diferència en ambdós casos és que cal afegir un gir addicional de 180\degree tal i com es veu en l’esquelet de codi presentat a continuació:

```c
//Calculem el gir de la flètxa guia
costatOposat = centreFinestraX-centreObjecteX;
costatContigu = centreFinestraY-centreObjecteY;
(...)
//Mirem on cau l'objecte en vers la flètxa
if(costatContigu > 0) angleZ = 180 - atan(costatOposat/costatContigu)*180/\pi;
else angleZ = -atan(costatOposat/costatContigu)*180/\pi;
(...)
```

Si bé és cert que la flètxa ha de tenir present el gir de la càmera i el seu moviment (pan), no s’ha de veure influïnciat directament per ells, ja que per simplificar càlculs l’interessant és que la flètxa sempre es vei igual (independentment del zoom) i que, mantenint la posició al centre, sigui vista per la càmera. Per acomplir aquestes característiques la solució passa per introduir una càmera addicional que enfoc a únicament a a flètxa guia.

El darrer aspecte important és el correcte visionat de la flètxa, si la flètxa es troba com un element més de l’escena come el risc de veure’s tapada per algun d’altre, així doncs, la millor manera és dibuixar la flètxa per sobre la resta d’elements. Per tal de dur a terme aquesta idea, el que cal fer és, un cop pintada l’escena, es desactiva el test de profunditat i es pinta la flètxa, d’aquesta manera sobreescriurà el valor dels pixels on sigui pintada, a part, per diferenciar la flètxa de manera més notòria, pintant-la amb mides i colors diferents es crea un efecte de “contorn” diferenciador.

Amb tots aquest criteris, l’aspecte final es pot veure en la figura 24, on una de les esferes està seleccionada (marcada de color verd) i es pot apreciar com la flètxa indica que cal dirigir-se cap amunt i a la dreta per tal de localitzar aquesta esfera la imatge esquerra, abaix a l’esquerra en la imatge central i amunt per la imatge central.

*Figura 24: Exemple de funcionament de la flètxa d’ajuda*
3.3. Gestió de materials

Amb els mètodes presentats anteriorment, l'aplicació ens permet navegar per diferents elements, l'objectiu d'aquesta funcionalitat és que un cop tenim un element seleccionat, a la "zona d'informació ampliada" de la dreta de la interfície (veure figura 16), se'n permet modificar-ne les propietats. Per aquests objectes, la gestió de propietats òptiques es vincula a materials, on un material, pot ser de dos tipus: predefinit o d'usuari. Independientment del tipus, tot material consta d'un seguit de propietats òptiques que defineixen com és comporta envers la llum, és a dir, quina part de l'espectre absorbeix, reflexa i transmet, si és un objecte transparent o opac, si reflexa o és completament mat, etc. A més a més, pot vincular-se una textura per dotar de major realisme. De manera més precisa, per a un material es pot definir el coeficient de refracció, les constants de reflexió i de transmissió de la llum i les constants d'emissivitat ambient, difosa i especular.

Com la idea de l'aplicació és experimentar amb el traçat de raïos, l'entorn ha de procurar facilitar aspectes més secundaris de gestió per poder centrar els esforços en l'experimentació, és per aquesta raó que hi ha dos tipus de materials per cobrir les diferents necessitats. Els predefinits són un conjunt de materials clàssics que estan completament configurats i l'usuari només els ha de seleccionar; desvinculant qualsevol tanteig previ a la cerca d'un resultat específic, així doncs l'aplicació ofereix materials reflectants com el ferro i l'or, transmissors com l'aigua i el vidre o sense comportaments accentuats com la fusta. Per contra, es permet seleccionar el material "usuari" que inicialment ve donat pel model llegit (o valor per defecte) i que permet modificar els diferents camps mostrats en la zona pertinent (figura 25 esquerra)

**Figura 25:** Gestor d'objectes (esquerra), gestor de fonts de llum (dreta)
3.4. Gestió de llums

Anàleg a la gestió de materials en els objectes, quan es tracta de gestionar les llums l'entorn d'informació ampliada canvia l'aspecte per poder modificar les propietats de les diferents fonts. Bàsicament es juga amb dos tipus de llums, les puntuals, que envien rajos en totes direccions i els focus, que envien la llum en una direcció privilegiada, en conseqüència, per al segon tipus de llum cal especificar addicionalment la direcció de la llum i com aquesta s'atenua a mesura que es propaga en l'espai (figura 25 dreta). De manera més precisa, per a una font de llum es pot definir un nom per a identificar-la, el tipus (puntual o focus), la seva posició en l'espai, les components ambient, difosa i especular, la direcció on irradia llum, l'angle del focus, l'exponent d'aquest i l'atenuació que percep en propagar-se.

Mentre els objectes són fixes un cop carregat el model tridimensional, les llums són elements totalment dinàmics, així doncs, la interfície atorga la possibilitat d'afegir i eliminar fonts per tenir diferents composicions lumíniques amb les que jugar; ara bé, el nombre mínim de llums és un, menys no tindria sentit, i el màxim és 8.

En la figura 26 es pot apreciar com pot variar la il·luminació en un model, en la imatge de més a la esquerra es veu l'esfera que simbolitza la font de llum a l'esquerra, després en la imatge central es veu com la font es resituada a la dreta del ninot, canviant de manera conseqüent la il·luminació inicial, finalment, a la dreta de tot s'enriqueix l'entorn afegint una nova font de llum que dota de major il·luminació el conjunt.

![Figura 26: Exemple de gestió de la lluminació](image)

4. Llençar rajos

Fins al moment s'ha presentat aquest primer mòdul com un entorn per configurar objectes i fonts de llum, però tot això és el preludi de la veritable interacció amb la resta de mòduls. El concepte és que un cop tenim l'escena configurada, en la subfinestra de selecció d'elements l'acció es reprèn en la subfinestra llançadora de rajos on per cada punt clicat amb el ratolí, s'inicia el traçat d'un raig.

La codificació d'aquest sistema es basa en traduir les coordenades sobre el visor que es mouen en el rang [0, 0] a [ampladaVisor, longitudVisor] a coordenades en l'espai de l'escena [centreEscena.x-radi,
centreEscena.y-radi, centreEscena.z+radi] a [centreEscena.x+radi, centreEscena.y+radi, centreEscena.z+radi], per tal de fer-ho, la idea passa per realitzar el procés invers d'OpenGL empleant les matrius inverses que utilitza aquest per renderitzar imatges. Amb tot això, primer el punt bidimensional en coordenades del visor es transforma en un punt de tridimensional, afegint-li un “z” amb valor 0 que significa un punt en el pla a partir del qual la càmera comença a veure (near), posteriorment, es multiplica aquest punt per la matriu inversa de visionat del model i de projecció, modelview i projection respectivament.

\[ \text{Figura 27: Model tridimensional amb 4 origen de raig acotats} \]

De cara a generar un realimentació per a l'usuari, els punts es dibuixen en pantalla, per tal de poder garantir que els punts es veinien bé, es pinta més d'un punt amb radi i color diferent per cada origen de raig (figura 27). D'aquesta forma, independentment del color dels elements del fons, els punts marcats podran ser reconeguts satisfactoriament.

Altres aspectes a destacar a l'hora de llançar raigs és que el nombre de raigs simultanis està restringit a 4, així és així perquè de cara a poder visualitzar la interacció de raigs, en la majoria de casos no té molt de sentit veure'n molts de cop, ja que donada la naturalesa de com es desenvolupen (es van dividint recursivament), a partir d'un cert nombre de raigs i depenent de les propietats òptiques dels elements de l'escena, resulta confús la visualització d'aquests enmig de l'entorn tridimensional. En aquesta línia, els punts d'inici poden ser esborrats per poder-los canviar per altres i també poden ser ocultats, aquest darrer aspecte és perquè la imatge generada en el visor de raigs és la mateixa que es mostra en el de traçat de raigs i de cara a crear un marc comparatiu, resulta profitós no veure els punts que indiquen on s'han llançat raigs per a l'animador, ja que aquest punts podrien ocultar zones interessants en l'àmbit de l'observació.

Resumint la interacció del llançador de raigs comença amb l'usuari que clica un punt de la subfinestra, aquest es tradueix utilitzant els mètodes d'OpenGL per a despreprojectció de punts (gluUnProject), el nou punt tridimensional es pinta en l'escena sempre que estigui activa la visualització de raigs, paral·lelament s'envia la informació del nou punt a l'animador de raigs perquè pugui gestionar la creació i animació del raig pertinent. Per altra banda, en el moment que ho desitgi l'usuari el punt serà eliminat a través de prèmer el botó pertinent de la barra d'eines.
5. Ampliacions futures

Les millores que podrien afegir-se al visor d'OpenGL en futures versions es dividirien en dos grans blocs, per una banda, aquelles que fan referència a enquirir la síntesis d'imatges d'OpenGL per tal de tenir un marc comparatiu més just entre el seu paradigma envers el de traçat de rajos i per altra banda, les que giren al voltant d'una interacció amb l'usuari més rica.

Dins el primer grup, les millores més directes serien la introducció d'efectes de transparència (blending) que emfatitzarien l'experimentació d'un càlcul aproximat d'objectes transmissors contra el comportament físic que proporciona el ray tracing. A més a més, els diferents sistemes d'ombres que s'aconsegueixen a través de shadow maps resultaria un altre punt interessant per validar diferències entre un mètode de pintat empíric i un de físic.

Ja en un altre nivell, l'adició de shaders i altres tècniques avançades suposaria un colofó a la síntesis d'imatges d'OpenGL. Lògicament aquestes millores es formulen com ampliacions que enquirin l'experiència, però que s'allunyen de les bases necessàries per assolir els objectius pactats ja que no interessa comparar OpenGL amb ray tracing, sinó entendre el correcte funcionament d'aquest segon.

Passant a aspectes de la interacció amb l'usuari, una de les possibles extensions seria poder interaccionar amb les llums directament, és a dir, en comptes d'un sistema senzill com l'actual on les coordenades de la llum són escrites en quadres de text, es proposaria que el fet de seleccionar l'esfera associada a la font de llum dotés a l'usuari de l'habilitat de poder-la moure amb simples directrius de ratolí i teclat, lògicament la manipulació de llums actual compleix els objectius marcats de poder determinar les característiques òptiques i aquesta interacció és un punt menor que no modifica el resultat final del programa, sinó que només agilitza la interacció i per contra requereix desviar-se de la planificació inicial del projecte i, a més a més, possiblement implicaria més canvis per part d'un usuari que sol fixitaria més gestions associades al ratolí i no a quadres de text.

A part d'aquesta interacció amb les llums, aquest concepte podria extendre's als objectes de l'escena, permeten modificar la seva posició de manera dinàmica per poder adaptar la seva col·locació als interessos de l'experimentació.

Finalment podria resultar interessant la creació d'un sistema de guardat de materials, així doncs, si l'usuari crea nous materials aquest serien accessibles per a qualsevol nova escena generada tal i com ho són els materials predeterminats, reduint els temps de gestió. Anàlogament, es podrien crear patrons de fonts de llum (Sol, fluorescent, bombeta vermella...) que permetessin generar il·luminació en l'entorn de manera més ràpida.
Animador de rajs

1. Introducció

L’animador de rajs té com a objectiu mostrar la interacció dels rajs de llum amb els diferents objectes de l’entorn al llarg del temps, d’aquesta manera es presenta un visor dinàmic que permet avançar i retrocedir en diferents instants per contemplar l’evolució del càlcul de color en els algorismes de traçat de rajs.

2. Estructura

L’estructura d’aquest mòdul està conformada per l’escena de l’aplicació, l’abstracció del visor (AnimadorRajs), una càmera, un seguit de rajs segmentats (enlloc de prologar-se indefinidament tenen un origen i destí acotats) i una estructura auxiliar per facilitar càlculs.

Diagrama UML 3: Diagrama de classes de l’animador de rajs

Analitzant el diagrama, l’animador de rajs s’encarrega de gestionar el comportament dels diferents rajs que generarà com també conté la informació de l’algorisme de traçat actual: tipus d’algorisme, nivell màxim de recursivitat, colors dels rajs segons criteri, patró de pintat, etc. Cada un dels rajs associats a l’animador es tradueix com un segment que viaja d’una intersecció a una altra i que adopta un color seguint un patró dictaminat per l’usuari en funció del que desitja destacar, de cara a ser estudiables el nombre de rajs està limitat a 4. Més concretament, un raig té un origen i destí, un color actual, un tipus (primari, reflectit o transmès) i un llistat de rajs derivats. Per altra banda, la càmera permet navegar en el mapat de rajs i poder veure correctament els angles que tracen
Tutor de ray tracing: Animador de raços

respecte els diferents punts d'impacte. Finalment, hi ha un arbre octal (arbre on tot node no fulla té 8 fills regulars) codificat en nodes “caixa” (reben aquest nom per simular una caixa contenedora en l'espai) que serveixen per classificar els diferents objectes de l'escena segons la regió de l'espai a la que pertanyen, agilitzant els càlculs d'intersecció raig-escena. Cada node està definit per un punt mínim de la caixa (xmin, ymin, zmin), les dimensions de la caixa i 0 o 8 nodes fills, com a principals operacions poden generar-se els fills del node actual o calcular si un raig intersecta amb aquest.

D'aquesta manera les funcionalitats de la subfinestra d'OpenGL encarregada d'anima rafos són:
- Navegació interactiva a través del ratolí i del teclat (rotació, zoom i pan).
- Navegació en el temps a través dels botons de la barra d'eines, mostrant els raços quan han executat de 0 a n interseccions amb l'entorn.
- Manipulació de l'algorisme de traçat de raços, definint l'algorisme i el nivell de recursivitat d'aquest.
- Selecció del color dels raços segons 3 criteris: tipus de raig, contribució de la llum absorbida o per identificador (un raig i tots el seus fills prenen el mateix color).

3. Disseny

3.1. Sistema d’interseccions

3.1.1. Creació del M.D.E. (model de divisió espaial)

La base de qualsevol de les variants del traçat de raços és que el color d’un píxel queda definit en base a la interacció dels raços de llum amb els elements de l’escena, aquesta no és altra cosa que el conjunt d’interseccions que té des de sortir de l’observador fins arribar a una font de llum o sortir de l’escena. De cara a calcular la intersecció d’un raig, provar si toca amb cada un dels objectes pot elevar de manera alarmant el cost computacional de l’algorisme, doncs en escenes amb un nombre elevat d’elements, no resulta factible que per cada un dels raços i dels seus raços derivats recorreguem la totalitat d’elements múltiples vegades. La solució per fer factible el càlcul és preparar una estructura prèvia que reduexi la cerca de la primera intersecció i que independentment del nombre d’elements que component l’escena, el càlcul d’interseccions es mantingui dins un rang acceptable.

L’opció escollida per a tal finalitat és un arbre octal, la base d'aquest sistema és que dividim l’espai de manera recursiva, adaptant l’estructura a com està la geometria repartida. Inicialment el node arrel de l’arbre representa una capsà envolupant de costat el diàmetre de l’escena, aquest node es veu dividit en 8 parts iguals, a partir d’aquest moment, cada una de les parts s’anirà dividint recursivament en 8 de noves fins que es compleixi una de les dues possibles condicions: el nivell de divisió és igual al màxim nivell permès o el nombre de primitives contingudes dins la part actual és inferior al mínim exigit, fet que garantitza un cost d’intersecció raig-escena aproximadament constant ja que en la majoria dels casos, enllà de visitar tots els objectes que seria com passava sense cap estructura, el que es farà serà visitar només aquells que potencialment poden ser intersectats per pertanyer a divisions per on passa el raig, on el cost d’interseccar un raig - divisió és inapreciable comparat amb el d’una intersecció raig - objecte, ja que per al primer cas es tracta de mirar si el raig talla una caixa i en el segon es veure si toca algun dels milers de triangles que poden formar un objecte.
En conclusió, un arbre octal és una estructura de dades que permet classificar la geometria en l'espai, on resulta més barat calcular interseccions amb raigs de llum i, per tant, serveix com a fitle per descartar objectes, resultant més eficient en escenes amb molts elements on el nombre de divisions de l'espai resulta ser molt petit en comparació amb el nombre d'objectes i, per tant, resulta molt més eficient fer poques interseccions amb caixes de l'arbre per acabar calculant interseccions sobre un conjunt petit de la totalitat d'objectes de l'escena.

En la figura 28 s'aprecia un exemple de les divisions superiors d'un arbre octal (s'entén que sota cada divisió en l'eix perpendicular al pla de la imatge hi hauria una més, completant els vuit fill que té un node). Com es pot apreciar, hi ha parts que ràpidament arriben al mínim de primitives (la del triangle) sense haver de subdividir-se, en canvi, per a zones més denses com la superior esquerra, la recursivitat dona lloc a més divisions (mètode adaptatiu).

Concretant de cara a la implementació, per tal de generar aquesta estructura, agafem tots els objectes de l'escena i, a través de la seva caixa englobant, calculem si interseccionen amb la subdivisió actual, lògicament, si un objecte no pertany a una subdivisió donada, tampoc pertanyerà a cap dels fills d'aquesta, per tant, passarem als 8 fills de cada node el subconjunt d'objectes que interseccionaven amb el pare i anirem repetint el procés fins arribar al nivell màxim de profunditat o a una cardinalitat d'objectes prou petita.

![Figura 28: Vista de les subdivisions superiors (4 de les 8 totals) d'una escena dividida per un arbre octal](image)

3.1.2. Càlcul de les interseccions

Muntada l'estructura, quan llencem un raig mirarem si toca un node d'aquesta, si es dóna el cas, en visitarem els fills de manera recursiva sempre que el raig els intersecti, si per contra no és així, deixarem d'avaluar interseccions. Jugant amb aquestes bases, estarem calculant interseccions entre raig-capsa que són de cost petit enfront d'interseccions entre raig-objecte, on cal mirar si el raig topa amb cada un dels milers de triangles que constitueixen la malla de l'element actual. Tota aquesta mecànica farà que al final haurem avaluat un grapat de caixes envolupants o divisions de l'espai i un subconjunt d'objectes bastant reduït. Ara bé, hi ha un factor a tenir present i és que de totes les interseccions del raig amb l'escena només volem la més propera a l'observador, gràcies a això es pot plantejar un nou afegit dins la mecànica presentada que acceleri encara més els càlculs.
Si establim una bijecció entre el món virtual i el de l’arbre octal, automatànicament podem treballar no només en un subconjunt dels 8 fills sinó que els podem visitar en ordre (en base a quan són intersectats pel raig), aturant el procés amb la primera intersecció amb una fulla de l’arbre vàlida (que intersecciona amb algun objecte de la fulla). La representació seleccionada per tal feta és definir l’octal com una matriu de dimensions $2 \times 2 \times 2$ on les dimensions es tradueixen respectivament com $x, y, z$ del món virtual (on està la geometria) dins els rangs $x/y/z$ mínima de la capsa a $x/y/z$ mínima més longitud del costat de la capsa. Tot això comporta que donat un punt en la capsa pare, podem saber a quin dels 8 fills pertany amb una transformació tan senzilla com

$$\text{fill}[(\text{punt.x - x mínima}) / \text{longitud costat}][(\text{punt.y - y mínima}) / \text{longitud costat}][(\text{punt.z - z mínima}) / \text{longitud costat}]$$

El mecanisme d’intersecció finalment trobarà el primer punt de tall amb la capsa arrel, ella traduirà el punt en un dels seus 8 fills i recursivament s’anirà traduint el punt fins trobar un fill fulla, si finalment la primera fulla no té cap objecte el següent pas és localitzar la següent divisió de l’espai que toparà amb el raig.

Donat el punt d’intersecció amb una divisió de l’espai i la direcció del raig és fàcil poder endevinar quina de les divisions germanes és la propera a observar, per fer-ho ens basem en les equacions paramètriques que representen el raig de llum i només hem de veure quina lambda és la menor tal que la coordenada actual esdevindrá igual a la frontera d’una divisió contigua. De manera més formal, la implementació aproximada seria quelcom semblant a:

```
calcularInterseccio(...)  

if(!fulla) 
{
    // Si no és una fulla, anem recorrent els fills en ordre a la cerca d’una intersecció
    while(estaDins(intersec)) 
    {
        if(fil[(intersec.x - xMin)/costat][(intersec.y - yMin)/costat][(intersec.z - zMin)/costat].calcularInterseccio(...)) 
        { 
            // Si ha donat una intersecció
            (...) 
        } 
        else // Calculem el següent fill a visitar
        { 
            lambdaX = (intersec.x - (vectorDirector.x > 0)?xMin+costat: xMin)/ vectorDirector.x; 
            (...) 
            lambda = max(abs(lambdaX), abs(lambdaY), abs(lambdaZ));
            intersec.x = intersec.x + vectorDirector.x * lambda;
            intersec.y = intersec.y + vectorDirector.y * lambda;
            (...) 
        } 
    } 
    else if(si és una fulla, avaluem interseccions amb els objectes que conté
    { 
        for(int i = 0; i < objectesFulla; i++) 
        { 
            interseccio = objectesFulla[i].calcularInterseccio(...);
            if(interseccio.lambda() < interseccioMesPropera.lambda())
            { 
                interseccioMesPropera = interseccio;
            } 
        } 
    } 
```
Amb el codi presentat es veu com el cas base de la recursivitat és quan la subdivisió de l’espai és una fulla de l’àrbre octal, en tal cas, es recorren els objectes de la fulla i es cerca la intersecció objecte-raig més propera a l’observador. Per altra banda, si la subdivisió no és un fulla, es mira recursivament els fills d’aquesta per si algun d’ells interseca, en cas de no produir-se cap intersecció, es viatja a les subdivisions germanes.

Introduït el factor d’ordre, només cal afegir un petit detall i és que visitar les subdivisions en ordre no implica que trobem les interseccions ordenades, en la figura 29 es mostra un exemple d’aquest fenomen on s’observa que quan el raig visita la divisió A, en ella hi ha contingut un objecte (triangle) que en avaluar si el raig hi impacta obtindrem resposta afirmativa.

![Diagrama de la subdivisió](image)

**Figura 29:** Exemple d’intersecció conflictiva

El problema radica en que el raig no impactarà amb l’objecte referenciat en la divisió A fins arribar a la divisió C, donant-se en aquest cas que, en la divisió B el raig topa amb un altre objecte. Tot això comporta que l’algorisme, si no es fa res al respecte, deixarà de cerca interseccions en trobar la intersecció de l’objecte referenciat en A, donant una sortida errònia.

La resolució d’aquest conflicte passa per una senzilla validació on, donada una intersecció d’un objecte referenciat per la divisió actual, només la donarem per bona si el punt d’intersecció pertany a la divisió, en cas contrari la descartarem i, només la donarem per vàlida, quan arribem a aquella divisió tal que el punt hi està contingut.

El darrer aspecte a tractar és que quan hi ha objectes que ocupen més d’una subdivisió de l’espai, aquests poden ser visitats més d’un cop per les diferents subdivisions, tal fet comporta tornar a visitar tota la malla de triangles i, un a un, mirar quins intersecta. Per tal d’evitar repetir càlculs, el truc es troba en fer que els objectes tinguin memòria i recordin la darrera intersecció que han sofert, d’aquesta manera, si es visita l’objecte en múltiples divisions, només es paga el cost de calcular la intersecció una vegada i, per la resta de vegades, es recuperen les dades de la darrera intersecció. Aprofundint en la problemàtica plantejada, no serveix recuperar la darrera intersecció a seques, cal el matís que aquesta ha de ser del mateix raig, és per aquest motiu que cal dissenyar un identificador unívoc que permeti reconèixer que la intersecció actual s’ha produït amb el raig actual i no amb un d’anterior. Partint de la base que a cada nou pintat de la imatge les dades es reinicien, un identificador unívoc és la tupla: les coordenades del pèxel del que el raig en definirà el color, nivell de recursivitat i fill dins el nivell (exemple: pèxel (0, 5), nivell de recursivitat 2, tercer fill). De manera més formal, la implementació aproximada seria quelcom semblant a:

46
A mode d'exemple, en la figura 30 es pot veure una escena on un objecte que ocupa 3 divisions de l'espai és intersectat per dos rajs de llum, el comportament que s'espera en aquest cas seria tal que: inicialment es traça el rai A, quan entra en la divisió A sol llicitar els objectes que pertanyen a aquesta divisió i en calcula la intersecció rai A - objecte, això provoca que el triangle verd passa a recordar la intersecció que té amb el rai A en el punt “p”, veient que “p” no pertany a la divisió A, viatja a la següent divisió, allà torna a calcular les interseccions, però en el cas del triangle, no la calcula ja que aquest li retorna “p”, novament el punt no pertany a la divisió i viatja fins a C, on sí que pertanyerà, havent estalviat dues vegades el càlcul de la intersecció. Posteriorment es llença el rai B, a l'arribar a la divisió A sol llicitar els objectes de la divisió i en calcula les interseccions, al sol licitar la intersecció amb el triangle, aquest no li retorna el punt “p” perquè al comparar els identificadors de rai veu que “p” era un punt associat al rai A i el rai a intersectar és un altre, conseqüentment es calcula el nou punt “q” que serà reutilitzat quan el rai B arribi a la nova divisió.
3.1.3. Càlcul de les dades associades a les interseccions

Un cop hem vist que un raig impacta en una regió amb objectes i que també ho fa amb la caixa englobant d'un d'ells, toca realitzar la part més costosa de l'algorisme, que es tracta de confirmar si efectivament intersecta amb l'objecte i l'obtenció de les característiques d'aquesta col·lisió. Per al primer punt, saber si un raig toca amb un objecte es tradueix en validar si intersecciona amb algun dels triangles de la malla que el defineixen, per tal de saber si un raig talla un triangle la metodologia passa per extreure el pla del triangle, posteriorment deduir el punt de tall entre el pla i la recta associada al raig i, finalment, esbrinar si el punt pertany al triangle.

Per extreure el pla, donats els tres vèrtex d'un triangle en ordre de pintat, el que fem és calcular la normal a través del producte vectorial entre dos vectors que formen en base als 3 vèrtex del triangle. Així doncs, si un triangle està definit pels punts p₀, p₁ i p₂, la seva normal s'expressa com: \( \mathbf{n} = (p₁ - p₀) \times (p₂ - p₀) \) on l'ordre és important de cara a mantenir l'orientació de la normal igual a la del triangle.

Definit el pla del triangle, calculem el punt de tall d'aquest amb el raig, per a fer-ho hem d'ajallar la variable lambda que expressa el nombre de cops que cal reproduir un segment de mòdul u des del punt d'origen del raig i en direcció el vector director d'aquest fins topar amb el pla: \( \text{lambda} = -\frac{(\text{origenRaig} - p₀)}{\text{vectorDirector}} \).

Arribats aquest pas, toca verificar si el punt trobat pertany al triangle, per fer-ho partim del concepte que donat un pla, el triangle és la regió que sorgeix de la intersecció de 3 semi-plans tal i com es mostra a la figura 31 esquerra, de la imatge també s'extreue que un punt està dins el triangle si, per a cada un dels tres plans troba en el semi-pla "esquerre".

![Figura 31: Esquemes sobre la indentificació de l'interior i exterior d'un triangle](attachment://image.png)

Amb aquesta informació el que hem de verificar és la banda on es troba el punt respecte la recta definida pel pla, aquesta verificació l'obtenim a través de calcular el producte vectorial entre la recta del pla i la que es forma amb un dels vèrtex del triangle i el punt a verificar, si el valor d'aquest producte vectorial per la normal del pla del triangle és major o igual a 0, significa que el gir que ha de
fer l’aresta del triangle és en sentit antihorari i, conseqüentment el punt es troba al semi-pla esquerre (figura 31 dreta). Validat que el punt d’intersecció està dins els límits del triangle, toca obtenir tota la informació d’aquest impacte, dins aquest conjunt d’informació cal entre d’altres aspectes la normal en el punt. Aquesta normal a diferència de la normal del pla neix de la ponderació de les normals definides en els tres vèrtexs del triangle, on el pes de cada una és proporcional a la distància a la que es troben del centre del triangle.

Figura 32: Esquemes sobre el càlcul de coordenades de textura amb mapat directe.

Addicionalment, també cal calcular les coordenades de textura que varien segons si són directes o per sphere mapping, en el primer cas, les coordenades de textura en un punt qualsevol del triangle sorgeixen de la suma ponderada de les coordenades de textura dels vèrtexs que formen el triangle actual, aquesta ponderació és en base a les tres àrees en les que queda dividit el triangle introduït el punt d’impacte (veure figura 32 esquerra), així doncs, donades les coordenades de textura “s” i “t” on “s” significa la coordenada associada al i-èssim vèrtex, el càlcul de coordenades quedarà com: $s_{\text{actual}} = (s_0 * a_0 + s_1 * a_1 + s_2 * a_2) / a_{\text{total}}$ i $t_{\text{actual}} = (t_0 * a_0 + t_1 * a_1 + t_2 * a_2) / a_{\text{total}}$. Ara bé, els vèrtexs del triangle que cal emprar per al càlcul no són els de la geometria original ja que quan es calculen les coordenades de textura, aquestes s’estan fent en base a un figura plasmada en el pla de visió i, en una càmera en perspectiva això implica que les distàncies entre punts no es mantenen, per comprendre millor aquest fenomen, en la figura 32 dreta hi ha un exemple on un segment dividit en dos troços iguals al projectar-se perd aquesta relació d’igualtat, i això és perquè en una càmera en perspectiva els objectes llunyans es veuen més petits que els propers. Tot això es tradueix en que les àrees empleades en la ponderació de les coordenades de textura es calculen amb el vèrtex projectats en el pla de visionat i no pas amb els originals de la geometria.

Per altra banda, per al cas del sphere mapping no hi ha coordenades de textura per vèrtex, sinó que la idea base és calcular el texel de la textura en base a simular un raig reflectit respecte el punt de vista de l’observador; més concretament el mapat esfèric és una tècnica on la imatge de la textura equival a l’hemisferi d’una esfera reflectora perfecta en una vista ortonormal (els elements mantenien la mida independentment de la distància a la que es troben de l’observador), aquest concepte s’aprecia en la figura 33 on en la part esquerra es veu la relació entre l’hemisferi d’una esfera i el mapa de bits que vindria a ser la textura, per altra banda, a la dreta de la figura hi ha un esquema de com els raigs
incidents es reflexen en l’esfera i retornen al pla de visió, donant-se la relació entre raig incident i coordenada de textura.

Figura 33: Esquemes sobre el càlcul de coordenades de textura amb mapat esfèric

Assentades les bases, el primer que cal fer és calcular el vector reflectit del raig respecte la normal ponderada del triangle on intersecta, concretament, si \( \vec{r} \) és el vector reflectit, un cop calculat s’obté el valor intermig “\( m \)” tal que \( m = 2 \cdot \arretool{r.x^2 + r.y^2 + (r.z + 1)^2} \) i amb ell se’n deriven les coordenades de textura on \( s = r.x/m + 0.5 \) i \( t = r.y/m + 0.5 \). Per a tots dos casos el resultat és un punt expressat en coordenades de textura que cal passar a coordenades de la imatge, multiplicant la coordenada per l’amplada i aïllada de la imatge respectivament i aplicant-ne mòdul amb aquestes \( \text{pixely} = (\text{txels} \cdot \text{amplada})/\text{amplada} \) \( \text{pixely} = (\text{txel.t} \cdot \text{alçada})/\text{alçada} \) ja que per definició s’aplica el criteri de repetició de textura, que significa que davant una coordenada de textura per sobre del 1.0, es torna a aplicar la textura des del principi de manera cíclica.

Amb tot el comentat només queda un matís dins l’àmbit de la intersecció més propera i és que quan intersectem amb un punt d’un triangle poden passar diferents coses: una d’elles és que el punt d’intersecció sigui el propi punt d’origen, per evitar aquest cas, només cal afegir un cert “èpsilon” (marge d’error) en la direcció del raig al punt. Un altre fenomen que es pot produir és que si tenim dos objectes que estan tocant-se o a una distància “èpsilon”, a l’hora de fer interseccions podria passar que primer avaluessim la intersecció amb l’interior de l’objecte actual, i en iniciar el següent raig a distància “èpsilon” del punt d’impacte, ens trobèssim que el nou punt d’origen ja està dins de l’objecte del costat, creient que estem a l’exterior quan realment no és així. Per solucionar aquest segon conflicte, només cal fer que quan trobem un punt d’intersecció mirem si es troba a menys distància que la intersecció a menor distància trobada fins al moment menys èpsilon, si és així, es substituirà la intersecció més propera per la intersecció actual, sinó es mirarà si la nova intersecció es troba dins un rang de +/- “èpsilon” respecte la més propera, si es dóna tal cas, es substituirà la intersecció més propera per l’actual si aquesta és una cara frontface, és a dir, una cara exterior i, per tant, estarem dient que si tenim dos objectes molt propers agafarem com a intersecció la que va del punt interior del primer objecte al punt exterior del segon i no a un altre punt interior del primer altre cop.

50
Per entendre millor el desenvolupament de la problemàtica plantejada sobre objectes propers, en la figura 34 esquerra es planteja un exemple on l’objecte A i B es troben que per un dels seus costats estan a distància menor d’èpsilon. Suposem que els dos objectes es troben dins una mateixa subdivisió de l’espai, d’aquesta manera, quan el raig A vol saber els objectes amb els que intersecta, rebrà com a resposta aquest dos objectes presentats, el problema és que no es pot saber si vindran en ordre A,B o B,A. Si partim que es dóna el primer cas, al rebre A sorgirà el punt “p” i la distància respecte l’origen serà de “d” unitats, posteriorment s’avaluarà l’objecte B, d’allà sorgirà el punt d’intersecció “q”, però en trobar-se una diferència de distàncies tan petita serà descartat per no ser una intersecció més propera, amb tot això l’execució haurà produït un resultat matemàtic correcte.

![Figura 34: Exemple d’objectes molt propers](image)

Ara partim que es dóna el segon cas, al rebre B, acabarem amb el punt “q”, després ens trobarem amb el punt “p”, però com la diferència de distàncies entre “p” i “q” és menor que èpsilon es descartarà, produint-se un resultat erroni. Si per contra s’optés per no fer un filtrat de distància èpsilon, donat el punt “p”, al avançar “èpsilon” distància per no topar amb “p” de nou, iniciariem el següent raig al punt “r”, creient que hauríem sortit de l’objecte A i que, per tant, estaríem a l’exterior; quan realment ens trobriem dins de l’objecte B i caldria aplicar les propietats que això comporta. Per descobrir que “q” és un punt d’un objecte diferent i que “p” és de l’objecte actual, només cal mirar la normal del triangle on s’intersecta, com es pot apreciar a la figura 34 dreta, la normal de sortida d’un objecte es troba en el mateix sentit al del raig, mentre que la d’entrada es troba en sentit invers (back i frontface respectivament), amb tot això, el que es fa és que si a distància menor d’èpsilon passem a entrar a un nou objecte, ens quedarem sempre amb aquell punt, en aquest cas el punt “q”, d’aquesta manera evitem que creïem estar a l’exterior i només estem modificant que els objectes que estan a distància menor que èpsilon s’interpreten com a junts completament.

### 3.2. Traçat de raigs

L’animador treballa amb diferents algorismes de traçat de raigs (ray tracing clàssic i ray tracing distribuït), encara que cada un té les seues peculiaritats, al final tots es basen en uns mateixos principis, que són llençar raigs fins que interseccionen amb algun element i fet això llençan-ne de nous en base certs criteris relacionats amb les propietats òptiques del punt d’intersecció.
3.3. Visualització de rajos

El punt fort de l’animador és veure l’evolució dels rajos de llum, per tal de poder-ho fer, de la mateixa manera que les fonts eren representades per esfèeres, els rajos ho seran per línies. Assentada aquesta base, el següent punt a tratar és com animar el pintat al llarg del temps, per tal de fer-ho es presenten diferents interaccions que són accessibles a la barra d’eines de l’animador (figura 35): tornar a l’inici de l’animació, anar un pas endarrere, anar un pas endavant i arribar fins al final de l’animació.

**Figura 35:** Secció de la barra d’eines de l’animador de rajos

El sistema d’animació s’estableix per nivells de profunditat, així doncs, un raig només es dibuixarà si tots els rajos de profunditat menor han estat dibuixats, és a dir, l’acció de fer un pas endavant dibuixa per a tot raig el seu següent fill, així si es té present l’abstracció d’arbre de rajos (veure figura 10), els pas endavant i pas endarrere dibuixen l’arbre fins a el nivell \( i + 1 \) o \( i - 1 \) de profunditat. Anàlogament, les accions d’inici d’animació i final dibuixen fins al nivell 0 o fins l’últim nivell respectivament.

**Figura 36:** Seqüència d’animació de rajos
3.4. Càlcut del color

Per a un estudi més elaborat del comportament dels raigs s'ha optat per proposar tres models de acolorat que permetin apreciar factors diferents: color per tipus, color per raig i color per contribució (veure figura 37).

![Figura 37: Escena amb raigs pintats segons el tipus (esquerra), un per raig (centre) o per contribució (dreta)]

El color per tipus fa referència a que el color del raig bé donat en funció del tipus d'aquest, depenent de si és primari, reflectit o transmès. En la figura 37-esquerra els raigs primaris estan pintats de color groc, els reflectits en verd i els transmesos en vermell, podent-se apreciar com de l'impacte dels primaris en sorgeixen els altres segons l'objecte on topen.

Per altra banda, el color per raig com el seu nom indica significa que donat un raig primari, ell i tots els seus fills mantindran el mateix color, poden estudiar el progrés d'un raig en concret al llarg del temps. En la figura 37-centre queda palès com hi ha quatre raigs: vermell, blau, rosa i verd que a diferència de l'acolorat per tipus, quan impacten amb objectes mantenen el seu color en els fills.

Finalment, el color per contribució és el color del raig en base a la seva interacció amb l'escena. En la figura 37-dreta es pot observar com els punts que surten de l'escena tenen el color del fons, mentre aquells que impacten amb objectes prenen el color de l'objecte amb una certa desviació originada pel color acumulat dels seus fills.

En els dos primers models d'acolorat el color ve per defecte en forma de llegenda que l'usuari pot modificar per tal d'adaptar les tonalitats a aquelles que resultin fàcils de diferenciar respecte l'entorn. En canvi, per al tercer mètode el color no pot ser canviat per l'usuari ja que ve pels propis càlculs dels algorismes pertinents com es comenta a continuació:

En els algorismes de traçat de raigs de l'aplicació, el color en un punt es descriu en base a dos contribucions, la directa i la indirecta. La contribució directa en un punt s'expressa com la suma del terme ambient, difós i especular. El primer terme és independent de la posició dels focus de llum i de l'observador i surt de la suma de les intensitats ambient de totes les fonts de llum (Iw) per la constant d'emissivitat ambient del material del punt actual (Ka):

\[
\text{Color ambient} = \sum (I_{wi} \cdot K_0) \text{ on } i : [0, \text{llums}-1]
\]
Els altres dos termes (difòs i especular) només s’originen per a una llum donada si el punt és il·luminat per aquesta cosa que es donarà sempre que no hi hagi cap element entre mig que el deixi a l’ombra. Així doncs, per a cada font de llum calculem la distància a la qual està del punt, després tracem un raig des del punt intersectat cap a la font de llum concreta i mirem si aquest impacta amb algun element, en cas de fer-ho comparem la distància a la que es dóna aquest impacte respecte la distància a la que està el punt del focus de llum, si aquesta distància és menor, significa que hi ha un oclusor i que per tant, el punt es troba a l’ombra del focus actual, en conseqüència no afegim contribució difosa ni especular. Per al cas en que la font de llum és un focus direccional, cal validar addicionalment si la recta traçada des del punt d’intersecció es troba dins el con d’il·luminació, sigui \( L \) el vector director de la recta de l’objecte a la font de llum i \( F \) el vector director cap on apunta el focus, amb arccos \( \langle -L,F \rangle/\|L\| \|F\| \) obtenim l’angle que descriuen aquest dos vectors i que hem de comparar amb el d’obertura del focus de manera que si l’angle obtingut és positiu i com a molt igual al del focus, el punt d’intersecció està il·luminat per la font de llum, altrament està a l’ombra (figura 38).

\[ \text{Figura 38: Esquema del càlcul d’il·luminació amb un focus de llum} \]

Quan un punt està il·luminat, la contribució difosa es calcula a través de la fórmula de Lambert on multipliquem la intensitat difosa de la llum \( l_b \) per la constant difosa del material \( K_d \) associat al punt i pel cosinus de l’angle entre la normal en la intersecció \( N \) i el vector incident des de la font de llum \( L \):

\[
\text{Color difosa} = \sum (l_b \ast K_d \ast \cos(N \ast L)) \text{ on } i : [0, llums-1]
\]

Lògicament, això només es donarà quan la cara sigui front face, que equivalent als casos en que el cosinus és positiu. A part, tenim el cas específic d’estar usant texturines, quan això succeeix el càlcul del color difós és el mateix, però enlloc d’utilitzar les característiques difoses del material, s’utilitza els colors del texel pertinent. Anàlogament per calcular la contribució especular multiplicarem la intensitat especular de la llum \( l_s \) i la constant especular del material \( K_t \) pel cosinus de l’angle entre el vector reflectit en el punt d’intersecció \( R \) i el vector que va fins a l’observador \( V \) elevat a la constant de Phong del material \( a \):

\[
\text{Color especular} = \sum (l_s \ast K_s \ast \cos(R \ast V))^a \text{ on } i : [0, llums-1].
\]

Respecte a l’altre tipus de contribució, la indirecta, aquesta sorgeix de la interacció lumínica amb la resta d’objectes de l’entorn, així doncs, el color vindrà com a suma de les aportacions dels raigs que sorgeixin del punt d’intersecció (reflexats i transmessenos), variant les característiques d’aquest i el seu
grau d’aportació segons l’algorisme de traçat empleat i les característiques del material en el punt d’impacte.

3.5. Algorismes de traçat

Fins al moment s’ha comentat els punts en comú de tot traçat de rajos, a continuació es pretén citar els diferents algorismes que l’aplicació deixarà a l’abast de l’usuari, comentant-ne les característiques clau i com s’han desenvolupat en termes d’implementació.

3.5.1. Ray tracing

Ray tracing és l’algorisme clàssic, la seva base radica en que cada raig es divideix en reflectit i transmès segons la llei de Snell de manera recursiva fins arribar a una profunditat màxima o a una contribució al color final no significativa.

Tenint aquest concepte en ment, en referència a la implementació cada cop que es generi una intersecció calcularem el raig reflectit en base a la llei de la reflexió, on donada la normal “N” en el punt d’intersecció i vector incident “I”, el raig reflectit “R” s’expressa com (figura 39 esquerra):

\[ R = (-2*NI)*N + I \]

Per altra banda, calcularem el raig transmès a través de descompondre’l com la suma de dos rajos T1 i T2 tals que (figura 39 dreta):

\[ T1 = -\cos(\beta)*N \]

\[ T2 = (\cos(\alpha)*N + I)*\mu \]

Figura 39: Esquemes sobre el càlcul del raig reflectit i transmès

Cal matitzar que \( \mu \) és el coeficient de refracció i que s’expressa com la relació entre els índex de refracció dels dos medis que participen en el fenomen (veure figura 40), per altra banda, el \( \cos(\beta) \) empletat en el càlcul de T1 surt com:

\[ \cos(\beta) = \arrel(1-\mu^2(1-\cos(\alpha)^2)) \]
on el nombre dins l’arrel pot prendre valors negatius quan es dona un fenomen de reflecció total, en tal cas, el raig transmès deixa d’avaluar-se en base a la llei de Snell i es calcula a través de la llei de la reflexió. Amb tot això, l’esquelet de l’algoritme prendrà la següent forma:

<table>
<thead>
<tr>
<th>Material</th>
<th>Índex de refracció</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vuit</td>
<td>1</td>
</tr>
<tr>
<td>Aire</td>
<td>1.00029</td>
</tr>
<tr>
<td>Aigua</td>
<td>1.333</td>
</tr>
<tr>
<td>Vidre flint</td>
<td>1.57 - 1.65</td>
</tr>
<tr>
<td>Diamant</td>
<td>2.417</td>
</tr>
</tbody>
</table>

Figura 40: Taula amb l’índex de refracció d’algunes substàncies

```c
void traçaRaig (...) {
    //Diferenciam segons si impacta o no el raig
    if (arbreOctal->calcularInterseccio(raig)) {
        color = calcularContribucioDirecta();
        //Seguim traçant raigs si encara no s’ha arribat al nivell de recursivitat màxim
        if (nivellActual < nivellMaxim) {
            if (interseccioActual.obtenirKr() > llindar) {
                //El material refleixa la llum
                raig = reflexat = (-2*N*y)*N+1;
                origenRaig = i_interseccioActual + margeError* reflexat.direccio();
                color += interseccioActual.obtenirKr() * reflexat.traçaRaig();
            }
            else (interseccioActual.obtenirKt() > llindar) {
                //El material transmet la llum
                cosA = N*y;
                cosB = 1 - cosRef* (1-cosA);
                if (cosB < 0) raig transmès = (-2*N*y)*N+1;
                else raig transmès = T1 + T2;
                color += interseccioActual.obtenirKt() * transmès.traçaRaig();
            }
        }
    }
    else //No intersecta amb res
        return colorFons;
}
```

Amb aquests criteris, una possible imatge que es podria veure en l’animador de raigs seria la de la figura 41, en ella hi ha una primera bola transmissora i una segona bola que reflexa la llum, d’aquesta
manera, el raig primari (color groc), impacta amb la primera bola, d’aquí es transmet llum (color vermell) que surt per l’altra de la bola fins arribar a la segona bola, allà el raig incident veu reflectit (verd).

![Imatge](image.png)

*Figura 41*: Exemple d’animació de raigs usant l’algorisme clàssic

3.5.2. Distributed ray tracing

Distributed ray tracing és una variant de l’algorisme clàssic que utilitza tècniques estocàstiques (regides per l’atzar) per tal de suavitzar els fenomenes de la llum en la generació d’imatges més realistes. En concret, mentre el ray tracing clàssic llança un raig per fer mostrejos dels fenòmens, (un raig per saber si un objecte està a l’ombra, un raig per saber la contribució reflectida, un per saber la transmessa), el distributed en llança “n”, els quals respecte el que llançava l’algorisme clàssic es formen com una desviació de la direcció base de manera aleatoria, tot això comporta que els fenomenes de la llum queden suavitzats al poder donar resultats parciais, per exemple, mentre en l’algorisme clàssic un punt només podia estar o no a l’ombra d’un focus de llum, en aquesta versió un punt pot estar parcialment a l’ombra a través de tenir que uns quants dels “n” raigs hagin donat que estaven il·luminats i uns altres no.

La base de l’aleatorietat en la reflexió parteix del raig teòric reflectit i genera raigs dins un rang respecte el raig original, en el cas de la implementació plantejada, donat el raig reflectit “r”, es generen “n” raigs que s’expressen com dos girs en els dos eixos perpendiculares de “r” sobre “r” (*veure figura 42*), d’aquesta manera s’aconsegueix un suavitzat que simula una superfície polida.

![Diagrama](diagram.png)

*Figura 42*: Esquema de vibració respecte el raig reflectit original
Anàleg al càlcul de la reflexió, per al raig transmès també es generen rajos aleatoris com desviacions del teòric bàsic, com també per als raigs d'ombra, a part, per qualsevol dels càlculs d'aquests fenòmens, a cada un dels "n" rajos se li assigna un pes de manera que la contribució final és una suma ponderada dels respectius "n" rajos, això provoca fronteres més suavitzades donant llocs a efectes com la penombra en el cas del càlcul d'ombres. Amb tot això, l'esquelet de l'algorisme prendrà la següent forma:

```c
void traçarRaig (...) 
if (interseccióActual.obtenirKr() > llinar) 
{ 
  // El material reflecta la llum
  raig derivat = reflexat.vibrat();
  origenRaig = i interseccióActual + margeError* derivat.direccio();
  color += interseccióActual.obtenirKr()/N * reflexat.traçarRaig();
} 
else (interseccióActual.obtenirKt() > llinar)
{ 
  // El material transmet la llum
  cosA = N*i;
  cosB = 1 - coefRef* (1-cosA*);
  (...) 
  for(int i = 0; i < N; i++)
  { 
    raig derivat = transmes.vibrat();
    origenRaig = i interseccióActual + margeError* derivat.direccio();
    color += interseccióActual.obtenirKt()/N * derivat.traçarRaig();
  } 
}
```

Amb aquests criteris, una possible imatge que es podria veure en l'animador de rajos seria la de la figura 43, anàloga a la mostrada anteriorment en la figura 41, es pot apreciar una primera esfera que és transmissora i una segona que és reflectora, d'aquesta manera, el raig primari (color groc), impacta amb la superfície de la primera bola, d'aquí es transmeten n-rajos de llum (color vermell) que surten per l'altra banda de l'esfera, on cada un, al tocar l'altra banda de l'esfera, deriva en "n" nous rajos fins arribar a la segona esfera, allà es tots ells generen "n" nous rajos reflectits (verd).

**Figura 43:** Exemple d'animació de rajos usant l'algorisme distribuït
Per a comprendre millor les diferències resultants d’aplicar un o l’altre algorisme en la figura 44 es pot apreciar com en el traçat clàssic els reflexos són perfectes del tot, no hi ha cap irregularitat i dóna la impressió d’una superfície pul lida de manera molt precisa, per contra, en el cas del traçat de raigs distribuït es veu una imatge que possiblement resulti més propera a la realitat, doncs els reflexos, les ombres i la il·luminació en general es veu afectada per un soroll, fent que les línies no siguin perfectes sinó que hi hagi lloc per a certa irregularitat, donant lloc per exemple a que el reflexe de les boles en la superfície del marbre quedï com difuminat. Lògicament, el mostreig utilitzat és només de 4 raigs, amb nombres més grans el soroll satenuaria i aleshores es veuria de manera més clara efectes com la penombra.

**Figura 44**: Imatge generada amb traçat clàssic (esquerra) i imatge amb traçat distribuït (dreta)

4. Ampliacions futures

Les ampliacions que podrien implementar-se en un futur en l’animador haurien de tenir com a principal prioritat enriquir la informació sobre els raigs, així doncs, seria interessant permetre la selecció dels raigs que es dibuixen i amb ella, la publicació de les dades associades, podent obtenir un control més acurat del que està passant.

Per altra banda, un altre aspecte que es podria afegir seria la visualització de les interseccions dels raigs amb les subdivisions de l’espai, així doncs, es podrien veure les subdivisió de color transparent i anar-se il·luminant quan fossin accedides o fer algun altre feedback que permetria veure com es descarten parts de la geometria gràcies a fer pre-tractament de la informació. Ja portant la idea al punt màxim, es podria crear un visor en dos dimensions que mostres la metàfora de l’arbre de raigs amb les dades de cada raig accessible des d’allà.

Un darrer aspecte interessant seria el de poder desar les animacions de raigs en format de vídeo, d’aquesta manera es podrien tenir gravacions de les interaccions més interessants, desvinculant-se la visualització dels fenòmens a l’ús concret d’aquesta aplicació.
1. Introducció

El visor de ray tracing és el mòdul colofó, en ell es presenta la imatge final renderitzada amb algorismes de traçat de rajos sobre l’escena matiscada en el visor d’OpenGL i en base a una càmera donada.

2. Estructura

L’estructura d’aquest mòdul està conformada pels mateixos elements que l’animador de Rajos, ja que en el fons aquest mòdul és una versió que pretén mostrar la mateixa informació però de diferent manera.

Diagrama UML 4: Diagrama de classes dels visors de traçat de rajos

Observant el diagrama, es repeteixen part de les classes sorgides en l’animador de rajos, així doncs, per una banda hi ha una nova classe raig que respecte el “raig segmentat” de l’animador de rajos no es veu limitat a 4, ja que aquest cop es traçen tants rajos com pixels ha de tenir la imatge final, a més a més, aquest rajos són més senzills que els “segmentats”, ja que no desitgen tenir-los en memòria doncs només interessa el color final, eliminant dades addicional de visualització i animació d’aquests.

En oposició, l’arbre octal, comels algorismes d’intersecció i algun que altre element que s’utilitzi per calcular la contribució de color en un punt són els mateixos en el visor de ray tracing i en l’animador de rajos, ja que la base que es desitja usar és la mateixa.
3. Disseny

3.1. Sistema de pintat

El sistema de pintat no afegeix cap aspecte que no hagués estat presentat en l'animador de raigs, el concepte base és recuperar les dimensions del visor i per cada un dels pixels llançar un raig que es dividirá recursivament en base a l'algorisme de traçat actual, d'aquesta manera l'estructura aproximada seria com:

```c
// Viatgem per tots els pixels de l'escena
for(int i=0; i < altura; i++)
{
    for(int j=0; j < amplada; j++)
    {
        cameraVisorRayTracing->obtenirPunt(puntActual[0], puntActual[1], puntActual[2], i, j);

        direcRaig[0] = puntActual[0] - observador[0];

        // Tracem el raig i assignem el color a la imatge final
        raigActual = new Raig(...);

        posicio = (i*amplada)+(j*3);
        imatgeRayTracing[posicio] = raigActual->obtenirColorContribucio(VERMELL);
        imatgeRayTracing[posicio+1] = raigActual->obtenirColorContribucio(VERD);
        imatgeRayTracing[posicio+2] = raigActual->obtenirColorContribucio(BLAU);
    }
}

// Dibuixem la imatge calculada
glDrawPixels(amplada, altura, GL_RGB, GL_FLOAT, (const GLvoid*) imatgeRayTracing);
```

El problema de l'algorisme presentat és que si s'obtenen els punts com en el visor de raigs, és a dir, a través de desprojectar amb les matrius inverses, el sistema resulta bastant ineficient. De cara a accelerar els càlculs una possible optimització resulta d'obtenir la desprojecció dels 4 extrems de la finestra i per interpolació lineal d'aquests, extreure el valor concret de cada punt.

Aprofundint més en aquesta millora, per fer que tot vagi el més ràpid possible, el millor sistema passa per no interolar entre 4 punts cada cop, sinó entre els dos que determinen una línia de la finestra, és a dir, quan s'inicia el pintat d'una nova línia de la finestra s'interpol a el punt d'inici i el punt final d'aquesta, a partir d'aquí, cada punt dins la línia s'expressa com: \((1-x)^*punt\text{\textunderscore initial} + x^*punt\text{\textunderscore final}\). D'aquesta manera, la major part del temps s'està emprant un parell de multiplicacions enlloc d'arrels per calcular les distàncies en el pla respecte als 4 extrems.

Finalment, els costos del pintat no només fan que no sigui raonable fer-lo de manera freqüent, sinó que també dificulten que aquest sigui de manera síncrona, per tal d'evitar que l'usuari vegui bloquejada la interacció que posseeix amb l'aplicació quan aquesta es troba calculant el pintat en traçat de raigs, el pintat es produeix de manera asíncrona (no és fa al mateix temps, sinó a trocets i al cap d'un temps
es motra el resultat final). En termes d’implementació el concepte passa per tenir un temporitzador que cada “t” unitats de temps, si escau, pinta un nombre de raïjos de la imatge final, lògicament, aquest nombre és molt menor respecte pintar-ne el total, de manera que al final el procés resulta transparent a l’usuari.

Aquest aspecte comporta que l’usuari podria apreciar imatges a mig pintar (figura 45), doncs al no pintar de cop a causa del sistema asíncron, l’estructura on es desen les dades podria no estar completament actualitzada.

![Zona Pintant-se actualment](image)

**Figura 45:** Exemple d’un pintat usant un sol buffer

Per tal d’evitar aquest nou problema, només cal imitar el sistema clàssic de dos “buffers”, així doncs, la idea és que quan es sol·licita repintar el visor, per pantalla es mostra una imatge obsoleta d’una estructura que estigui del tot escrita i de mentre tenim una segona estructura pintant-se. Quan s’han càlculat i escrit els valors de tots els pixels en l’estructura auxiliar, aquesta passa a ser l’estructura primària (la que s’utilitza per repintar el visor) i la primària passar a ser l’auxiliar (on s’escriuràn les dades dels nous raïjos traçats).

### 3.2. Sistema de refresc

En un altre ordre de coses, el re-pintat complet de l’escena resulta costós, per tant no es pot fer tan sovint com a la resta de visors, així es planején 3 repaints: per sincronització de càmera, per refresc per part de l’usuari i per re-escalat de la finestra.

Per al primer cas, el de sincronització de la càmera, el concepte és que quan l’usuari que navega en el visor d’OpenGL troba la vista que vol renderitzar, sol·licita la sincronització de la càmera, fent que el visor de raïjos i el de traçat de raïjos passin a veure’s de la mateixa manera, produint-se un pintat del visor.

Per al segon cas, el de refresc per part de l’usuari, la idea és que fixat un punt de vista es poden canviar els materials, les fonts de llum i altres elements que provoquen que el resultat de la síntesi d’imatges amb el traçat de raïjos variï. A nivell d’usuari el comportament més normal és fer tot un
seguit de canvis i despàs visualitzar el resultat, si per cada un d'ells es repintés tot, l'aplicació patiria re lentitzacions constants, per tal d'evitar això i tenint present el comportament normal d'un usuari, el pintat es planteja com un procés que s'originà quan l'usuari sol llicita un refresc, acció que tendirà a produir-se despàs de realitzar un seguit de canvis en les propietats de l'escena.

El darrer fenomen que pot originar un repintat és el re-escalat de l'aplicació (canviar les dimensions de la finestra) ja que provocarà el canvi de les dimensions del visor; ara bé, en termes d'usabilitat, quan un usuari re-escala l'aplicació tendeix a estar variant la dimensió un temps prou gran com per ser apreciat a escala humana, això implica que a nivell de l'aplicació hi hauria una seqüència de constants re-escalats des del moment en que comença a canviar les dimensions fins que deixar anar el cursor. Davant aquest panorama torna a no ser raonable repintar cada cop que el sistema informa d'un canvi en les dimensions de l'aplicació, com a solució es planteja introduir un temporitzador que cada "x" segons miri si s'havia fet un re-escalat, d'aquesta manera no s'anirà repintant constantment durant el re-escalat sinó només en aquestes interrupcions cada "x" segons, així en el pitjor dels casos trobarem un desfasament de pocs segons entre que finalitzem un canvi de dimensions i el traçat de rajos es mostra correctament, però a canvi el sistema funcionarà de manera més flúida.

Apart de la gestió del re-pintat, per tal de reduir encara més els efectes d'un re-escalat, la idea mostrada en el pintat de dues estructures s'acaba d'acotar amb l'ús d'una textura, és a dir, un cop s'ha generat una imatge aquesta es desa en una textura (estructura primària) i es pinta en un polígon per pantalla, d'aquesta manera, davant nous canvis, es continua escriuant l'estructura interna, mentre la textura segueix mostrant la darrera imatge completament generada. Els avantatges d'aquest mètode és que al variar les dimensions de la finestra, el polígon s'adapta i amb ell la textura mapada damunt, fent que la imatge es re-escalà, lògicament això pot comportar pèrdua de qualitat, però quan s'hagi acabat de recalcuar la imatge amb les noves dimensions aquestes incorreccions seran corregides.

En qualsevol cas, tenir refrescant imatges de manera passiva en determinats sistemes pot suposar realentitzacions notables en la interacció, per aquest motiu, s'ha afegit addicionalment l'opció de bloquejar el sistema de traçat de rajos, permetent que, mentre l'usuari no estigui interessat en veure el resultat final, la interacció amb la resta de mòduls no es vegi penalitzada.

3.3. Gestió d'imatges

L'objectiu final del mòdul de traçat de rajos és comparar imatges generades amb característiques diferents, per tal de dur-lo a terme, una funcionalitat indispensable és el fet de poder desar qualsevol imatge generada en memòria, poden a posteriori i amb l'ajuda de programes de tractament i síntesi d'imatges la comparació entre els diferents resultats al llarg d'una sessió de treball.

Per aquest propòsit Qt aporta ja un suport per poder desar la informació d'OpenGL, així que la implementació passa per extreure la informació del visor i plasmar-la en una entitat de Qt que finalment desarà en memòria i en format de "jpeg" el resultat.
4. Ampliacions futures

Com ampliacions interessants per aquest darrer mòdul, primer es podrien plantejar nous sistemes per dotar de millor rendiment al traçat de raigs per tal de reduir els costos de computació actuals. Per altra banda, es podria enriquir les variants d'algorismes existents permeten que qualsevol sistema de traçat de raigs actual pugui ser observat amb el programari plantejat.

De cara a donar un valor afegit a la informació, podria resultar interessant poder activar/desactivar les estructures auxiliars que optimitzen el càlcul d'interseccions (per exemple els arbres octals) com la variació d'aquestes, obtenint també els temps de càlcul per poder valorar també l'aspecte computacional i no només el qualitatiu que queda expressat en la imatge renderitzada al final.

En un altre ordre de coses, l'actual sistema de pintat podria veure's millorat amb un sistema de qualitats graduals, és a dir, idear un pintat menys costós a base de sacrificar la qualitat final, així doncs, en un temps reduït l'usuari podria veure una imatge aproximada del resultat final sense haver d'esperar el temps de càlcul de la veritable solució. Això comportaria accelerar molts tràmits per part de l'usuari final ja que podria replantejar-se modificar aspectes de la configuració de l'escena amb la "vista prèvia" de menor qualitat sense haver d'esperar el renderitzat final.

Com a colofó, un darrer grup d'ampliacions es podrien focalitzar en la gestió de les imatges generades, més concretament, augmentant el ventall de formats en el que es puguin desar, com enriquint l'aplicació amb característiques que facilitessin la visualització de múltiples imatges i comparació entre elles.
1. Introducció documental

L'apartat següent és una anàlisi de la desviació sofera respecte la planificació inicial, avaluant les traves més representatives durant el desenvolupament del projecte i els canvis realitzats en base a les tendències observades.

2. Anàlisi

En la planificació inicial es platejava una desenvolupament de 6 mesos estructurat en una fase de documentació tecnològica i preparatius pre-projecte, una etapa destinada per a cada un dels mòduls a especificar-ne les característiques i implementar-les i un interval final per detalls com manual d'usuari o test de cohesió.

La fase inicial de documentació ha seguit el calendari establert, per contra, en les etapes posteriors han sorgit desviacions significatives per diferents causes. Un dels punts de canvi respecte la planificació inicial ha estat la data de començament de la fase d'implementació, ja que davant unes necessitats difícils d'acotar sobre que havia de fer el mòdul, arribats al disseny del primer d'ells, es va canviar la filosofia d'acció plantejada, passant d'un sistema a dos fases on primer calia fer una especificació total del sistema i després implementar tot allò pactat, per una nova metodologia on una primera etapa d'especificació marçava els aspectes més importants i després s'implementava un prototipus per comprendre millor les necessitats del mòdul, a partir d'aquesta implementació es rebia un feedback que ajudava en la comprensió del mòdul, acabant de delimitar les necessitats en una segona especificació i plasmant-ho tot plegat en una darrera fase d'implementació.

La fórmula de prototipatge usada en el primer mòdul va donar bons resultats, executant-se per a la resta de mòduls amb resultats bastant satisfactoris, reduint notablement els temps per a l'especificació. En oposició a aquesta reducció, la implementació es va veure allargada, però per una infravaloració dels canvis entre versions de Qt, més que pel sistema de prototipatge on l'especificació era més débil.

Aprofundint amb la problemàtica sorgida amb Qt, un dels principals problemes ha estat l'ús de múltiples widgets d'OpenGL, fet que ha provocat l'aparició de canvis de context gràfic amb resultats inesperats en l'execució, tot això ha fet que les solucions inicialment plantejades haguessin de modificar-se per sistemes més complexes, així per exemple es va haver d'afegir tot un seguit de tràmits per recuperar el context com per evitar que els threads de cada visor poguessin contaminar-se entre ells. Les peculiaritats de treballar usant OpenGL amb Qt van fer-se realment patents en la gestió de textures, on cada widget reservava textures de manera que fossin visibles només per qui les hagués creat, plantejant un nou disseny de la gestió d'aquestes on es reservaven múltiples identificadors per tenir copies per a cada visor. En definitiva, a l'iniciar el projecte es va seleccionar Qt perquè era una tecnologia coneguda, però la darrera versió d'aquesta llibreria va ocultar més canvis dels esperats.
Un altre aspecte que ha obligat a rejustar el desenvolupament ha estat els horaris calculats, no sempre ha estat possible mantenir la premissa inicial d'una dedicació de 4 hores diàries i s'ha compensat aprofitant els marges que s'havien deixat per al darrer mes de desenvolupament, com utilitzant alguna que altra hora de dies no laborals.

De cara a la generació de documentació no hi ha hagut cap incidència, s'ha anat desenvolupant paral·lelament a les tasques principals del projecte, documentant-se de manera ràpida i àgil gràcies a redactar en tot moment informació sobre les tasques contemporanies, evitant tràmits carregosos de recuperació de feines anteriors.

3. Conclusions

Com a conclusions finals es pot extreure que de cara a propers desenvolupaments, la idea del prototipatge és un concepte a tenir present com a possible metodologia per a la creació de software, apart, cal posar més esforç en les etapes d'especificació, ja que en determinats punts s'ha observat el fet d'haver de refer coses per no haver fet un estudi previ prou acurat.

En termes de tecnologies, cal tenir present que no dominar completament una tecnologia s'hauria de contemplar amb uns marges d'error més generosos ja que després sorgeixen imprevistos per una infravaloració del procés d'estudi tecnològic. Seguint en el marc de les estimacions, un dels elements de feedback d'aquest projecte seria l'aprenentatge sobre rangs per acotar hores de feina de les diferents tasques, de manera que en posteriors projectes, l'experiència adquirida servirà per acurar més les estimacions temporals.

Finalment, el seguiment d'un projecte en base a estimacions temporals en forma de diagrama Gantt o similar es mostra com sistema molt útil gràcies a permetre comprendre el nivell de desviació present en tot moment, de manera que resulta factible rectificar amb temps i reduir els costos finals de qualsevol projecte plantejat, a part permet comprendre i tenir present els punts que queden, evitant eludir certes fites com ajudant a mantenir un ordre cronològic coherent que optimitzi el desenvolupament.

La ciència es compon d'errors, que a la vegada, son passos cap a la veritat
Jules Verne (1828-1905) - escriptor francès
Bibliografia

1. Pàgines web

A continuació es presenten el conjunt de pàgines web que s'ha utilitzat per documentar i realitzar el projecte.

http://es.wikipedia.org
http://www.howstuffworks.com
http://www.edndoc.esri.com
http://www.google.es
http://www.cppreference.com
http://www.sc.ehu.es/sbweb/fisica/ondas/snell/snell.htm

2. Llibres

Tot seguit es mostra el llistat de llibres sense els que no hauria estat possible la realització del projecte.

OpenGL Programming Guide, Sixth Edition - Shreiner, Woo, Neider, Davis - Addison Wesley
**Glossari**

**Ambient**
Llum no direccional i uniformement distribuïda en l'espai. Aquesta llum és reflectida pels objectes independentment de la localització i orientació de les seves superfícies.

**Antialiasing**
Tècnica de síntesi d'imatge que minimitza la distorsió produïda per els artefactes coneguts com aliasing, on l'aliasing és l'efecte que causa que senyals contínues diferents es tornin indistingibles quan són mostrejades digitalment.

**Clipping**
Eliminació de les porcions de les primitives que queden fora del semi-espai definit per determinants plans.

**Diffuse**
La lluminació i reflexió difusa té present la localització del focus de llum. La intensitat de la llum al irradiar una superfície varia segons l'angle entre la orientació de l'objecte i la direcció del focus de llum.

**Specular**
La lluminació i reflexió especular incorpora la reflexió dels objectes brillants i la posició de l'observador. La reflexió especular màxima sorgeix quan l'angle entre l'observador i la direcció de la llum reflectida és zero.

**Fragment**
Pixeles potencialment visibles, cada fragment correspon a un únic píxel i inclou color, profunditat i, sovint, les coordenades de textura.

**Framebuffer**
Matriu on cada cel la acaba codificant la informació dels píxels que es dibuixaran en el dispositiu gràfic final (pantalla). Tots els buffers d'una finestra o context donat.

**Mapat de textures**
Procés d'aplicar una imatge (textura) a una primitiva. El mapat de textures sol utilitzar-se per afegir realisme a una escena, per exemple es pot aplicar una imatge d'una façana a un polígon per representar un mur.

**Material**
Propietat d'una superfície utilitzada en el càlcul de la lluminació d'aquesta.
Model
En gràfics per computador és la representació geomètrica d'una escena.

Pixel
Element d'una imatge. Conjunt dels bits a la localització \((x, y)\) del pla de bits del framebuffer.

Primitiva
En gràfics per computador, una primitiva és un punt, una línia, un polígon, un mapa de bits o una imatge.

Snell, llei de
Fórmula utilitzada per calcular l'angle de refracció de la llum al travessar la superfície de separació entre dos medis d'índex de refracció diferent. Aquesta llei diu que el producte de l'índex de refracció pel sinus de l'angle d'incidència és constant per a qualsevol raig de llum que incideix sobre la superfície que separa dos medis.

Subfinestra
Veure visor.

UML (Unified Modeling Language)
Llenguatge gràfic per a visualitzar, especificar, construir i documentar un sistema software.

Visor
En el context d'aquest projecte, s'enten per visor el component d'una interfície gràfica que síntetitza imatges utilitzant OpenGL.