Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques
Universitat de Barcelona

Ray Casting de volum: CPU vs GPU

Sergi García Trancoso

Director: Anna Puig
Realitzat a: Departament de
Matemàtica Aplicada i
Anàlisi. UB

Barcelona, 7 de gener de 2013

Índex de la memòria

Índex de Figures i Taules

Abstract

The motivation of this project is to provide interactive displays of volumetric data captured from medical devices. In this project we analyze two strategies based on Ray Casting algorithm: the classic approach in the CPU and optimized approach on the GPU in order to study and analyze the various costs both in computational times. We propose a validation system that displays the results of both the CPU and the GPU so that they can be compared visually and computationally. Moreover, we want to go further and show the results in different ways, such as by using Look up Tables, the use of models with display lighting in the scene, such as Phong Blinn methods or other kind of projection like MIP (Maximum Intensity Projection).
This involves the analysis, design and implementation of application able to visualize volumetric data in space, bearing in mind that we want to compare results of the CPU with the GPU.

The implementation of volume rendering in real time methods, to directly transmit the information content of objects with large volumetric data, remains a challenge for the community of the discipline of computer graphics.

Most approaches based Volume Ray Casting CPU get some great results thanks to advanced memory layers, line spaces and excessive pre-computing. These approaches usually require a lot of memory. These approaches are limited in size, which usually do not satisfy the medical data used in daily routine now.

Therefore techniques have been presented based on Volume Ray Casting GPU, which give high quality results all interacting in real time. The computational power of the GPU and its flexibility has been exploited in the implementation of these techniques based on the OpenGL framework which allow easy integration of shaders for Volume Ray Casting.

Finally, in this project we will be able to see how the approaches of Volume Ray Casting on GPU can interact in real time and also the high computational cost of the approach based on Volume Ray Casting on CPU. In addition, the project implementation will be based on low-level graphics libraries such as OpenGL and GLSL shaders, that is, the application must be created from scratch. In this case, we will include and modify some class of a previous project which allowed viewing volumes through the z-buffer algorithm on CPU. Implement the GUI, events, reading ppm files and creating classes required for rendering volumetric data on screen.

1 Introducció

La motivació d'aquest projecte és la de proporcionar visualitzacions interactives de dades volumètriques procedents d'aparells de captació mèdics. En aquest projecte s'analitzen dues estratègies basades en l'algorisme de Ray Casting: l'aproximació clàssica en la Unitat Central de Processament (CPU d'ara en endavant) i l'aproximació optimitzada en la Unitat de Processament Gràfic (GPU d'ara en endavant), per tal de poder estudiar i analitzar els diferents costos computacionals. Es proposa un sistema de validació dels dos mètodes que visualitza tant els resultats de la CPU com els de la GPU de tal manera que es puguin comparar visual i computacionalment. A més a més, es vol arribar més enllà i mostrar els resultats de formes diferents, com ara amb l'ús de Look up Tables (LUTs d'ara en endavant), l'ús de models de visualització amb il·luminació de l'escena, com ara Phong Blinn i mètodes de MIP (Maximum Intensity Projection).
Això implica l'anàlisi, disseny i implementació de la aplicació capaç de poder visualitzar dades volumètriques a l'espai, tot tenint present que es vol poder comparar els resultats de la CPU amb els de la GPU.

1.1 Àmbit del projecte

En aquest projecte es desenvoluparà un sistema de visualització per a visualitzar models de volum amb l'algorisme de Ray Casting implementat en la CPU i en la GPU. Per tal que es puguin visualitzar de forma interactiva diferents regions o estructures anatòmiques de les dades captades directament des d'aparells com ressonàncies magnètiques i escàners, és necessari realitzar un procés d'etiquetatge o classificació i una interfície d'interacció amb algorismes de classificació ja desenvolupats prèviament en CPU i en GPU. En aquest projecte es focalitza la part de la visualització més acurada dels models de volum, integrant les característiques dels models classificats i les de les dades originals. L'algorisme de Ray Casting permetrà diferents il·luminacions que emfatitzaran les diferents estructures.

Utilitzats des de la dècada dels anys 70, l'escàner mèdic, permet la visualització dels òrgans interns de l'organisme gràcies a la diferent densitat dels teixits que el composen. En aquest projecte, es pretén poder treballar amb aquestes dades, aprofitant la potència de les targetes gràfiques i aplicant-li a aquests models diferents tipus d'interpolació de color, inclús que els models puguin interaccionar amb una llum puntual aplicada sobre ells mateixos, dotant-los d'un cert tipus de material i comprovar com aquest reflecteix la llum, obtenint d'aquesta manera, una imatge gràfica molt real sobre el volum.

Generalment, a la medicina, aquests models de volum, estan encapsulats a l'anomenat món de vòxels. Un món de vòxels, es pot imaginar com Z superposicions de imatges 2D amb una amplada X i una alçada Y, de manera que aquestes imatges 2D, un cop superposades, generen el cub 3D anomenat món de vòxels, on un vòxel constituirà un punt del cub 3D el qual tindrà com a propietat, un cert valor de densitat. En aquest projecte es treballarà amb aquest tipus de models tridimensionals, degut a que són àmpliament utilitzats al món de la medicina i al seu fàcil tractament i/o manipulació.

A més, les aplicacions gràfiques solen disposar d'una interfície la qual facilita a l'usuari la interacció cap als seus objectius, de tal manera que aquest últim, pugui, de forma unívoca i sense errors, obtenir els resultats desitjats d'una manera intuïtiva i fàcil. En aquest projecte, es planteja la idea de realitzar, una interfície gràfica, intuïtiva i fàcil d'utilitzar, de manera que per aconseguir resultats, l'usuari no hagi de tenir coneixements previs sobre els algorismes utilitzats.

El perfil d'usuari que suposem utilitzarà l'aplicació, serà el de qualsevol usuari interessat en la visualització tridimensional, de les dades captades de ressonàncies magnètiques i escàners, de manera ràpida, gràcies a l'evolució de les GPU, a més a més de la possible visualització per capes que l'algorisme de Ray Casting permet sobre el model de volum.

Per últim, fent referència al pla d'estudis de la carrera, aquest projecte està estrictament relacionat amb l'assignatura de Gràfics i Visualització de Dades, on es tracten i es donen els primers passos a les aplicacions gràfiques, els coneixements bàsics sobre com posicionar un objecte a una escena i ser capaços de visualitzar-los en un viewport o imatge final, així com els pipelines bàsics i frameworks que ens poden ajudar a aconseguir els objectius de la assignatura.

A més a més, s'han de mencionar, les assignatures prèvies i que tenen relació amb Gràfics i Visualització de Dades, tals com:

- Programació I: On s'introdueixen els coneixements bàsics de la programació.
- Programació II: On s'introdueixen coneixements avançats de programació.
- Introducció a la Computació Científica: On s'introdueixen els coneixements bàsics de la programació computacional científica.
- Algorísmica: On s'introdueix què són els algorismes i la importància dels costos computacionals.
- Disseny de Software: On s'introdueixen les metodologies a seguir per al correcte disseny i posterior desenvolupament del software.
- Estructura de Dades: On s'introdueixen què són i com utilitzar les diferents estructures de dades.
- Sistemes Operatius I i II: On s'introdueixen els coneixements necessaris per tal de gestionar els diferents recursos disponibles a un ordinador i a com gestionar la concurrència.

1.2 Motivació

La representació gràfica de models tridimensionals o renderització de volum, es basen en un conjunt de tècniques utilitzades per a mostrar una projecció en 2D d'un conjunt de dades discretament mostrejat. Com s'ha esmentat abans, un conjunt de dades 3D es pot veure com un conjunt de imatges 2D superposades, i en el cas de la medicina, aquestes imatges 2D serien preses per aparells de ressonàncies magnètiques o escàners mèdics.

A l'assignatura de Gràfics i Visualització de dades ens varen introduir a la renderització d'una escena, la qual es composava d'un o varis objectes, visualitzats gràcies a la projecció en perspectiva o paral·lela, a partir d'una càmera situada a una certa posició i dirigida directament contra el model volumètric. A més vàrem aprendre diferents tècniques de shading o interpolació de color. Els objectes es modelaven segons la seva superfície i no es tenia en compte el seu interior.

Aquesta renderització, bàsicament la realitzàvem a partir de l'algorisme del z-buffer, el qual consisteix en guardar en un buffer de profunditat les dades a pintar, de tal manera que es visualitzin sempre abans els píxels amb menor profunditat degut a que es troben més endavant dels píxels amb major profunditat i també estan més a prop de qui els està visualitzant. A l'assignatura de Gràfics i Visualització de dades, ens van mencionar altres tècniques de renderització, com per exemple, el Ray Tracing o el Ray Casting, els quals comparteixen un denominador comú, per tal de poder visualitzar l'escena, llancen un conjunt de rajos amb els quals es pot calcular la il·luminació o detectar quins objectes estan més a prop de l'observador i quins estan més lluny.

En les dades procedents de escàners o ressonàncies magnètiques, els objectes són volumètrics, és a dir, estan representats implícitament amb punts mostrejats a l'espai de forma regular (vòxels) i a cada punt es mostreja un valor de densitat, d'activitat, etc...
Els objectes ja no són simplificables amb una superfície i és necessari adoptar les tècniques clàssiques de visualització de models de volum.

Finalment, amb ganes de conèixer amb més profunditat aquestes tècniques, aplicades a dades de volum, poder implementar-les, realitzar un sistema de visualització amb el qual es visualitzin els resultats i a més, degut a que podem aprofitar el paral·lelisme de la multitud de processadors que romanen a les GPU, comparar aquestes tècniques en CPU i GPU i els seus costos computacionals.

1.3 Objectius generals

La finalitat d'aquest projecte és la d'analitzar, dissenyar i implementar un sistema de visualització el qual sigui capaç de mostrar la renderització de volum mitjançant els diferents algorismes de Ray Casting aplicats a dades volumètriques a la CPU i a la GPU, per tant serà necessari l'estudi de:

- Estudi previ dels mètodes de Ray Casting aplicats a volum en CPU i GPU.

- Disseny d'un sistema de visualització que permeti la càrrega i visualització d'un món de vòxels. Aquest món de vòxels pot haver estat prèviament segmentat o bé ser directament un món de vòxels procedent d'un aparell mèdic.

- Anàlisi, disseny i implementació del **Ray Casting de volum** a la **CPU**.

- Anàlisi, disseny i implementació del **Ray Casting de volum** a la **GPU**.

- Context en el què s'implementarà l'aplicació.

- Validació dels mètodes: anàlisi visual i computacional.

1.4 Objectius específics

Els objectius generals es detallen com els següents objectius específics.

1. *Estudi previ dels mètodes de Ray Casting aplicats a volum en CPU i GPU.*

   Estudiar la manera que es composa un món de vòxels, així com entendre les seves propietats i per a què poden ser útils. Analitzar i estudiar el Ray Casting de volum en CPU i GPU i entendre les seves diferències.

2. *Disseny d'un sistema de visualització que permeti la càrrega i visualització d'un món de vòxels.*

   Analitzar, dissenyar i implementar el sistema, el qual farà ús d'una interfície gràfica com a mecanisme de comunicació entre aquest i l'usuari.

3. *Disseny de la interfície gràfica.*

   Estudiar i realitzar un disseny fàcil i intuïtiu per a l'usuari. A més la interfície a d'esser capaç de mostrar en el mateix instant de temps, la visualització del resultat del Ray Casting de volum de la CPU i la GPU.

4. *Anàlisi, disseny i implementació del **Ray Casting de volum** a la **CPU**.*

   Estudiar i implementar les diferents parts de les que consta aquest algorisme de renderització volumètrica, com ara:
   - 4.1 Càlcul dels rajos.
   - 4.2 Intersecció dels rajos amb el volum.
   - 4.3 Càlcul de la il·luminació, acumulació del color.
   - 4.4 Ús de textures per a mostrar el resultat final.

5. *Anàlisi, disseny i implementació del **Ray Casting de volum** a la **GPU**.*

   Estudiar quines tècniques es poden utilitzar per tal de fer un *port* a la GPU de l'algorisme de Ray Casting de volum tradicional i implementarles:
   - 5.1 Ús de shading Language (Scripts executats per la GPU).
   - 5.2 Ús de framebuffers (Off-screen rendering).
   - 5.3 Ús de textures tridimensionals.
   - 5.4 Càlcul del raig a la GPU.
   - 5.5 Iteració sobre la textura 3D a la GPU.
   - 5.6 Càlcul de la il·luminació, acumulació del color a la GPU.

- *Definició del context en el que s'implementarà l'aplicació.*

  Utilització del framework Qt Creator, el cual permet implementar aplicacions C++, OpenGL i GL Shading Language (GLSL d'ara en endavant).

- *Anàlisi dels costos computacionals CPU vs GPU.*

  Estudiar i analitzar els resultats obtinguts dels diferents costos computacionals.

1.5 Organització de la memòria

El present document s'organitzarà en els següents capítols:

Capítol 1: Introducció.
En aquest capítol s'introdueixen quines són les motivacions d'aquest projecte, els objectius i en quin àmbit està situat.

Capítol 2: Antecedents.
En aquest capítol s'introduiran els antecedents d'aquest projecte, els problemes a tractar i la determinació de tecnologies a utilitzar per tal de dur a terme la solució.

Capítol 3: Anàlisi.
Anàlisi dels requeriments de l'aplicació mitjançant casos d'ús i el model de domini, de les solucions als problemes a tractar.

Capítol 4: Disseny.
Disseny de l'aplicació mitjançant diagrames de classes i diagrames de seqüència associats als casos d'ús del capítol anterior.

Capítol 5: Validació i simulacions.
Exposició dels resultats obtinguts amb comparatives visuals i computacionals.
Simulacions amb descripció del funcionament de l'aplicació, captures de la interfície, visualització dels resultats i taules de temps.

Capítol 6: Conclusions i futures línies de continuació.
Conclusions del projecte. Assoliments d'objectius i enumeració de futures línies de continuació.

Capítol 7: Referències bibliogràfiques.

Annex I: Instal·lació, requeriments mínims i passos a seguir.
Annex II: Apèndix A. Manual del desenvolupador.
Apèndix B. Manual d'usuari.

2 Antecedents

La finalitat d'aquest capítol és la d'analitzar els antecedents del tema que tracta aquest projecte, de manera que podem situar-lo dins d'un context més general i determinar-ne l'àmbit de l'aplicació.

El capítol per tant, es dividirà en quatre apartats:

2.1 Background: Explicació dels mètodes més utilitzats per a la visualització de volums i breu explicació de sobre com es classifiquen.

2.2 Ray Casting de Volum a la CPU.

2.3 Ray Casting de Volum a la GPU.

2.4 Conclusió.

2.1 Background

Seguidament s'explicaran alguns dels diferents mètodes utilitzats per a la visualització de volums.

2.1.1 Visualització de Volums

A la disciplina dels gràfics per computador, la visualització d'un model tridimensional o volum, implica la projecció en 2D de l'escena que conté aquest volum. Aquesta projecció esdevé gràcies a la definició d'una càmera la qual determina on es situa l'observador.

Els volums que es visualitzaran en aquest projecte, corresponen a dades extretes de aparells mèdics tals com ressonàncies magnètiques o escàners. Les dades obtingudes d'aquests aparells estan organitzades d'una forma determinada, generalment el que obtenim és l'anomenat món de vòxels, i aquest, serà el volum amb el que tractarem, on a cada vòxel es mostreja el valor de densitat associat.

Món de vòxels

Un **vòxel** (Fig. 1) representa una mostra unitària, o punt de les dades, amb espai regular. Aquest punt pot contenir diverses propietats tals com una opacitat, o un color i una opacitat, entre d'altres. Un vòxel mai no representa un volum, sinó un punt a l'espai volumètric.

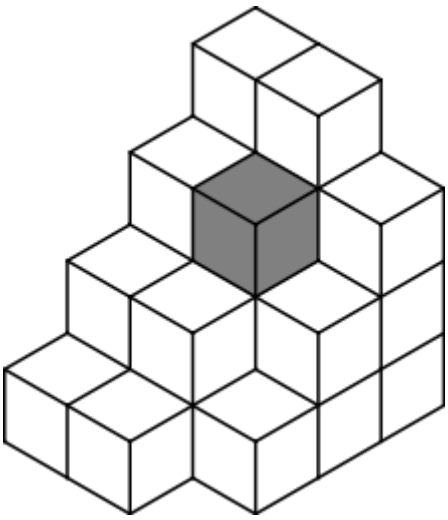


Fig. 1 - Vòxel a l'espai

Mentre que un vòxel proporciona el benefici de la precisió i la profunditat de la realitat, generalment un conjunt, o món de vòxels pot ésser un gran conjunt de dades i això els fa difícils d'administrar degut a la limitació de l'ample de banda dels ordinadors comuns.

Doncs un **món de vòxels** (Fig. 2) es considerarà com el conjunt de vòxels que formen un volum. Aquests vòxels es repartiran al llarg dels eixos del volum, de tal manera que la quantitat de vòxels que ocupen un eix ens donarà la resolució d'aquell eix. La resolució total del volum vindrà donada per la quantitat de vòxels repartits a cadascun dels eixos i la mida de cadascun d'ells. Aquesta resolució determina la quantitat d'espai utilitzada per el model.

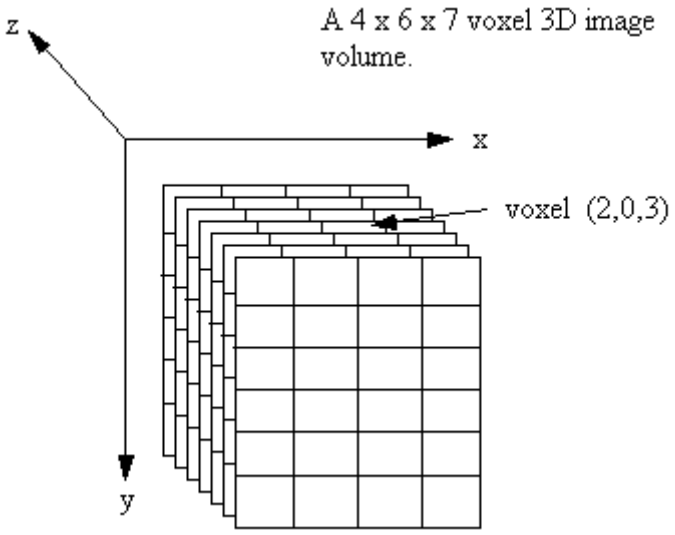


Fig. 2 - Món de vòxels i la seva distribució al llarg dels eixos

Renderització de volum

En aquest projecte, visualitzarem el tipus de dades tridimensionals descrites en els paràgrafs anteriors. El procés necessari per a la visualització del món de vòxels, rep el nom de **renderització de volum** [17].

A la disciplina de gràfics per computador, la renderització de volum consisteix en un conjunt de tècniques emprades per a visualitzar projeccions 2D de models tridimensionals, com per exemple un món de vòxels.

La renderització directa d'un volum requereix del mapeig de cada mostra a una opacitat i un color. Això pot ser directament obtingut gràcies a l'ús d'una funció de transferència, amb la qual es mapeja el valor de densitat del vòxel a un color i opacitat concret. Cal mencionar que el color és composa de quatre components, les denominades RGBA (Red Green Blue Alpha), les tres primeres indiquen quina intensitat tindrà el color en el seu canal corresponent, i la quarta indica la opacitat d'aquella mostra. Un cop s'han obtingut els RGBA, aquests són projectats als píxels corresponents del framebuffer. La manera com es fa depèn de la tècnica de renderització de volum utilitzada.

Així doncs, a la renderització de volum hi ha diferents tècniques, la més bàsica és la tècnica projectiva de Z-buffer [18], altres de més complexes, són les de Ray Casting [12] i Ray Tracing [13].

A més, aquestes tècniques són perfectament compatibles amb l'ús de mapejat de textures, les quals poden, en un moment donat, dotar de realisme l'escena que es visualitza.

Z-Buffer

A la disciplina dels gràfics per computador, la tècnica de z-buffer [18], és basa en la gestió de la profunditat de coordenades de la imatge en models tridimensionals. Bàsicament és una solució al problema de la visibilitat de les parts del model tridimensional a la imatge final, decidir quines parts són visibles i quines no han de ser-ho.

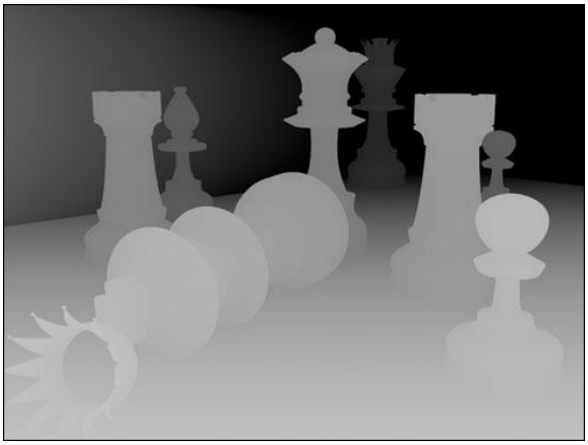


Fig. 3 - Representació d'una escena mitjançant z-buffer

La tècnica consisteix en emmagatzemar a un buffer (z-buffer), la profunditat dels píxels generats. Aquest buffer es representa generalment com una matriu bidimensional amb coordenades x i y, amb un element per a cada píxel. Si ha d'ésser pintat algun altre element de l'escena al mateix píxel on tocava pintar, el mètode tracta de comparar les dues profunditats i finalment es seleccionarà la més propera a l'observador. Aquesta nova profunditat substitueix a la antiga en el z-buffer. El resultat és el fet de poder representar la percepció de profunditat usual, veure cóm un objecte fa oclusió a un altre.

Els problemes de qualitat de la tècnica de z-buffering poden aparèixer quan la precisió en els valors de distància del z-buffer no es distribueixen per igual. Els valors més propers són més precisos (i així es poden mostrar objectes propers millor) que els més llunyans. Normalment això és desitjable, però pot causar artefactes de manera que els objectes semblin més distants de com estan en realitat.

Ray Casting i Ray Tracing de models superficials

A la disciplina dels gràfics per computador, el **Ray Casting** [12] fa referència a l'ús de la intersecció raig-superfície [11] per a solucionar una varietat de problemes. No confondre amb l'algorisme que s'implementa en aquest projecte, el Ray Casting de volum.

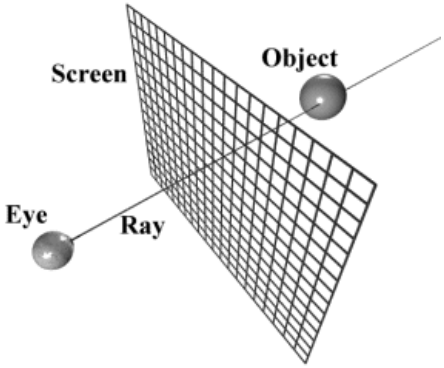


Fig. 4 - Representació del Ray Casting

En l'algorisme de Ray Casting [12] es determinen les superfícies visibles en l'escena que es vol sintetitzar traçant rajos des de l'observador (càmera) fins a l'escena a través del pla de la imatge. Es calculen les interseccions del raig amb els diferents objectes de

l'escena i aquella intersecció que estigui més a prop de l'observador determina quin és l'objecte visible.

El **Ray Tracing** [13] és un altre algorisme per a la síntesi d'imatges tridimensionals, basat en l'algorisme de determinació de superfícies visibles anomenat Ray Casting.

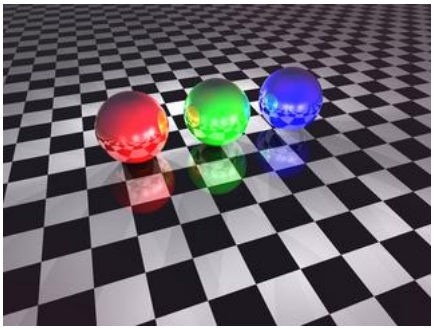


Fig. 5 - Imatge sintetitzada amb Ray Tracing

A partir de la primera intersecció del raig amb una superfície, es calcula un segon raig (reflectit) en aquell punt d'intersecció. Amb el segon raig es busca la següent intersecció amb una altra superfície per a veure com influeix en la il·luminació del primer punt.

L'algorisme de Ray Tracing estén la idea de traçar els rajos per determinar les superfícies visibles amb un procés d'ombrejat (càlcul de la intensitat del píxel) que té en compte efectes globals d'il·luminació com ara reflexions, refraccions o ombres llançades.

Mapeig de Textures

El mapeig de textures [15] és un mètode per afegir detalls, colors o textures exteriors a un gràfic generat per ordinador o un model 3D.

El mapa de la textura és aplicat a la zona exterior de la figura o el polígon. Aquest procés és similar a aplicar paper de regal a una caixa blanca.

El multi-texturat és l'ús de més d'una textura alhora en un polígon. Per exemple, una textura es pot utilitzar a manera de mapa de llums per il·luminar la superfície de l'objecte com a alternativa a recalcular la llum en aquesta part de l'objecte cada vegada que és renderitzat. Una altra tècnica del multi-texturat és el mapa topològic que pot donar una bona aparença en superfícies complexes, com l'escorça d'un arbre o ciment aspre, això ofereix detalls de llums afegit als detalls del color.

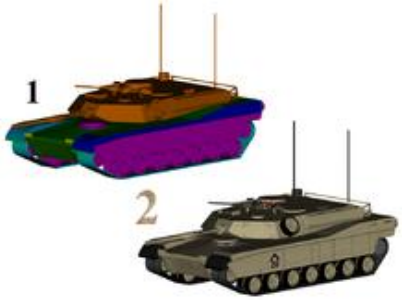


Fig. 6 - Objecte sense i amb textures

2.1.2 Classificació de Volums

Entre les diferents tècniques de definició de volum es troben els processos de segmentació i classificació del volum [3]. En aquest projecte podem tractar amb dades segmentades o directament amb el món de vòxels del volum sense haver estat segmentat.

La segmentació de volum en 3D és el procés de divisió en regions vòxels 3D (subvolums) que representen entitats físiques significatives que són més fàcil d'analitzar i que es puguin utilitzar en aplicacions futures.

La segmentació de volum assigna els vòxels d'imatges 3D en particions o regions en 3D que representen entitats físiques significatives. L'objectiu és distingir entre les diferents regions en el volum 3D i cobrir els contorns extrets de tot el volum.

La classificació de vòxels a les regions es realitza d'acord a una regió determinada a la qual pertanyen els vòxels, i algunes de les propietats compartides. Aquests vòxels comprendran un objecte aïllat o segmentat d'interès (OOI) del volum d'entrada.

La classificació permet mapejar el color i la opacitat a una estructura o volum d'interès donat en el procés de segmentació. Usualment s'utilitzen funcions de transferència entre valors i colors, implementades en Look up Tables.

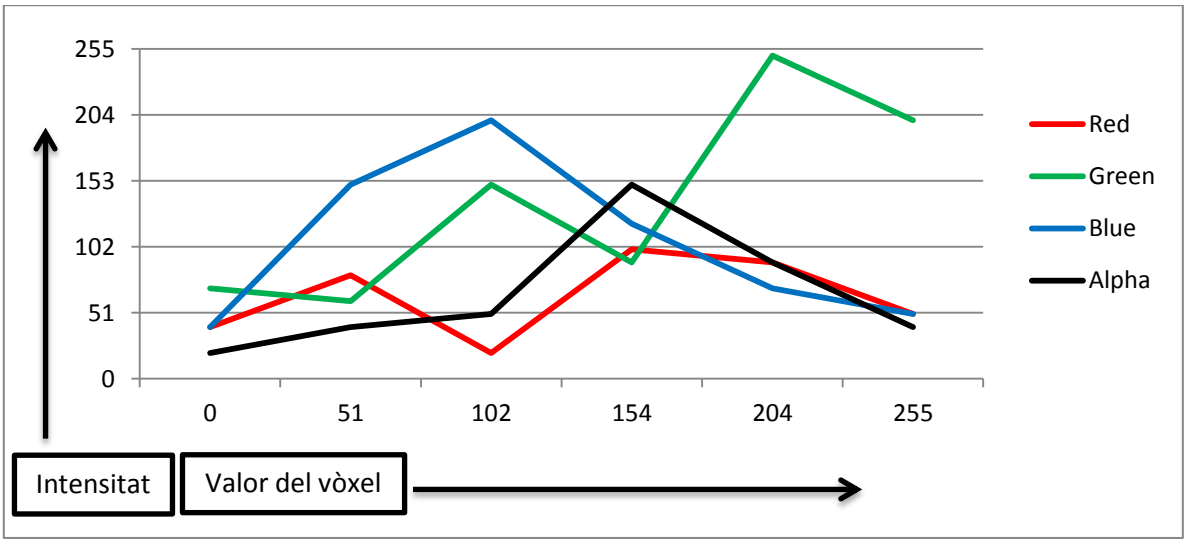


Fig. 7 - Funció de transferència (LUT)

Algunes de les tècniques més emprades a la segmentació i classificació de volum són:

- *Thresholding:* Segmentació del volum a partir del valor de intensitat directe del vòxel. La idea és agrupar els vòxels que es troben amb un cert valor de intensitat major que l'umbral d'intensitat establert.

- *Clustering:* És el procés de segmentació de cada grup dels píxels d'un volum en una classe, cada classe té les mateixes propietats o similars de tal manera que avaluïn una part específica del volum.

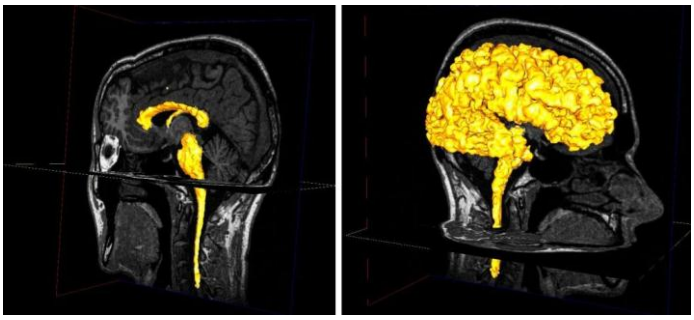


Fig. 8 - Representació de la segmentació d'un Volum

Entre d'altres tècniques com per exemple, les transformació d'ones (mitjançant Fourier), transformació d'ones discretes...

2.2 Ray Casting de Volum a la CPU

La implementació de mètodes de renderització de volum en temps real, per tal de transmetre directament el contingut d'informació dels objectes amb grans dades volumètriques, continua sent tot un repte per a la comunitat de la disciplina dels gràfics per computador.

Bàsicament s'ha observat que la capacitat d'una sola CPU de propòsit general no és suficient per aconseguir interactivitat en temps real amb els grans volum de dades que conformen els models volumètrics en l'actualitat. Tot i això s'ha invertit un gran esforç en les tècniques d'acceleració per al renderitzat de volum basat en CPU i en el disseny i explotació de maquinari dedicat als gràfics.

La majoria de les aproximacions de Ray Casting de Volum basades en CPU aconsegueixen uns grans resultats gràcies a avançades capes de memòria, subdivisió d'espais i pre-computació excessiva [1]. Aquestes aproximacions normalment necessiten d'una gran quantitat de memòria. Estan limitades a les dimensions, les quals normalment no satisfan les dades mèdiques utilitzades a la rutina diària actualment.

El Ray Casting de Volum està sent àmpliament utilitzat en la representació d'alta qualitat que es desitja sobre grans volums de dades. S'han proposat diverses tècniques d'acceleració per al Ray Casting de Volum en els últims anys. Aquestes tècniques aconsegueixen frame-rates significatius utilitzant un disseny de propagació de la memòria i gradients pre-computats. De totes maneres, aquestes tècniques necessiten d'una gran quantitat de memòria addicional que no poden esser subministrades pels computadors tradicionals.

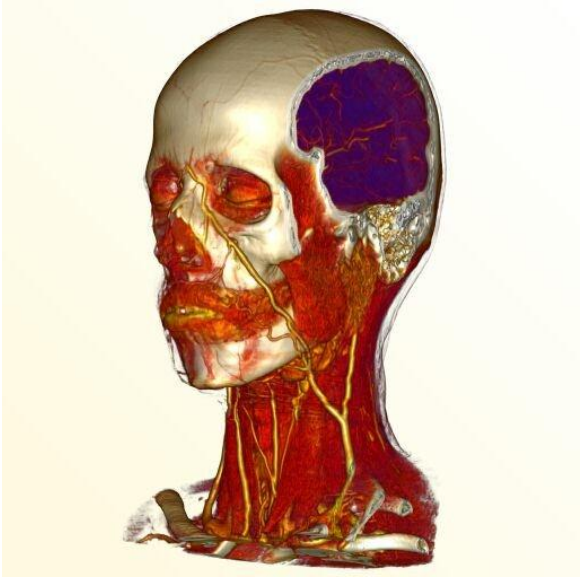


Fig. 9 - Resultat de Ray Casting de Volum basat en CPU

2.3 Ray Casting de Volum a la GPU

En els últims anys la programabilitat de les estacions de treball i el maquinari de gràfics a nivell de consum ha evolucionat a un ritme cada vegada més gran. Impulsada per les demandes constantment creixents de la indústria dels videojocs, el rendiment dels processadors gràfics moderns ha superat la potència de càlcul de CPU en xifres brutes i en la seva extraordinària taxa de creixement. Les GPU d'avui s'han convertit en unitats de processament molt sofisticades i altament programables SIMD (Single Instruction Multiple Data). Amb l'arribada dels processadors gràfics que donen suport a les noves característiques de la dinàmica de bucle i de ramificació, les GPU s'estan tornant cada cop més en les unitats generals de processament amb comparació a la CPU.

Per aquest motiu s'han presentat tècniques de Ray Casting de Volum basades en GPU [6], les quals donen resultats d'alta qualitat tot interaccionant en temps real. El poder computacional de les GPU i la seva flexibilitat han estat explotades en la implementació d'aquestes tècniques amb frameworks basats en OpenGL [7] els quals permeten la fàcil integració dels shaders per al Ray Casting de Volum. Aquests frameworks es basen en la idea d'una aplicació conductora per una banda, i per l'altra, un conjunt de programes anomenat "*fragment shaders*" els quals poden ser carregats en temps d'execució i s'executen en paral·lel a cada píxel projectat. Tots aquests shaders implementen l'aproximació del Ray Casting de Volum, explotant la funcionalitat de looping que es troba implícita en aquests tipus de programes, de manera que tot quedi executat en només un pas de renderitzat.

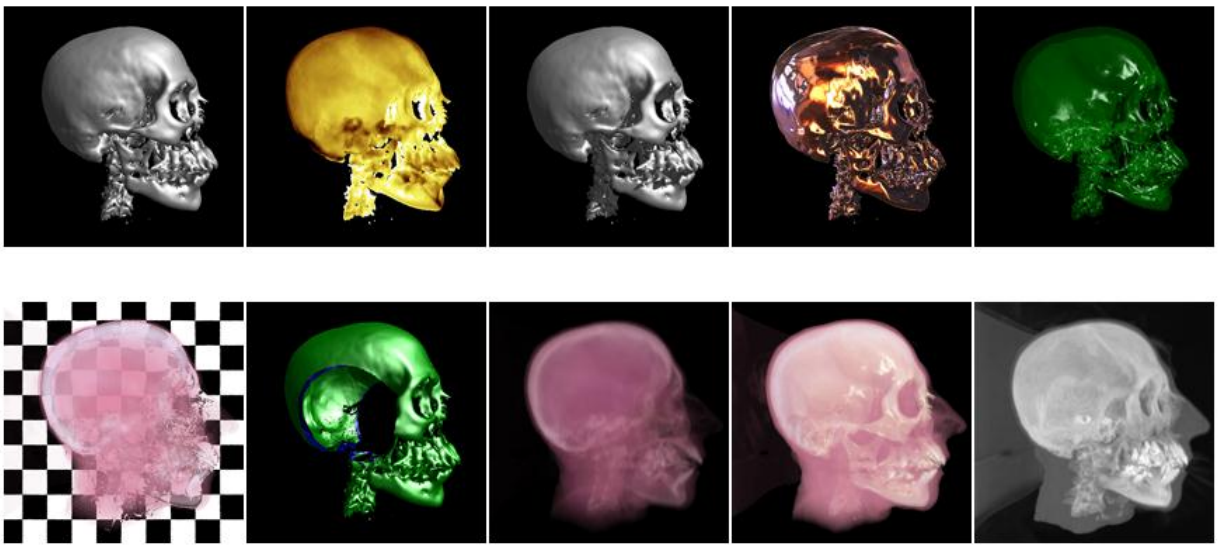


Fig. 10 - Resultats de Ray Casting de Volum basat en GPU

2.4 Conclusions

A la disciplina dels gràfics per computador, la visualització d'un model tridimensional o volum, implica la projecció en 2D de l'escena que conté aquest volum.

Els volums que es visualitzaran en aquest projecte, corresponen a dades extretes de aparells mèdics tals com ressonàncies magnètiques o escàners. Les dades obtingudes d'aquests aparells estan organitzades d'una forma determinada, generalment el que obtenim és l'anomenat món de vòxels.

Un vòxel representa una mostra unitària de la densitat, o punt de les dades, amb espai regular. Un món de vòxels es considerarà com el conjunt de vòxels que formen un volum.

El procés necessari per a la visualització del món de vòxels, rep el nom de renderització de volum. A la renderització de volum hi ha diferents tècniques, la més bàsica és la tècnica projectiva de Z-buffer, altres de més complexes, Ray Casting i Ray Tracing. A més, aquestes tècniques són perfectament compatibles amb l'ús de mapejat de textures, les quals poden, en un moment donat, dotar de realisme l'escena que es visualitza.

Els volums amb els que es treballarà en aquest projecte poden haver estat prèviament segmentats, o ser directament el món de vòxels. La segmentació de volum en 3D és el procés de divisió en regions vòxels 3D (subvolums). La classificació de vòxels a les regions es realitza d'acord a una regió determinada a la qual pertanyen els vòxels, i algunes de les propietats compartides. Algunes de les tècniques més emprades a la segmentació de volum són el "*Thresholding"* i el *"Clustering"*, entre d'altres.

Passats els anys, s'ha observat que la capacitat d'una sola CPU de propòsit general no és suficient per aconseguir interactivitat en temps real amb els grans volum de dades que conformen els models volumètrics en l'actualitat. Per aquest motiu s'han presentat tècniques de Ray Casting de Volum basades en GPU, les quals donen resultats d'alta qualitat tot interaccionant en temps real. El poder computacional de les GPU i la seva flexibilitat han estat explotades en la implementació d'aquestes tècniques amb frameworks basats en OpenGL els quals permeten la fàcil integració dels shaders per al Ray Casting de Volum.

Finalment, en aquest projecte serem capaços d'observar com es pot interaccionar en temps real amb l'aproximació del Ray Casting de volum basat en GPU i el gran cost computacional de l'aproximació del Ray Casting de volum basat en CPU. A més, la implementació del projecte estarà basada en les llibreries de gràfics a baix nivell com OpenGL, és a dir, s'haurà de crear l'aplicació des de zero. En aquest cas, s'inclourà i es modificarà alguna classe d'un projecte anterior el qual permetia la visualització de volums mitjançant l'algorisme de Z-buffer en CPU. Implementar la interfície gràfica, els events, la lectura de fitxers ppm i la creació de classes necessàries per a la renderització en pantalla de les dades volumètriques.

3 Anàlisi

En aquest capítol s'analitzaran les diferents aproximacions de Ray Casting de Volum en CPU i en GPU implementades en aquest projecte, a més també s'analitzaran els diferents casos d'ús de l'aplicació final i el model de domini.

Per tant, dividirem aquest capítol en els següents subapartats:

3.1 Anàlisi de l'aproximació de Ray Casting de Volum en CPU.

3.2 Anàlisi de l'aproximació de Ray Casting de Volum en GPU.

3.3 Anàlisi de les dades parametritzables.

3.4 Casos d'ús de l'aplicació.

3.5 Model de Domini de l'aplicació.

3.6 Conclusió.

3.1 Anàlisi de l'aproximació de Ray Casting de Volum en CPU

La tècnica del Ray Casting de Volum [16] (Fig. 11) proporciona resultats d'alta qualitat. La idea bàsica d'aquesta tècnica és la de llençar un raig contra el volum per a cada píxel de la imatge final. Utilitzant un model simple de càmera, el raig serà llençat contra el volum des de la posició de l'observador. La finalitat és la de calcular per a cada píxel de la imatge final, el color i la opacitat d'aquest gràcies a la possibilitat de poder mostrejar els vòxels que són travessats pel raig.

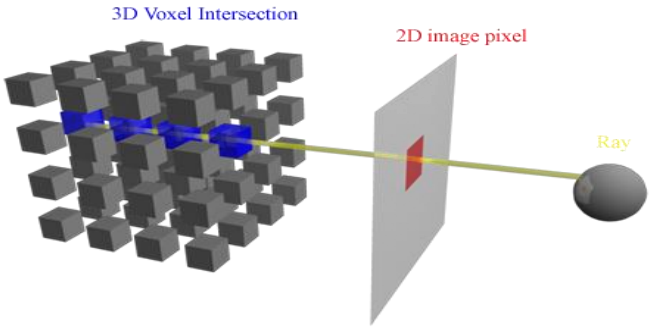


Fig. 11 - Representació del Ray Casting de Volum per a un píxel de la imatge final

L'algorisme de Ray Casting de Volum en CPU, es composa de quatre parts:

1- *Ray casting:* Per a cada píxel de la imatge final, un raig és llençat contra el volum des de la posició de l'observador.

   Un raig és una línia recta el qual ve donat per la següent equació:

$$Raig = p_0 + \lambda \vec{v}$$

   On p_0 és el punt inicial del Raig en coordenades de món, λ és l'increment de desplaçament per al mostreig i $\vec{v}$ és el vector director de la línia recta que formarà el raig.

2- *Sampling:* Al llarg de la part del raig que roman dins el volum, es seleccionen punts equidistants, això vol dir que caldrà comprovar si el raig intersecciona o no amb el món de vòxels. En general, el raig no es troba alineat amb el volum, de tal manera que aquests punts no seran consecutius i estaran entre els vòxels, és per això que serà necessari interpolar els punts a mostrejar amb els punts que envolten el punt de mostreig.

La interpolació trilineal (Fig. 12) genera una mostra balancejada en el seu espai tridimensional. Això s'aconsegueix prenent els valors dels 8 vòxels veïns que envolten el punt a mostrejar.

$$V_{xyz} = V_{000}(1-x)(1-y)(1-z) + V_{100}x(1-y)(1-z) + V_{010}(1-x)y(1-z) + V_{001}(1-x)(1-y)z + V_{101}x(1-y)z + V_{011}(1-x)yz + V_{110}xy(1-z) + V_{111}xyz$$

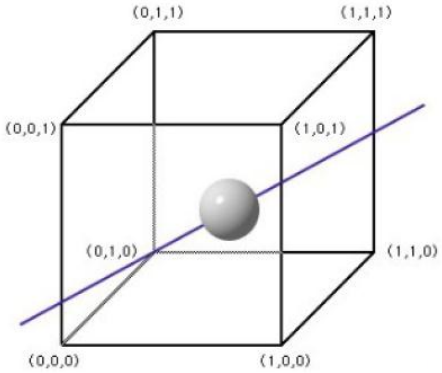


Fig. 12 - Equació i representació gràfica de la interpolació trilineal

3- *Shading:* Per a cada punt de mostreig, es calcula la seva il·luminació o color.

En aquest projecte es calcularà la il·luminació o color amb diverses tècniques:

- *Escala de grisos:* El color serà calculat mapejant directament els valors de densitat dels vòxels, els quals estan entre 0 i 255. Als seus valors de grisos corresponents.

- *LUT (Look up Table):* Els colors dependran dels valors de densitat dels vòxels. Segons quin sigui el valor de densitat del vòxel actual, s'anirà a buscar la correspondència de color a la look up table (Procés de classificació).

- *Phong Blinn:* Tècnica de càlcul d'interpolació del color segons els materials aplicats al volum i les llums que l'il·luminen. A més es combinarà aquesta tècnica amb les LUTs, de tal manera que segons el valor de densitat del vòxel actual, s'anirà a buscar la correspondència de component difusa del material a la look up table. Cal mencionar que per a calcular el Phong Blinn, calen les normals del model 3D, això s'ha aconseguit gràcies al càlcul del gradient del volum, que s'ha considerat directament com les normals del volum.

- *MIP:* Tècnica de projecció del valor màxim de intensitat.

4- *Compositing:* Després d'haver calculat la il·luminació i el color de cada punt de mostreig, aquests són composats al llarg del raig, obtenint així el color resultant per al píxel que s'està processant.

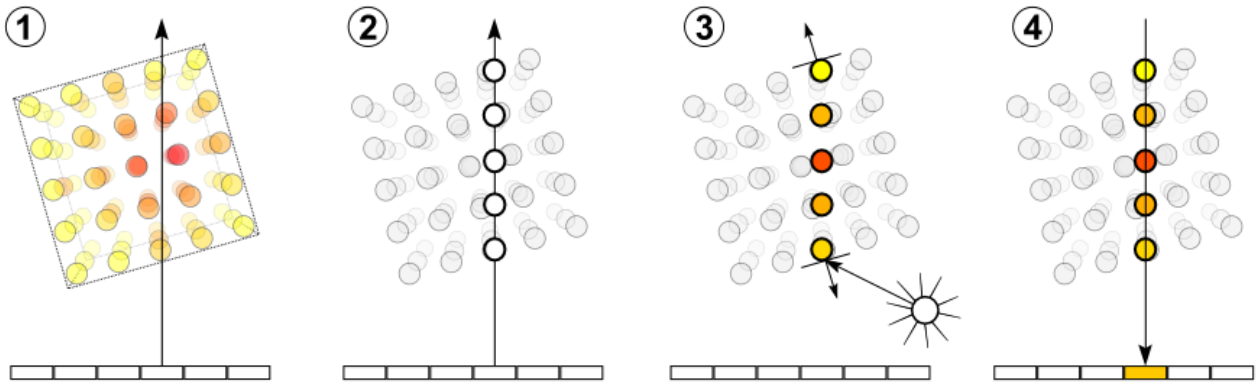


Fig. 13 - Representació de les quatre parts de l'algorisme del Ray Casting de volum

Així doncs, el recorregut de mostreig del volum pot ser d'endarrere cap endavant (**back to front**) o d'endavant cap endarrere (**front to back**).

El recorregut **back to front** consisteix en recórrer el model des del final d'aquest fins al inici en direcció cap a l'observador. D'aquesta manera l'usuari podrà obtenir informació de parts del volum les quals haurien de quedar amagades segons la percepció de profunditat usual.

El recorregut **front to back** consisteix en recórrer el model des de l'inici d'aquest fins al final en la mateixa direcció en que l'observador està mirant. Aquest recorregut ens permet implementar la **early ray termination**, amb la qual podem reduir el temps de mostreig. La early ray termination consisteix en parar de mostrejar quan la opacitat acumulada sigui igual a la màxima opacitat que es pot acumular, de tal manera que quan s'hagi arribat a opacitat màxima es passarà a computar el següent raig.

3.2 Anàlisi de l'aproximació de Ray Casting de Volum en GPU

El nucli principal de l'aproximació de Ray Casting de volum en GPU [5] continua sent el mateix que en CPU, enviar un raig per a cada píxel de la imatge final i mostrejar aquest raig a través del volum. Això es pot implementar en un fragment shader i la representació es pot fer en temps real. La tècnica és bastant flexible i permet la implementació en poques línies de codi les diferents tècniques d'interpolació de color.

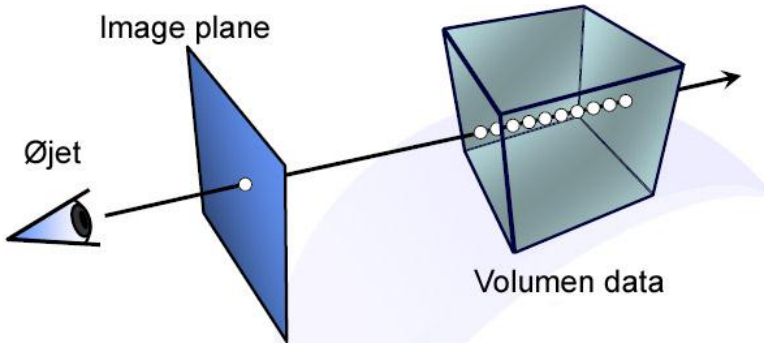


Fig. 14 - Nucli principal del Ray Casting de Volum

En aquesta aproximació, per tal de generar els raigs farem ús de l'habilitat d'interpolació implícita a OpenGL per a renderitzar una geometria.

De la mateixa manera, un raig és una línia recta el qual ve donat per la següent equació:

$$Raig = p_0 + \lambda \vec{v}$$

On p_0 és el punt inicial del Raig en coordenades de món, λ és l'increment de desplaçament per al mostreig i $\vec{v}$ és el vector director de la línia recta que formarà el raig. Per tal de generar el raig necessitarem trobar el punt d'origen i el vector director.

Podem aconseguir el raig renderitzant un cub de les mateixes dimensions que el món de vòxels, on els colors representen les coordenades. Aquest cub s'haurà de renderitzar d'una forma determinada, en concret, volem renderitzar el frontface (cares frontals) i el backface (cares posteriors), tal i com es mostra a la següent figura.

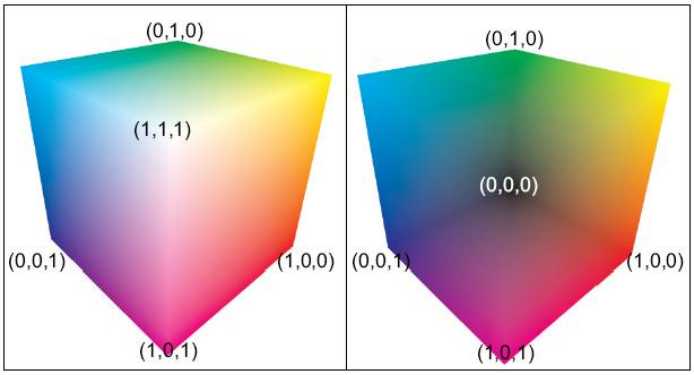


Fig. 15 - Renderització del frontface i el backface respectivament

El punt inicial del nostre raig serà directament el punt actual del frontface a la renderització, i el vector director l'obtindrem fent la següent operació:

$$\vec{v} = backface - frontface$$

D'aquesta manera calcularem el raig actual.

Per tal de ser capaços de renderitzar les frontface i les backface, farem ús dels FrameBufferObject d'OpenGL [8], els quals permeten la renderització fora de pantalla de qualsevol geometria.

Es farà en dos passos de renderitzat, un per les frontface i un altre per a les backface. Per a renderitzar les backface haurem d'activar el frontface culling, el qual permet amagar les parts davanteres de tal manera que només renderitzi les posteriors. El renderitzat dels frontface s'utilitzarà per a determinar els punts inicials dels raigs al procés de Ray Casting de volum.

Finalment, abans d'implementar el Ray Casting de volum al fragment shader, haurem de passar les dades necessàries a la GPU, és a dir, la textura 2D de les backface, les coordenades de textura del cub que s'ha utilitzat per a renderitzar les frontface, els punts del mateix cub i la textura volumètrica la qual representarà el món de vòxels a la GPU.

Un cop tenim totes les dades necessàries a la GPU, implementem el nucli principal del Ray Casting de volum en el fragment shader considerant que s'executa un fragment shader per cada píxel del frontface renderitzat. Calculem el raig actual, i fem el mostreig del raig sobre la textura volumètrica, calculem il·luminació i color i finalment fem la composició de color per al píxel actual.

Versió optimitzada

En aquest projecte, s'ha implementat una versió optimitzada de l'aproximació de Ray Casting de volum en GPU, gràcies a la implementació del model de càmera, el qual conté dades necessàries per al càlcul del vector director del raig.

Bàsicament, ens estalviarem el fet d'haver de renderitzar fora de visualització final les backface, això implica el no haver de inicialitzar *framebuffers*, ni *renderbuffers*, és a dir, estalviar-se passos de procés en CPU, de manera que s'executin abans els *shaders* de la GPU. Per tant, podrem visualitzar l'aproximació de Ray Casting de volum en GPU amb un sol pas de renderitzat, en contra dels dos passos de renderitzat del mètode no optimitzat, per aquest motiu anomenarem a aquest mètode *One step.*

Com ara no tindrem les backface, no podrem obtenir el vector director amb la operació de diferència que hi ha en els paràgrafs anteriors, però sí amb dades de la càmera. El model

de càmera conté el *vrp* que és la posició d'origen del volum i l'*observador*, que és la posició on es troba la càmera. Doncs amb la següent operació, obtindrem també el vector director del raig:

$$\vec{v} = vrp - observador$$

A partir d'aquí, es procedeix de la mateixa forma que a la versió no optimitzada.

3.3 Anàlisi de les dades parametritzables

Les diferents aproximacions de Ray Casting de volum a implementar en aquest projecte, permeten la parametrització d'algunes dades amb les quals l'usuari pot obtenir diferents resultats. Es poden dividir aquestes dades en dos grups:

- Paràmetres del Ray Casting de volum.
- Paràmetres de la càmera.

Paràmetres del Ray Casting de volum

Les diferents aproximacions de Ray Casting de volum en CPU i GPU comparteixen una sèrie de dades parametritzables amb les quals l'usuari pot obtenir diferents resultats a la imatge final. Aquestes dades corresponen als llindars d'opacitat i threshold i al valor de d'increment de mostreig.

Amb el llindar d'opacitat establert per l'usuari, obtindrem una imatge amb tendència a mostrar les capes exteriors del volum amb transparència, deixant d'aquesta manera que es pugui visualitzar la part interna del volum.

El llindar de threshold ens permet segmentar el volum. Bàsicament, es mostraran a la imatge final, aquelles parts del volum o vòxels, que tenen un valor de densitat superior al del llindar de threshold establert per l'usuari.

El valor d'increment de mostreig, el qual pot ser establert per l'usuari, ens permetrà determinar si el procés de mostreig de les aproximacions de Ray Casting de Volum en CPU i GPU, es realitzarà agafant més o menys mostres al llarg del raig. D'aquesta manera l'usuari podrà obtenir imatges més precises a costa d'augmentar el temps d'execució.

Paràmetres de la càmera

En aquest projecte s'implementa una càmera la qual permet la visualització del volum i la interacció volum-càmera. L'usuari podrà rotar la càmera, de manera que visualitzarà el volum en diferents angles, moure la càmera (*panning)* o fer zoom, per a visualitzar el volum més a prop o mes lluny.

Els paràmetres que permeten aquest tipus d'interacció corresponen a els angles de rotació del sistema de coordenades de la càmera i la finestra de visualització de la càmera. L'usuari no en serà conscient de les actualitzacions d'aquests paràmetres.

3.4 Casos d'ús de l'aplicació

En aquest projecte s'implementarà un sistema de visualització, sota l'entorn Qt [10] el qual ens permet crear aplicacions amb C++, OpenGL i GLSL. Aquest sistema ha de poder validar les diferents aproximacions de Ray Casting de volum en CPU i GPU. Cal recordar que en aquest projecte s'ha introduït alguna classe d'un projecte anterior el qual permetia la visualització de volums en z-buffering, de manera que no s'especificaran els casos d'us referents a la funcionalitat inicial del sistema, com per exemple, la obertura d'un fitxer que conté les dades d'un volum.

Per tant, els casos d'us que es detallaran a continuació seran els següents:

1. Permetre a l'usuari visualitzar l'aproximació de Ray Casting de volum a la CPU sobre un volum.
2. Permetre a l'usuari visualitzar l'aproximació de Ray Casting de volum a la GPU sobre un volum.
3. Permetre a l'usuari interaccionar amb el volum en temps real.
4. Permetre a l'usuari canviar el mode de il·luminació amb el que renderitzar el volum.
5. Permetre a l'usuari seleccionar entre l'aproximació en GPU normal, o la optimitzada (One Step).

3.4.1 Cas d'ús 1 (UC1)

Permetre a l'usuari visualitzar l'aproximació de Ray Casting de volum a la CPU sobre un volum.

Actor Principal: Usuari.

Personal involucrat i interessos: L'usuari vol visualitzar un volum concret amb l'aproximació del Ray Casting de volum en CPU.

Pre condicions: L'usuari ha obert mitjançant el menú de fitxer, el volum a renderitzar donant els valors d'alçada, amplada, profunditat i tipus de dades de les mostres correctament.

Escenari principal:

1. L'usuari sel·lecciona la opció RayCasting de la barra de menús.
2. L'usuari sel·lecciona la opció CPU.
3. El sistema mostra la visualització del món de vòxels complet calculada en CPU.

Post condicions: El resultat final dependrà dels valors de la càmera, llindars d'opacitat i threshold i el valor de mostreig del raig.

3.4.2 Cas d'ús 2 (UC2)

Permetre a l'usuari visualitzar l'aproximació de Ray Casting de volum a la GPU sobre un volum.

Actor Principal: Usuari.

Personal involucrat i interessos: L'usuari vol visualitzar un volum concret amb l'aproximació del Ray Casting de volum en GPU.

Pre condicions: L'usuari ha obert mitjançant el menú de fitxer, el volum a renderitzar donant els valors d'alçada, amplada, profunditat i tipus de dades de les mostres correctament.

Escenari principal:

1. L'usuari sel·lecciona la opció RayCasting de la barra de menús.
2. L'usuari sel·lecciona la opció GPU.
3. El sistema mostra la visualització del món de vòxels complet calculada en GPU.

Post condicions: La visualització en GPU queda activada permanentment.

3.4.3 Cas d'ús 3 (UC3)

Permetre a l'usuari interaccionar amb el volum en temps real.

Actor Principal: Usuari.

Personal involucrat i interessos: L'usuari vol aplicar thresholding al volum o variar les opacitats del volum en temps real.

Pre condicions: L'usuari està visualitzant el volum amb l'aproximació de Ray Casting de volum en GPU.

Escenari principal:

1. L'usuari selecciona diferents valors als lliscadors de Threshold, Opacitat i Valor de mostreig del raig.
2. El sistema mostra en el mateix widget la visualització del món de vòxels amb els nous valors dels lliscadors.
3. L'usuari rota, mou o fa zoom al volum.

4. El sistema mostra en el mateix widget la visualització del món de vòxels amb els mateixos valors dels lliscadors però amb els nous paràmetres de la càmera.

Post condicions: -

3.4.4 Cas d'ús 4 (UC4)

Permetre a l'usuari canviar el mode de il·luminació amb el que renderitzar el volum.

Actor Principal: Usuari.

Personal involucrat i interessos: L'usuari vol visualitzar els diferents modes de color que es poden aplicar al volum.

Pre condicions: L'usuari està visualitzant el volum amb l'aproximació de Ray Casting de volum en GPU.

Escenari principal:

1. L'usuari inicialment visualitza el volum en el mode de color Gray Scale.
2. L'usuari selecciona el mode de color LUT.
3. El sistema mostra el volum amb la taula de colors inicial.
4. L'usuari canvia la taula de colors situada a baix a l'esquerra.
5. El sistema mostra com els colors que selecciona l'usuari, es van assignant al volum segons el valor de densitat dels vòxels.

Extensions:

4.1 L'usuari selecciona el mode de color Phong Blinn.
4.2 El sistema mostra el mode de color Phong Blinn amb la mateixa taula de colors que al mode anterior.
4.3 L'usuari selecciona el mode de color MIP.
4.4 El sistema mostra el mode de color MIP aplicat al volum.

Post condicions: El sistema deixa activa la darrera visualització seleccionada.

3.4.5 Cas d'ús 5 (UC5)

Permetre a l'usuari seleccionar entre l'aproximació en GPU normal o la optimitzada.

Actor Principal: Usuari.

Personal involucrat i interessos: L'usuari vol visualitzar la aproximació optimitzada del Ray Casting de volum en GPU.

Pre condicions: L'usuari està visualitzant el volum amb l'aproximació no optimitzada de Ray Casting de volum en GPU.

Escenari principal:

1. L'usuari clicka el check de mode de Ray Casting One Step.
2. El sistema mostra al widget corresponent el volum amb l'aproximació optimitzada del Ray Casting de volum en GPU.

Post condicions: El sistema deixa actiu el mode optimitzat per el Ray Casting de volum en GPU.

3.5 Model de Domini de l'aplicació

Es presenta el següent model de domini per a l'aplicació:

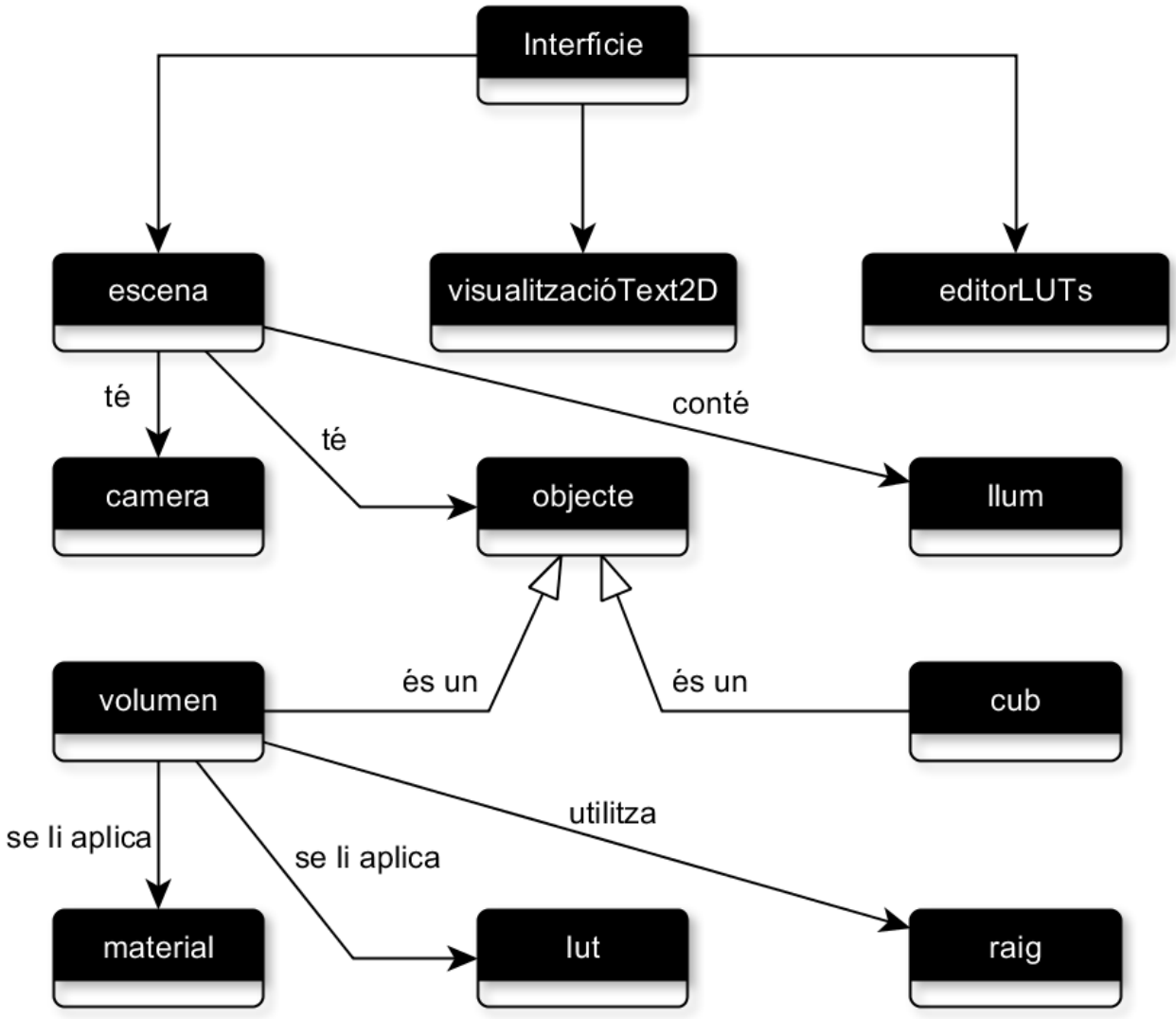


Fig. 16 - Model de Domini de l'aplicació

Detall de les entitats:

Interfície: És la entitat que correspon a la interfície gràfica, en ella es crearan tots els widgets necessaris per a la visualització i la interactivitat amb l'usuari sobre els volums.

escena: Entitat controladora del renderitzat de la aproximació de Ray Casting de volum en GPU i CPU, de les rotacions, panning o zoom de la càmera i de la llum de l'escena.

visualitzacióText2D: Entitat controladora de la visualització en textura 2D.

EditorLUTs: Entitat controladora de les taules de color.

Camera: Entitat que representa una càmera. Contindrà els atributs i mètodes necessaris per a calcular la càmera que projectarà sobre els volums.

Objecte: Entitat abstracta la qual representa qualsevol tipus de volum.

Llum: És a entitat que representarà la llum de l'escena per a certs mètodes d'interpolació de color. Contindrà els atributs i paràmetres necessaris per a simular el comportament de la llum puntual.

Volumen: Entitat filla d'objecte. Representa el món de vòxels. Contindrà els atributs i mètodes necessaris per a la creació de volums representables en l'aplicació.

Cub: Aquesta entitat serà utilitzada per a enviar a la GPU les coordenades màximes del món de vòxels, de manera que puguem calcular els raigs a la GPU.

Material: Entitat que representarà el material del volum. Contindrà els atributs i mètodes necessaris per tal de poder simular el material d'un objecte real el qual reflecteix la llum.

Lut: Entitat que representa una look up table. Contindrà els atributs i mètodes necessaris per a poder guardar taules de colors.

Raig: Entitat que representa un Raig. Utilitzada per a l'aproximació del Ray Casting en CPU. Contindrà els atributs i mètodes necessaris per tal de calcular els raigs.

3.6 Conclusió

Després d'analitzar les diferents aproximacions de Ray Casting de volum en CPU i GPU, som capaços de distingir les diferències i similituds existents entre cadascuna d'elles. A més, es pretén millorar els speed ups respecte la CPU amb la versió optimitzada de Ray Casting de volum en GPU.

També hem analitzat els paràmetres que compartiran aquestes aproximacions per tal d'obtenir els mateixos resultats a l'hora de comparar-los visualment. Els mateixos paràmetres ens permetran donar a l'usuari interacció en temps real.

S'han analitzat els cinc casos d'ús més rellevants per a l'aplicació, aquests casos d'us cobreixen totes les interaccions interfície-usuari que afecten a la visualització i mostra de resultats.

Finalment, el model de domini conté les entitats més importants del domini necessàries per al context conceptual de l'aplicació.

4 Disseny

En aquest capítol es mostrarà el disseny de l'aplicació mitjançant els diagrames de seqüència i el diagrama de classes de l'aplicació.

Per tant el capítol es dividirà en els següents apartats:

4.1 Diagrama de classes de l'aplicació.

4.2 Diagrames de seqüència.

4.3 Conclusió.

4.1 Diagrama de classes de l'aplicació

El diagrama de classes de l'aplicació contindrà les mateixes entitats que el model de domini, però en aquest cas, amb els seus corresponents mètodes i atributs, de tal manera que s'exposarà a continuació el diagrama de classes i aquestes seran detallades per separat.

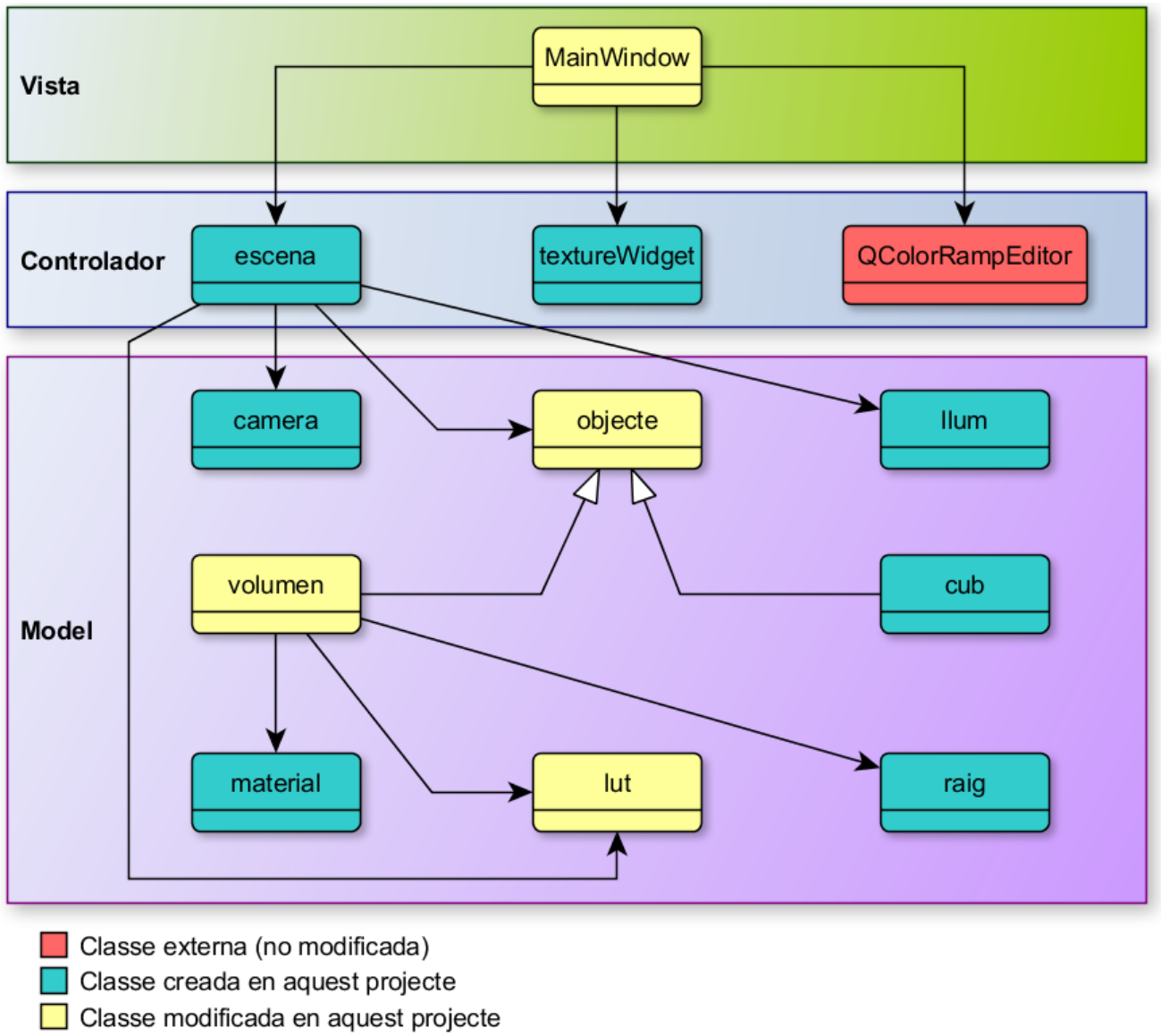


Fig. 17 - Diagrama de Classes de l'aplicació

Les classes en color vermell, són classes utilitzades com a recurs extern.

Les classes de color blau, són classes que seran creades per a aquest projecte.

Les classes de color groc, són classes de l'anterior projecte, les quals han estat modificades en aquest projecte.

La aplicació seguirà el patró de disseny MVC (Model Vista Controlador). Aquest patró es basa en separar les dades de l'aplicació, la interfície gràfica i la lògica de l'aplicació.

El Model és la representació específica de la informació amb la qual el sistema opera.

La Vista s'encarrega de presentar el Model en un format adequat per a interactuar amb l'usuari.

El Controlador respon als events o accions de l'usuari.

A continuació es detallaran les classes de manera individual.

Fig. 18 - Detall de la classe MainWindow

Es tracta de la única classe que forma part de la vista, és a dir, és la classe encarregada de generar i controlar la interfície gràfica de l'aplicació la qual servirà per a mostrar els resultats i interaccionar amb l'usuari.

Com es pot observar, conté tota una sèrie de components d'interfície, com són els *sliders* (lliscadors), les *labels* (etiquetes), entre d'altres. També conté constants per a referenciar quin serà el valor associat a algun mode de color i valors relacionats amb les aproximacions de Ray Casting de volum els quals seran enviats als controladors.

Tal i com s'aprecia tots els atributs són de visibilitat privada, de la mateixa manera que els mètodes a excepció del constructor.

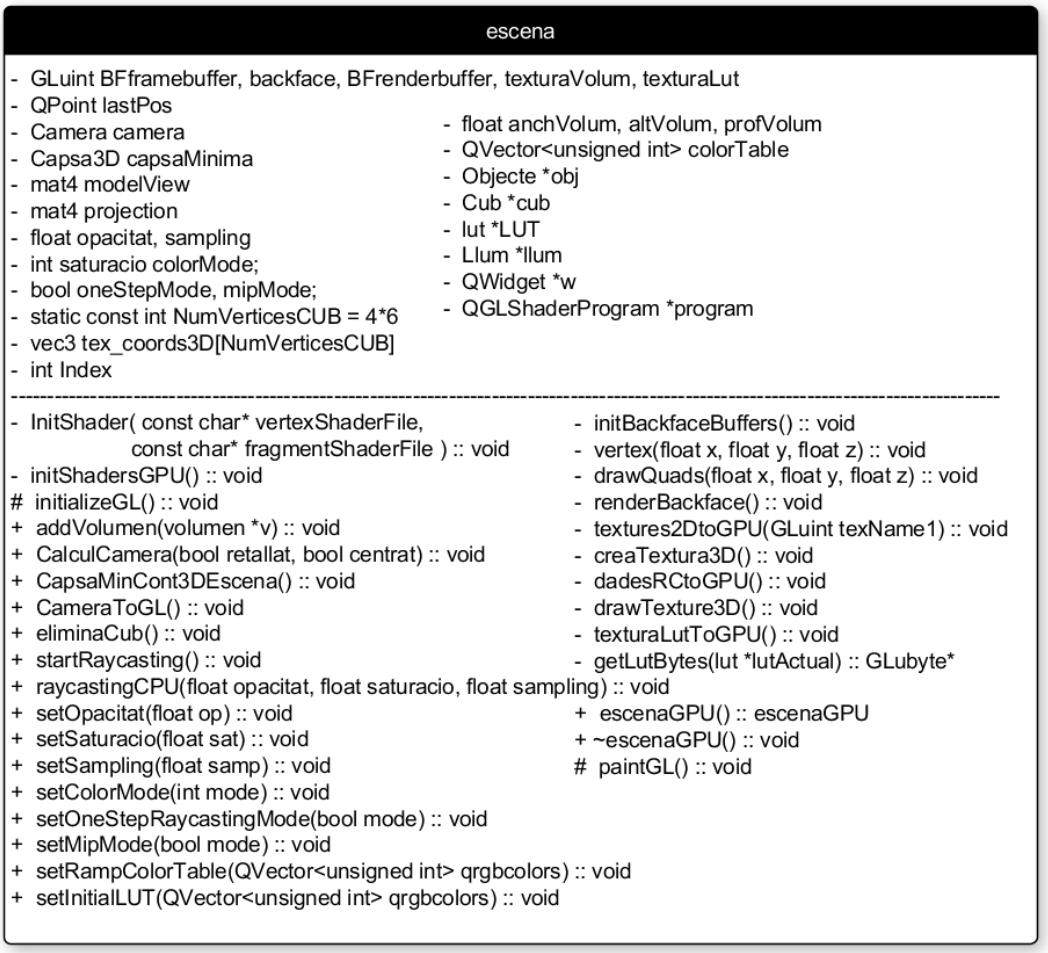


Fig. 19 - Detall de la classe escena

Es tracta de la classe controladora de l'escena que conté el volum renderitzat amb les aproximacions de Ray Casting de Volum en CPU i GPU. Conté els atributs i mètodes necessaris per a visualitzar una escena, a més dels programes de la GPU (*shaders*), els quals interpolaran el color de l'escena que es renderitza.

Conté a més, una sèrie de mètodes públics, els quals seran cridats des de la classe que la contingui. Aquests mètodes serviran per a actualitzar els valors que afectaran al resultat final del Ray Casting de volum en CPU i en GPU.

L'escena que representa aquesta classe, conté una càmera i un volum, els quals són compartits per les dues aproximacions de Ray Casting de volum en CPU i GPU, de tal manera que podem visualitzar i comparar els dos resultats amb els mateixos paràmetres.

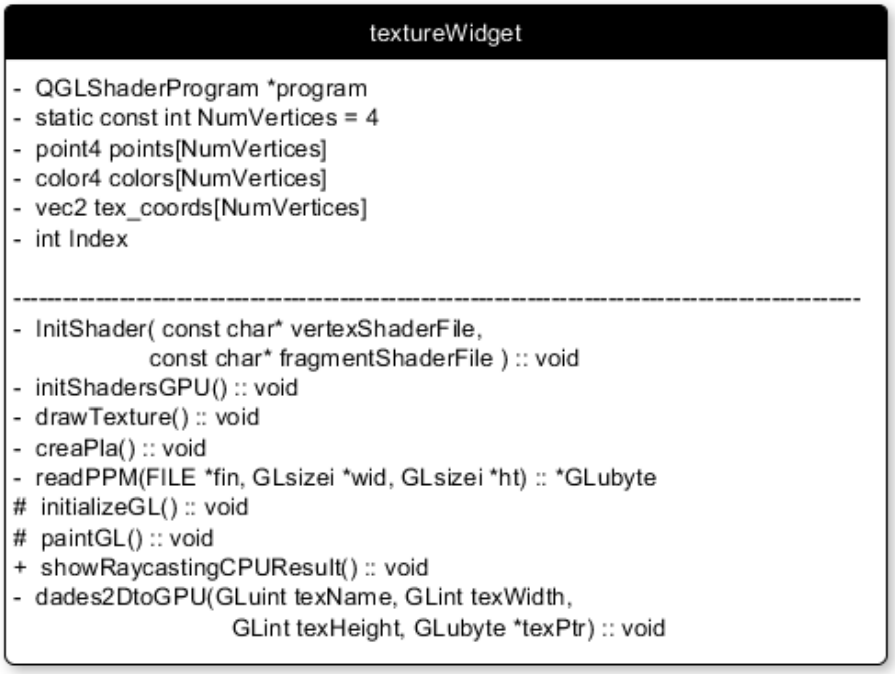


Fig. 20 - Detall de la classe textureWidget

Classe controladora encarregada de la visualització del resultat de l'aproximació del Ray Casting de volum en CPU. Conté els atributs i mètodes necessaris per tal de poder llegir la imatge en format ppm [2], la qual conté el resultat del Ray Casting de volum en CPU, i generar un pla, on es mapejarà aquest resultat com a textura.

També conté programes de GPU (*shaders*), ja que la visualització final, el pla amb la textura de resultat mapejada, es realitza en GPU.

Aquesta classe no conté cap volum i cap càmera, es limita a llegir d'una imatge el resultat del Ray Casting de volum en CPU i finalment, permetre la seva visualització.

A continuació es mostrarà el detall de les classes pertanyents al model de l'aplicació. Aquestes classes contindran els atributs i mètodes necessaris per tal d'instanciar a l'aplicació allò que pretenen representar.

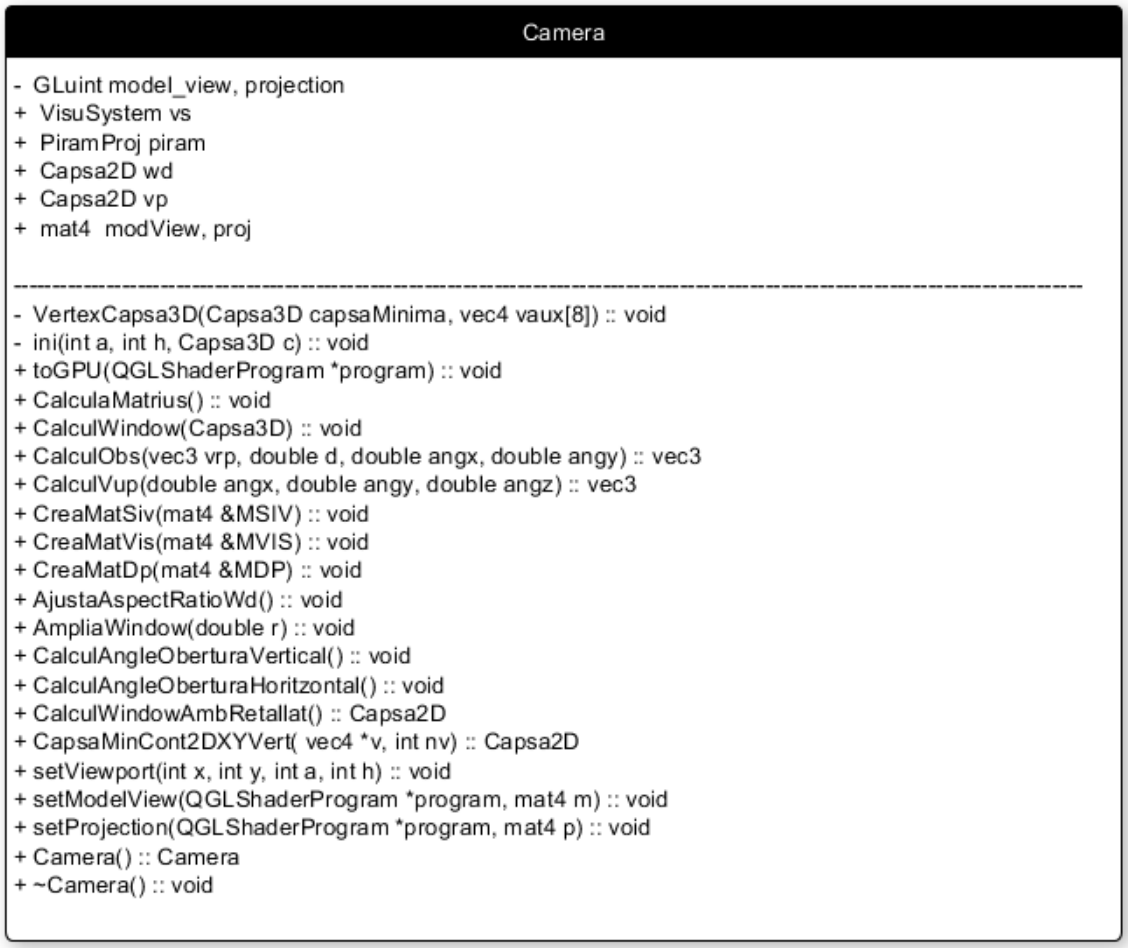


Fig. 21 - Detall de la classe Camera

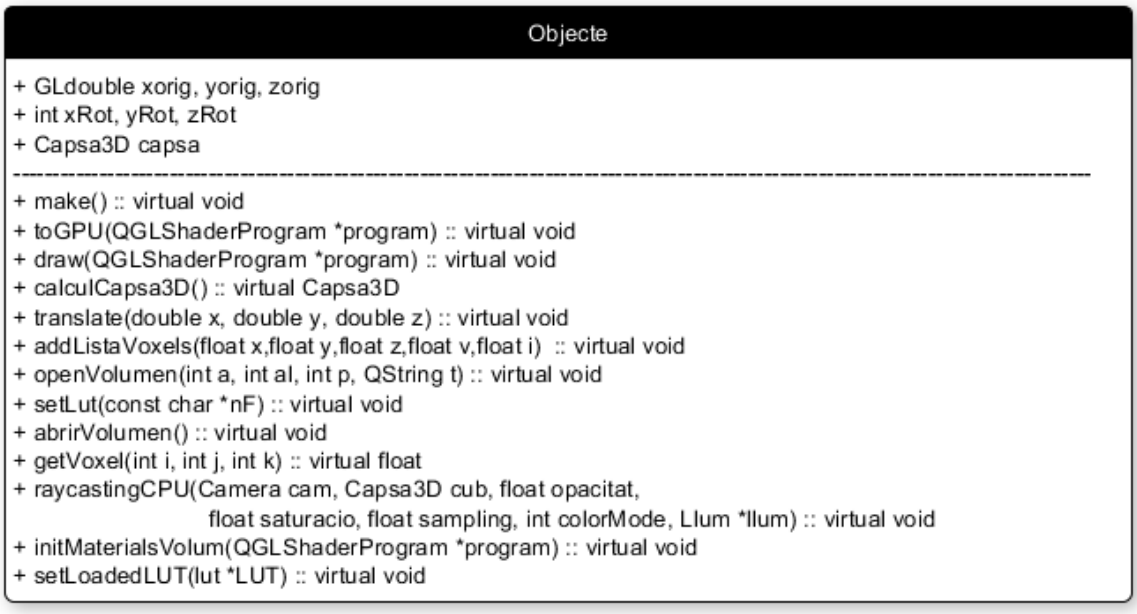


Fig. 22 - Detall de la classe Objecte

A continuació es mostraran les dues classes que hereten de la classe abstracta Objecte. Els atributs i mètodes heretats no es reflectiran degut a que ja han estat mostrats al diagrama anterior.

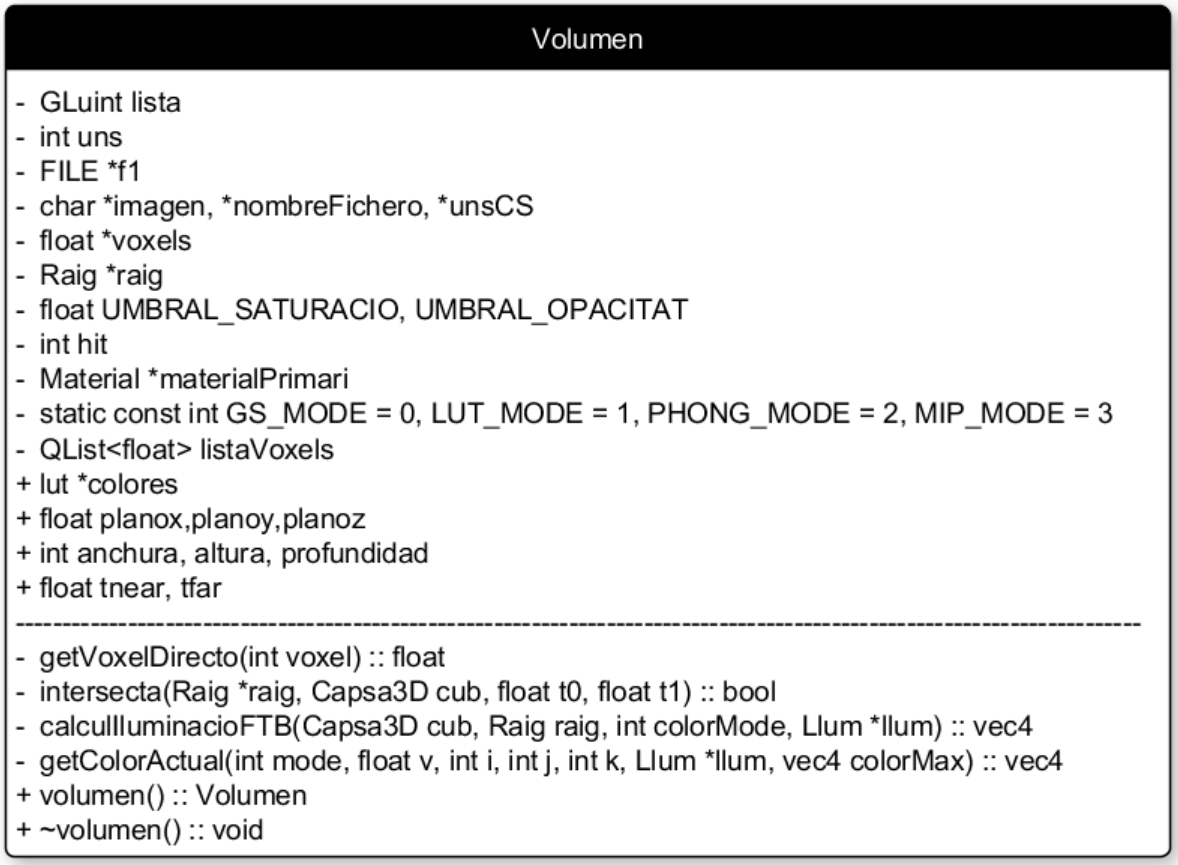


Fig. 23 - Detall de la classe Volumen

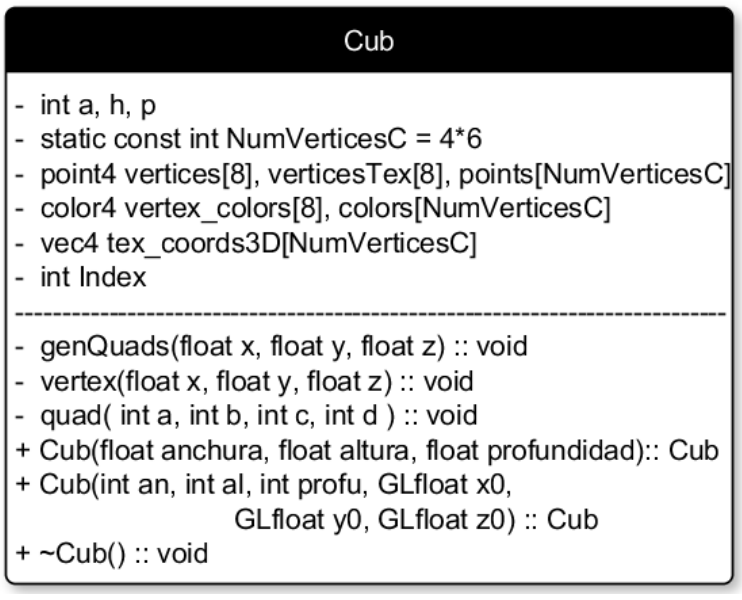


Fig. 24 - Detall de la classe Cub

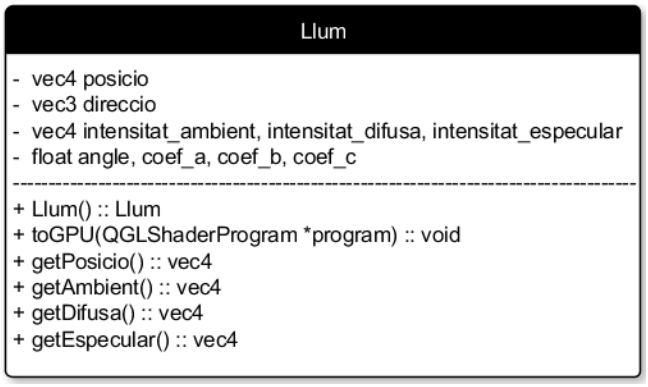


Fig. 25 - Detall de la classe Llum

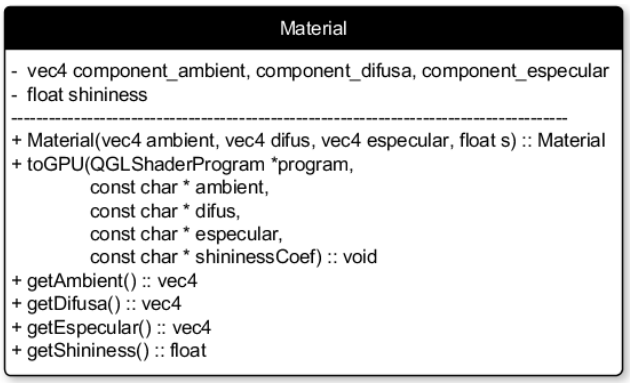


Fig. 26 - Detall de la classe Material

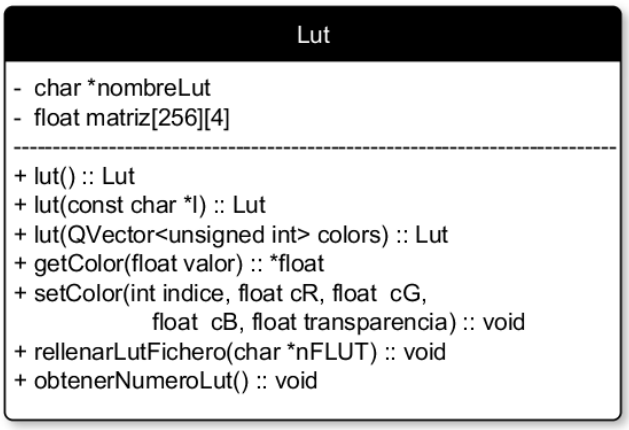


Fig. 27 - Detall de la classe Lut

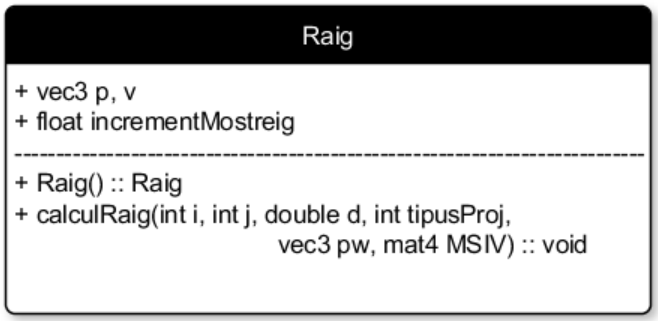


Fig. 28 - Detall de la classe Raig

4.2 Diagrames de seqüència

Els següents diagrames mostraran la interacció entre els objectes de l'aplicació. Aquests diagrames estaran relacionats amb els casos d'ús exposats al capítol anterior. De la mateixa manera que en el capítol anterior, les funcionalitats inicials referents a les del projecte anterior del qual parteix aquest, no es veuran reflectides en els següents diagrames. "*Per motius de llegibilitat, els diagrames de seqüencia es separaran per mòduls, a més, les implementacions d'algunes parts de l'aplicació són molt extenses, de manera que alguns dels diagrames es trobaran en versió simplificada amb el fi de fer-los més intel·ligibles*".

4.2.1 Diagrama de seqüència 1

Permetre a l'usuari visualitzar l'aproximació de Ray Casting de volum a la CPU sobre un volum.

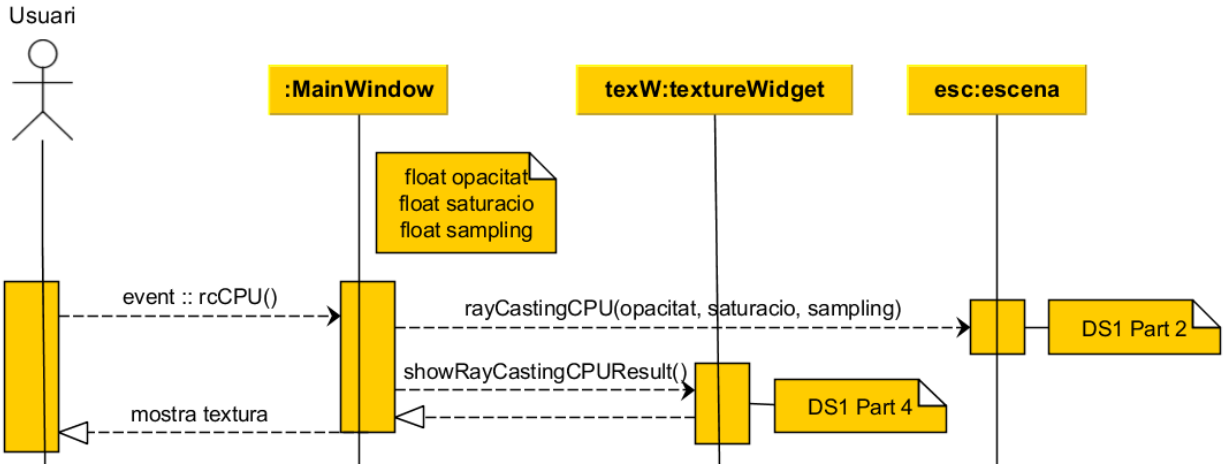


Fig. 29 - DS1 Part 1

Com podem observar, la implementació es separa en dues parts ben diferenciades. Una, la implementació de l'aproximació de Ray Casting de volum en CPU, i l'altra, la visualització dels resultats. Això és degut a que el Ray Casting de volum en CPU no permet la visualització en temps real i per tant, emmagatzemem el resultat en una imatge, la qual després és passa com a textura i es visualitza al widget corresponent.

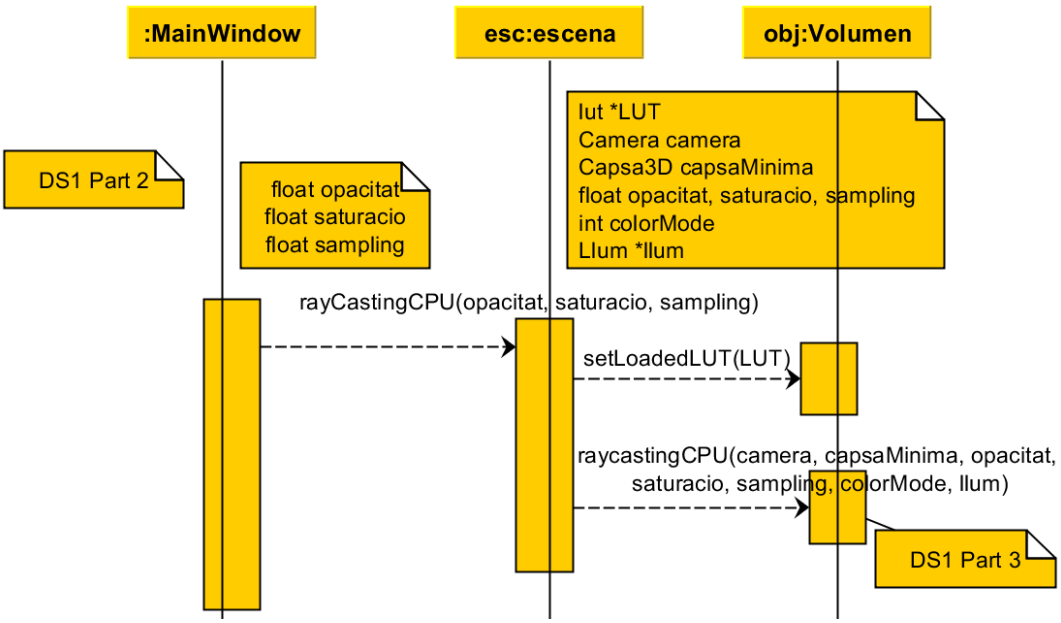


Fig. 30 - DS1 Part 2

En el diagrama DS2 Part 2, observem com es farà la crida que donarà pas al Ray Casting de volum en CPU i com es passen els paràmetres necessaris per a completar-ho.

A continuació es mostrarà el diagrama de seqüència simplificat que precedeix a l'anterior.

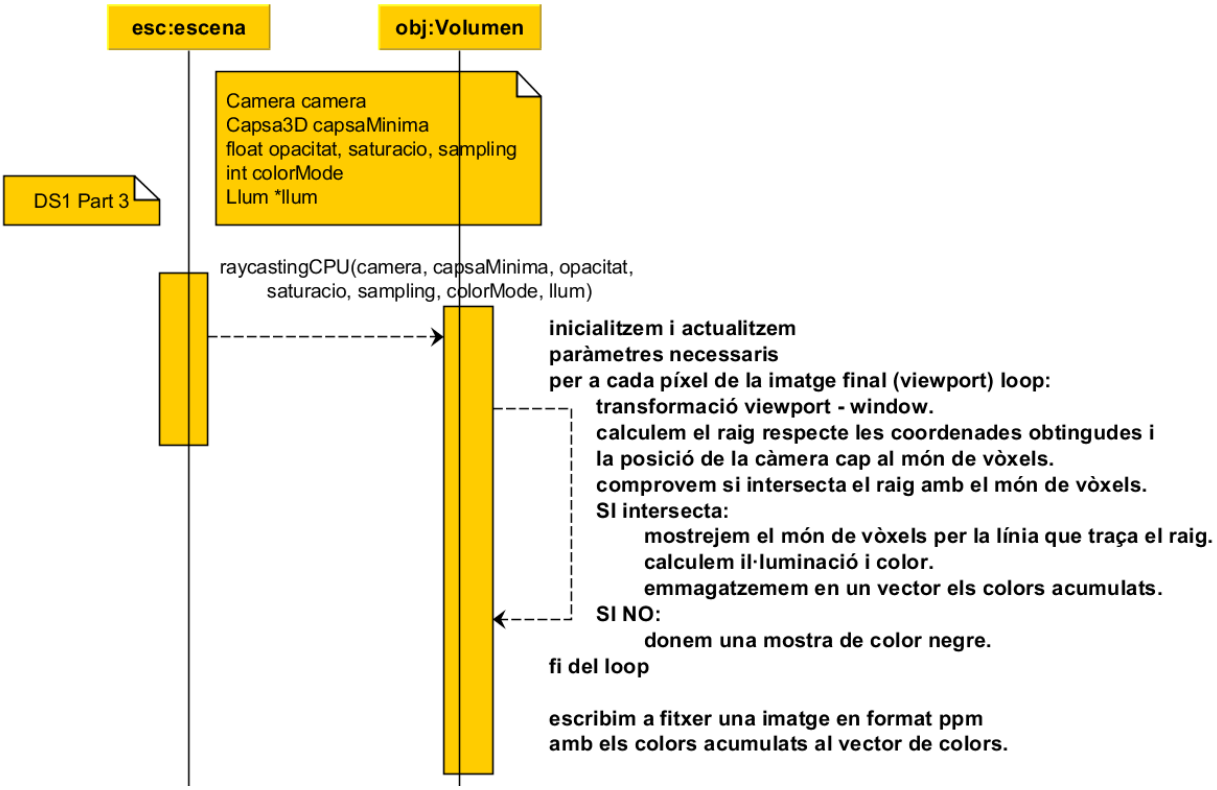


Fig. 31 - DS1 Part 3

Observem com el diagrama de seqüència DS2 Part 3 implementa l'aproximació del Ray Casting de Volum en CPU.

Un cop s'ha emmagatzemat el resultat a una imatge en format ppm, s'inicia el procés de visualització de la imatge al widget corresponent de l'aplicació.

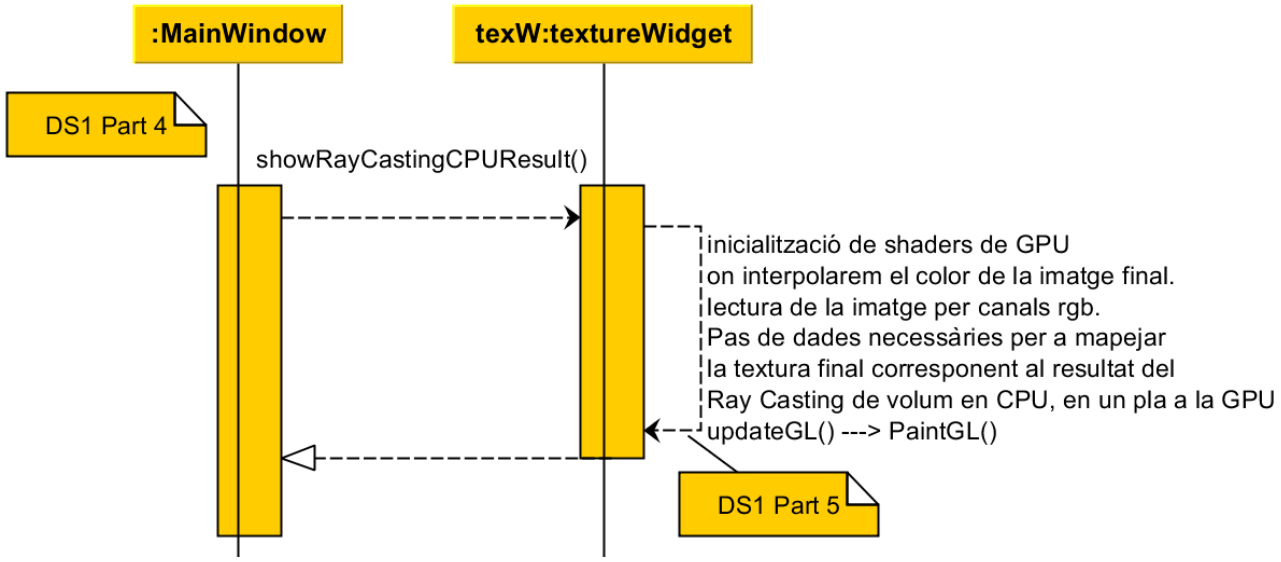


Fig. 32 - DS1 Part 4

Finalment es fa la crida al PaintGL() de manera que s'executaran els programes de la GPU, els shaders.

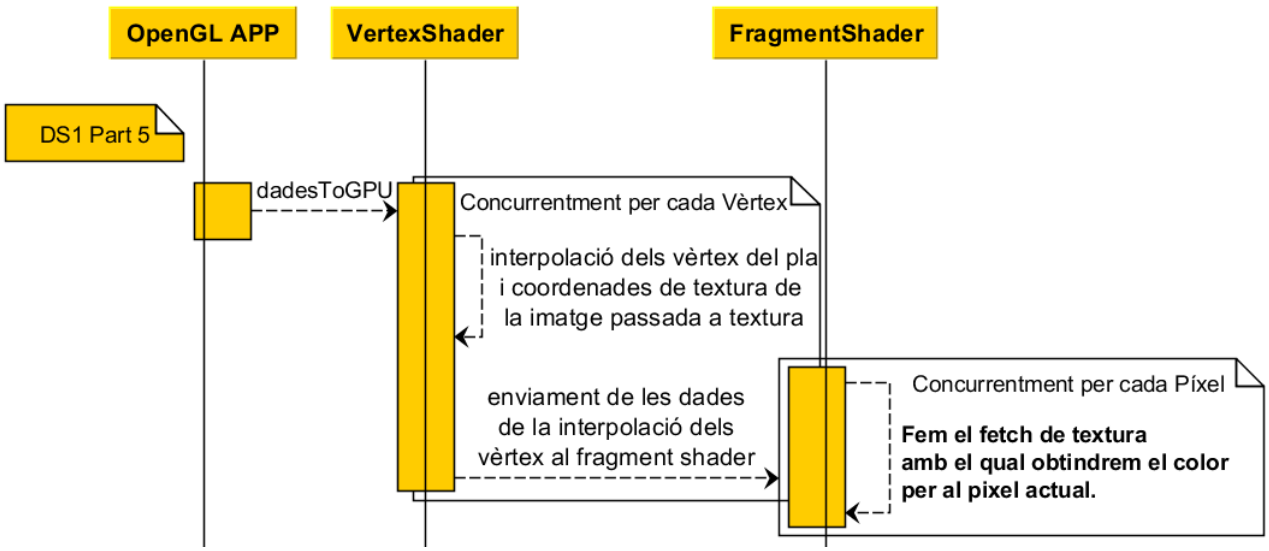


Fig. 33 - DS1 Part 5

4.2.2 Diagrama de seqüència 2

Permetre a l'usuari visualitzar l'aproximació de Ray Casting de volum a la GPU sobre un volum.

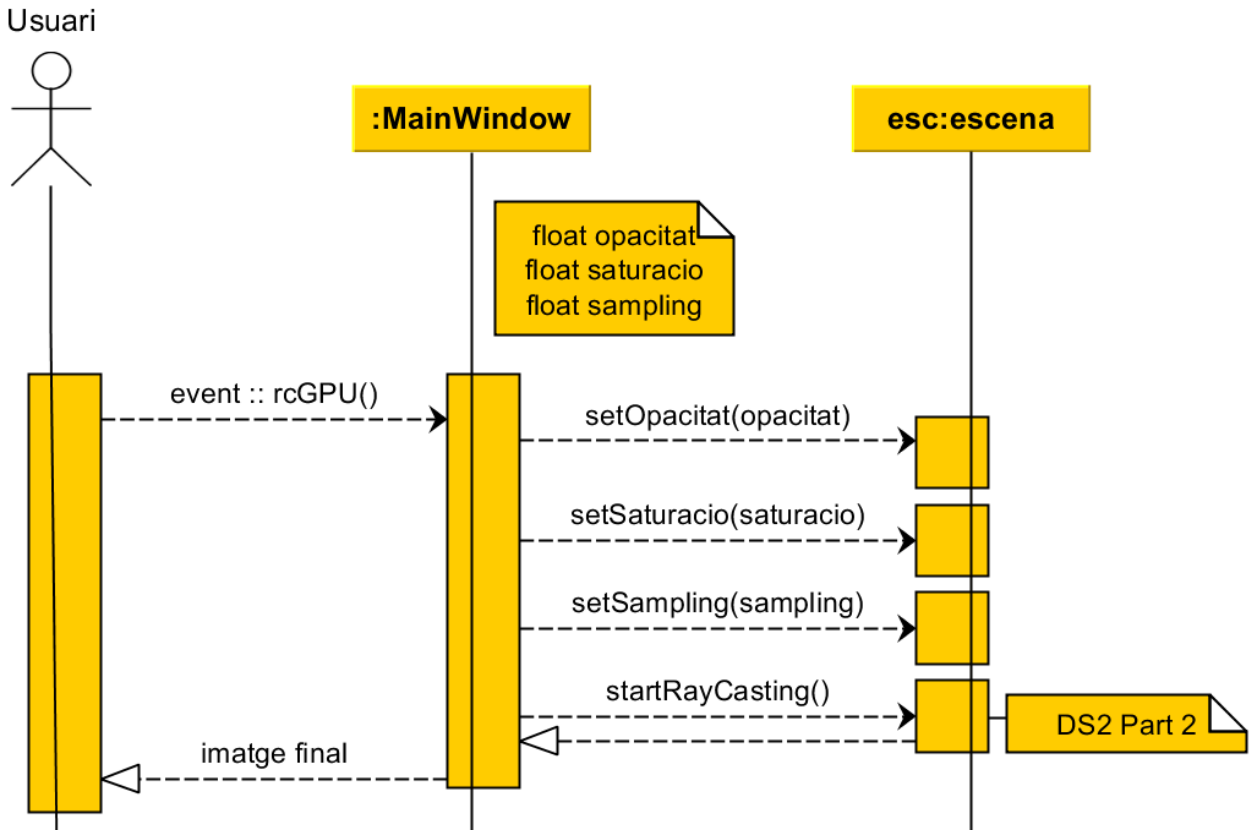


Fig. 34 - DS2 Part 1

Inicialment, l'usuari és qui crida mitjançant la interfície, a l'event que permet el Ray Casting de volum en GPU. Aquest event és escoltat pel listener i acte seguit es crida al mètode rcGPU(), el qual actualitza els valors d'opacitat, saturació i sampling a la classe escena i crida al mètode startRayCasting() de la classe escena.

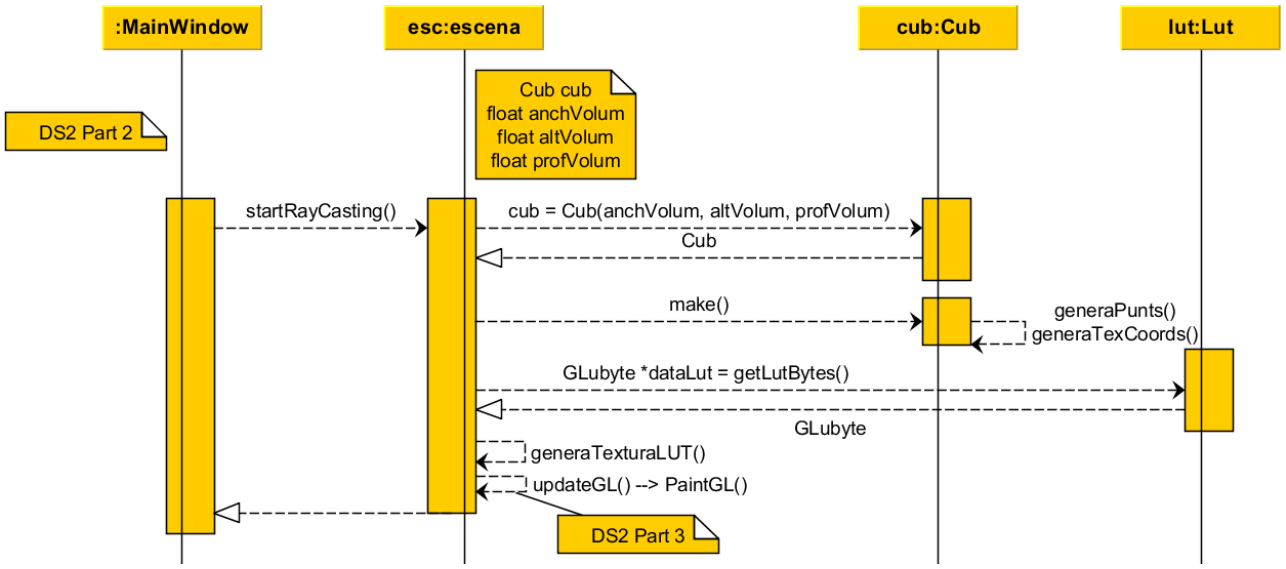


Fig. 35 - DS2 Part 2

Inicialment el genera el cub del qual aprofitarem els punts i les coordenades als shaders de la GPU. Generem la textura LUT per tal de mostrar els colors als modes de color que utilitzin LUT i seguidament actualitzem el widget amb updateGL, aquesta crida finalment cridarà a PaintGL.

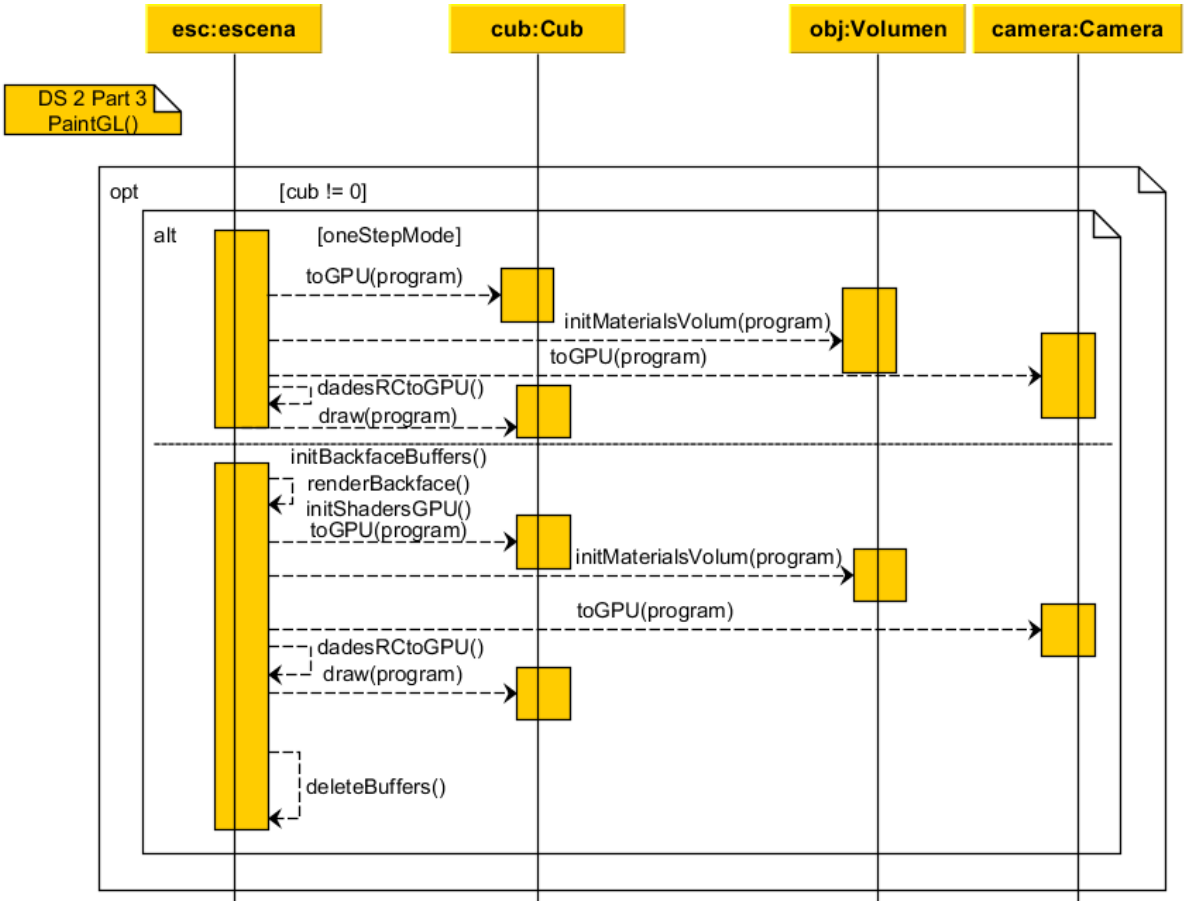


Fig. 36 - DS2 Part 3

El diagrama DS2 Part 3 pertany al mètode PaintGL el qual es crida cada cop que s'ha de pintar quelcom al widget de visualització. Segons el mode de Ray Casting de volum en GPU (optimitzat o no optimitzat), es cridaran a uns mètodes o a uns altres, però sempre amb un denominador comú, enviar les dades necessàries a la GPU per tal d'executar amb èxit els shaders que permetran la visualització de l'aproximació de Ray Casting de volum en GPU.

Finalment, es mostren els diagrames de seqüència simplificats dels shaders, que s'executen de manera concurrent a cadascun dels píxels projectats.

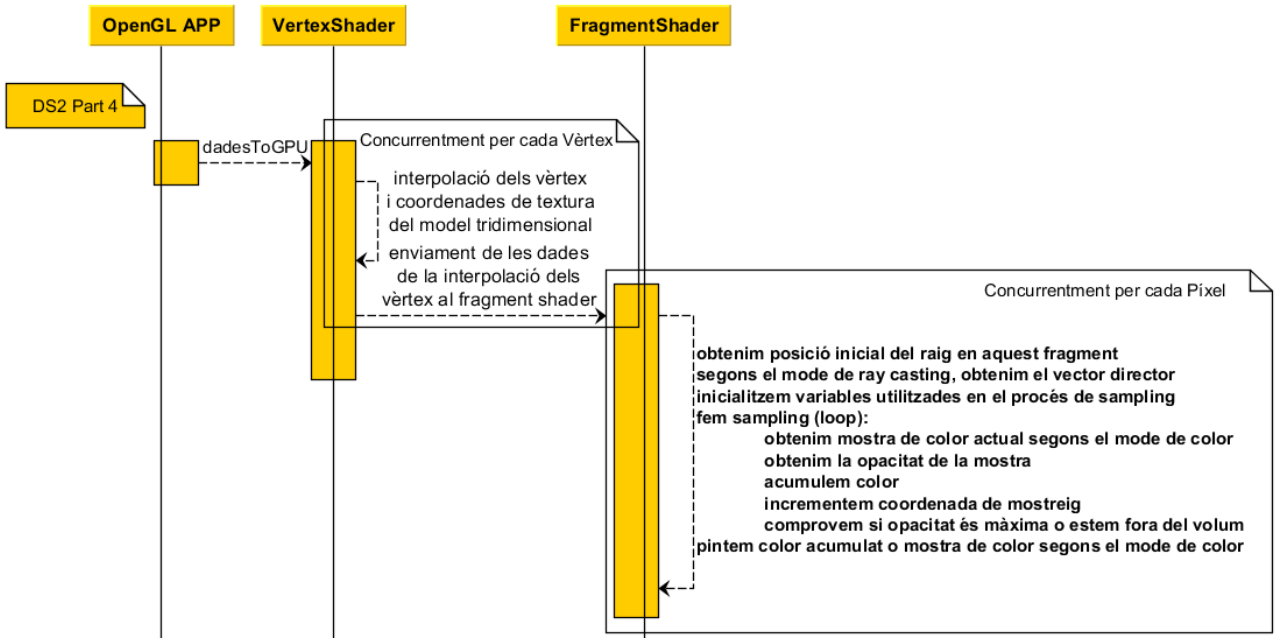


Fig. 37 - DS2 Part 4

Un cop finalitzada la seqüència, l'usuari visualitza el volum al widget corresponent amb l'aproximació de Ray Casting de volum en GPU.

4.2.3 Diagrama de seqüència 3

Permetre a l'usuari interaccionar
amb el volum en temps real.

La interacció usuari – aplicació en temps real, dependrà dels sliders o de la càmera, per tant podem separar en dos diagrames aquest tipus d'interaccions.

Per una banda, trobarem la interacció amb els sliders, els quals permetran visualitzar canvis en el volum en temps real. Hi hauran diferents sliders, però tots tindran un denominador comú, el flux d'execució serà el mateix, per tant només mostrarem els diagrames de seqüència d'un dels modes d'interacció en temps real amb sliders. D'altra banda, l'usuari podrà interaccionar amb el widget de visualització i rotar la càmera amb la finalitat de crear l'efecte de rotació del volum, moure el volum (panning de càmera) o fins i tot fer zoom.

Interacció amb lliscadors (*sliders*)

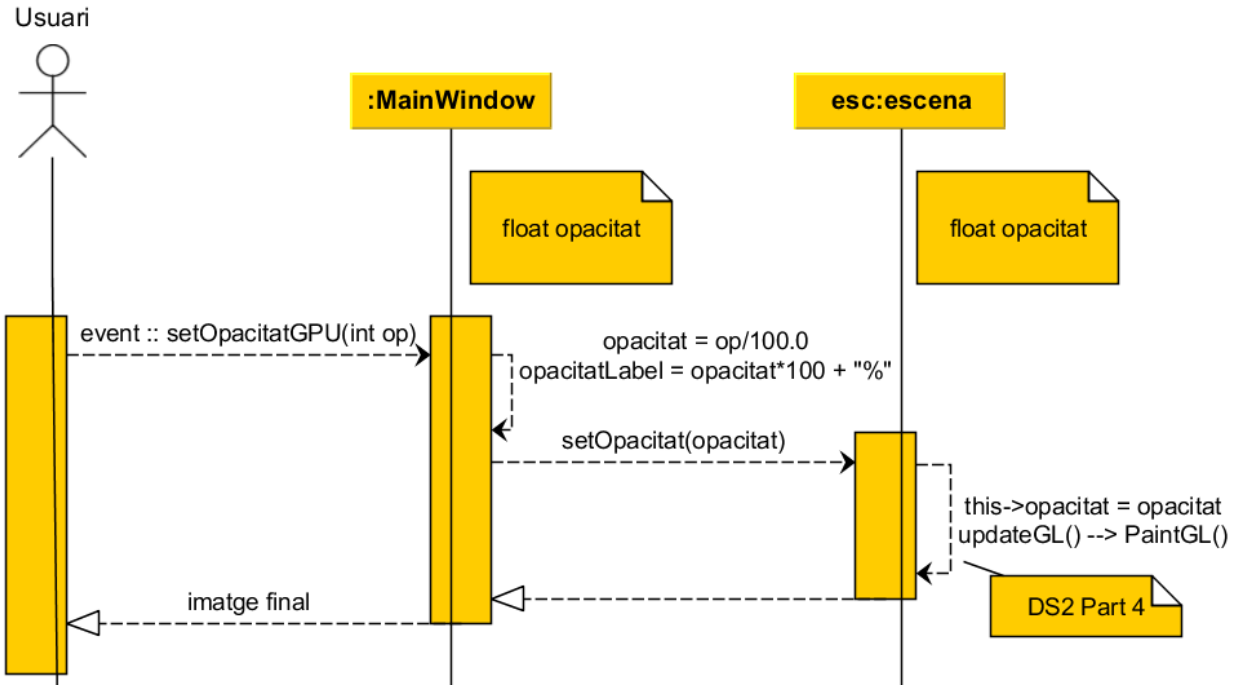


Fig. 38 - DS3

Interacció amb la càmera

Cada cop que l'usuari faci zoom, roti o mogui la càmera, es faran les crides a updateGL i PaintGL, de manera que tornarà a calcular-se l'aproximació de Ray Casting de volum en GPU. Tot i així, aquest tipus d'interacció no afecta a la tècnica, de manera que no es mostraran els diagrames de seqüència relacionats.

4.2.4 Diagrama de seqüència 4

Permetre a l'usuari canviar el mode de color
amb el que renderitzar el volum.

Tots els modes de color sel·leccionables seguiran el mateix flux d'execució, per tant només es mostrarà el diagrama de seqüència relacionat amb la sel·lecció del mode de color "*Phong Blinn".*

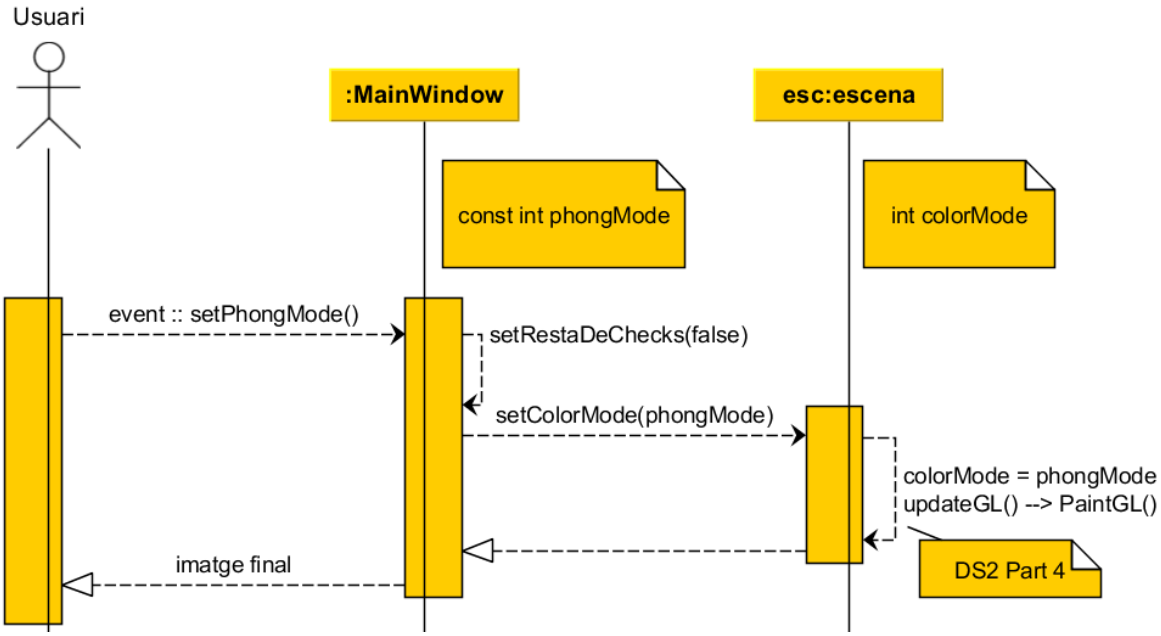


Fig. 39 - DS4

Com es pot observar, l'usuari interacciona amb l'aplicació seleccionant el botó Phong Blinn a la secció de modes de color, i aquest és passat al controlador de l'escena, de manera que els quan es cridi a PaintGL, aquest mode sigui passat a la GPU com mode actual de color i es calculin les noves mostres de color mitjançant el mode desitjat.

4.2.5 Diagrama de seqüència 5

Permetre a l'usuari seleccionar entre l'aproximació en GPU normal o la optimitzada.

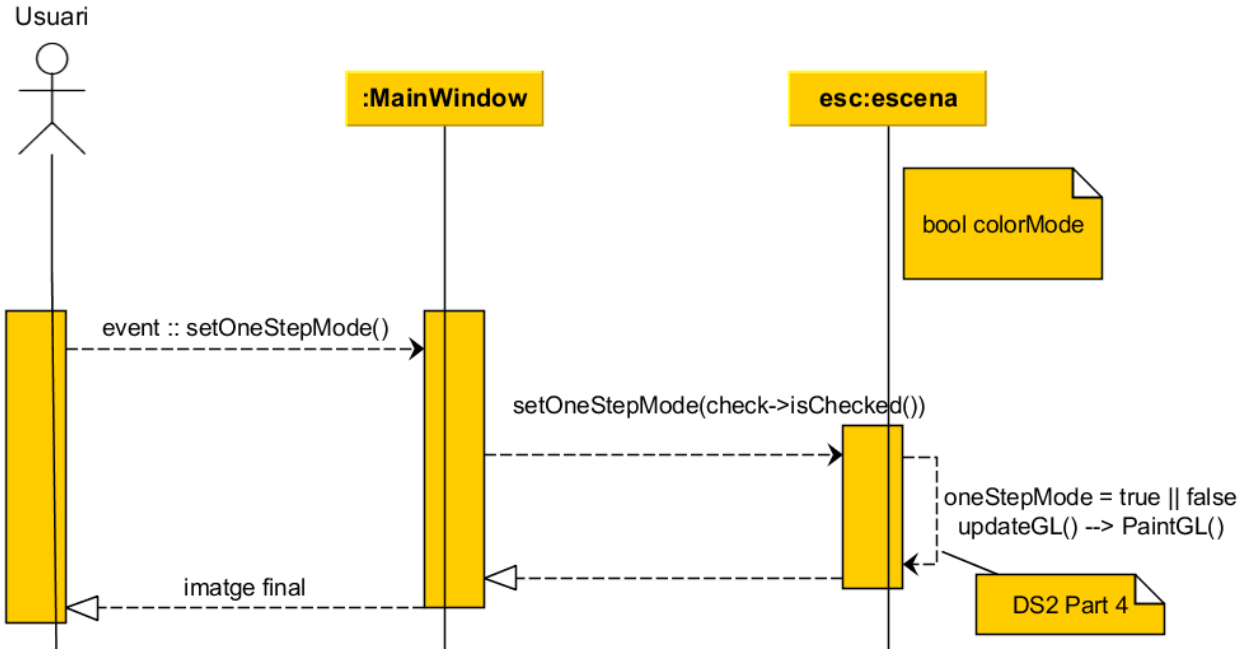


Fig. 40 - DS5

Segons l'estat del check (true o false), s'executarà la versió optimitzada o no optimitzada de l'aproximació de Ray Casting de volum en GPU.

4.6 Conclusió

En aquest capítol hem pogut observar el diagrama de classes, del qual s'ha detallat cada classe individualment. La aplicació seguirà el patró de disseny MVC (Model Vista Controlador), de manera que separarem les dades de l'aplicació, la interfície gràfica i la lògica de l'aplicació.

A més, s'han mostrat els diagrames de seqüència relacionats amb els casos d'ús analitzats al capítol anterior. Alguns diagrames s'han presentat en versió simplificada per tal de fer-los més intel·ligibles.

5 Simulacions

En aquest capítol es mostraran els resultats obtinguts amb les simulacions, després de la implementació del sistema de visualització el qual permet validar les diferents aproximacions de Ray Casting de volum en CPU i GPU. Alhora s'indicaran els temps d'execució de cadascuna de les aproximacions.

Les simulacions s'han realitzat sobre els models de vòxels següents:

- "*foot2.img*" que conté el peu d'un pacient, format per valors escalars de tipus "*unsigned char"* entre 0 i 255, amb dimensions 128x128x128 i 2MBytes.
- "*ncat_phantom_medium.img*" que conté el tors d'un pacient, format per valors escalars de tipus "*unsigned char"* entre 0 i 255, amb dimensions 400x400x400 i 62MBytes.

El computador emprat per a realitzar les simulacions ha estat un Acer 5755G equipat amb els següents components:

- Processador Intel Core i5 2430m @ 2.4~3.00GHz amb tecnologia TurboBoost, 3MB Cache.
- 4GB de memòria RAM DDR3.
- Targeta gràfica Nvidia Geforce GT 540m @ 1344MHz amb 96 CUDA Cores i 2GB de memòria vídeo RAM DDR3.

5.1 Simulacions amb el model 1

Primerament, es mostra el primer model tal qual després d'haver executat l'aproximació no optimitzada de Ray Casting de volum en GPU amb el mode de color GrayScale.

Seguidament, el mateix volum en la mateixa posició i mateix mode de color però amb l'aproximació de Ray Casting de volum en CPU.

Fig. 41 - Simulació del model 1 amb mode de color Gray scale (GPU i CPU)

Temps d'execució en GPU: 0.053s.

Temps d'execució en CPU: 1.878s.

Simulació de les mateixes característiques però canviant paràmetres de la càmera:

Fig. 42 - Simulació del model 1 amb mode de color Gray scale (GPU i CPU)

Temps d'execució en GPU: 0.048s.

Temps d'execució en CPU: 1.507s.

Mateixa simulació però ara amb l'aproximació optimitzada en GPU:

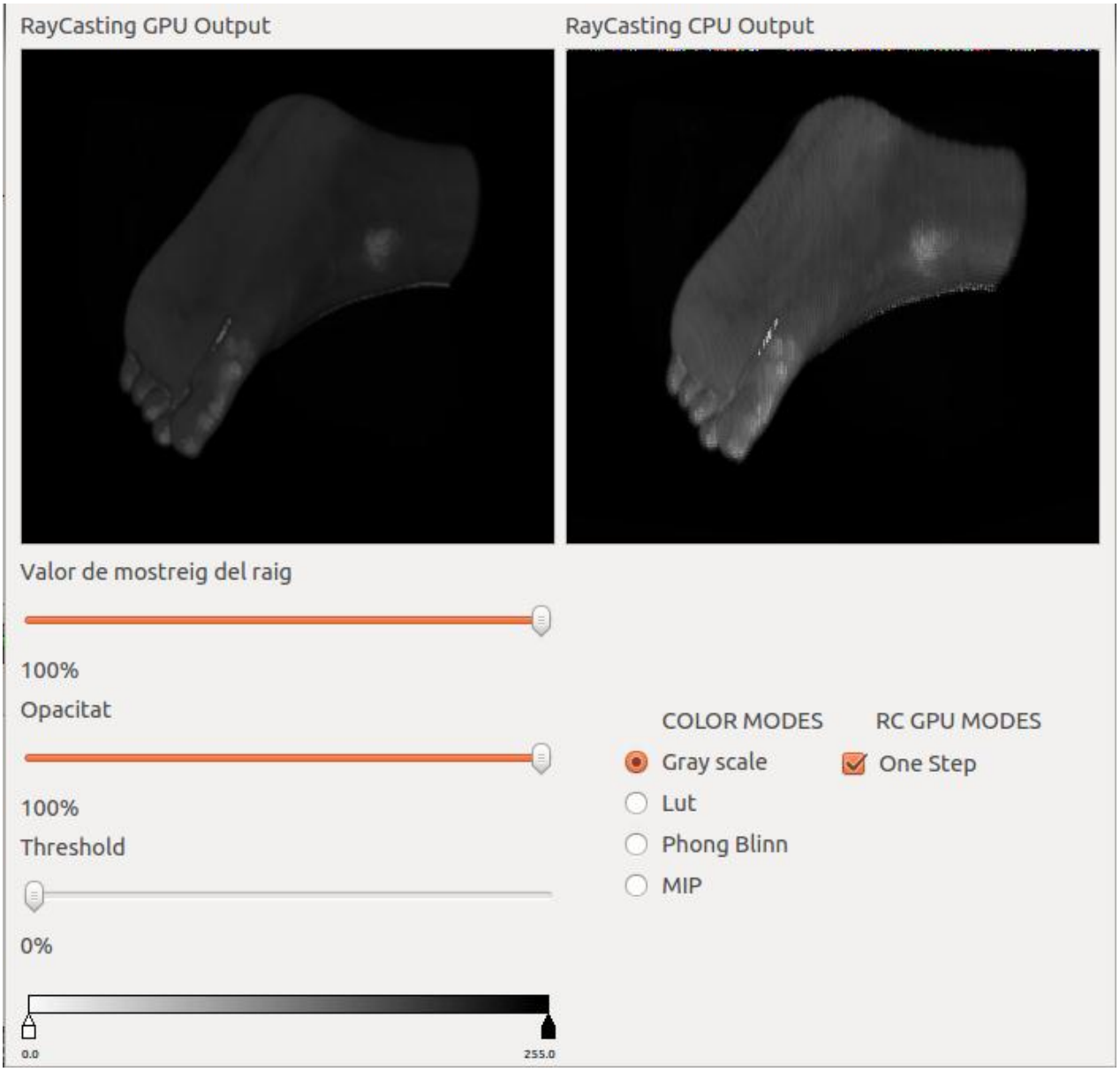


Fig. 43 - Simulació del model 1 amb mode de color Gray scale (GPU i CPU)

Temps d'execució en GPU optimitzat: 0.022s.

Temps d'execució en CPU: 1.509s.

Simulació amb la opacitat al 12%:

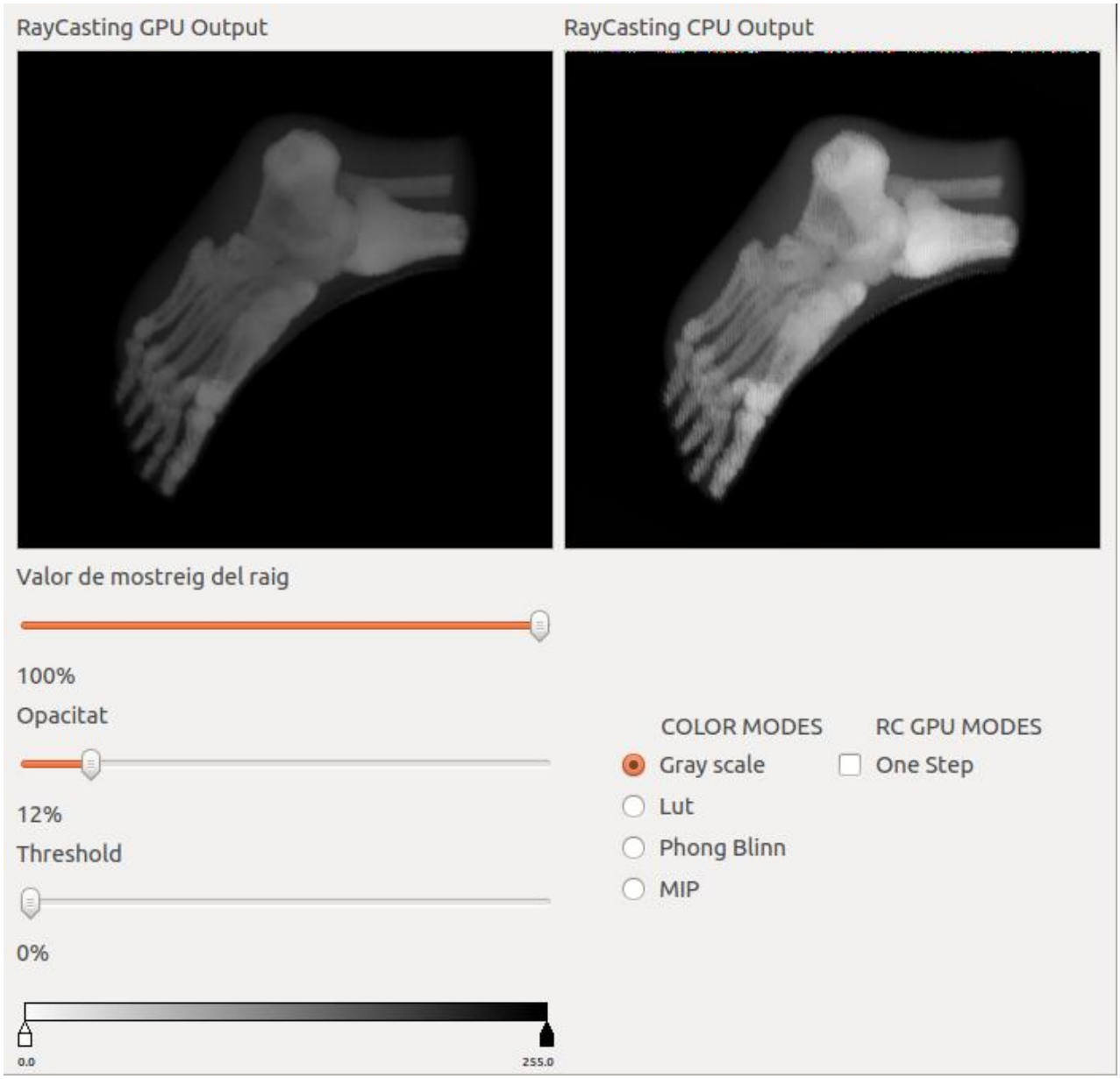


Fig. 44 - Simulació del model 1 amb mode de color Gray scale (GPU i CPU)

Temps d'execució en GPU: 0.041s.

Temps d'execució en CPU: 1.790s.

Mateixa simulació però ara, amb l'aproximació optimitzada en GPU:

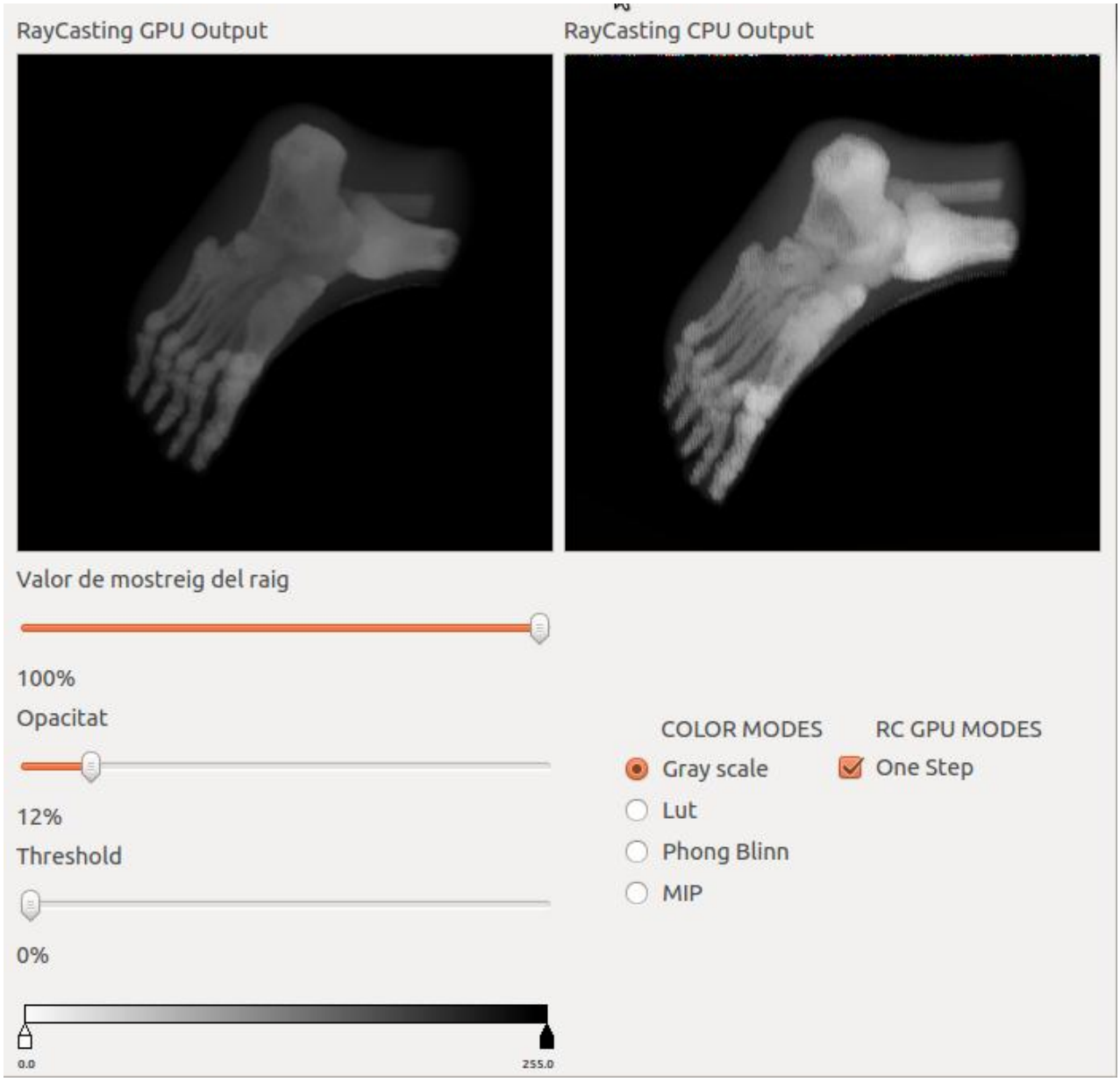


Fig. 45 - Simulació del model 1 amb mode de color Gray scale (GPU i CPU)

Temps d'execució en GPU optimitzat: 0.012s.

Temps d'execució en CPU: 1.790s.

Els efectes visuals són els mateixos, encara que a temps inferiors.

A continuació es mostraran resultats sobre el mateix model amb el mode de color LUT.

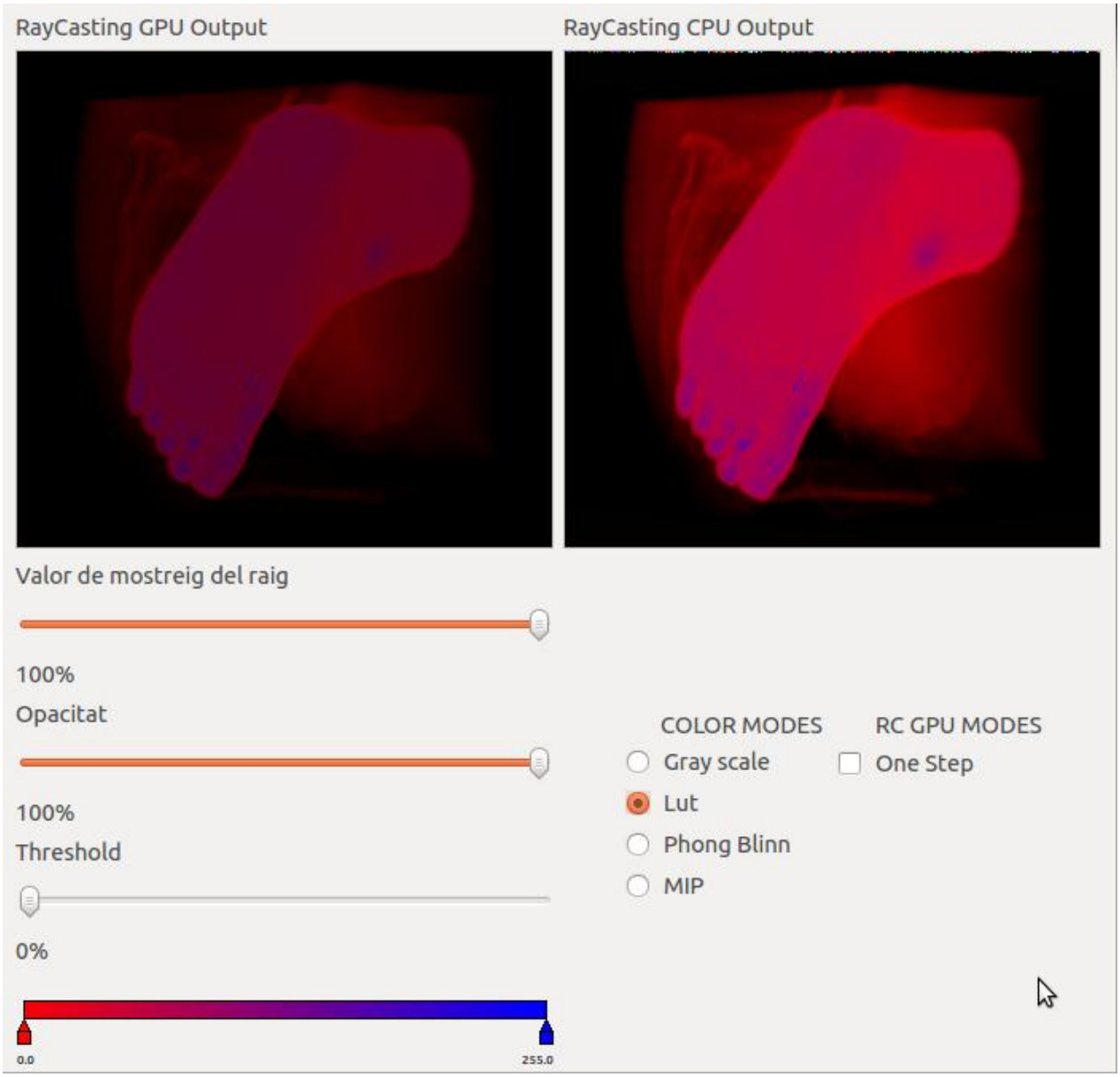


Fig. 46 - Simulació del model 1 amb mode de color LUT (GPU i CPU)

Temps d'execució en GPU: 0.152s.

Temps d'execució en CPU: 1.465s.

Com es pot observar, en aquest mode de color el model de vòxels adopta els mateixos colors que els de la taula de colors de la part inferior de la interfície, a més, els valors a sota els indicadors de colors, estan directament relacionats amb les densitats dels vòxels del model.

A la següent simulació, segmentarem el model mitjançant *Thresholding,* el qual serà possible gràcies al lliscador de *Threshold*.

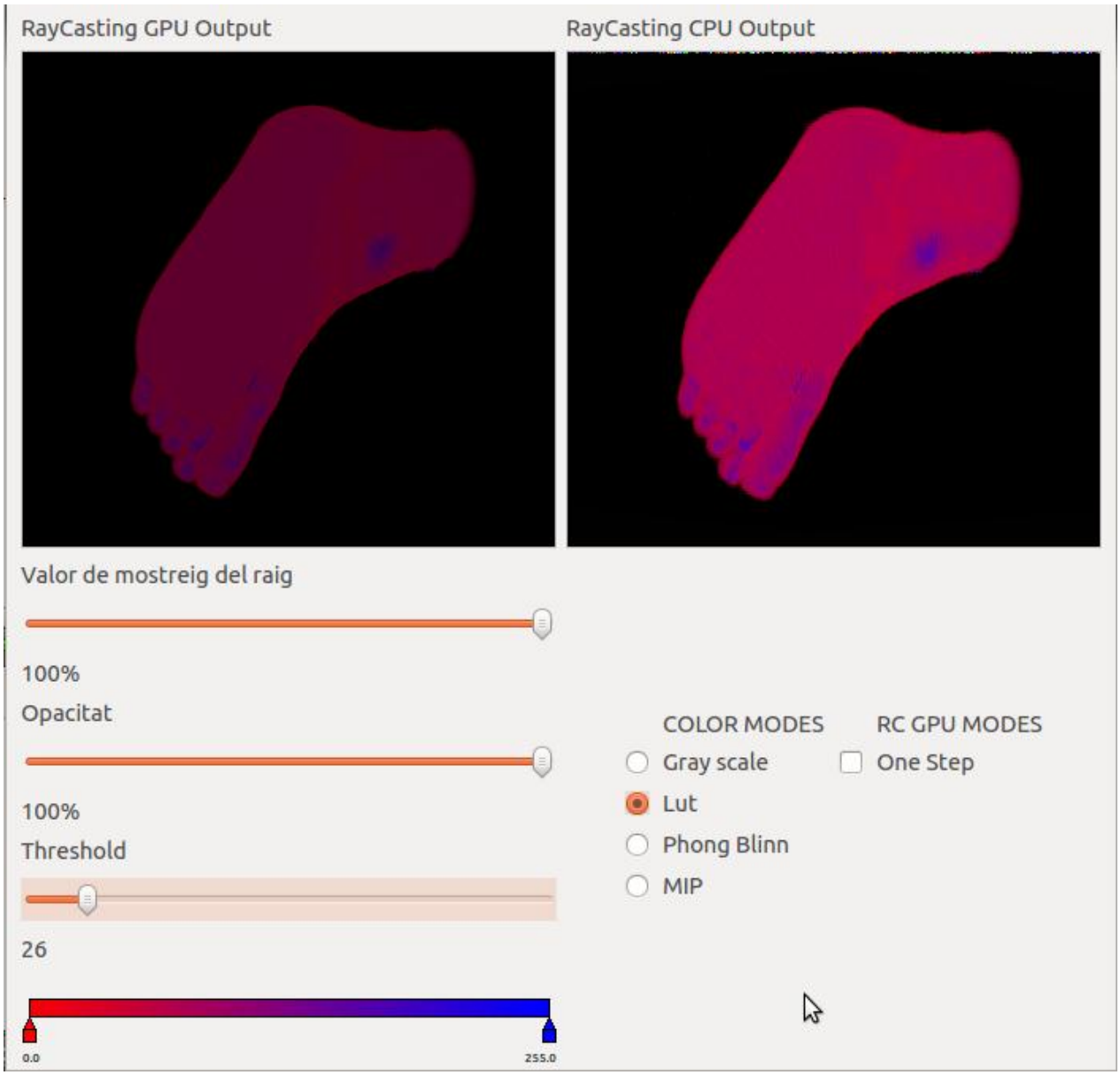


Fig. 47 - Simulació del model 1 amb mode de color LUT (GPU i CPU)

Temps d'execució en GPU: 0.016s.

Temps d'execució en CPU: 1.227s.

Més segmentació i mètode optimitzat en GPU:

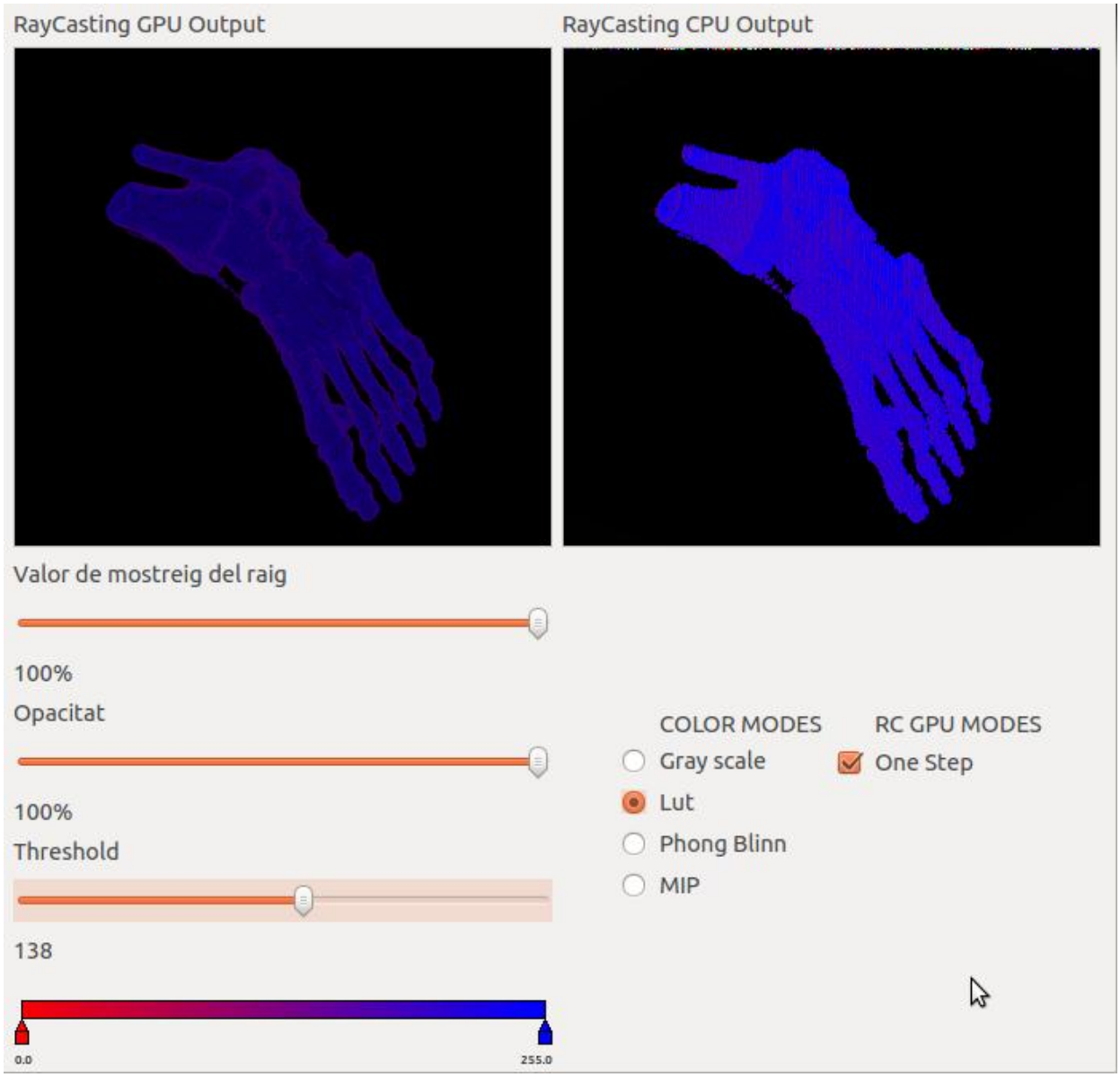


Fig. 48 - Simulació del model 1 amb mode de color LUT (GPU i CPU)

Temps d'execució en GPU optimitzat: 0.014s.

Temps d'execució en CPU: 1.167s.

Simulació amb mode de color LUT, opacitat 20% i sense segmentació. Aproximació optimitzada en GPU:

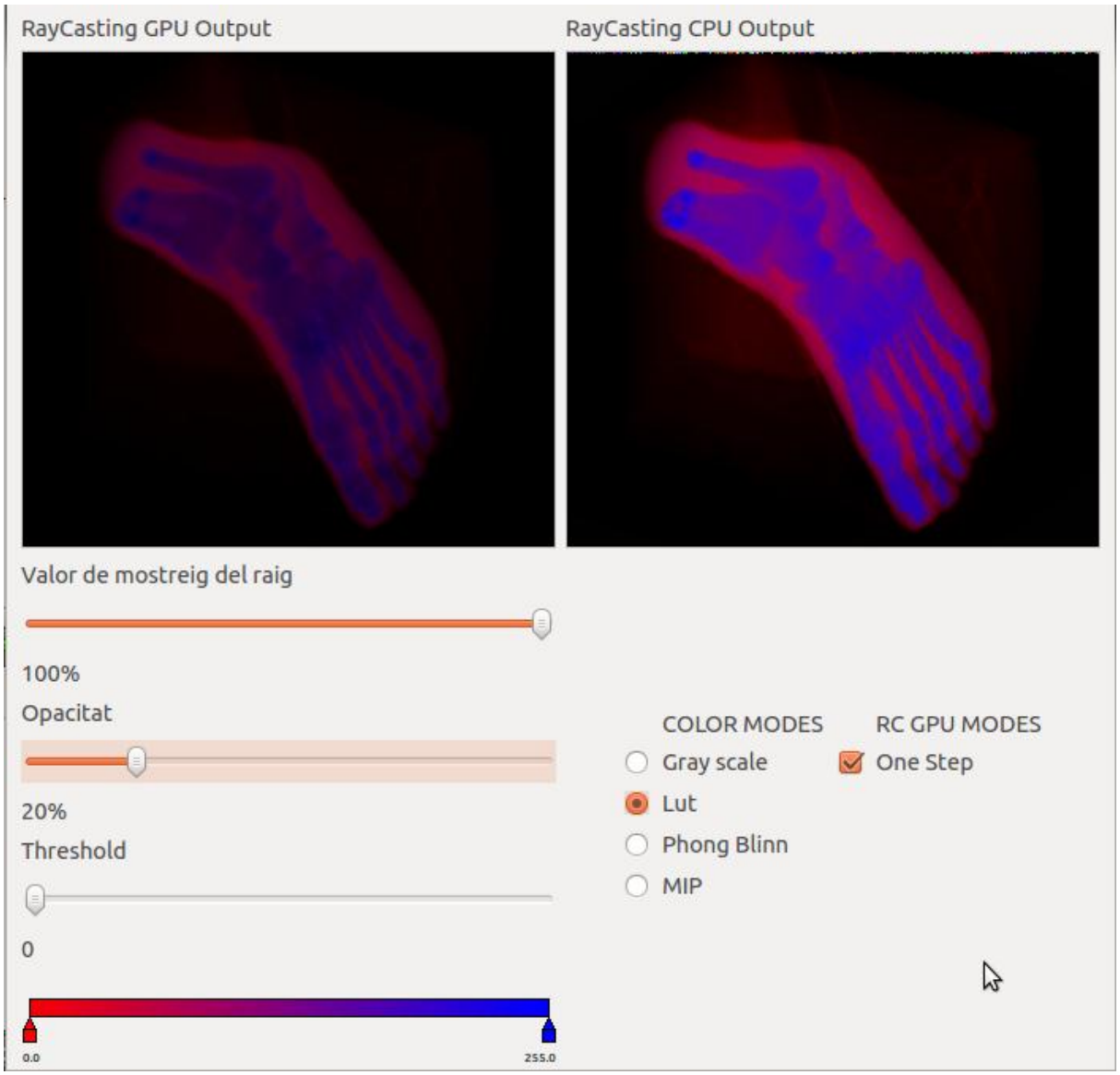


Fig. 49 - Simulació del model 1 amb mode de color LUT (GPU i CPU)

Temps d'execució en GPU optimitzat: 0.013s.

Temps d'execució en CPU: 1.707s.

A continuació es mostraran les simulacions amb el mode de color Phong Blinn.

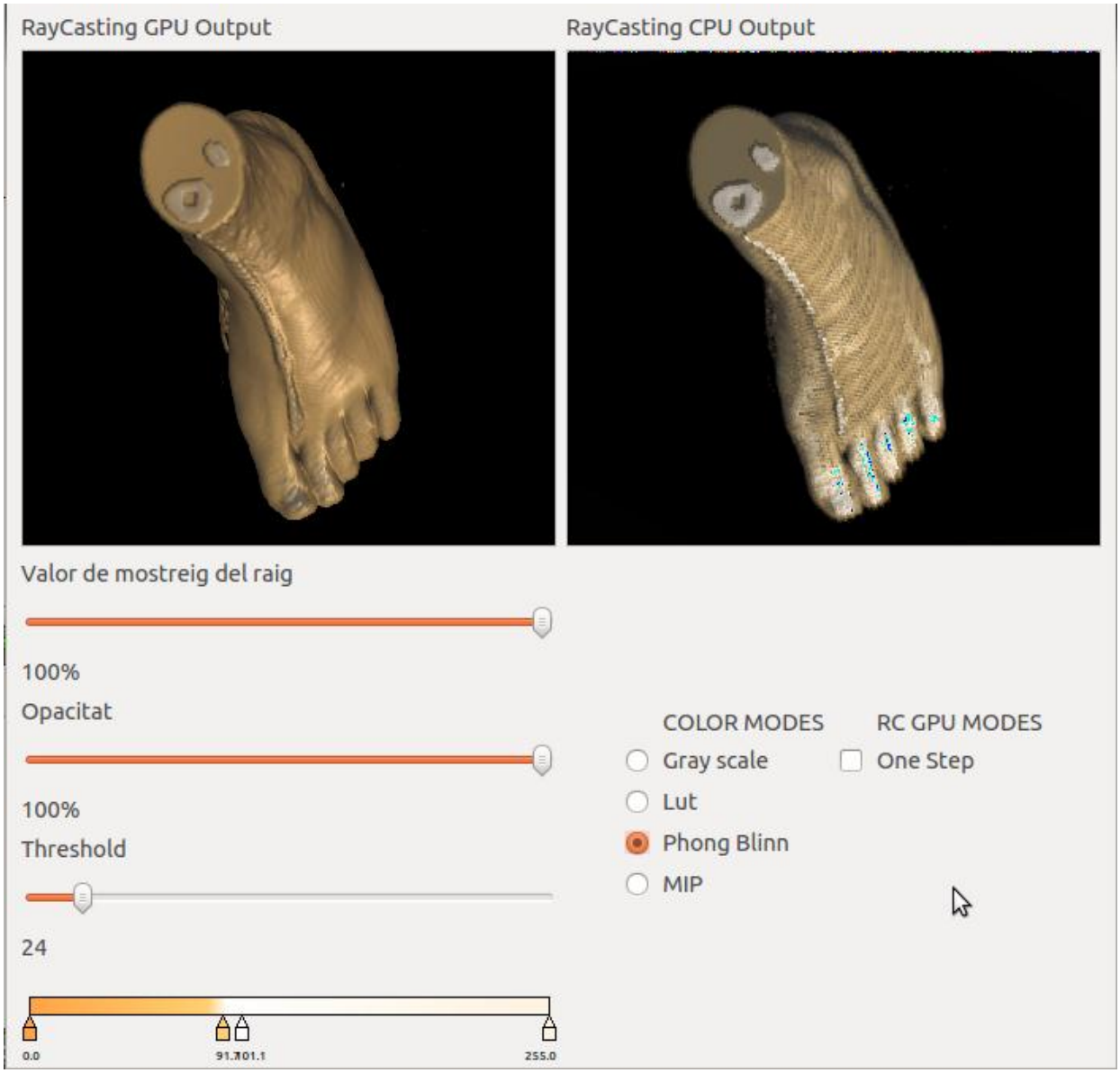


Fig. 50 - Simulació del model 1 amb mode de color Phong Blinn (GPU i CPU)

Temps d'execució en GPU: 0.029s.

Temps d'execució en CPU: 1.867s.

Com podem observar, el model reflecteix la llum que li arriba des d'un punt concret de l'escena. A més, els colors que es reflecteixen corresponen als de la taula de colors. Aquesta taula és totalment personalitzable per a l'usuari.

Simulació amb mode de color Phong Blinn, segmentació i aproximació optimitzada en GPU:

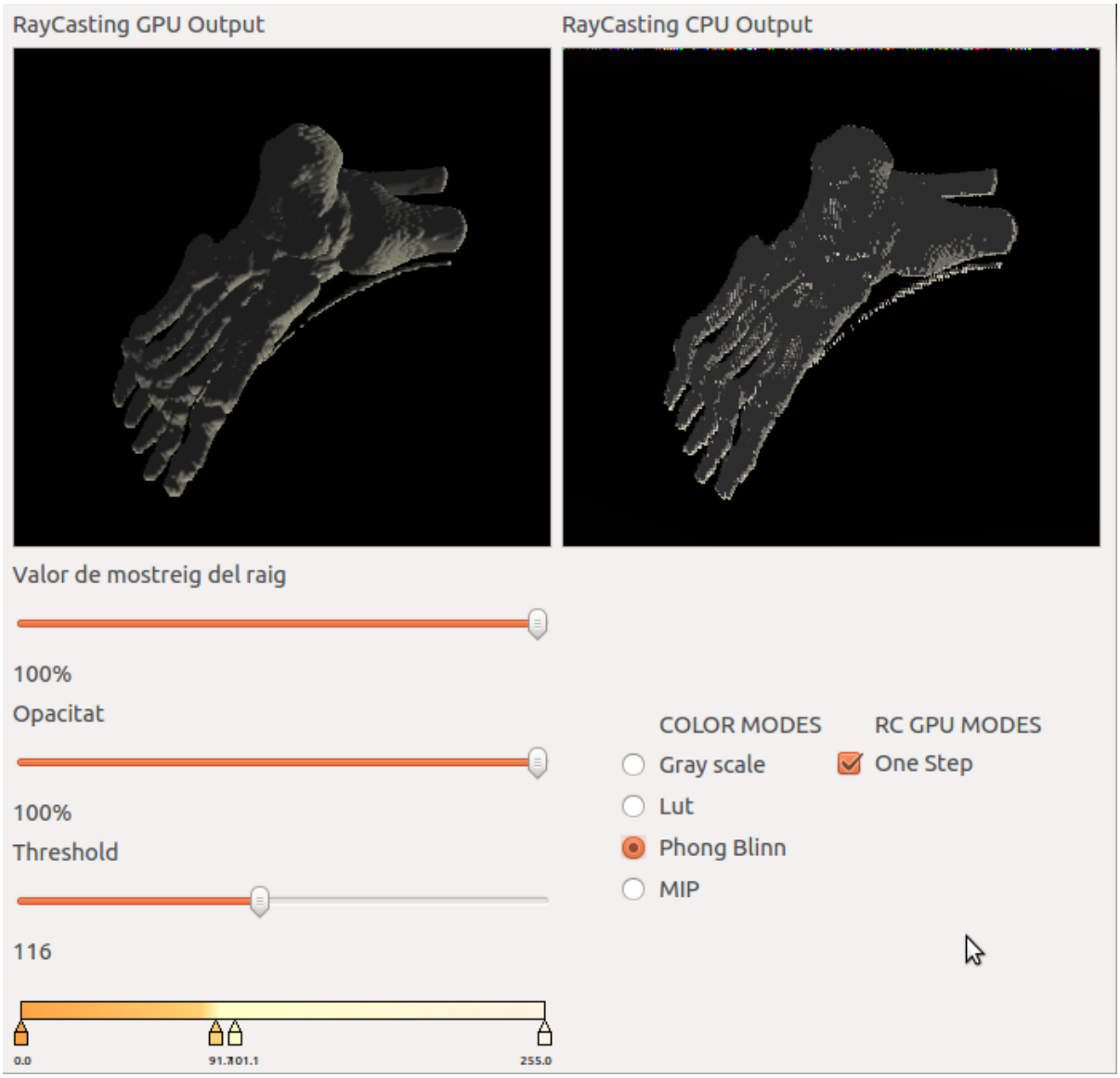


Fig. 51 - Simulació del model 1 amb mode de color Phong Blinn (GPU i CPU)

Temps d'execució en GPU optimitzat: 0.017s.

Temps d'execució en CPU: 1.242s.

Mateixa simulació però ara amb l'aproximació no optimitzada en GPU:

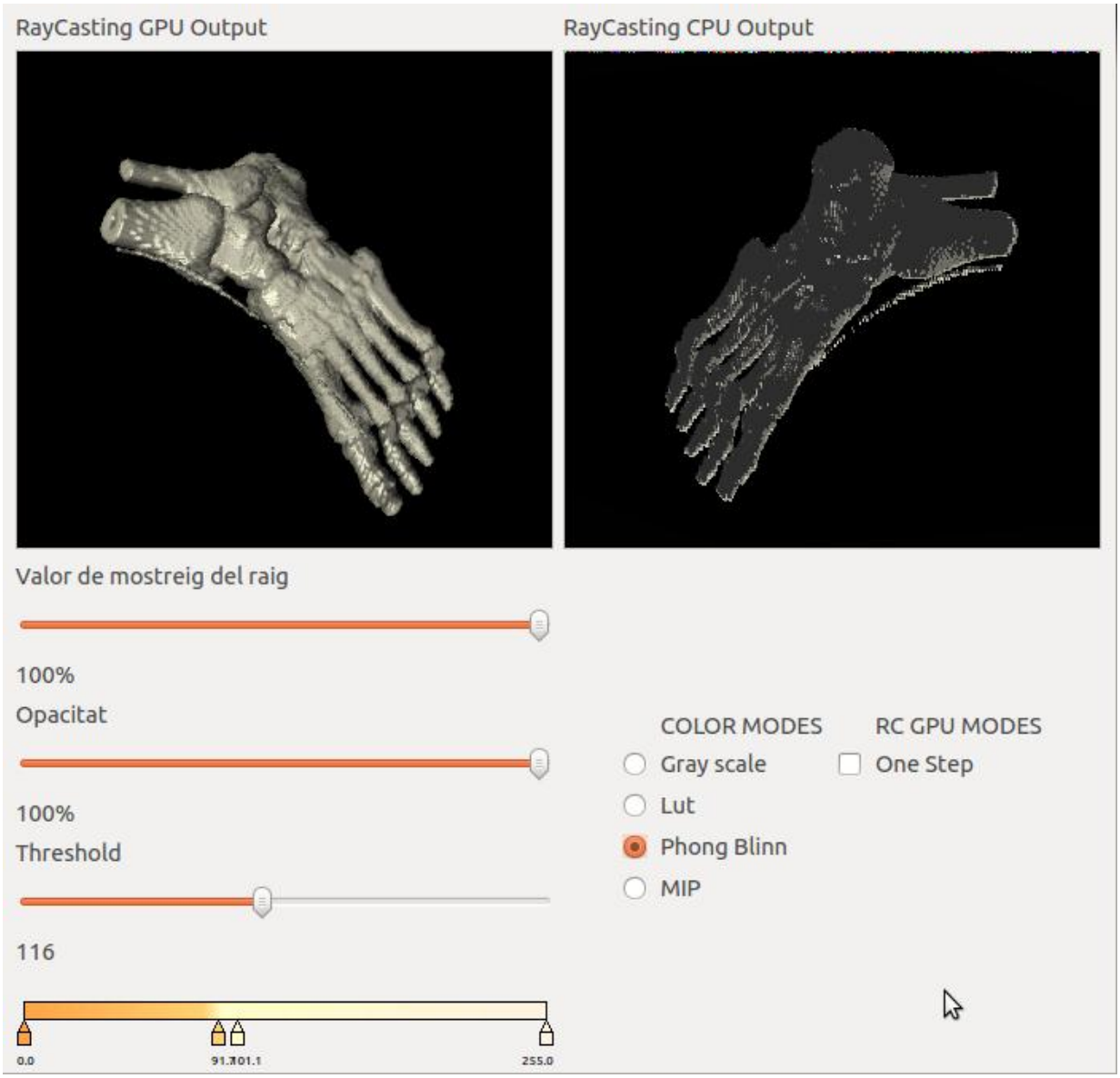


Fig. 52 - Simulació del model 1 amb mode de color Phong Blinn (GPU i CPU)

Temps d'execució en GPU: 0.053s.

Finalment, simulacions amb el mode de color MIP.

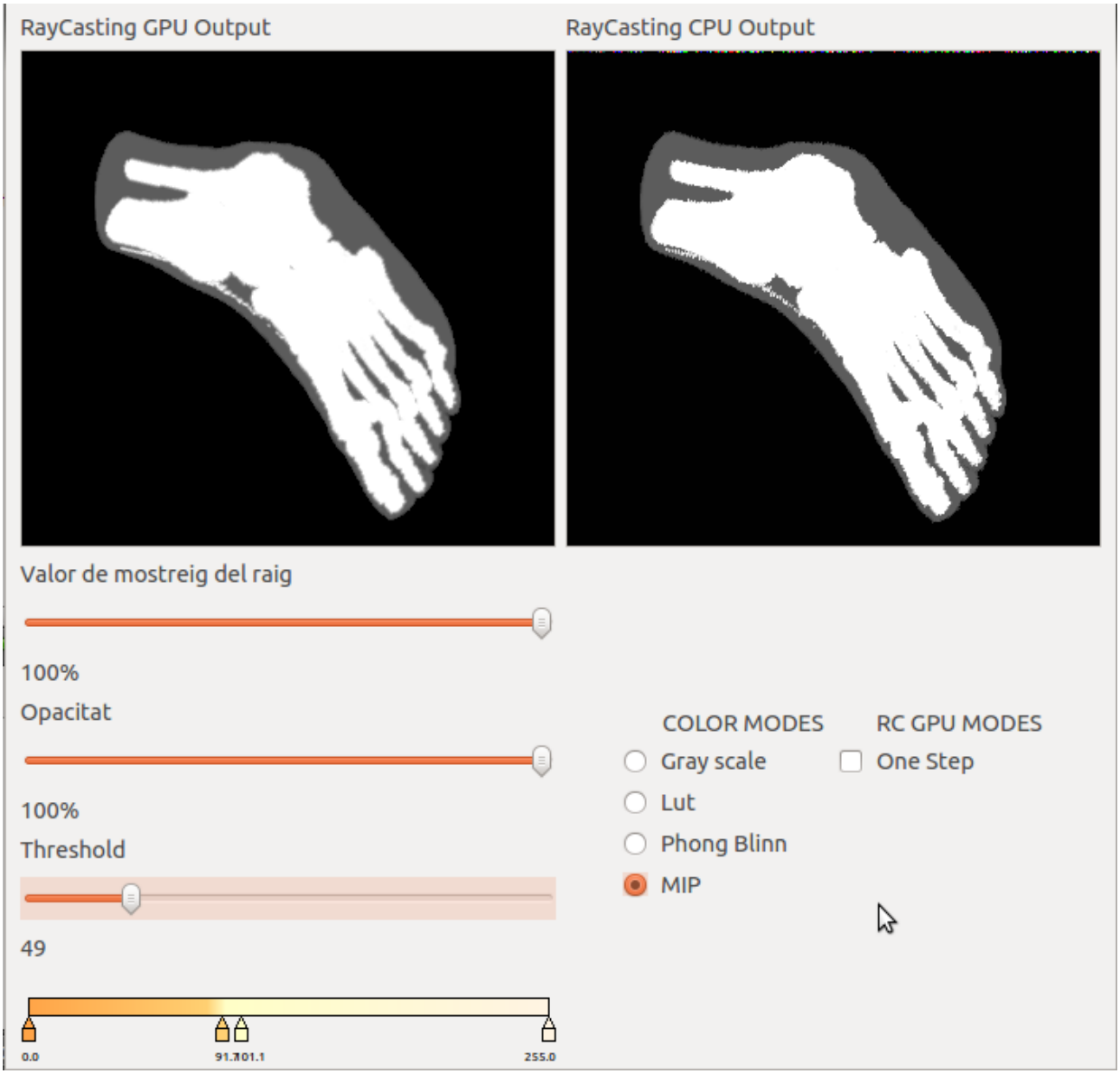


Fig. 53 - Simulació del model 1 amb mode de color MIP (GPU i CPU)

Temps d'execució en GPU: 0.026s.

Temps d'execució en CPU: 1.206s.

Mateixa simulació amb l'aproximació de Ray Casting de volum en GPU optimitzada:

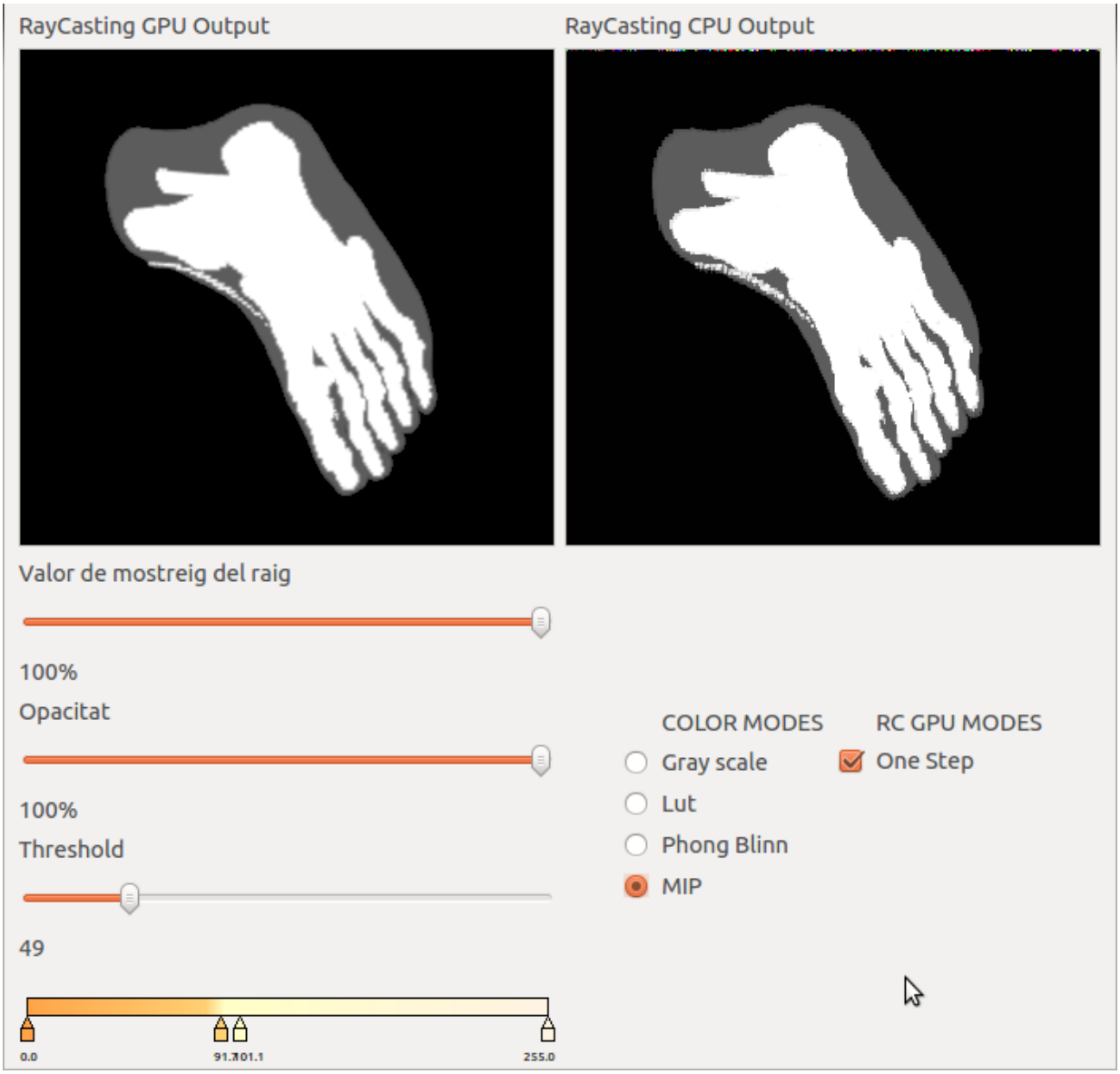


Fig. 54 - Simulació del model 1 amb mode de color MIP (GPU i CPU)

Temps d'execució en GPU optimitzat: 0.012s.

Temps d'execució en CPU: 1.256s.

5.2 Simulacions amb el model 2

A continuació es mostraran les simulacions i resultats amb el segon model. Aquest model correspon al tors d'un pacient i les dimensions del model són molt grans (400x400x400).

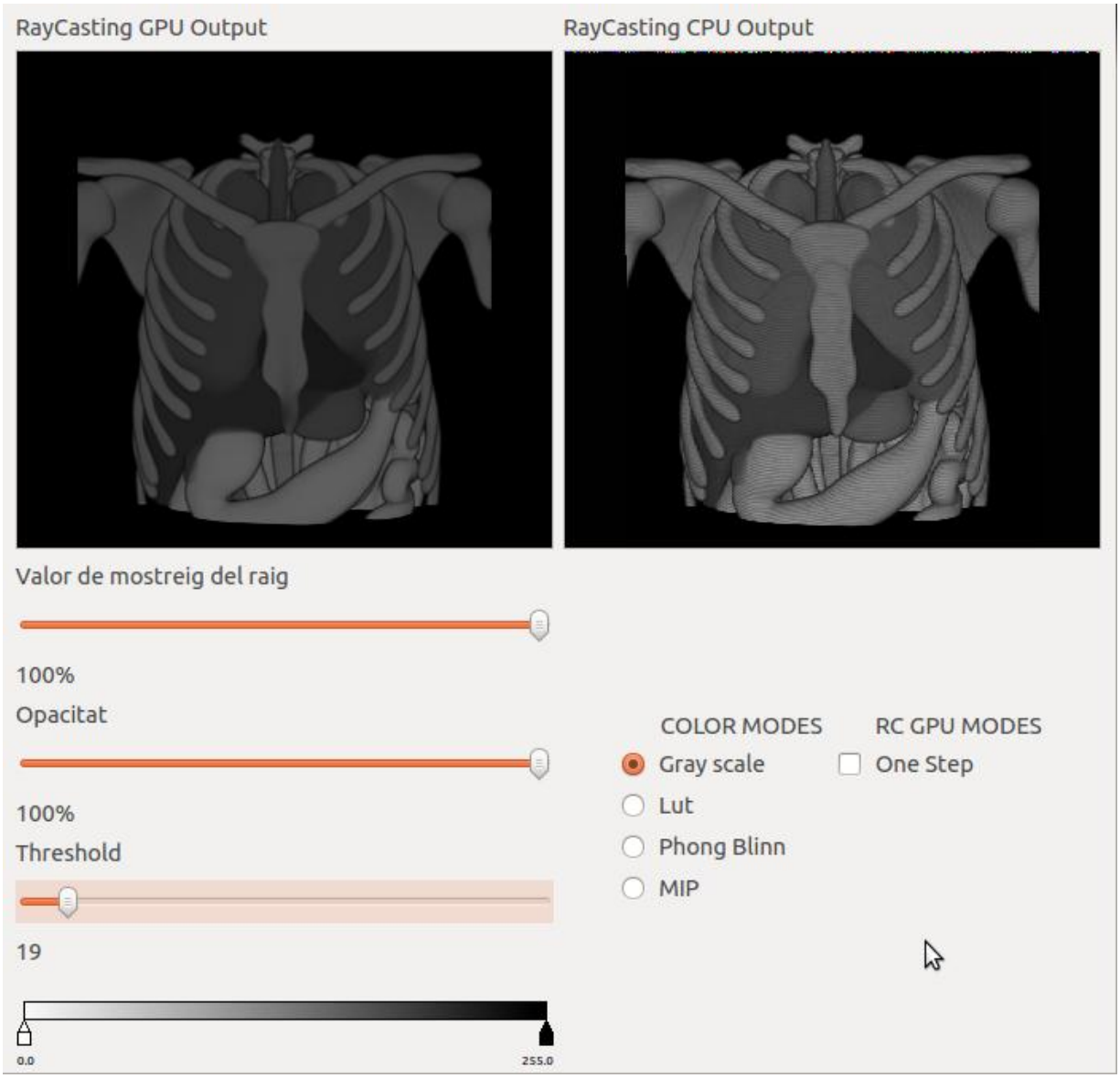


Fig. 55 - Simulació del model 2 amb mode de color Gray scale (GPU i CPU)

Temps d'execució en GPU: 0.067s.

Temps d'execució en CPU: 3.745s.

Mateixa simulació amb l'aproximació optimitzada en GPU:

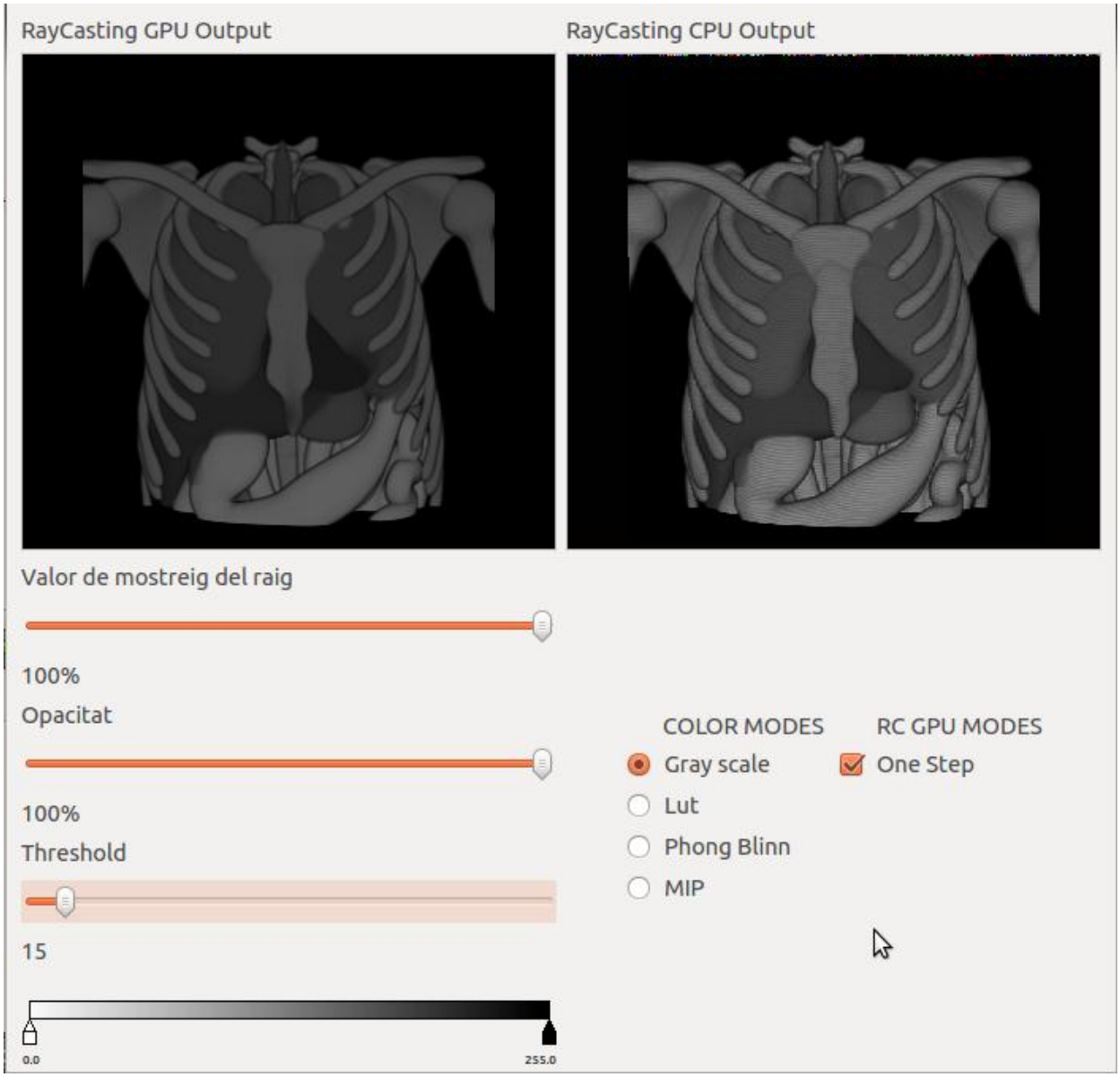


Fig. 56 - Simulació del model 2 amb mode de color Gray scale (GPU i CPU)

Temps d'execució en GPU optimitzat: 0.046s.

Temps d'execució en CPU: 3.767s.

A continuació, es mostraran simulacions amb el mode de color LUT:

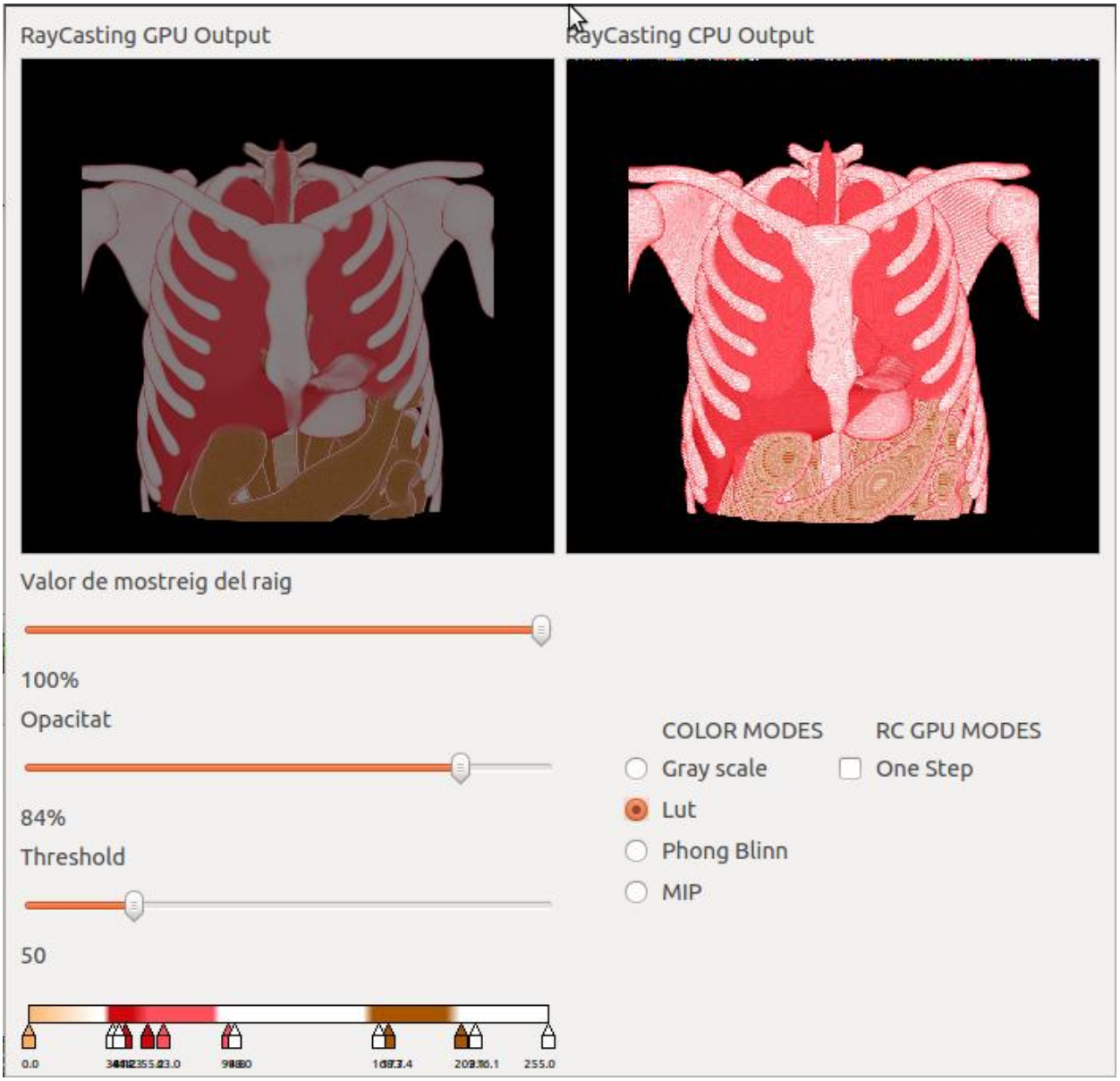


Fig. 57 - Simulació del model 2 amb mode de color LUT (GPU i CPU)

Temps d'execució en GPU: 0.130s.

Temps d'execució en CPU: 3.829s.

Mateixa simulació amb l'aproximació optimitzada en GPU:

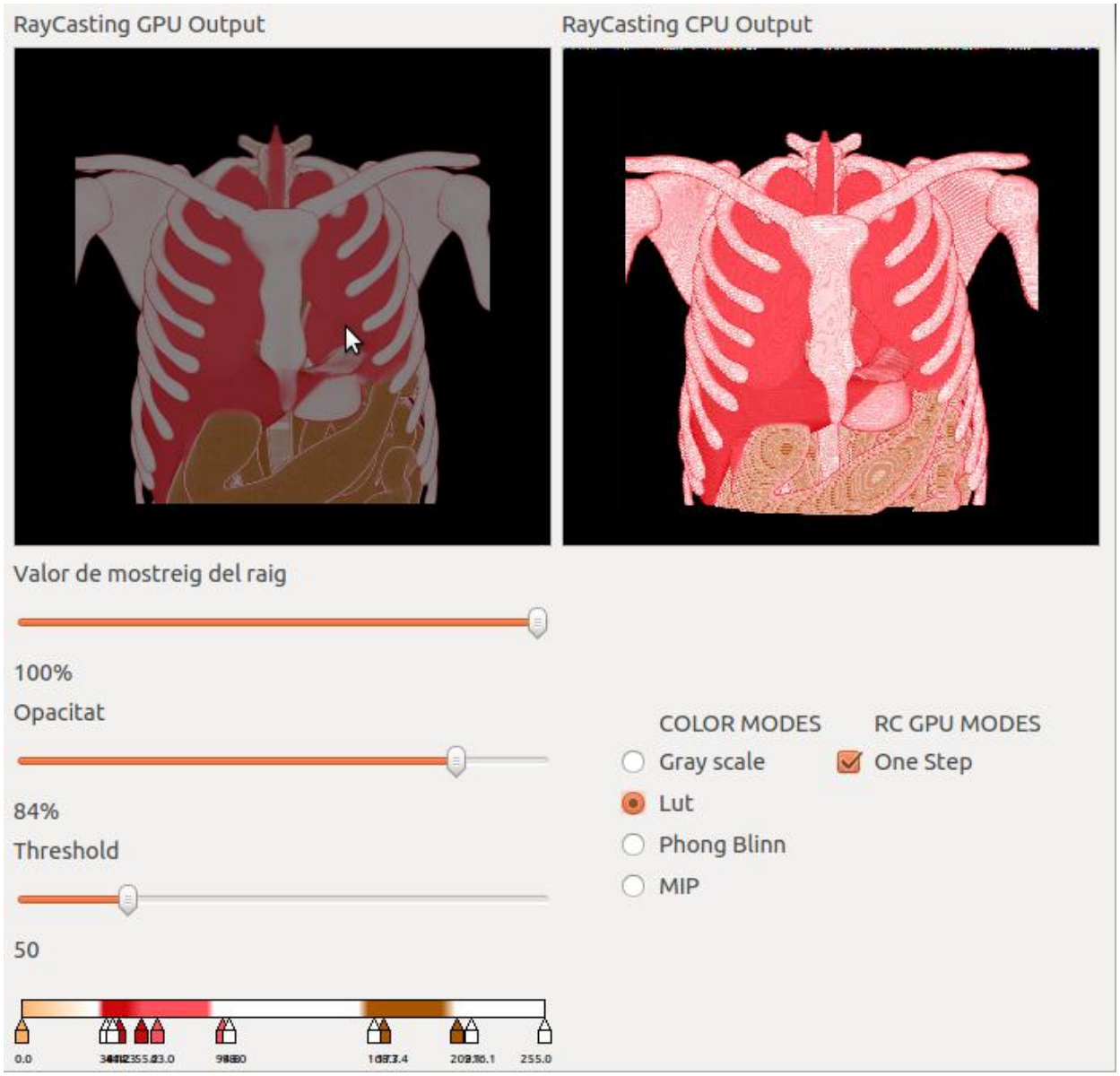


Fig. 58 - Simulació del model 2 amb mode de color LUT (GPU i CPU)

Temps d'execució en GPU optimitzat: 0.047s.

Temps d'execució en CPU: 3.767s.

A continuació, es mostraran simulacions amb el mode de color Phong Blinn:

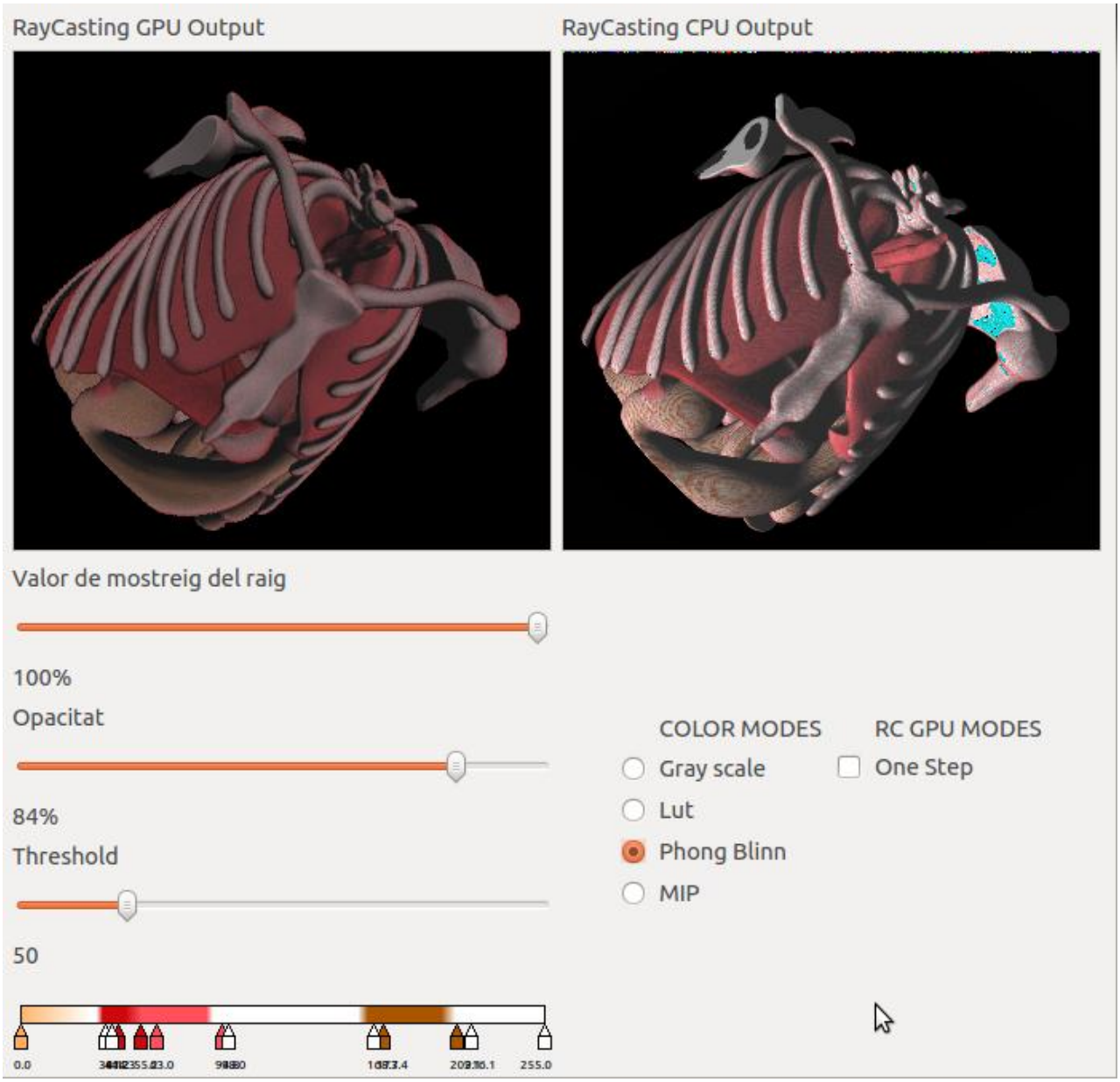


Fig. 59 - Simulació del model 2 amb mode de color Phong Blinn (GPU i CPU)

Temps d'execució en GPU: 0.215s.

Temps d'execució en CPU: 11.378s.

Mateixa simulació amb l'aproximació optimitzada en GPU:

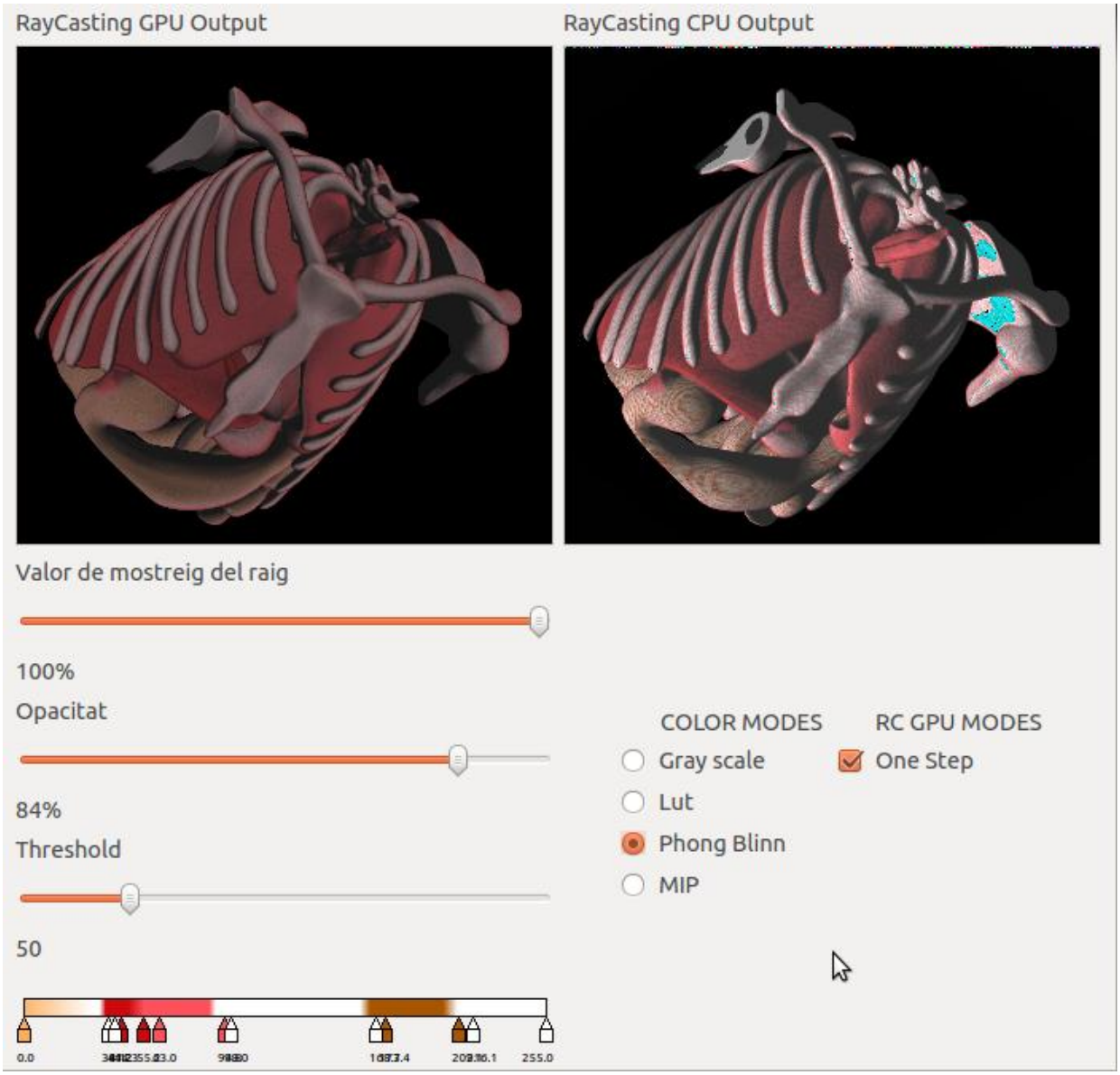


Fig. 60 - Simulació del model 2 amb mode de color Phong Blinn (GPU i CPU)

Temps d'execució en GPU optimitzat: 0.094s.

Temps d'execució en CPU: 3.767s.

A continuació, es mostraran simulacions amb el mode de color MIP:

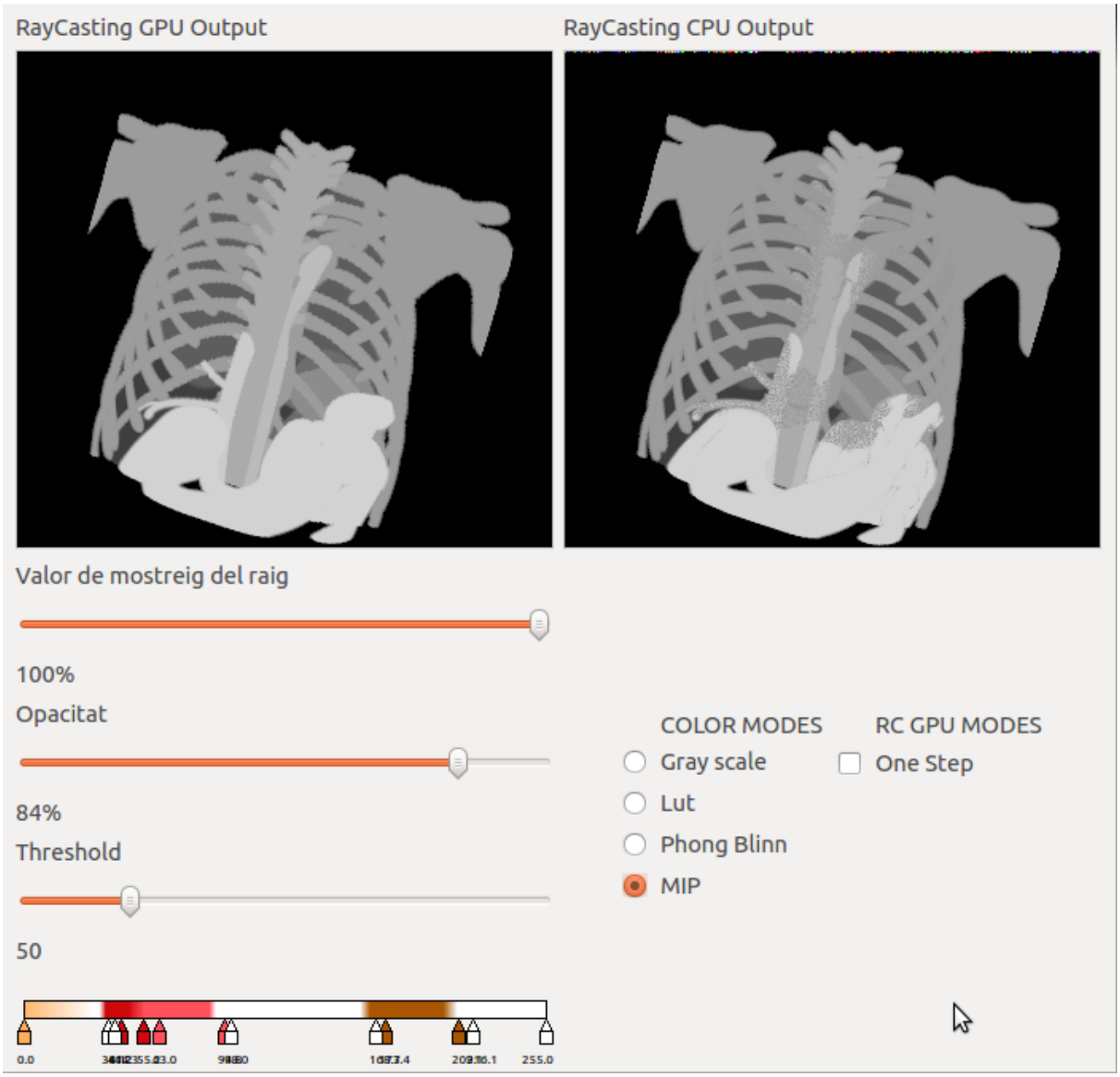


Fig. 61 - Simulació del model 2 amb mode de color MIP (GPU i CPU)

Temps d'execució en GPU: 0.119s.

Temps d'execució en CPU: 4.349s.

Finalment, última simulació, similar a l'anterior però amb l'aproximació optimitzada en GPU:

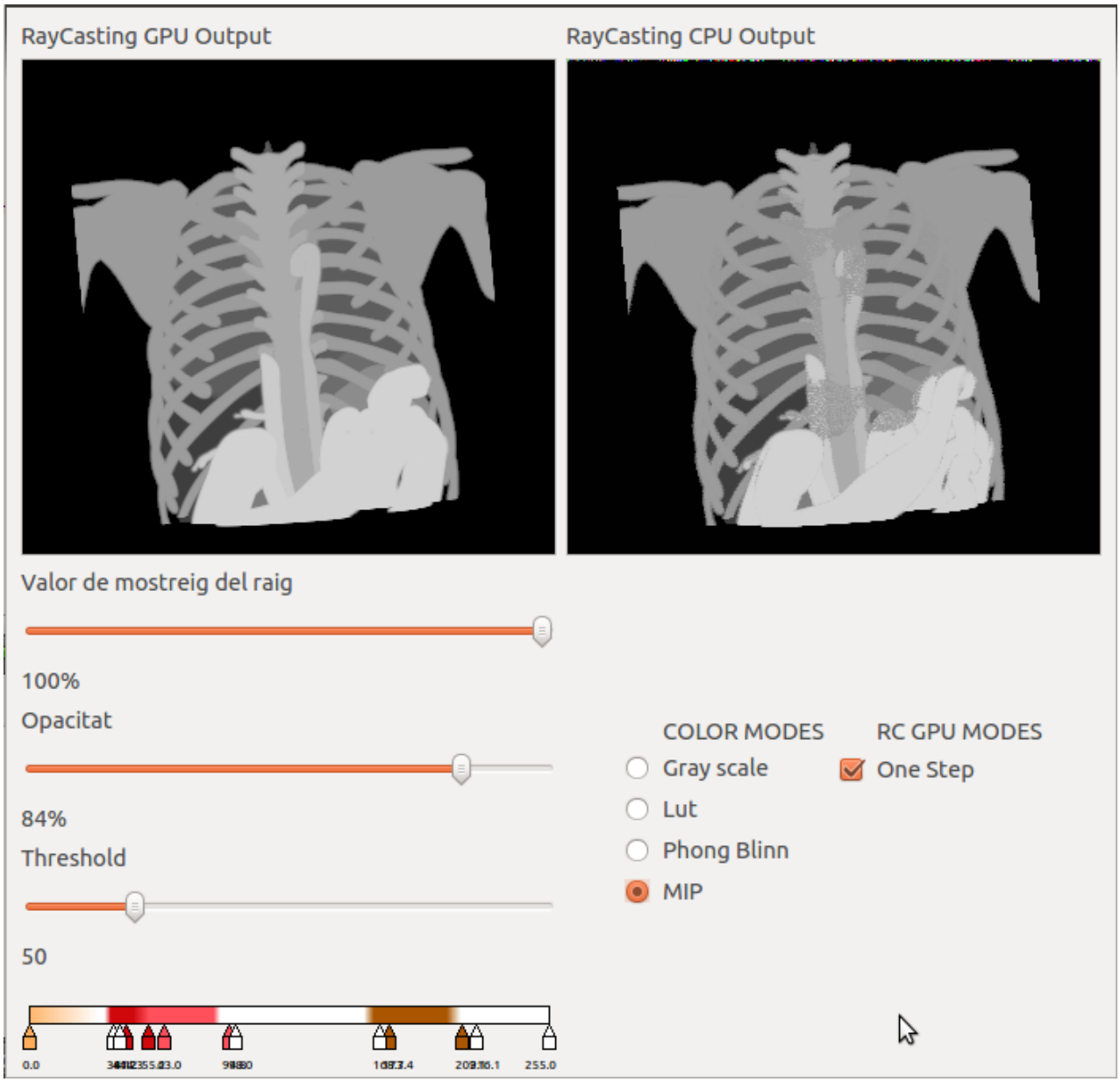


Fig. 62 - Simulació del model 2 amb mode de color MIP (GPU i CPU)

Temps d'execució en GPU optimitzat: 0.086s.

Temps d'execució en CPU: 3.923s.

5.3 Taula de Temps

A continuació es mostrarà la taula de temps relativa a les simulacions anteriors:

Dimensions del Model	Tipus de Processador	Mode de Color	Optimitzat	Temps
128x128x128	CPU	Gray scale	-	1.878s
128x128x128	GPU	Gray scale	No	0.053s
128x128x128	GPU	Gray scale	Si	0.022s
128x128x128	CPU	LUT	-	1.465s
128x128x128	GPU	LUT	No	0.152s
128x128x128	GPU	LUT	Si	0.014s
128x128x128	CPU	Phong Blinn	-	1.867s
128x128x128	GPU	Phong Blinn	No	0.053s
128x128x128	GPU	Phong Blinn	Si	0.017s
128x128x128	CPU	MIP	-	1.206s
128x128x128	GPU	MIP	No	0.026s
128x128x128	GPU	MIP	Si	0.012s
400x400x400	CPU	Gray scale	-	3.745s
400x400x400	GPU	Gray scale	No	0.067s
400x400x400	GPU	Gray scale	Si	0.046s
400x400x400	CPU	LUT	-	3.829s
400x400x400	GPU	LUT	No	0.130s
400x400x400	GPU	LUT	Si	0.047s
400x400x400	CPU	Phong Blinn	-	11.378s
400x400x400	GPU	Phong Blinn	No	0.215s
400x400x400	GPU	Phong Blinn	Si	0.094s
400x400x400	CPU	MIP	-	4.349s
400x400x400	GPU	MIP	No	0.119s
400x400x400	GPU	MIP	Si	0.086s

Fig. 63 - Taula dels temps d'execució de les simulacions

Els resultats de la GPU posen en evidència el fet que les CPU actuals no estan preparades per a grans quantitat de càlculs i operacions.

A la CPU no s'ha aconseguit baixar de 1 segon, la qual cosa faria la interacció molt lenta i fins i tot molesta. La GPU sempre es troba per sota dels 0.3 segons, de manera que fa possible la interacció en temps real i demostra la gran potència de les GPU modernes.

Amb la versió optimitzada en la GPU, s'aconsegueixen speed ups d'entre 90 i 20 vegades més ràpida que la CPU depenent del joc de les dades o el tipus de visualització.

6 Conclusions

L'objectiu d'aquest projecte era el de analitzar, dissenyar i implementar un sistema de visualització el qual permetés validar les diferents aproximacions de Ray Casting de volum en CPU i GPU. A més es volia poder visualitzar els models de vòxels amb diversos mètodes de càlcul d'il·luminació o color, de manera que s'ampliés el rang de possibles visualitzacions.

Aprofitant els bons resultats del Ray Casting de volum, i la potència de les GPU actuals, es vol també, poder interaccionar en temps real amb els volums per tal que el propi usuari sigui capaç d'obtenir imatges de gran qualitat sense tenir coneixements dels algorismes que implementa la aplicació. Això comportava el disseny d'una interfície gràfica suficientment intuïtiva com per a que l'usuari aconseguís els seus objectius d'una manera fàcil, unívoca i sense errors.

L'anàlisi, disseny i implementació de l'aplicació final, amb la qual es considera que s'han assolit els objectius, comportava una pronunciada corba d'aprenentatge, degut a que per a la implementació dels algorismes de Ray Casting de volum, calien uns coneixements que no van ser donats a l'assignatura de Gràfics i Visualització de Dades i per tant, s'ha hagut de estendre el coneixement que es va oferir a l'assignatura. Aquests coneixements correspondrien a la generació de textures en GL, el mapeig de textures en GL i GLSL [14], el pas de textures a GPU, el mapeig de textures a GPU, la renderització fora de pantalla amb *framebuffers*, les característiques dels mons de vòxels, l'ús de *look up tables*, mètodes d'interpolació trilineal, càlcul de normals mitjançant el gradient del volum, les diferents competències implícites a les aproximacions de Ray Casting de Volum en CPU i en GPU, entre d'altres.

Finalment, conclou amb èxit l'anàlisi, disseny i implementació del sistema de visualització el qual permet la validació de les diferents aproximacions de Ray Casting de volum en CPU i en GPU. El sistema consta d'una interfície gràfica, fàcil i intuïtiva, la possible visualització simultània de les diferents aproximacions en CPU i GPU, interactivitat en temps real amb els volum, quatre modes de color, cadascun amb diferents característiques i dos modes de Ray Casting de volum en GPU que corresponen als analitzats al capítol 3.

A continuació s'enumeraran les possibles línies de continuació del projecte.

Possibles línies de continuació:

- Implementar més modes de il·luminació per tal d'ampliar el rang de visualitzacions possibles sobre el volum amb el Ray Casting de volum.

- Implementar la possible visualització de volums ja segmentats, de tal manera que al llegir des de fitxer la classificació del volum a renderitzar, es pintessin els vòxels amb un color acord amb els valors de densitat que han estat classificats.

- Permetre guardar en un fitxer, el volum segmentat amb el lliscador de Threshold, de manera que reduíssim el volum traient possibles capes que no fossin útils, o fins i tot per a poder renderitzar més ràpid les parts d'interès. També es podria permetre guardar en fitxer un subvolum per intervals de densitat dels vòxels, els quals poden ser visualitzats gràcies al lliscador de Threshold.

- Implementar diferents variants a les aproximacions de Ray Casting de volum, com per exemple, el Ray Casting de volum adaptatiu, amb el qual es redueix el temps d'execució, gràcies a que en el procés de mostreig del volum, es mostreja amb més o menys freqüència depenent de si s'està mostrejant una zona d'interès o no. Aquesta variant pràcticament no es notaria a les GPU actuals degut a la seva gran potència, en canvi és una tècnica molt adequada per a les CPU modernes de més d'un nucli.

- Aprofitar la implementació de les aproximacions de Ray Casting de volum per a ampliar les funcionalitats implementant processos d'ombrejat o altres tècniques com el Ray Tracing.

7 Referències bibliogràfiques

[1] **A Simple and Flexible Volume Rendering Framework for Graphics-Hardware-based Ray Casting** [Simon Stegmaier, Magnus Strengert, Thomas Klein]

[2] **Arxius PPM –** http://rosettacode.org/wiki/Bitmap/Write_a_PPM_file#C

[3] **Classificació del Volum** - http://www.hindawi.com/journals/aai/2010/520427/

[4] **GLSL –** http://www.opengl.org/documentation/glsl/

[5] **GPU Ray Casting** [Peter Triers]

[6] **Memory Efficient Acceleration Structures and Techniques for CPU-based Volume Raycasting of Large Data** [Sören Grimm, Stefan Bruckner, Armin Kanitsar, Eduard Gröller], Acceleration Techniques for GPU-Based Volume Rendering [J. Krüger, R. Westermann]

[7] **OpenGL** – http://www.opengl.org/

[8] **OpenGL Framebuffer objects** – http://www.songho.ca/opengl/gl_fbo.html

[9] **QColorRampEditor**

http://qt-apps.org/content/show.php/QColorRampEditor?content=147481

[10] **Qt SDK** – http://qt-project.org/

[11] **Ray-Box Intersection: An Efficient and Robust Ray-Box Intersection Algorithm** [Amy Williams, Steve Barrus, R. Keith Morley, Peter Shirley]

[12] **Ray Casting** – Wikipedia

[13] **Ray Tracing** – Wikipedia

[14] **Textures i el seu accés a GLSL**

http://www.lighthouse3d.com/tutorials/glsl-tutorial/simple-texture/

[15] **Texture Mapping –** Wikipedia

[16] **Volume Ray Casting –** Wikipedia

[17] **Volume Rendering** [Francisco Morillo, Ciro Durán]

http://www.inmensia.com/articulos/raytracing/planotrianguloycubo.html?pag=3

[18] **Z Buffering** – Wikipedia

Apèndix A

A.1 Instal·lació: Requeriments mínims i passos a seguir

- Requeriments:
  Els requeriments software per a executar aquesta aplicació són:
    - Sistema operatiu Linux, a ser possible Ubuntu 12.04, per ser openSource i per la fàcil trobada de documentació a internet.
        - Fonts: http://www.ubuntu.com/
    - G++ 4.6.
        - Fonts: http://gcc.gnu.org/
    - FreeGLUT3 o algun altre conjunt de llibreries per OpenGL.
        - Fonts: http://freeglut.sourceforge.net/
    - Qt SDK.
        - Fonts: http://qt-project.org/
    - Controladors de la targeta gràfica (Nvidia) o drivers de glsl que permetin shaders posteriors a la versió 150.
        - Fonts: http://www.nvidia.es/Download/index.aspx?lang=es
- Per tal d'instal·lar el g++ i el FreeGLUT3, es recomana utilitzar el gestor de paquets Synaptic.
- Per tal d'instal·lar la resta de components, descarregar les fonts de la pàgina i seguir el manual d'instal·lació.

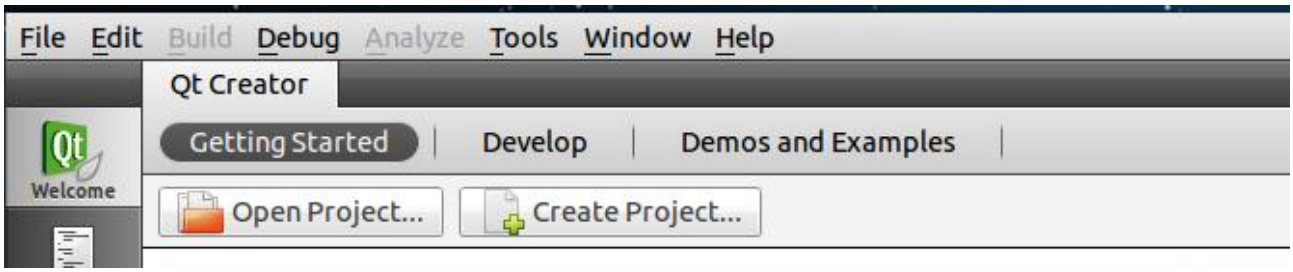
A.2 Manual del desenvolupador

Per al desenvolupament d'aquesta aplicació, s'han utilitzat les eines comentades a l'apartat 1 de l'apèndix A. A més, com a IDE, s'ha utilitzat el QtCreator, el qual pot ser descarregat des del mateix enllaç que el QtSDK.

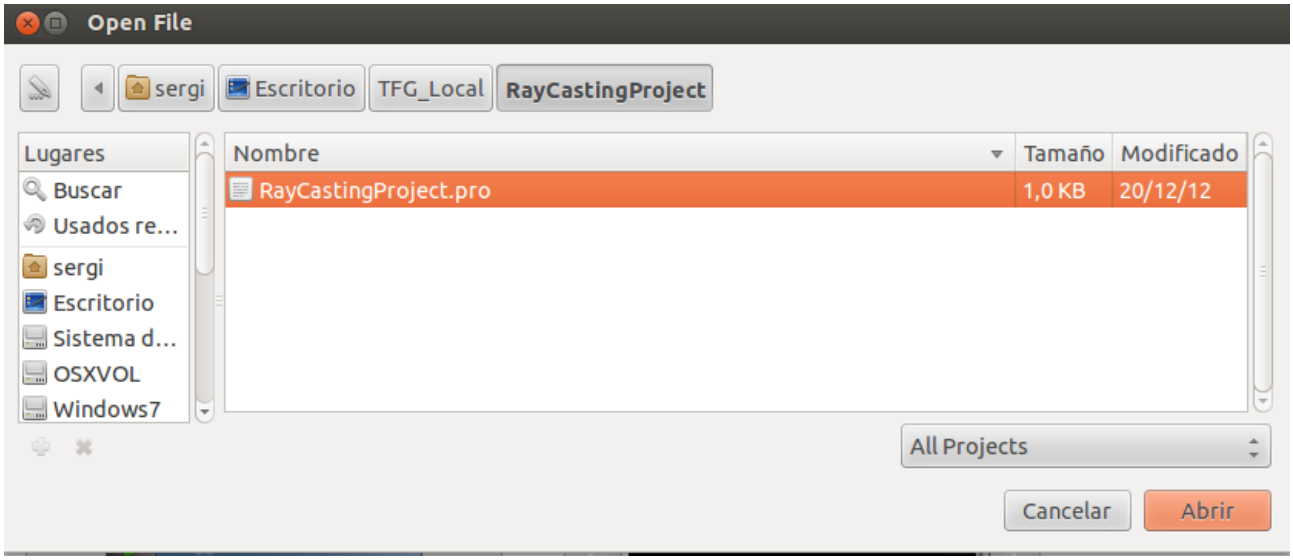
Per a continuar amb el desenvolupament de l'aplicació es recomana seguir els següents passos:

a. Instal·lar els components de l'apartat anterior més l'entorn de desenvolupament QtCreator. Les guies d'instal·lació de l'entorn de desenvolupament estan a l'enllaç facilitat.

b. Un cop instal·lats tots els components i, el projecte localitzat a un directori, obrir el projecte de la següent forma:

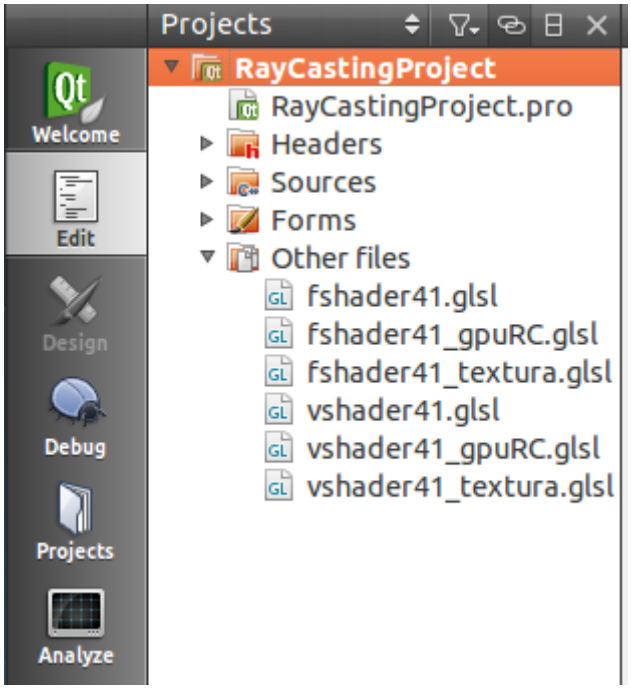


Fem click a Open Project i s'obrirà el gestor de fitxers.



Busquem la carpeta on es troba el nostre projecte i obrim el fitxer de projecte amb extensió ".pro".

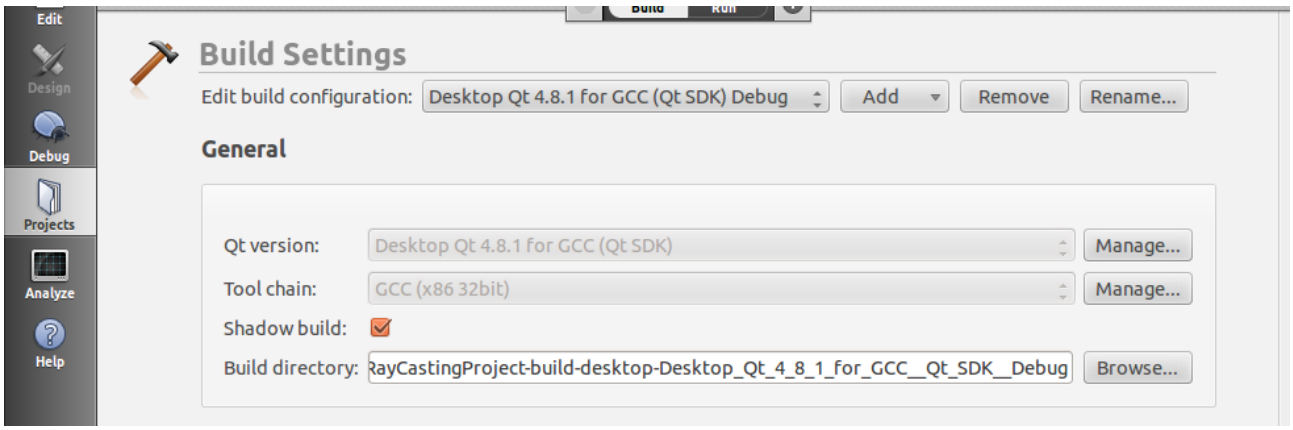
Un cop obert, veurem el nostre projecte a l'arbre de fitxers del projecte.



On a la carpeta Headers trobarem les capçaleres amb extensió ".h", a la carpeta Sources trobarem les fonts de codi amb extensió ".cpp" i a Other files trobarem els *shaders* els quals són executats per la GPU amb extensió ".glsl".

Es recomana un cop obert el projecte, configurar a on es trobarà la carpeta de compilació. Aquesta carpeta hauria d'estar al mateix nivell que la carpeta del projecte, degut a que la aplicació anirà a buscar als *shaders* a la carpeta del projecte però l'execució es fa des de la carpeta de compilació, que és on es troba l'executable.

Per a configurar a on es compilaran les fonts, clicar a la icona "*Projects*".



A on veurem el "*Build directory*" que és a on aniran a parar els fitxers compilats i l'executable de l'aplicació.

Finalment, per executar l'aplicació, fer click al botó d'execució o fer ús del *shortcut* Ctrl+R.

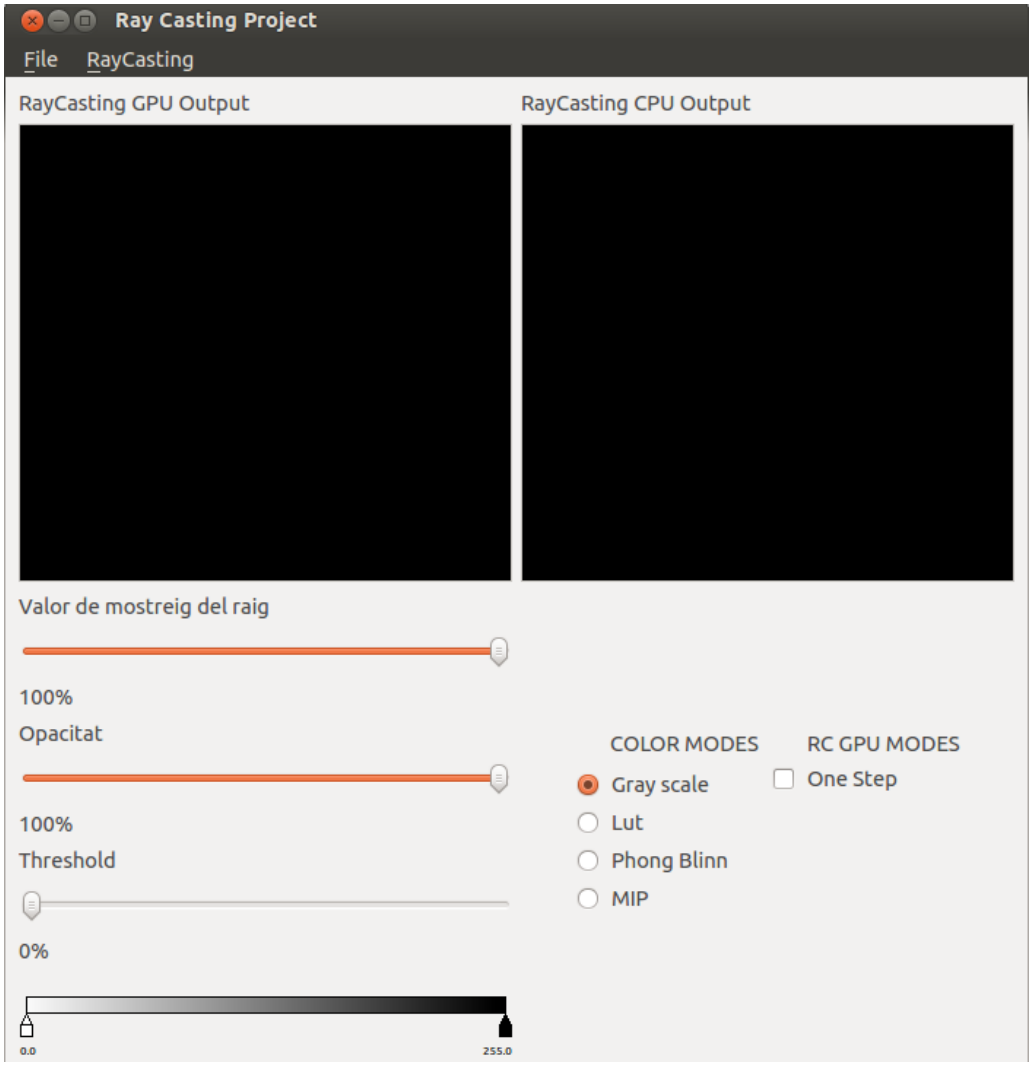
Apèndix B

B.1 Manual de l'usuari

Primerament, s'explicarà com executar l'aplicació:

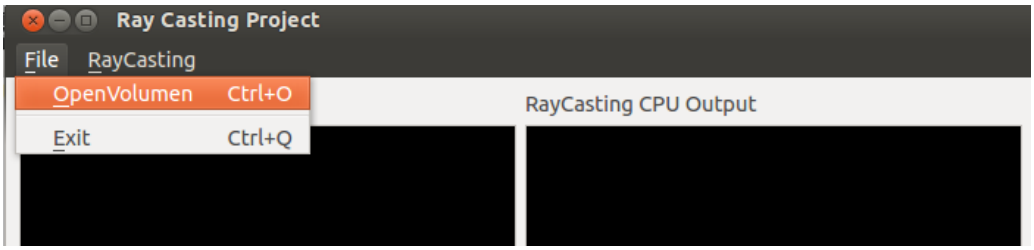
- Obrir una terminal.
- Navegar a la terminal fins a la carpeta de compilació de fonts del projecte on es troba l'executable.
- Un cop es troba a la carpeta, escriure la següent instrucció sense les cometes:
  ./RayCastingProject

Un cop executada, el primer que veurem serà la següent interfície gràfica.

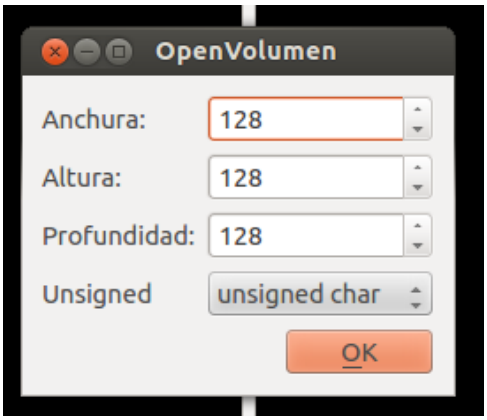


La interfície gràfica consta d'una barra de menús, amb el menú File i el menú RayCasting, dues finestres de visualització, tres lliscadors per a interactuar amb els volums, una taula de colors, i els checks de mode de color i mode de Ray Casting en GPU.

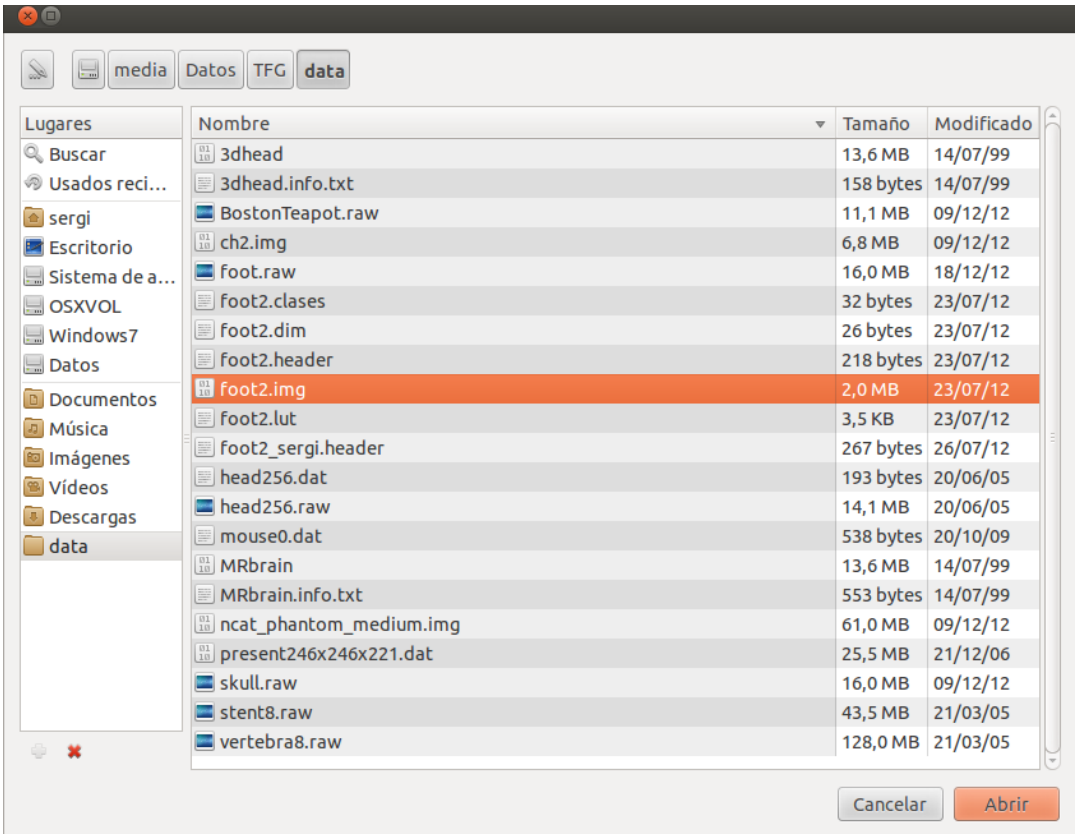
Per tal d'obrir un volum, farem clic al menú File i posteriorment, a la opció OpenVolumen o directament fem ús del *shortcut* Ctrl+O.



Després, se'ns obrirà un diàleg el qual demana dades sobre el volum. Aquestes dades corresponen a les dimensions i el tipus d'informació de la qual es composa el model.

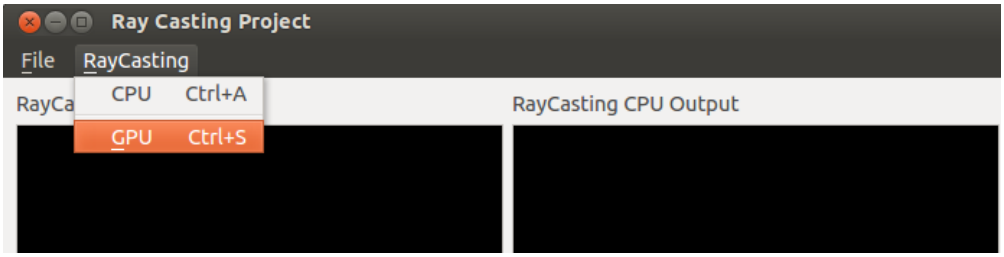


Quan acabem d'introduir les dades, fem clic al botó d'OK. Acte seguit s'obrirà el gestor de fitxers per a seleccionar el fitxer que conté les dades del volum en qüestió.

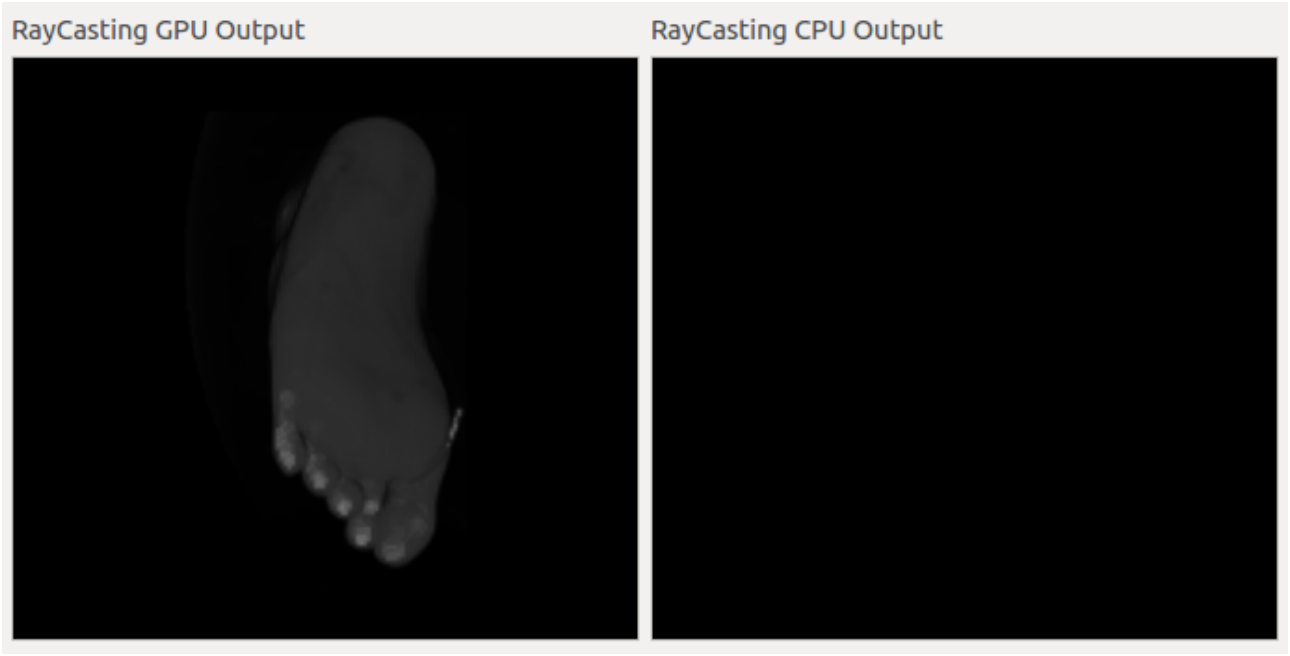


En obrir el fitxer de volum desitjat, encara no visualitzarem res a les finestres de visualització, per tal de visualitzar quelcom, s'ha d'activar el Ray Casting de volum del menú de RayCasting de la barra d'eines.

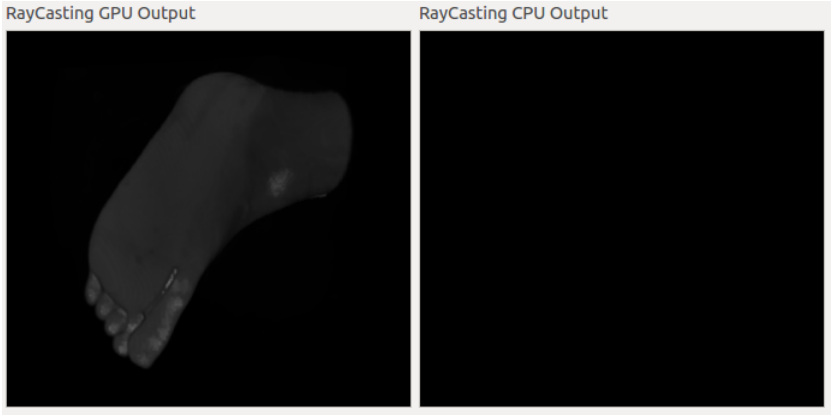
Es recomana activar primer el Ray Casting de volum en GPU ja que aquest controla la posició en la que sortirà el volum en el Ray Casting de volum en CPU.



Després d'activar el Ray Casting de volum en GPU, obtindrem la següent visualització, depenent del volum seleccionat anteriorment.

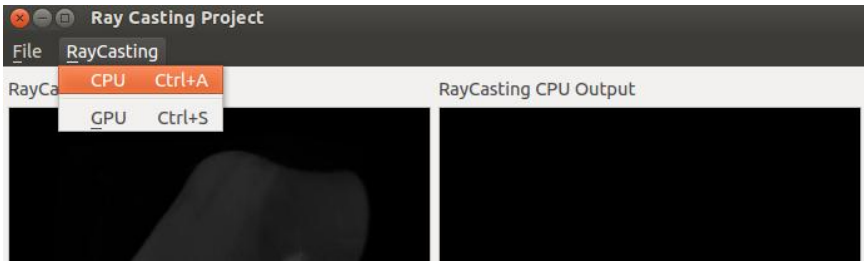


El volum renderitzat per GPU té l'avantatge de poder ésser rotat, mogut o fins i tot apropat.

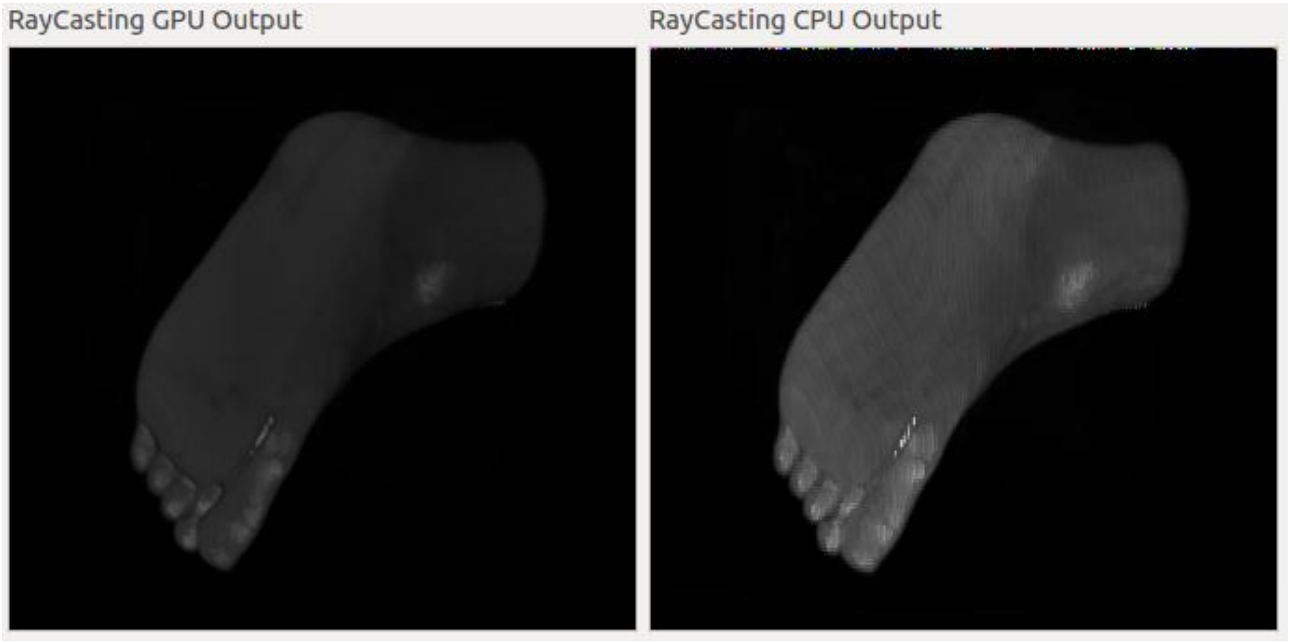


Per rotar el volum, clicar amb el botó esquerre del ratolí sobre el volum i rotar-lo fins obtenir la posició desitjada. Per moure el volum, fer clic amb el botó dret del ratolí sobre el volum i moure'l fins obtenir la posició desitjada. Per fer zoom, clicar amb el botó del mig del ratolí sobre el volum i moure el ratolí cap endavant per a apropar o cap endarrere per a allunyar.

Després podem activar el Ray Casting de volum en CPU i veurem el volum en la mateixa posició que el que esta sent renderitzat per la GPU.



Esperem uns segons, cal recordar que el Ray Casting de volum en GPU és en temps real, però en CPU és costós, tal i com s'ha analitzat al capítol de simulacions.



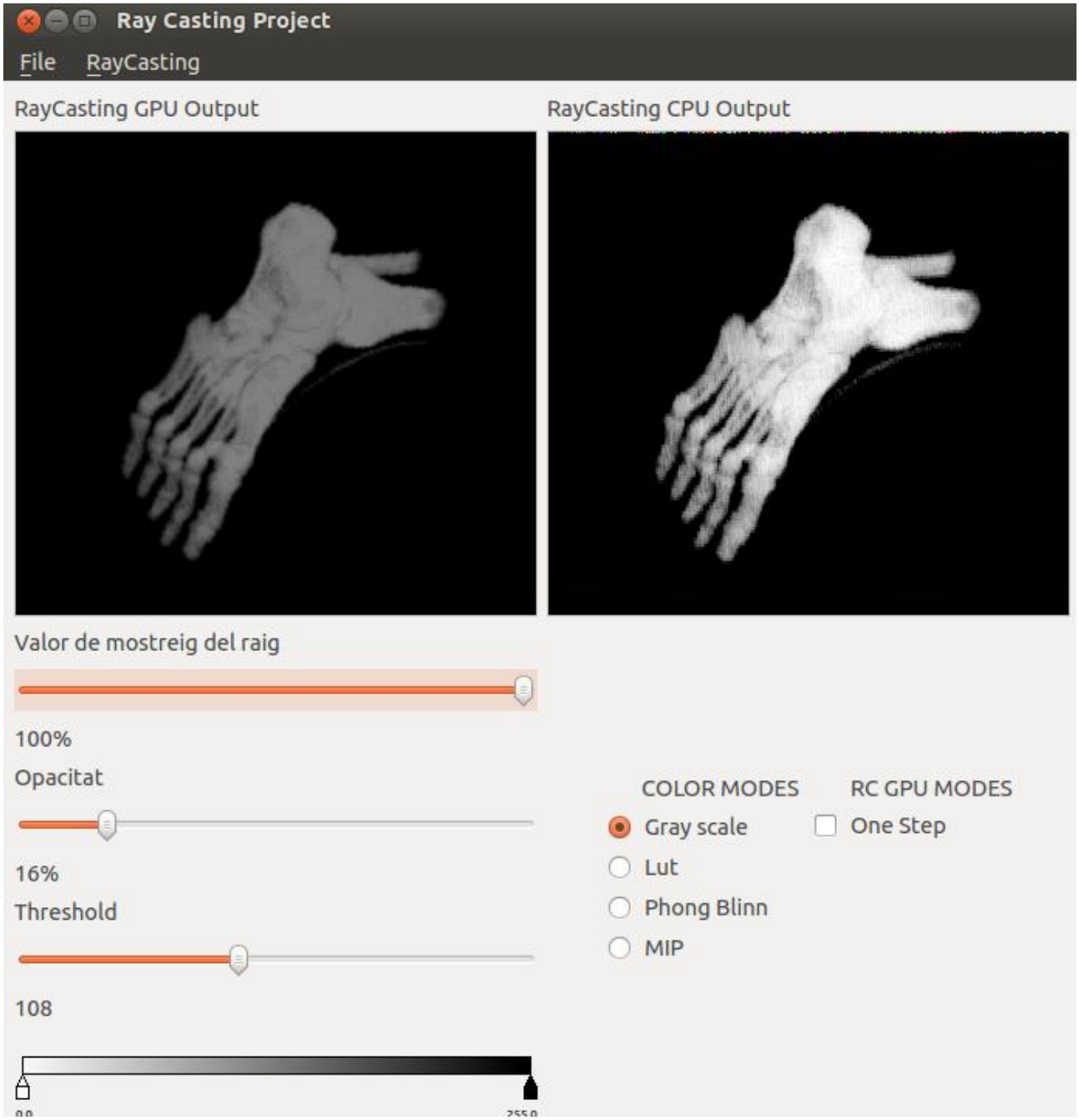
A continuació s'explicaran com funcionen els components interactius.

Com a elements d'interacció tenim els lliscadors i la taula de colors, aquesta última només pot ser utilitzada quan els modes de color LUT o Phong Blinn estan activats.

Si baixem la opacitat, obtindrem el següent resultat.

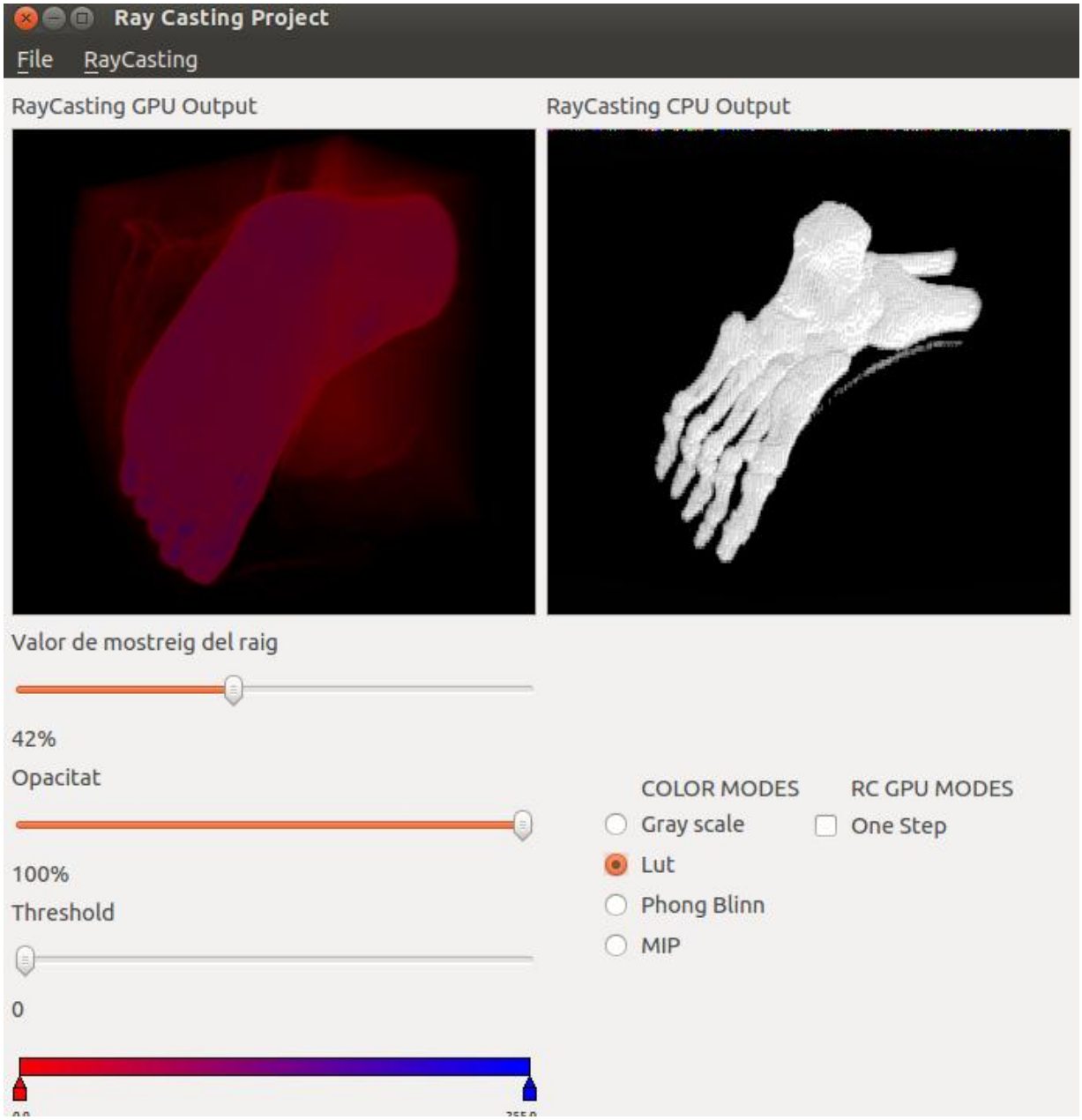
Com es pot observar, es poden veure les capes interiors del volum, sense que aquest hagi estat segmentat.

Si volem segmentar el volum, utilitzarem el lliscador de Threshold.

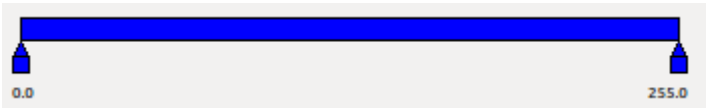


Com es pot observar, han desaparegut les capes exteriors, que en aquest cas corresponen a les que tenen densitat de vòxel inferior al valor del lliscador de Threshold.

El lliscador de valor de mostreig del raig, afectarà més al output del Ray Casting de volum en CPU, com més petit sigui el valor de mostreig, més mostres es faran al procés i més lent serà, a canvi d'obtenir una imatge més acurada del volum. A la següent imatge es pot observar com afecta el valor de mostreig del raig del volum en CPU respecte a la imatge anterior.

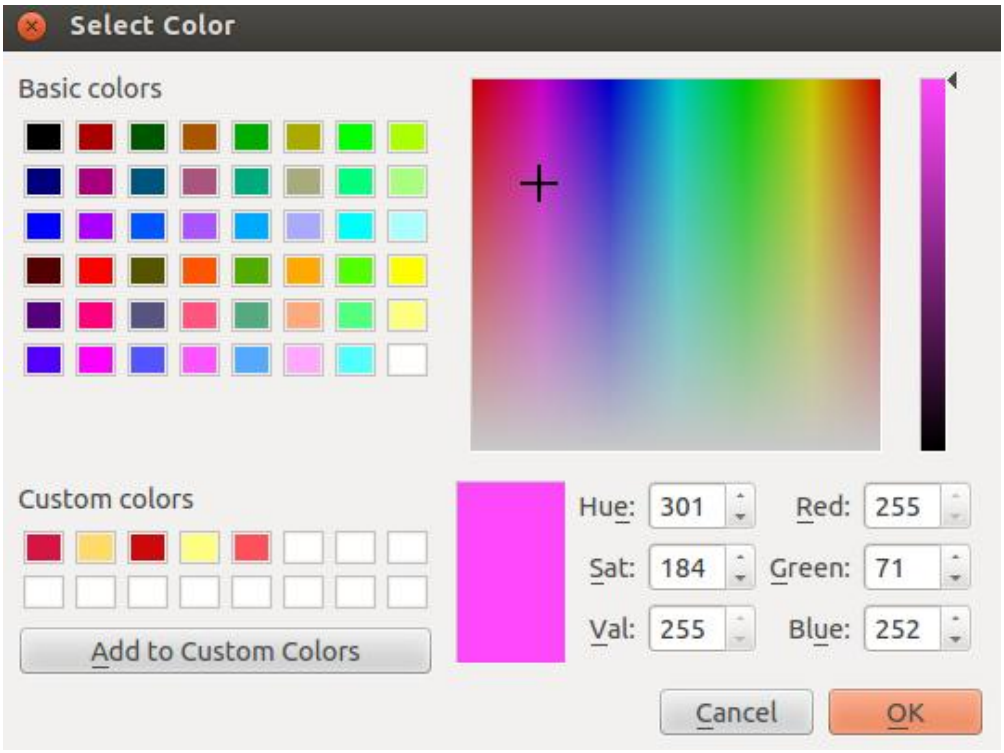


A més podem observar com s'ha canviat el mode de color. En el mode de color LUT, els colors del volum correspondran als de la taula de colors inferior.

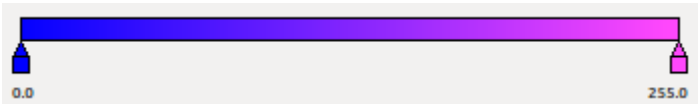


On els valors van de 0.0 a 255.0, que són el mínim i el màxim de valors que poden prendre les densitats dels volums. D'aquesta manera les mostres de color aniran en funció de la densitat del vòxel i el valor que hi hagi a la taula de colors per a aquella densitat.

La taula de colors és totalment personalitzable i pot ser canviada en temps real. Per a canviar els colors només cal fer doble clic als indicadors de colors que tenen forma de llapis. S'obrirà un diàleg el qual permet seleccionar el color desitjat.



Un cop escollit, el resultat es veurà reflectit tant a la taula de colors, com al volum renderitzat per GPU.



A més, si es volen afegir més colors o més indicadors de colors, només cal fer clic a la barra de colors i s'afegirà un nou indicador. Els indicadors dels extrems no poden ser moguts, són fixes, en canvi els nous que s'afegeixin podran ser lliscats per tota la barra. Finalment veurem tants colors com indicadors hi hagin.