UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Treball Final de Grau

Menció en ciències de la computació

Segmentation of 3D volume models in virtual reality

Oriol Vidal Teruel

Director

Pere Pau Vázquez Alcocer

ppau@cs.upc.edu

Tutora

Eva Monclús Lahoya

emonclus@cs.upc.edu

Alumne

Oriol Vidal Teruel

oriol.vidal.teruel@estudiantat.upc.edu

January 18, 2021

Resum

La visualització mèdica és un camp d'estudi molt important per a millorar el coneixement i la diagnosi sobre les diferents patologies que pateixen tota classe d'organismes, amb especial èmfasi a la humanitat. La recerca de tècniques de gràfics per computador destinades a l'avenç d'aquest camp és molt àmplia, i ha tractat diferents aspectes d'aquest procés de visualització. La segmentació és una tècnica que permet separar regions d'interès contingudes en un model volumètric més gran, oferint així una oportunitat de millorar la visualització i inspecció d'aquestes regions. L'alt rendiment que ofereixen les targetes gràfiques desenvolupades els últims anys ha permès implementar tècniques que treballen sobre models volumètrics amb resultats més ràpids, que ha permès millorar de la interacció amb les aplicacions de visualització. La realitat virtual es presenta com a un camp que ha evolucionat molt els últims anys, oferint un canvi de paradigma en la interacció amb les aplicacions. En aquest projecte implementarem algorismes de segmentació en una aplicació de visualització de models volumètrics que permet la interacció en un entorn de realitat virtual.

Resumen

La visualización médica es un campo de estudio muy importante para mejorar el conocimiento y la diagnosis sobre las diferentes patologías que sufren toda clase de organismos, con especial énfasis en la humanidad. La búsqueda de técnicas de gráficos por computador destinadas al avance de este campo es muy amplia, y ha tratado diferentes aspectos de este proceso de visualización. La segmentación es una técnica que permite separar regiones de interés contenidas en un modelo volumétrico mayor, ofreciendo así una oportunidad de mejorar la visualización e inspección de estas regiones. El alto rendimiento que ofrecen las tarjetas gráficas desarrolladas en los últimos años ha permitido implementar técnicas que trabajan sobre modelos volumétricos con resultados más rápidos, que ha permitido mejorar la interacción con las aplicaciones de visualización. La realidad virtual se presenta como un campo que ha evolucionado mucho en los últimos años, ofreciendo un cambio de paradigma en la interacción con las aplicaciones. En este proyecto implementaremos algoritmos de segmentación en una aplicación de visualización de modelos volumétricos que permite la interacción en un entorno de realidad virtual.

Abstract

Medical visualization is a very important field of study to improve knowledge and diagnosis about the different pathologies that all kinds of organisms suffer from, with special emphasis on humanity. The search for computer graphics techniques aimed at advancing this field is very broad, and has addressed different aspects of this visualization process. Segmentation is a technique that allows to separate regions of interest contained in a larger volumetric model, thus offering an opportunity to improve the visualization and inspection of these regions. The high performance offered by graphics cards developed in recent years has allowed the implementation of techniques that work on volumetric models with faster results, which has improved the interaction with display applications. Virtual reality is presented as a field that has evolved a lot in recent years, offering a paradigm shift in interaction with applications. In this project we will implement segmentation algorithms in a volumetric model visualization application that allows interaction in a virtual reality environment.

Índex

Índex de figures

Índex de taules

1 Introducció i contextualització

1.1 Introducció

A Preim i Botha (2013) es defineix el terme *visualització* com a l'ús de tècniques de gràfics per computador per a crear models interactius de representacions visuals de dades, amb l'objectiu de facilitar i millorar l'avenç del coneixement humà. La contínua millora del *hardware* que s'encarrega de gestionar i processar informació relacionada amb els gràfics ha obert un ventall de camps d'investigació molt ampli, com la visualització científica, la indústria dels videojocs i les arts visuals entre d'altres (Engel, 2006). Tan àmplia i diversa n'és la recerca que de la visualització científica es pot estudiar per separat la visualització mèdica.

Gràcies a la investigació en el marc de la visualització mèdica, els mecanismes de diagnosi i exploració del cos humà han experimentat una gran revolució des que es van publicar els primers resultats d'una mostra en 3D a partir d'una tomografia computeritzada el 1978 (Preim i Botha, 2013), tot i que a la literatura hi ha referències anteriors. La importància de l'avenç i millora de la visualització mèdica és molt important perquè es considera un procés directament vinculat a la millora de la comprensió de la informació que tenen les persones professionals del sector.

La interacció juga un paper fonamental en el disseny de sistemes de visualització mèdica. Aquesta ha d'ajudar a l'usuari a navegar pel conjunt de dades, *seleccionar* les parts que consideri rellevants durant l'exploració, comparar la informació de les diferents regions d'interès, i ajustar i perfeccionar els paràmetres de visualització que defineixen les propietats òptiques observables per l'ésser humà.

La importància per a la visualització mèdica de treballar amb estructures anatòmiques d'interès i les possibilitats de millora d'interacció que ofereixen els sistemes immersius de realitat virtual (RV) enfoquen l'eix central sobre el qual recaurà aquest Treball Final de Grau (TFG): *Algorismes de segmentació de models volumètrics 3D en un entorn de realitat virtual.*

Aquest projecte es desenvoluparà dins del grup ViRVIG (Department of Computer Science, 2020) -vegeu la figura 1, un equip de recerca enfocat a la visualització, el modeleatge geomètric i de models volumètrics, i els sistemes de realitat virtual, el qual ofereix un entorn de RV suportat pel motor gràfic Unity3D (Unity Software Inc., 2020), que permet la visualització i interacció amb models 3D obtinguts a partir de tècniques d'obtenció d'imatges mèdiques com podrien ser tomografies computeritzades (CT) o ressonàncies magnètiques (MR). Concretament, partirà del TFG *Inspecció interactiva i immersiva de models volumètrics* realitzat per Joan Fons Sànchez (Sànchez, 2017), el qual fent ús del dispositiu de RV HTC Vive (H. Corporation, 2020) posa a disposició totes les eines necessàries per implementar la funcionalitat que pretén oferir el TFG al qual es refereix aquest document.

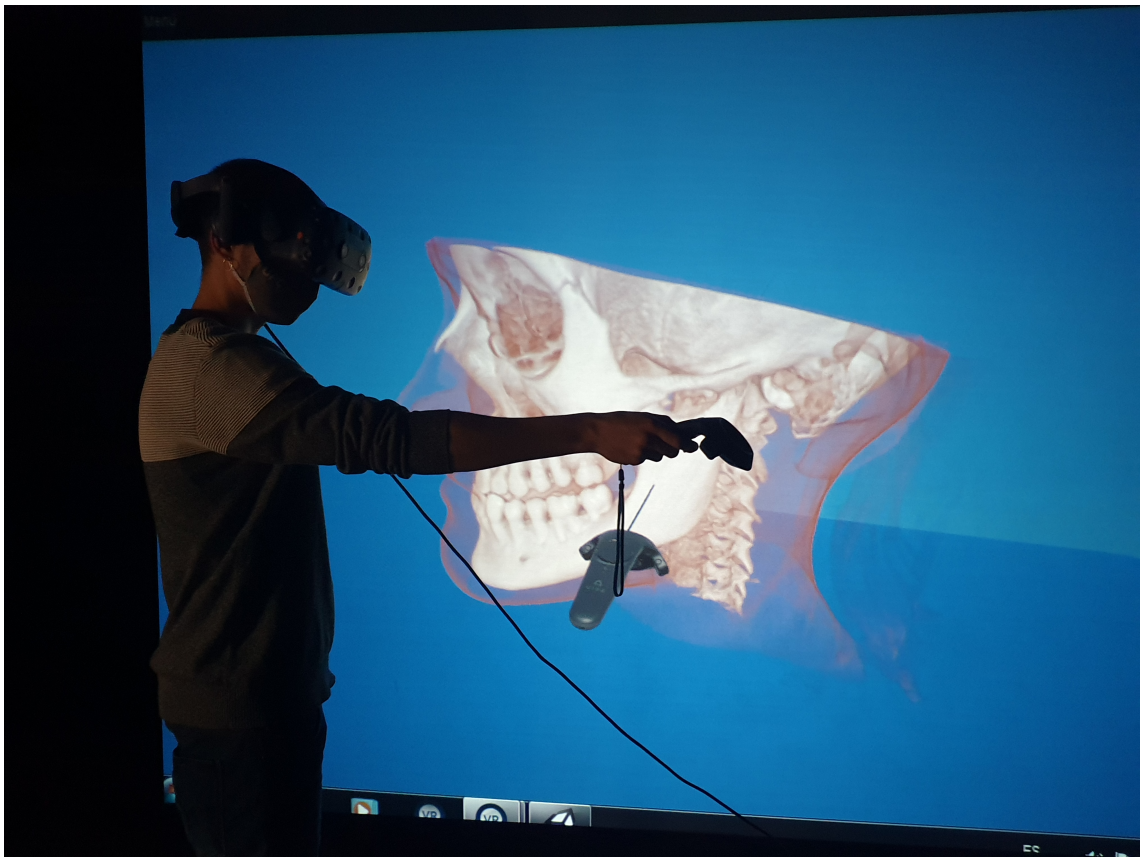


Figura 1: Usuari provant una aplicació de visualització de models volumètrics a un entorn de realitat virtual immersiu.

1.2 Formulació del problema

Els sistemes d'obtenció d'imatges mèdiques, com la CT, la MR i l'ultrasò són el centre de referència del tractament i l'estudi de malalties que afecten a l'anatomia humana i animal. Els metges i cirurgians solen utilitzar la segmentació de models volumètrics per computador per identificar i analitzar les estructures anatòmiques d'interès del conjunt d'imatges sobre les quals han de treballar. Per exemple (veure figura 2), els neuroradiòlegs solen segmentar i examinar l'artèria caròtida interna per poder determinar el grau d'estenosi de pacients que pateixen atacs isquèmics transitoris[1] (AIT). El grau d'estenosi de la caròtide és un factor crític per a determinar si els pacients d'AIT han de sotmetre's a una cirurgia per obrir el vas sanguini. Durant el procés de segmentació també es poden obtenir altres mesures, com la forma, la topologia i el volum. Per tant, la segmentació de volums és un pas essencial i important per al processament d'imatges mèdiques (Chen et al., 2006).

La complexitat de certes estructures anatòmiques i les irregularitats que poden presentar les patologies que les afecten fa que la segmentació segueixi sent a dia d'avui un problema obert. Un

[1] https://rb.gy/kst4zh, 25/07/2020

Figura 2: Eina per la segmentació 2D d'imatges mèdiques i visualització 3D de l'estructura segmentada (artèries caròtides). Aplicació en un entorn de sobretaula. Font: https://www.researchgate.net, 27/09/2020

exemple d'això és la recerca publicada a la revista *Medical Image Analysis*[2]. Moltes implementacions que ofereixen bons resultats requereixen de la intervenció manual de l'usuari sobre l'estructura anatòmica per poder obtenir un etiquetatge correcte dels objectes d'interès, ja que en molts casos el procés d'etiquetatge s'efectua seleccionant píxels d'una pantalla (2D) que mostra el model volumètric, fet que dificulta la tasca de segmentació davant de problemes més adversos.

En aquest punt ens trobem davant de dos reptes: seleccionar, estudiar i implementar tècniques de segmentació que ofereixin solucions a l'hora d'etiquetar estructures anatòmiques complexes; integrar aquestes solucions a un entorn de RV per mirar de minimitzar la intervenció manual de l'expert que vol estudiar els objectes d'interès a fi de realitzar una diagnosi.

1.3 Actors implicats

Com s'ha pogut anar intuint, aquest projecte va molt enfocat a l'ús que en pot donar la comunitat científica, i més concretament, la comunitat mèdica. És un fet que l'estudi dels algorismes de segmentació, així com la seva evolució i millora estan relacionats amb oferir solucions per a facilitar l'estudi de patologies que realitzen les persones que treballen al camp de la medicina, que a la vegada pretenen curar i millorar la salut d'altres persones i animals.

També és un fet que estudiants de medicina i veterinària es formen amb l'ús d'eines que apliquen tècniques de segmentació, ja que aquestes permeten fer estudis detallats i acurats de diferents parts del cos. A més, utilitzar volums 3D segmentats suposa una reducció de costos en material per a realitzar pràctiques i laboratoris on es pretén formar l'estudiantat, ja que l'alternativa per a

[2]journals.elsevier.com/medical-image-analysis, 28/09/2020

l'estudi de l'anatomia és treballar directament sobre cossos reals.

1.4 Definició de conceptes fonamentals

Aquesta secció pretén recopilar i oferir informació d'estudis i algorismes d'aplicacions que realitzen segmentació de volums 3D. Aquesta recerca ha permès entendre l'estat de l'art de la segmentació i la RV i saber les diferents tècniques i eines que són objecte d'estudi d'aquest camp, que ens permetrà definir posteriorment què hi ha per fer, què volem fer, i com ho voldrem fer. Començarem però, explicant el funcionament de la visualització de models volumètrics, ja que és la funcionalitat principal de l'aplicació a la que s'emmarca aquest projecte.

1.4.1 Visualització de models volumètrics

El *renderitzat* de volums (Engel, 2006) és un terme que agrupa totes les tècniques usades per representar un conjunt de dades discretament mostrejades en 3D (podeu veure el pipeline complet a la Figura 1.3). Aquestes dades reben el nom de models volumètrics i presenten una estructura diferent a la dels models geomètrics d'ordinador més comuns, formats per malles de triangles. La seva creació es pot interpretar com l'apilament d'imatges 2D, obtingudes mitjançant alguna tècnica d'obtenció d'imatges (com la CT). La figura 3 mostra un esquema general d'aquest procés.

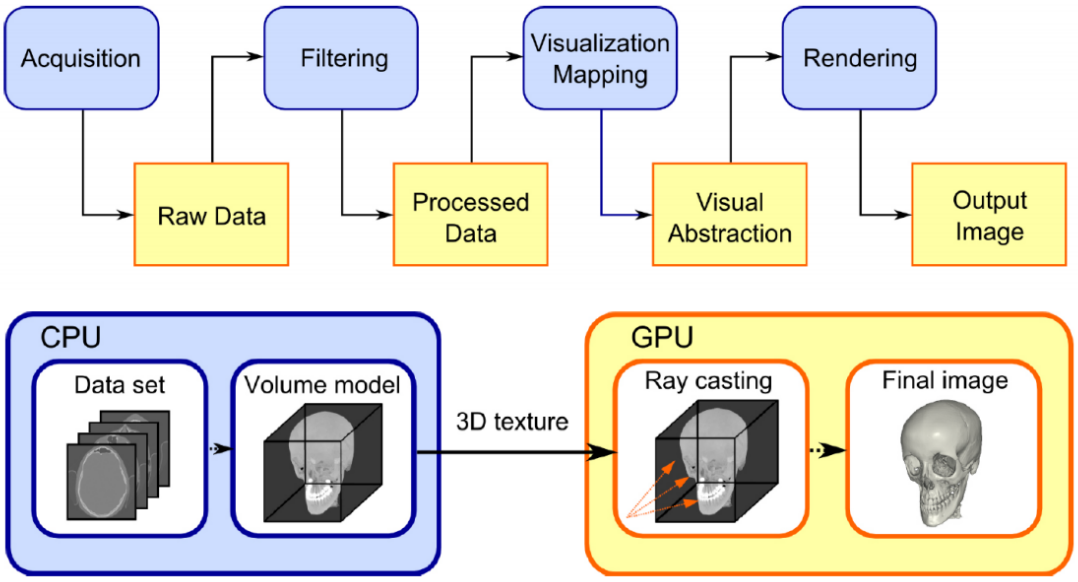


Figura 3: Procés de visualització de models volumètrics. Font: Herrera, 2020.

Una tècnica molt utilitzada és el *ray casting*[3], que consisteix en produir un raig per cada píxel de la finestra de visualització, el qual recorre l'escena i pren mostres del model volumètric en intervals regulars (figura 4). Hi ha l'interès de poder millorar la visualització de l'usuari del model

[3]Existeixen altres mètodes, però la millora de les targetes gràfiques dels últims anys ha popularitzat utilitzar aquesta tècnica.

volumètric, sobretot en una aplicació de diagnosi mèdica, ja que generalment els valors que es llegeixen dels models són valors de densitat (intensitats). És per això que durant el procés de visualització s'utilitzen les funcions de transferència, les quals assignen un color i opacitat a cada píxel del model volumètric en funció d'unes propietats concretes.

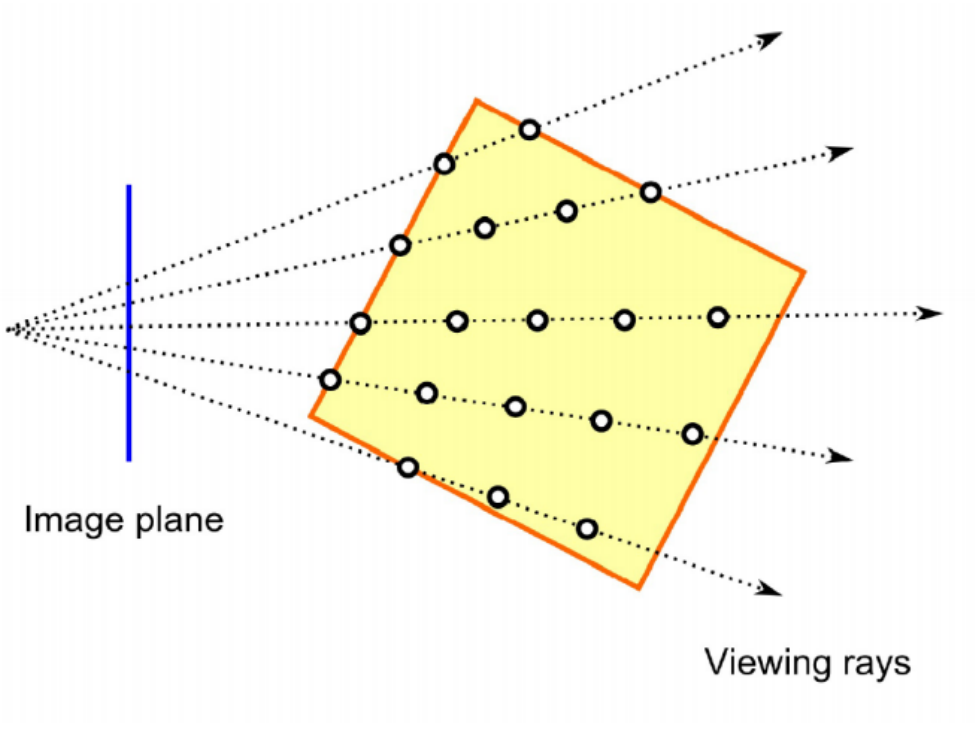


Figura 4: Funcionament de l'algorisme de *raycasting*. Font: Engel, 2006.

1.4.2 Segmentació

La segmentació (o etiquetatge) d'imatges és el procés de dividir una imatge en regions que no es creuen, de manera que cada regió sigui homogènia respecte a algun criteri de similitud predefinit (Schenke, Wünsche i Denzler, 2005). Les aplicacions de la segmentació van des de la visió per computador fins a l'anàl · lisi d'imatges mèdiques, que generalment implica un conjunt de dades d'imatges en 3D (segmentació de volum).

És possible categoritzar els mètodes de segmentació en tres classes:

- Els mètodes basats en píxels (*pixel-based methods*) classifiquen cada píxel basant-se en el seu valor característic utilitzant histogrames o altres mesures d'informació global.
- Els mètodes basats en la frontera (*edge-based methods*) utilitzen informació de les arestes i funcions de càlcul de cost mínim per poder determinar i delimitar regions homogènies.
- Els mètodes basats en regió (*region-based methods*) construeixen els límits dels objectes utilitzant criteris predefinits de similitud per determinar quins píxels han de pèrtanyer a una

certa regió.

En el marc d'aquest projecte es desenvoluparà una versió clàssica basada en regió, ja que es presenta com un algorisme bàsic però útil per a treballar amb estructures anatòmiques de difícil accés, a més, tenint en compte que el projecte del qual partim no disposa de cap eina de segmentació, és adient incorporar els algorismes més bàsics i versàtils encara que existeixin d'altres molt més sofisticats per segmentar estructures anatòmiques concretes, ja es que vol disposar de quelcom d'un abast més ampli. Més endavant se'n donaran més detalls.

1.4.2.1 GPU vs. CPU

Les GPU programables ofereixen un major poder computacional que les CPU ja que han estat dissenyades explícitament per al processament simultàni de múltiples primitives, que poden ser tractades paral · lelament amb independència. Per altra banda, les GPU ofereixen un conjunt més reduït d'instruccions i els *shaders* estan més limitats alhora de prendre i treballar amb memòria i informació global, que a més poden veure's obligats a haver d'accedir-hi vàries vegades, fet que pot suposar una pèrdua de rendiment.

Molts algorismes de segmentació es basen en calcular dinàmicament estadístiques de les imatges, díficil d'implementar a la GPU. N'és un exemple l'ús d'histogrames de les imatges, que requereixen construir una taula amb el nombre de píxels per cada valor de gris en un interval d'interès. Un mètode senzill per aconseguir això és llegir el valor de gris d'un píxel i incrementar un comptador corresponent al valor de gris a la taula. Això requereix fer escriptures a memòria, que suposa una forma de dispersament de memòria i és impossible fer-ho des d'un *fragment processor*. És per aquest motiu entre d'altres (Schenke, Wünsche i Denzler, 2005), que molts algorismes estan dissenyats híbridament per portar els càlculs complicats per a una GPU a la CPU.

1.4.2.2 Implementacions per GPU

En aquest punt és important saber quins mètodes basats en GPU exsiteixen. Sabem que estan dissenyats per oferir solucions basades íntegrament en GPU que ofereixen una millora important respecte a les solucions basades íntegrament en CPU. Conèixer quins són, com funcionen i les limitacions que presenten en ajudarà a enfocar la nostra implementació i a valorar si és necessària o pot ser útil una implementació híbrida d'aquests amb la CPU.

A Smistad et al. (2015) es defineixen els paràmetres per avaluar l'eficiència d'un algorisme implementat per GPU. Sense entrar en detall per no excedir l'extensió del document, aquests paràmetres són el (I) paral · lelisme de dades, el (II) nombre de *threads*, la (III) divergència de branques, l' (IV) ús de memòria i la (V) sincronització. A la figura 5 es pot veure de forma genèrica l'arquitectura d'una GPU.

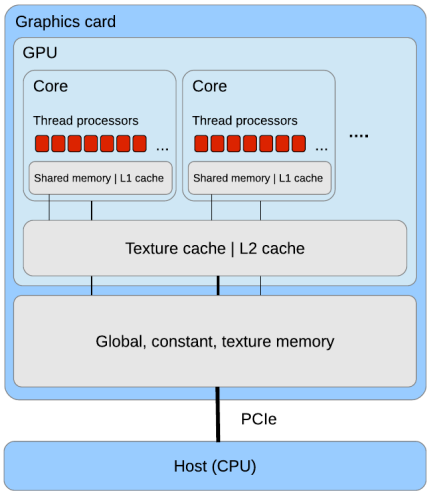


Figura 5: Disposició general d'una GPU i la seva jerarquia de memòria. Font: Smistad et al., 2015

Generalment, un algorisme obtindrà un bon rendiment de la GPU si és *data parallel*, té molts *threads*, no té molta divergència de branques, utilitza menys memòria de la total de la GPU i si utilitza la mínima sincronització possible. A continuació es presenten dos mètodes molt populars a la literatura per la seva simplicitat, la qualitat dels resultats que ofereixen i el seu rendiment.

La segmentació per *threshold* (veure figura 6) consisteix en renderitzar o descartar cada vòxel basant-se en el seu valor d'intensitat utilitzant un o més *thresholds*. Aquest mètode és completament *data parallel*, ja que cada vòxel es classifica independentment dels altres, i no requereix de sincronització. El nombre de *threads* que calen equival al nombre de píxels o vòxels. Encara que presenti divergència de branques, el mètode és massa simple, i l'ús de memòria que fa és baix, ja que només necessita espai per guardar la segmentació resultant (Smistad et al., 2015).

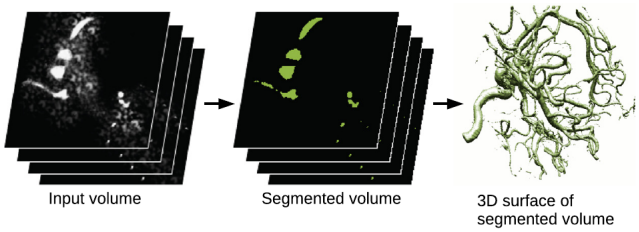


Figura 6: Segmentació per *threshold* d'un escaneig per CT. La intensitat de cada vòxel del volum d'entrada és comparada amb el valor de llindar. Si aquest és major que el valor de *threshold* el vòxel és segmentat com a part del vas sanguini. La segmentació resultant s'utilitza per generar el volum 3D que se li presenta a l'usuari. Font: Smistad et al., 2015.

Un altre mètode de segmentació molt popularitzat és el *seeded region growing*. Aquest mètode comença amb un conjunt de píxels llavor que són coneguts de pertànyer a un objecte d'interès. A partir d'aquestes llavors, les regions que contenen l'objecte d'interès s'aniran extenent pels píxels veïns si aquests satisfan alguns criteris predefinits. Aquest mètode és similar a la cerca en amplada,

i és especialment útil quan el fons i la regió d'interès tenen intensitats de píxels que es sobreposen, i estan separats en l'espai per un mur o una altra regió. Un exemple és una CT de tòrax, on els vòxels de les vies respiratòries i de la parènquima tenen una intensitat baixa, i estan separades per un teixit de sang amb intensitat alta.

Region growing permet la paral · lelització de dades, ja que tots els píxels al voltant de la vora de la regió a segmentar són evaluats amb la mateixa instrucció. Per altra banda, a mesura que la vora creix, el nombre de *threads* canvia. Això és problemàtic perquè modificar el nombre de *threads* típicament implica tornar a carregar el *kernel*[4], fet que suposa llegir valors de la memòria global un altre cop. Això introdueix divergència de branques, a més, es necessita sincronització, que també limita la velocitat. L'ús de memòria és baix (2N), ja que necessita les dades d'entrada i la informació de la segmentació resultant (Smistad et al., 2015).

Per conduir el lector al següent apartat, volem destacar que la segmentació té dos objectius fonamentals segons Preim i Botha (2013): (I) identificar els objectes rellevants, és a dir, reconèixer diferents parts particulars d'una estructura anatòmica; (II) aquests objectes han d'estar ben delimitats, és a dir, els seus contorns han d'especificar-se amb molt bona precisió. El *reconeixement* (I) és una tasca que generalment les persones poden realitzar amb millors resultats que les màquines, mentre que la *delimitació* (II) és una tasca on la precisió n'és un tret essencial. D'aquesta manera el repte de la segmentació és saber combinar la destresa de l'usuari amb el potencial d'un ordinador.

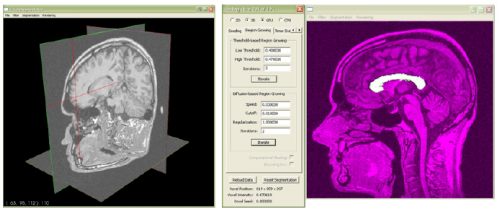
Els diferents mètodes exposats depenen de paràmetres definits per l'usuari per al seu funcionament. Els articles consultats, més enllà d'explicar l'algorisme, proposen mètodes d'interacció amb els volums per tal de poder definir bé aquests paràmetres i obtenir bons resultats. Tots aquests mètodes d'interacció requereixen a l'usuari interactuar amb el volum a través d'una pantalla i marcar els paràmetres de l'usuari a través d'un punter controlat amb el ratolí. Aquesta interacció pot arribar a ser feixuga quan es pretén segmentar objectes o regions que presenten una topologia més complexa o que intersecten amb altres objectes. A la figura 7 es poden veure algunes aplicacions proposades pels articles amb els quals ens basarem per a la implementació de la segmentació, més detallat a la següent secció.

1.4.3 Realitat virtual

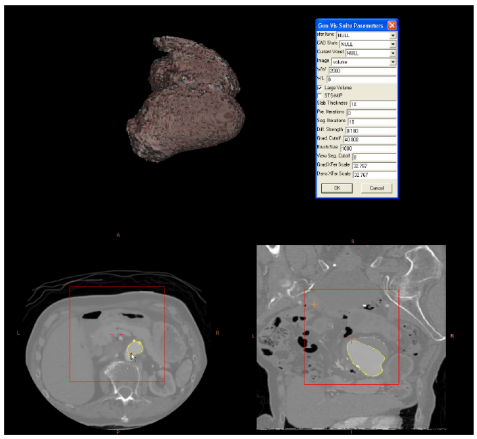
La RV és una simulació interactiva per computador des del punt de vista del participant, a la qual se substitueix o s'augmenta la informació sensorial que rep. Els elements bàsics que han d'estar presents a qualsevol sistema de RV són el modelatge digital en 3D, la visualització interactiva, la interacció implícita i la immersió sensorial (Department of Computer Science, 2020).

Els dispositius de RV es poden classificar en dues classes: semi-immersius o immersius. Els primers

[4]Denominem *kernel* a un conjunt d'instruccions que s'han d'executar sobre una única dada.



(a) Font: Schenke, Wünsche i Denzler, 2005

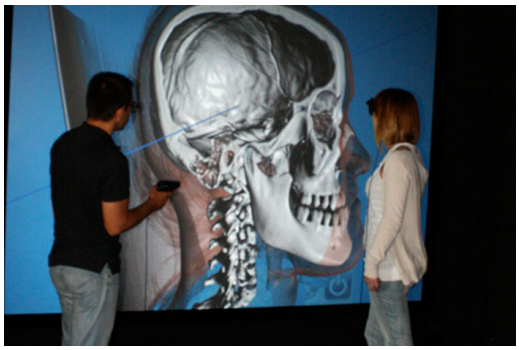


(b) Font: Sherbondy, Houston i Napel, 2003

Figura 7: Exemples d'interfície que es proposen als articles corresponents. En aquestes propostes la interacció es realitza sobre les imatges mèdiques (interacció 2D). Volem avaluar com realitzar la segmentació directament sobre un model volumètric 3D, i per tant haurem d'adaptar la interfície dissenyada originalment a un entorn 3D.

disposen d'una o més pantalles estèreo que simulen la presència d'objectes en un entorn des del punt de vista de l'usuari. Els segons consisteixen en disposar les pantalles davant dels ulls de l'usuari, de manera que cada ull veu una imatge lleugerament diferent, creant un efecte estereoscòpic. Les figures 8.a i 8.b mostren un sistema de RV immersiu i semi-immersiu, respectivament.

(a) Sistema de RV immersiu.



(b) Sistema de RV semi-immersiu.

Figura 8: (a) Dispositiu de RV immersiu HTC Vive que disposa el centre de recerca ViRVIG. (b) Sistema semi-immersiu en anatomia humana, del ViRVIG. Font: (Department of Computer Science, 2020).

En el marc d'aquest projecte, l'eina de RV ja mencionada HTC Vive que utilitzarem es tracta d'un dispositiu de RV immersiu. Aquest consta d'un monitor muntat al cap (HMD - *head mounted display*), que és el principal dispositiu de sortida de les dades, i permet veure l'entorn de RV. També disposa d'un o més comandaments que funcionen com a dispositius d'entrada juntament amb l'HMD, que també envia informació a l'aplicació com la posició i orientació del cap (veure imatge 8 a).

Tota la interacció amb l'entorn virtual es pot fer amb la gesticulació de braços i mans. A més, l'usuari també es pot moure per l'entorn per tenir punts de vista des de l'interior del model, per exemple. Els comandaments també ofereixen botons i altres elements de control, com un gallet, un *joystick*, etc. A la figura 9 es pot veure amb més detalls els components del dispositiu de RV HTC Vive.

Figura 9: Lot estàndard del dispositiu immersiu HTC Vive. Font: H. Corporation, 2020

L'avenç i millora dels sistemes de RV (veure secció 1.4.3) dels últims anys ha portat al mercat eines d'immersió d'ús genèric i fàcilment adquiribles gràcies a la reducció de costos de la producció. Com s'ha comentat anteriorment, el centre de recerca ViRVIG disposa d'un entorn de RV creat amb el motor gràfic Unity3D i que utilitza el dispositiu HTC Vive, que permet carregar i interactuar amb models volumètrics creats a partir d'imatges mèdiques. Pensem que podem mirar d'aprofitar aquestes eines per oferir als usuaris una millor experiència d'interacció amb els volums per a poder realitzar la segmentació, i comprovar si utilitzar un dispositiu de RV pot aportar més comoditat alhora de tractar amb els models a segmentar. És un fet que haurem d'avaluar la nostra proposta i comparar-la amb les que ja existeixen, que utilitzen tècniques d'interacció en 2D.

2 Abast i obstacles

2.1 Objectius i subobjectius

L'objectiu d'aquest TFG és dotar l'aplicació del grup de recerca ViRVIG de tècniques bàsiques de segmentació de forma interactiva en un entorn de RV immersiu. Els mètodes seleccionats són dels més bàsics i utilitzats per qualsevol aplicació orientada a la inspecció i edició de models volumètrics, si més no, la versió per CPU de la segmentació per *threshold* i el *region growing* basat en *threshold*. És per aquest motiu que tenir a disposició aquests mètodes pot ser molt útil per

a futures investigacions dutes a terme pel grup de recerca en el mateix marc que descriu aquest projecte.

Volem implementar els dos algorismes de segmentació de volums esmentats a secció anterior (1.4.2.2) i integrar-los al projecte ja en producció, partint de la solució per a la visualització de models volumètrics en un entorn de RV controlat amb el dispositiu HTC Vive (Sànchez, 2017). Concretament, es donarà una solució de *seeded region growing* basat en difusió implementada a la GPU publicada a Sherbondy, Houston i Napel (2003), i una solució híbrida que utilitza segmentació per *threshold* i un *seeded region growing*, també presentada per ser implementada íntegrament a la GPU (Schenke, Wünsche i Denzler, 2005), ja que pensem que són mètodes que poden ser útils com a eina de segmentació interactiva en un entorn de RV.

Durant el procés d'implementació, s'adaptaran les solucions descrites per ser executades híbridament entre la CPU i GPU, de manera que es tindran més models per comparar-ne l'eficiència. Un cop es tinguin les solucions funcionals, es plantejarà aprofundir en adaptar o modificar la interacció del projecte actual per facilitar (si és necessari) la tasca de segmentació.

Concloem aquest apartat oferint un llistat que formalitza el què acabem de comentar:

- **Objectius**
  - *Seeded region growing* per difusió implementat a la GPU.
  - *Seeded region growing* basat en *threshold* implementat a la GPU.
- **Subobjectius**
  - Adaptar les implementacions de forma híbrida (CPU - GPU).
  - Oferir millores d'interacció amb l'aplicació, si escau.

2.2 Obstacles

Un dels problemes amb els que ens podem trobar és que, el motor gràfic i el dispositiu de RV utilitzats són productes de llicència privada. Això dificulta l'abstracció d'informació d'altres articles que proposen solucions similars, ja que poden treballar amb altre software que utilitzi APIs diferents, fet que pot canviar la representació del model de dades i l'enfoc de les implementacions.

A part dels dos algorismes que s'implementaran en aquest projecte, existeixen altres solucions per la tasca de segmentació. Això provoca una gran diversitat d'implementacions que podem trobar fent més recerca, moltes d'elles pensades per segmentar estructures més específiques, mentre que nosaltres pretenem oferir una solució genèrica i adaptable a cada problema. Així doncs optarem per prendre els algorismes més populars i referenciats i adaptar-los a l'entorn sobre el qual treballarem.

Aquest projecte requereix de l'ús de material proveït pel grup de recerca ViRVIG per a les proves amb l'entorn de RV. La situació d'emergència sanitària actual no és prou estable com per garantir la total llibertat d'ús d'aquest material. El projecte desenvolupat al motor Unity3D ofereix un mode de simulació sense RV que, encara que no pot oferir els mateixos resultats, servirà per comprovar la funcionalitat dels mètodes a implementar.

2.3 Riscos

Serà molt important tenir cura amb la implementació de les solucions, ja que com s'ha mencionat, *region growing* és un algorisme costós. Es tindrà molt en compte els detalls per tal de d'obtenir solucions en temps real i poder realitzar més proves fàcilment i poder afegir, si són necessàries, les millores d'interacció. Cal tenir en compte que les aplicacions que s'executen en entorns de RV requereixen una velocitat de refresc mínim de 60 fps (Sànchez, 2017) per tal que l'experiència de l'usuari sigui bona, ja que l'alta latència pot implicar marejos.

Un altre risc amb el què ens podem trobar és que, malgrat la pujada de popularitat dels sistemes de RV, aquests han anat molt destinats a l'entreteniment, de manera que metges i estudiants no solen trobar-se amb aquestes eines alhora de treballar. Això pot provocar que encara que per nosaltres sigui fàcil fer proves d'interacció amb la solució proposada, aquesta tingui una corba d'aprenentatge més llarga, i no pugui suposar una solució eficaç per la comunitat a la que va destinada. Intentarem poder realitzar proves d'usabilitat amb agents externs per comprovar l'efectivitat de la interacció.

3 Desenvolupament del projecte

Aquesta secció conté tots els detalls de la implementació de la solució presentada fins ara. S'han identificat tres topics centrals que utilitzarem per explicar tot el procés de desenvolupament de l'aplicació. Concretament, a 3.2 i 3.3 es donaran detalls dels algorismes de segmentació implementats per ser executats íntegrament a la CPU i a la GPU, respectivament. A 3.4 es donaran detalls de les dues metàfores d'interacció que s'han implementat per provar l'aplicació.

A cada secció es donarà una definició formal del problema que estem tractant (3.2.1, 3.3.1 i 3.4.1, respectivament) i s'explicarà com l'hem implementat a l'entorn gràfic de Unity3D (3.2.2, 3.3.2 i 3.4.2, respectivament). Per la part de la implementació dels algorismes (3.2 i 3.3) també es donaran detalls de proves de rendiment que ens han permès avaluar l'escalabilitat de la nostra aplicació.

Començarem, però, explicant els aspectes més importants de l'aplicació de la qual partim, explicada amb més detall a Sànchez (2017), ja que el seu enteniment ha estat molt important per poder implementar la nostra solució.

3.1 Visualització de models volumètrics

L'aplicació *core* d'aquest projecte compleix la tasca de visualitzar (*renderitzar*) el model volumètric que un usuari ha escollit. L'algorisme de visualització és un *Muli-pass Raytracing* que es realitza íntegrament a la GPU. Aquest consisteix en un *fragment shader* on s'envia tota la informació necessària per poder assignar a cada vòxel del volum una opacitat i un color utilitzant un algorisme de *raycasting* (Engel, 2006) que consulta una funció de transferència per obtenir aquests valors. La figura 10 mostra el *pipeline* simplificat de l'aplicació original.

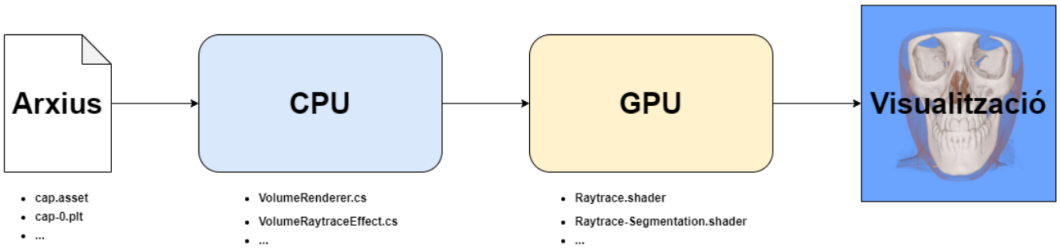


Figura 10: Fil principal d'execució de l'aplicació. Podem veure que les dades d'entrada són el model volumètric (fitxer **.asset*) i la funció de transferència (fitxer **.plt*). El fitxer *VolumeRenderer.cs* gestiona el model de dades de l'aplicació (textura d'intensitats del volum, informació de la segmentació i funció de transferència), i el fitxer *VolumeRaytraceEffect.cs* gestiona la lògica que s'enviarà als *shaders* que aplicaran l'algorisme de *raycasting* per visualitzar el model volumètric. Font: Herrera, 2020.

Per interès d'aquest projecte, l'aplicació ofereix la possibilitat de visualitzar informació de segmentació del model volumètric si es carrega un *Asset* de Unity (Unity Software Inc., 2016) que conté l'etiquetatge dels vòxels del volum. En aquest cas, el *pipeline* comentat a la figura 10 assigna un *shader* (fitxer *Raytrace-Segmentation.shader* de la figura 10) que *renderitzarà* el vòxel segons l'etiqueta de segmentació. Aquest nou *shader* rep una segona textura 3D amb les etiquetes de cada vòxel (valors entre 0 i 255, normalitzats) i una textura 2D on cada posició conté els colors associats a les etiquetes corresponents (l'etiqueta 254 està reservada per indicar l'"absència de segmentació").

Així doncs, l'usuari podrà visualitzar les regions d'interès marcades per l'*Asset* de segmentació. Per la nostra part, ens podrem centrar en la tasca que ens pertoca sense perill de trencar la implementació original a l'hora d'integrar la nostra solució.

3.2 Segmentació per CPU

L'ús de la CPU per a la implementació i execució d'algorismes de segmentació ha estat un pas essencial per iniciar el desenvolupament. La caracterització iterativa dels algorismes presentats fa que sigui senzill obtenir una primera versió de cada un d'ells, que ens ha ajudat a entendre el

seu funcionament (i les seves limitacions) i el model de dades que té l'aplicació de visualització de models volumètrics que estem utilitzant.

3.2.1 Formulació del problema

Ens disposem a explicar amb més detall els dos algorismes de segmentació implementats en aquest TFG. Les característiques principals que es detallen sobre els algorismes a 3.2.1.1 i 3.2.1.2 s'obviaran a la secció d'implementacions per GPU, que es centrarà en introduir els canvis que apareixen alhora de portar la implementació per a ser-hi executada (per a més informació, vés a 3.3).

3.2.1.1 Segmentació per llindar desenvolupat a la CPU

Tal com s'explica a la secció 1.4.2.2, la segmentació per llindar (*threshold*) consisteix en decidir si un vòxel pertany o no a una estructura d'interès en funció de si el seu valor d'intensitat compleix uns requisits marcats per un o més valors de llindar. A la literatura hem trobat que existeixen diferents mètodes basats en *threshold* que es diferencien els uns dels altres en com relacionen els valors de *threshold* i els valors d'intensitat dels vòxels. Per exemple, a la figura 6 es seleccionen aquells vòxels que tenen un valor d'intensitat major que el valor de *threshold*. En canvi, per altres models volumètrics o objectes d'interès ens podria interessar fer l'operació inversa.

Pel nostre projecte, hem decidit treballar amb un rang de valors de *threshold*, de manera que es pugui treballar amb models volumètrics que presentin més de dos rangs d'intensitat, i poder així seleccionar aquelles regions d'interès que no presenten els valors d'intensitat més alts ni més baixos.

És ben cert que ens podem trobar amb que alguns vòxels que pertanyen a una certa regió tinguin valors d'intensitat que són estrictament més grans o petits que els valors màxim o mínim de *threshold*, respectivament. Això sol passar sobretot quan aquests vòxels es troben a les fronteres entre dues o més regions d'interès. Per oferir més flexibilitat a l'algorisme davant d'aquest problema, es sol afegir una tolerància τ al valor o valors de *threshold*. Pel nostre cas, aquesta tolerància serà un percentatge que indicarà quant més gran ha de ser el valor de *threshold* màxim i quant més petit ha de ser el valor mínim, respecte els valors originals.

D'aquesta manera, la funció per decidir si cal etiquetar o no un vòxel serà la següent:

$$\_segmID(voxel) = \begin{cases} \_selectedID & \_VolumeIntensity(voxel) \in [threshold_{min} - \tau, threshold_{MAX} + \tau] \\ \_emptyID & otherwise \end{cases} \tag{1}$$

La seva senzillesa limita el seu ús a certes modalitats d'imatges mèdiques (com la CT) i a estructures ressaltades prèviament per contrast (com el cas de la figura 6) o per segmentar regions d'interès

clarament distingibles per la seva densitat (exemple??). Més endavant també veurem exemples de la falta de precisió que ens ofereix un mètode tan poc sofisticat com aquest. L'algorisme 1 mostra el pseudo-codi de la segmentació basada en *threshold*.

Algorithm 1 Segmentació per threshold a la CPU. $O(V)$

1: $V \leftarrow VolumeTexture$
2: S ▷ texture that will contain segmentation labels
3: $threshold_{min}$, $threshold_{MAX}$, τ
4: **for** $x = 1, 2, \ldots, V.resolution.x$ **do**
5: **for** $y = 1, 2, \ldots, V.resolution.y$ **do**
6: **for** $z = 1, 2, \ldots, V.resolution.z$ **do**
7: $voxel \leftarrow (x, y, z)$
8: $S[voxel] \leftarrow ComputeVoxelID(V, voxel, threshold_{min}, threshold_{MAX}, \tau)$ ▷ ComputeVoxelID is a function similar to equation 1.
9: **end for**
10: **end for**
11: **end for**

3.2.1.2 Segmentació per creixement de regió basat en llindar desenvolupat a la CPU

Els algorismes de segmentació basats en el creixement d'una regió es caracteritzen per etiquetar els vòxels en funció de la zona on es troben, de manera que intuïtivament la funció que avalua l'etiqueta d'un determinat vòxel (avaluant si la intensitat del vòxel es troba dins d'un rang de valors de *threshold*, per exemple) s'aplicarà sobre aquells vòxels que es trobin en una regió d'interès determinada per l'usuari. Aquesta característica fa que aquests algorismes siguin més sofisticats que els comentats a la secció anterior, ja que en un principi discriminarà tots aquells vòxels que no sigui propers a la regió d'interès, reduïnt així el nombre d'iteracions respecte la segmentació per *threshold*, que com es veu a l'algorisme 1, recorra sempre tot el volum.

Per exemple, en un crani humà els valors de intensitat del maxil · lar són ben diferenciats al de les dents. Com que *region growing* discrimina vòxels per valor i per ubicació, podem separar el maxil · lar superior de l'inferior si apliquem l'algorisme amb tres *seeds* i etiquetes diferent, com es veu a la figura 11. Aquest resultat és impossible d'obtenir aplicant la segmentació per *threshold*.

Com es comenta a la introducció d'aquesta secció (vés a 3.2), el pas d'implementar els algorismes a la CPU ha servit com a presa de contacte amb ells i l'aplicació, ja que gairebé tota la metodologia present en aquest TFG (models volumètrics, algorismes de segmentació i processament d'informació a la GPU) estava fora de l'abast de les assignatures del grau, de manera que com a desenvolupador he pogut anar incorporant gradualment els diferents conceptes i procediments que requereix aquest projecte. Tenir implementacions per les dues unitats de processament (CPU i GPU) també ha servit

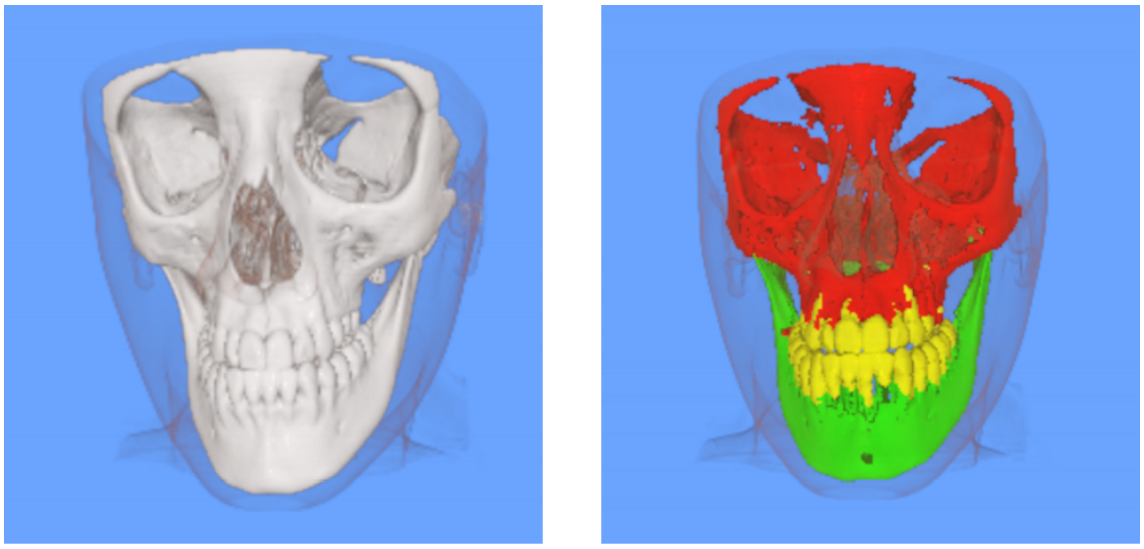


Figura 11: A l'esquerra, es veu el model volumètric del crani. A la dreta, es veu el mateix model on les tres regions d'interès s'han etiquetat per poder-les diferenciar. Font: Herrera, 2020.

per fer un estudi de primera mà de l'eficiència que es guanya amb la capacitat de paral · lelització de les targetes gràfiques. És per això que per aquesta part del desenvolupament s'ha escollit implementar un esquema iteratiu de *region growing* amb el criteri de decisió basat en *threshold*, mentre que la versió de la GPU tindrà un criteri de decisió per difussió, una tècnica més costosa però que ofereix a l'usuari més control sobre l'evolució de la segmentació (per a més informació, vés a 3.3).

Region growing parteix d'un o més vòxels que l'usuari defineix per formar part d'una o més regions d'interès. A partir d'aquests vòxels, que es solen anomenar llavors (*seeds*), es marquen els seus vòxels veïns com a pendents de visitar. A partir d'aquí, s'avaluarà cada vòxel pendent per visitar i, si el criteri de selecció decideix que el vòxel s'ha de segmentar, es marcaran els seus veïns com a pendents d'avaluar.

La figura 12 mostra un exemple d'execució d'un *region growing*, i l'algorisme 2 mostra un pseudocodi molt similar a la solució que hem implementat en aquest projecte.

Podem veure que la complexitat dels dos algorismes presentats és la mateixa, però la caracterització del *region growing* de discriminar vòxels també per la seva ubicació portarà a realitzar menys iteracions que la segmentació purament basada en *threshold*, que sempre recorrerà tot el volum. A la secció 3.2.3 veurem més en detall aquesta consideració.

3.2.2 Integració a Unity3D

S'ha creat una jerarquia de classes per definir els algorismes descrits i la resta de solucions que presentarem en aquest projecte per facil · litar la reutilització de les característiques que totes elles

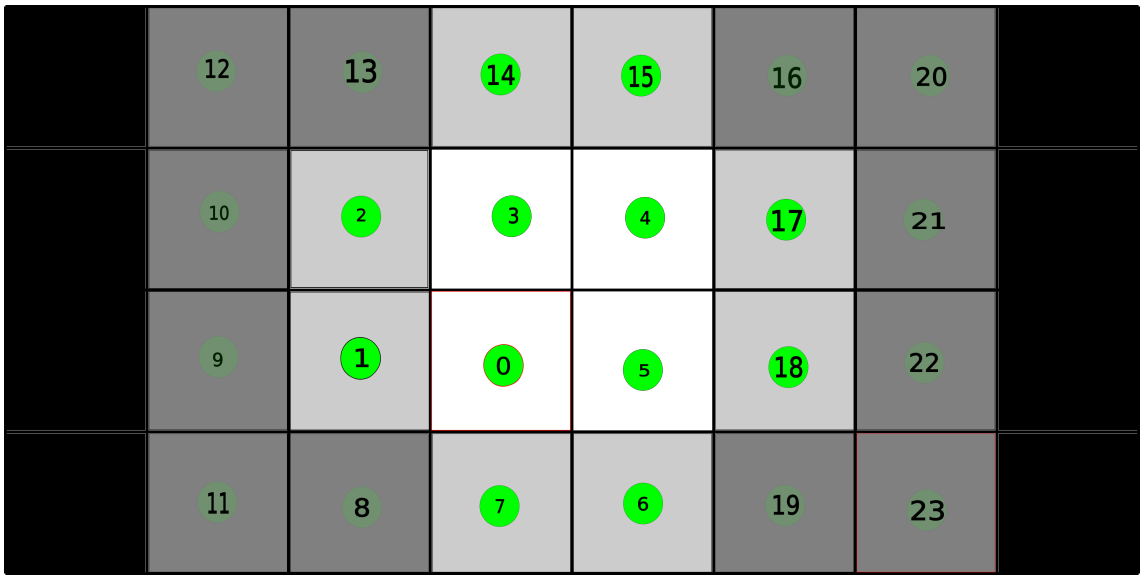


Figura 12: Exemple d'execució d'un *region growing* basat en *threshold* sobre una imatge. Cada píxel que l'algorisme ha visitat s'etiqueta amb un cercle verd opac o transparent en funció de si el píxel compleix o no el criteri de segmentació. A més, les etiquetes contenen un nombre que indica en quin ordre s'ha avaluat el píxel. Aquest exemple comença amb una *seed* (píxel 0), i segmenta tots els píxels que el seu valor d'intensitat sigui igual a 1 (blanc) amb una tolerància del 20% per a valors més petits (0.8, gris clar). Podem observar que després de descartar els píxels amb valor 0.5 (gris), ja no posa a la cua els píxels amb valor 0 (negre). Elaboració pròpia.

Algorithm 2 Region Growing basat en threshold a la CPU. $O(V)$

$V \leftarrow VolumeTexture$
S ▷ texture that will contain segmentation labels
$C[|V|]$ ▷ Constant access data structure to check visited voxels. $space = O(V)$.
$threshold_{min}$, $threshold_{MAX}$, τ
$Q \leftarrow seed$ ▷ It may content more than one seed.
while $!Q.empty()$ **do**
    $voxel \leftarrow Q.head()$ ▷ head() function retrieves and removes queue's head element.
    $S[voxel] \leftarrow ComputeVoxelID(V, voxel, threshold_{min}, threshold_{MAX}, \tau)$ ▷ ComputeVoxelID is a function similar to equation 1.
    $C[voxel] \leftarrow true$
    for $neighbour$ *in* $voxel.neighbours()$ **do**
        if $C[neighbour] == false$ **then**
            $Q.enqueue(neighbour)$
        end if
    end for
end while

comparteixen. A partir d'aquí, a través del sistema d'interacció que oferim als usuaris (veure secció 3.4) es poden crear instàncies d'aquests algorismes amb diferents paràmetres, que alhora ofereixen un mètode *Run()* que implementa les idees exposades als algorismes 1 i 2, respectivament, de manera que es modifica la textura de segmentació del volum que ofereix l'aplicació sobre la que estem treballant (per a més informació, vés a 3.1). La figura 13 mostra la modificació del *pipeline* de la figura 10 un cop implementats els algorismes de segmentació per CPU.

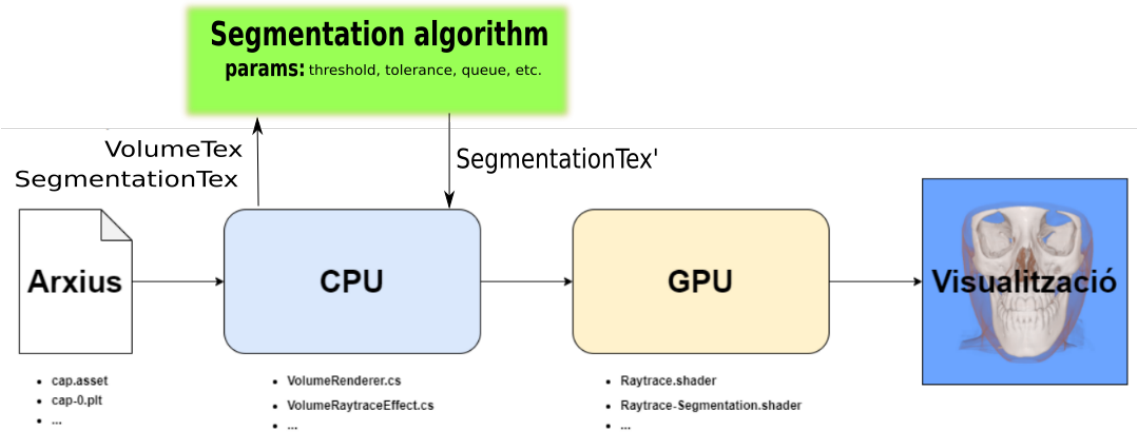


Figura 13: *Pipeline* de l'aplicació que incorpora la funcionalitat de segmentar regions d'interès del volum a través del còmput d'algorismes de segmentació per la CPU. Cada algorisme de segmentació té els seus propis paràmetres, definits per l'usuari des d'una metàfora d'interacció. Aquests prenen la informació del model volumètric que necessiten i modifiquen la textura de segmentació (*SegmentationTex* → *SegmentationTex'*). A partir d'aquí, el *pipeline* segueix el mateix curs per a *renderitzar* el model volumètric i la segmentació.

Per a poder provar la funcionalitat dels algorismes d'una forma més còmode en situacions on no es disposa l'equip de RV, s'ha implementat una finestra a l'editor de Unity3D que permet modificar-ne els paràmetres (veure figura 14).

La figura 15 mostra un exemple de l'aplicació de visualització amb la funcionalitat de segmentar per la CPU les regions d'interès.

Un cop hem aconseguit una primera implementació de segmentació per *threshold* i un *region growing*, hem pogut observar clarament les ventatges del segon respecte el primer anal · litzats al final de la secció 3.2.1. Per exemple, la figura 16 mostra diferents resultats alhora de voler segmentar l'esfera més gran de les interiors segons quin mètode de segmentació utilitzem. Podem observar que en aquest model volumètric alguns vòxels de cada esfera tenen valors d'intensitat que no corresponen a la majoria, deixant en evidència la imprecisió de la segmentació per *threshold*.

A la següent secció mostrarem l'estudi del rendiment de les implementacions explicades fins ara. Mencionem aquí, però, que quan la textura de segmentació del model és modificada, cal fer una crida a l'API de Unity3D (*SetPixels*) per informar a la GPU que la informació de la textura

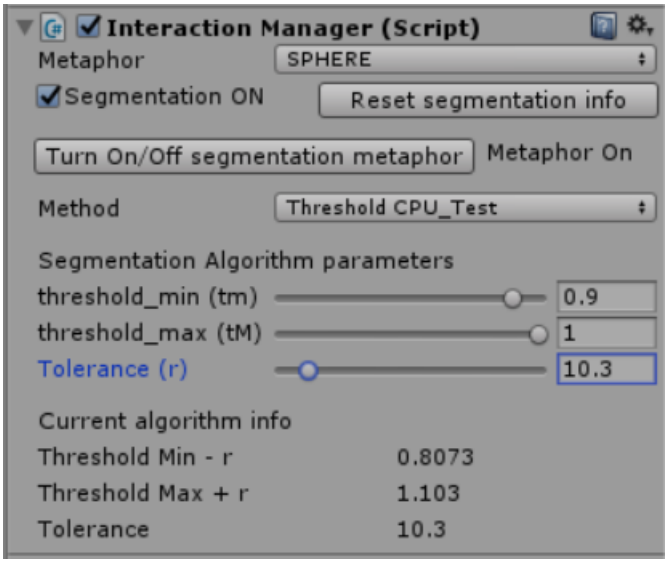
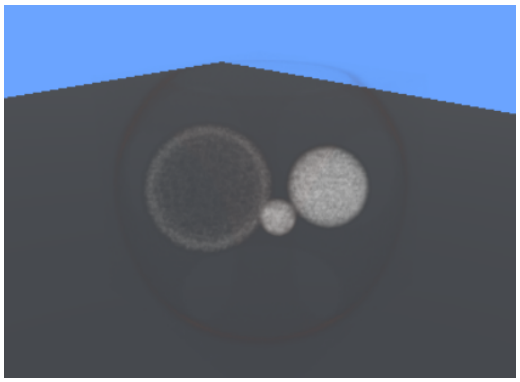
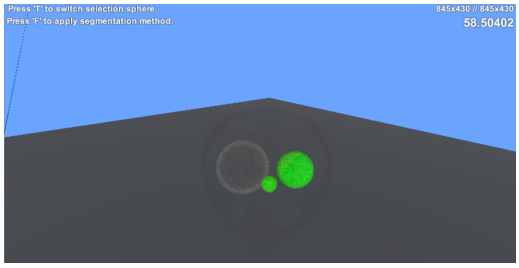


Figura 14: Editor de Unity3D que permet visualitzar i modificar la segmentació d'un model volumètric.

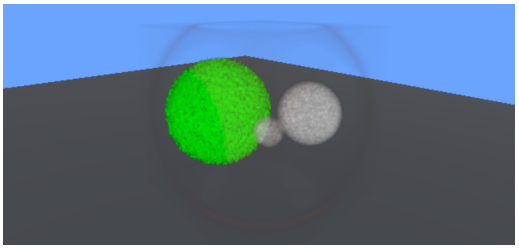


(a) Model volumètric.

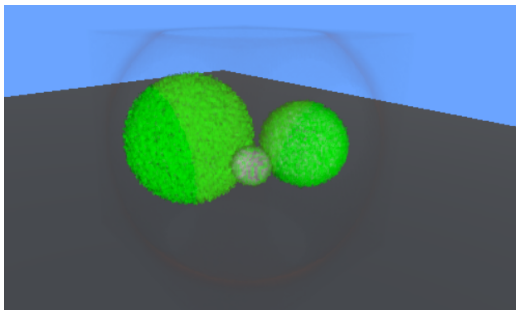


(b) Model volumètric, segmentat.

Figura 15: A l'esquerra, un model que representa tres esferes que es troben dins d'una més gran. Totes elles presenten valors d'intensitat diferents. A la dreta es pot veure la segmentació de l'esfera mitjana i petita segons els paràmetres definits a l'editor de la figura 14.



(a) Segmentació basada en regió per *threshold* de l'esfera gran.



(b) Segmentació per *threshold* de l'esfera gran.

Figura 16: Segmentació de l'esfera gran del model volumètric. A l'esquerra, s'ha col · locat una llavor a l'esfera i s'ha executat un *region growing* amb els valors de *threshold* que s'han obtingut llegint el valor d'intensitat del vòxel (0.78) on l'hem col · locat, amb una tol · lerància de l'11.7% per tenir un rang de *threshold* de [0.69, 0.87], que es sap que és el rang de valors que oscil · la entre els vòxels de l'esfera. A la dreta, veiem que aquest rang també prèn vòxels de les altres esferes, de manera que la segmentació per *threshold* etiqueta aquests vòxels.

ha canviat. Aquest pas d'informació penalitza molt el temps de refresc de l'aplicació. És per això que encara que la següent secció mostri diferents aspectes de rendiment dels algorismes, anticipem al lector que la penalització del pas d'informació a la GPU ens va evidenciar la necessitat d'implementar solucions (també d'interacció, vés a 3.4) íntegrament a la GPU.

3.2.3 Anàl · lisi de rendiment

Per a estudiar l'escalabilitat de la nostra aplicació disposem de tres models volumètrics de diferents resolucions (128x128x128, veure figura 15; 256x256x256, veure figura 17; i 512x512x256, veure figura 11), que ens permetran observar com afecta el nombre de vòxels a visitar als algorismes implementats. A més, s'han fet proves amb equips diferents per veure com canvia la velocitat d'execució en funció del *hardware*.

S'ha utilitzat la funcionalitat *Profiler* de Unity3D per observar el consum de CPU i GPU de l'execucions que estem anal · litzant. La documentació (Unity Software Inc., 2016) adverteix que tenir aquesta funcionalitat activada augmenta el consum de recursos de l'aplicació, però els resultats ofereixen una intuïció del rendiment real de l'aplicació. La figura 19 mostra un exemple d'una captura del *Profiler*.

La figura 20 mostra el temps d'execució de CPU de la segmentació per *threshold* als diferents models comentats. La figura 21 mostra el temps d'execució del *region growing* basat en *threshold*, però mostra el temps d'execució en funció del nombre de vòxels a visitar, ja que com s'ha comentat, *region growing* no visita tots els vòxels del model volumètric, de manera que si la regió d'interès d'un model amb més resolució comprèn menys vòxels que una altra regió d'interès d'un model de més baixa resolució, obtindrem millor *performance* a la primera execució.

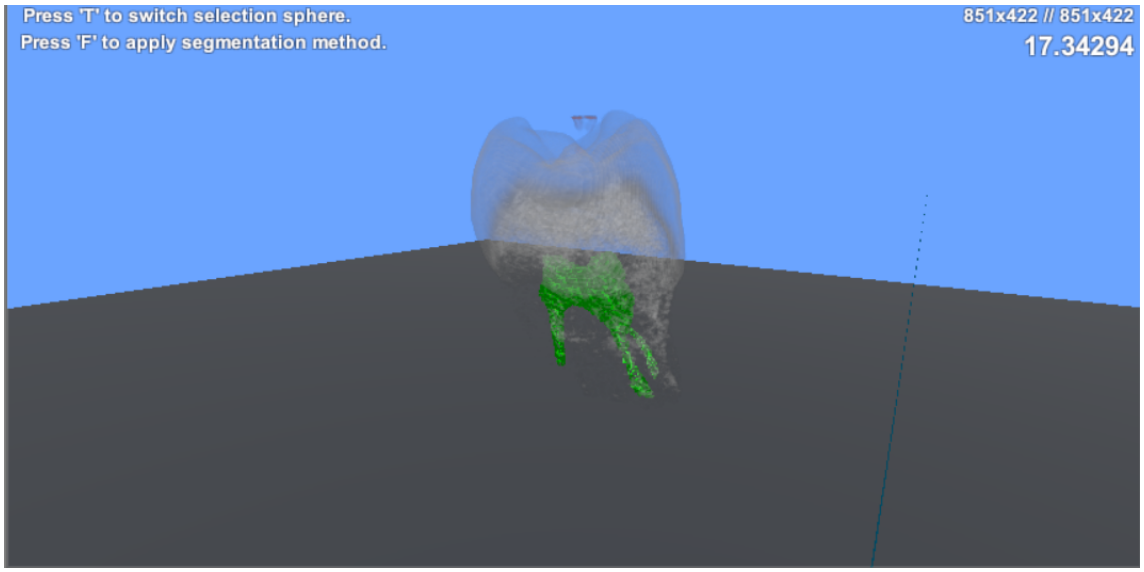


Figura 17: Model volumètric d'un queixal amb resolució 256x256x256, del qual se n'ha segmentat l'arrel mitjançant un *region growing* basat en *threshold*.

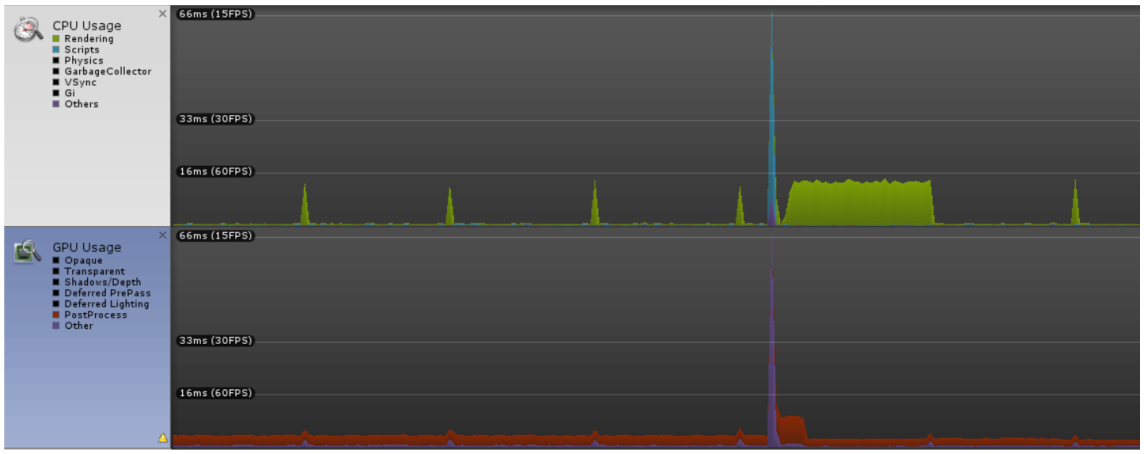


Figura 18: Gràfic del *Profiler* de Unity.

Figura 19: Un gràfic que mostra una trama del consum de CPU i GPU durant un conjunt de *frames*. Podem veure un pic quan s'està executant un algorisme de segmentació.

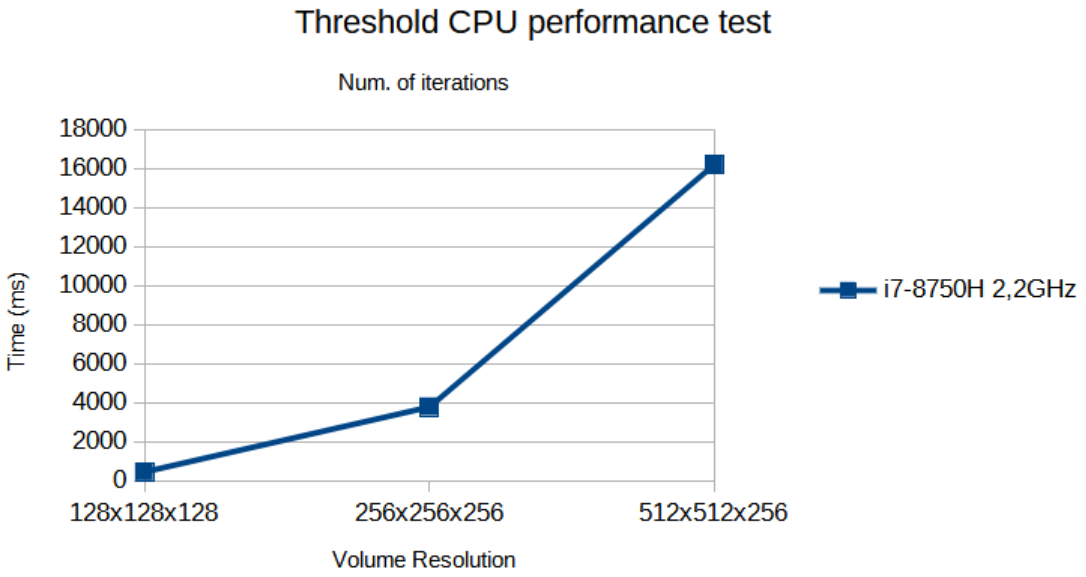


Figura 20: Temps d'execució de la segmentació per *threshold* a la CPU de models volumètrics de diferents resolucions. Elaboració pròpia.

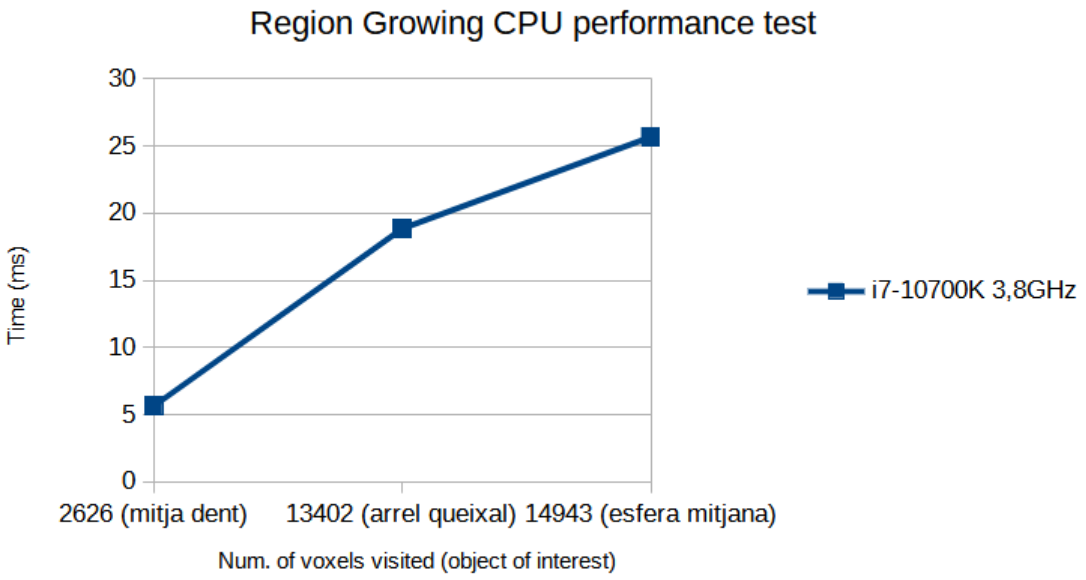


Figura 21: Temps d'execució d'un *region growing* basat en *threshold* a la CPU de models volumètrics de diferents resolucions. Elaboració pròpia.

La informació recopilada fins ara mostra que *region growing* es presenta no només com un algorisme més sofisticat sinó que més eficient que la segmentació per *threshold*, que encara que sigui més trivial, el fet d'haver de recórrer sempre tot el volum el fa més lent. De totes maneres, *region growing* tampoc escala bé quan augmentem el nombre de vòxels a visitar, i totes les proves d'execució dels algorismes per CPU han comportat una congelació del refresc de l'aplicació d'entre 10 i 15 segons. Concloem doncs, afirmant la inviabilitat d'implementat algorismes de segmentació per la GPU. A la següent secció veurem que com la GPU millora els temps d'execució.

3.3 Segmentació per GPU

Com ja s'ha comentat a la secció 1.4, la capacitat de paral · lelització de les targetes gràfiques les converteix en una eina molt potent alhora de realitzar un nombre molt gran d'operacions si aquestes permeten, evidentment, el càlcul simultani. Com que les característiques fonamentals dels algorismes s'han explicat a la secció 3.2, aquí ens centrarem en explicar els conceptes claus per a portar les seves implementacions a la GPU (apartats 3.3.1.1 i 3.3.1.2) i la difusió de Perona i Malik (Sherbondy, Houston i Napel, 2003) per al *region growing*, una variant més costosa però que ofereix millors resultats i més flexibilitat per decidir com creix la segmentació (3.3.1.3), ja que es presenta com un mètode que permet fer una segmentació interactiva.

3.3.1 Formulació del problema

L'objectiu de portar la implementació dels algorismes de segmentació a la GPU és aconseguir etiquetar els vòxels en paral · lel, ja que com hem vist, el nombre d'iteracions i el pas d'informació de la CPU a la GPU és el què en penalitza els temps d'execució (de l'ordre de 10 a 15 segons amb la millor targeta gràfica que disposàvem al Centre de Realitat Virtual). Per altra banda, ens interessa treballar sense trencar el *pipeline* de l'aplicació original (veure figura 10). Primer veurem la solució que hem trobat a la literatura sobre com modificar la destinació d'un *fragment shader* per guardar-ne la sortida a una textura (3.3.1.1), després veurem l'ús de *Compute Shaders* (3.3.1.2) per simular aquesta tècnica com a solució a les complicacions que hem trobat al llarg del desenvolupament a l'haver d'utilitzar l'API de Unity3D per redireccionar la sortida de *fragment shaders*.

3.3.1.1 Render to Texture

Aquest és el nom amb el que popularment es refereix a *renderitzar* una escena (o un determinat objecte) a una textura, en comptes de fer-ho directament a un dispositiu de sortida de la targeta gràfica. Aquesta tècnica consta de tres tasques: crear la textura on volem *renderitzar*; *renderitzar*-hi alguna cosa; utilitzar-la (en el nostre cas, pel *pipeline* de visualització).

Les diferents APIs que ens permeten configurar el *pipeline* de *renderitzat* de les GPU ofereixen versions diferents de la mateixa funcionalitat. Per exemple, a OpenGL cal crear un *frame buffer* ($glGenFramebuffers(1, \&fbo)$) i assignar-lo com a actiu (*glBindFramebuffer(GL_FRAMEBUFFER, fbo)*) i crear una textura i assignar-la al *frame buffer* (*glFramebufferTexture()*) (Vries, 2021). DirectX ofereix una amigable interfície per obtenir el mateix procediment ($CreateRenderTargetView()$) (Soft, 2015).

En el marc de la nostra aplicació volem projectar l'etiqueta de segmentació de cada píxel en una textura 3D, però sabem que la sortida d'un *fragment shader* només correspon a valors per un espai

(*viewport*) 2D, de manera que cal adaptar el procediment comentat per a *renderitzar* a la textura 3D llesca a llesca. La figura 22 mostra com podem concebre una textura 3D com a un conjunt de llesques (*slices*) de textures 2D.

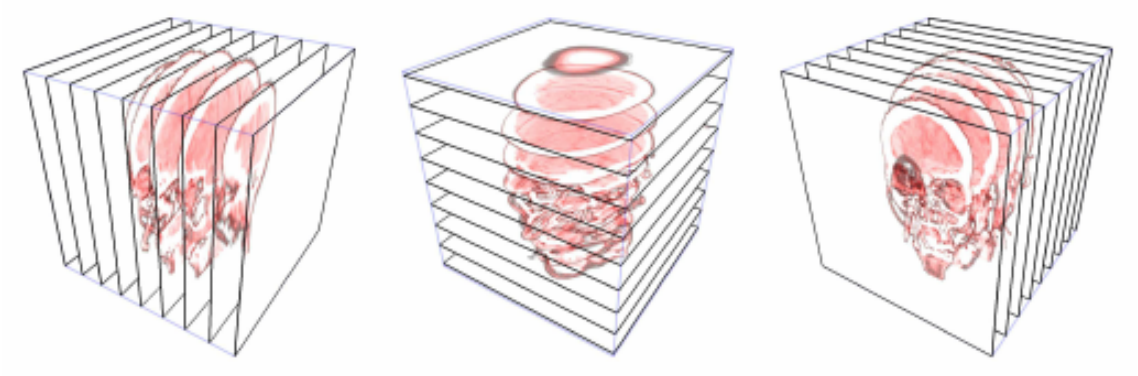


Figura 22: Partició d'una textura 3D d'un model volumètric en llesques. Veiem que podem concebre les llesques en qualsevol dimensió. Font: Benassarou et al., 2005.

L'algorisme 3 mostra la idea a seguir per aplicar la tècnica *render to texture* a una textura 3D. L'objectiu és que per cada *slice* de la textura 3D on volem *renderitzar* (el *render target*), cridem a pintar un quadrat que ocupi tot l'espai de *renderitzat*. Com que pels algorismes que estem desenvolupant necessitem les coordenades de la textura del volum (una textura 3D), amb cada vèrtex que volem pintar enviem les coordenades de la textura 3D del volum, on la coordenada z dependrà de la *slice* que estem visitant. A la secció 3.3.2 mostrarem amb més detall com hem implementat aquesta tècnica amb l'API que ofereix Unity3D.

3.3.1.2 Compute Shaders i Compute Buffers

Un *compute shader* és un *shader* programable que expandeix la progrmació a les targetes gràfiques més enllà de la tasca de *renderitzar*. Ofereix un sistema de computació de propòsit general d'alta velocitat aprofitant la capacitat de paral · lelització de les GPU. També implementa mecanismes de memòria compartida i sincronització de *threads* per oferir més flexibilitat alhora de dissenyar i implementar mètodes de programació en paral · lel. Aquest projecte implementa els algorismes de segmentació amb *compute shaders* utilitzant l'API de Microsoft Direct3D 11 (Microsoft, 2018), però els conceptes que presentarem en aquest apartat es poden relacionar fàcilment amb les solucions que ofereixen altres API, com OpenGL (Khronos, 2019).

La idea fonamental per treballar amb *compute shaders* recau en dos paràmetres: el nombre de grups de *threads* a executar (es defineix des de l'aplicació), i el nombre de *threads* que tindrà cada grup (es defineix quan es declara un *kernel* al *compute shader*). El primer es defineix cridant a un mètode *ID3D11DeviceContext::Dispatch* que permet definir tres dimensions de grups de *threads*. Des de la versió 11 de Direct3D, es poden definir fins a un màxim de 65535 grups de *threads*

Algorithm 3 Render To (3D) Texture. $O(V's\ z\ dimension\ resolution)$

1: $V \leftarrow VolumeTexture$
2: S ▷ texture that will contain segmentation labels
3: **for** $i = 1, 2, \ldots, V.resolution.z$ **do**
4: $slice = i$
5: $GraphicsAPI.SetRenderTargetMethod(texture : S, slice)$ ▷ your API's method to set render target
6: $z = clamp(0, 1, \frac{slice}{V.resolution.z-1}$
7: $GraphicsAPI.Draw(QUAD)$ ▷ your API's method to draw 4 vertexs.
8: $GraphicsAPI.SendVertex(0, 0, 0)$
9: $GraphicsAPI.SendTexCoord(0, 0, texCoordZ)$
10: $GraphicsAPI.SendVertex(1, 0, 0)$
11: $GraphicsAPI.SendTexCoord(1, 0, texCoordZ)$
12: $GraphicsAPI.SendVertex(1, 1, 0)$
13: $GraphicsAPI.SendTexCoord(1, 1, texCoordZ)$
14: $GraphicsAPI.SendVertex(0, 1, 0)$
15: $GraphicsAPI.SendTexCoord(0, 1, texCoordZ)$
16: $GraphicsAPI.EndDraw()$
17: **end for**

per cada dimensió[5]. A partir d'aquí, per cada grup definit per la funció *Dispatch*, s'executen en paral · lel el nombre de *threads* que indica el *kernel*, també definit com un paràmetre de tres dimensions. Cada grup de *threads* que s'executa en paral · lel tindrà accés a recursos compartits només dins de la seva regió. Les API solen oferir modificadors per fer un recurs compartit entre tots els grups de *threads*. A més, podem definir el mètode inicial del nostre *kernel* perquè rebi un o més paràmetres, entre els quals hi ha l'identificador del grup ($SV\_GroupID$), l'identificador del *thread* dins del grup ($SV\_GroupThreadID$), i l'identificador del *thread* d'entre el nombre total de *threads* que es crea amb la crida al *Dispatch* ($SV\_DispatchThreadID$). La figura 23 mostra un exemple del què acabem d'explicar.

El mètode principal d'un *compute shader* no retorna cap valor, de manera que a diferència de la tècnica explicada a la secció anterior (3.3.1.1), hem d'explicitar al nostre *kernel* que després de fer les operacions pertinents guardi el resultat on li pertoca per la nostra aplicació: la textura de segmentació. Per aconseguir-ho, podem declarar la textura com un objecte de tipus UAV (*unordered-access view*), de manera que podem llegir i escriure directament sobre la textura com si fos una estructura de dades convencional, tenint en compte que cal fer-la visible coherentment entre els *threads*, de manera que diferents grups puguin veure les escriptures d'altres. A part d'aquest ús que ofereixen les textures, els UAV també poden ser referenciats com a *búffers* d'un mateix

[5] A la versió 10 el nombre màxim de grups de *threads* per la dimensió "z"és de 1.

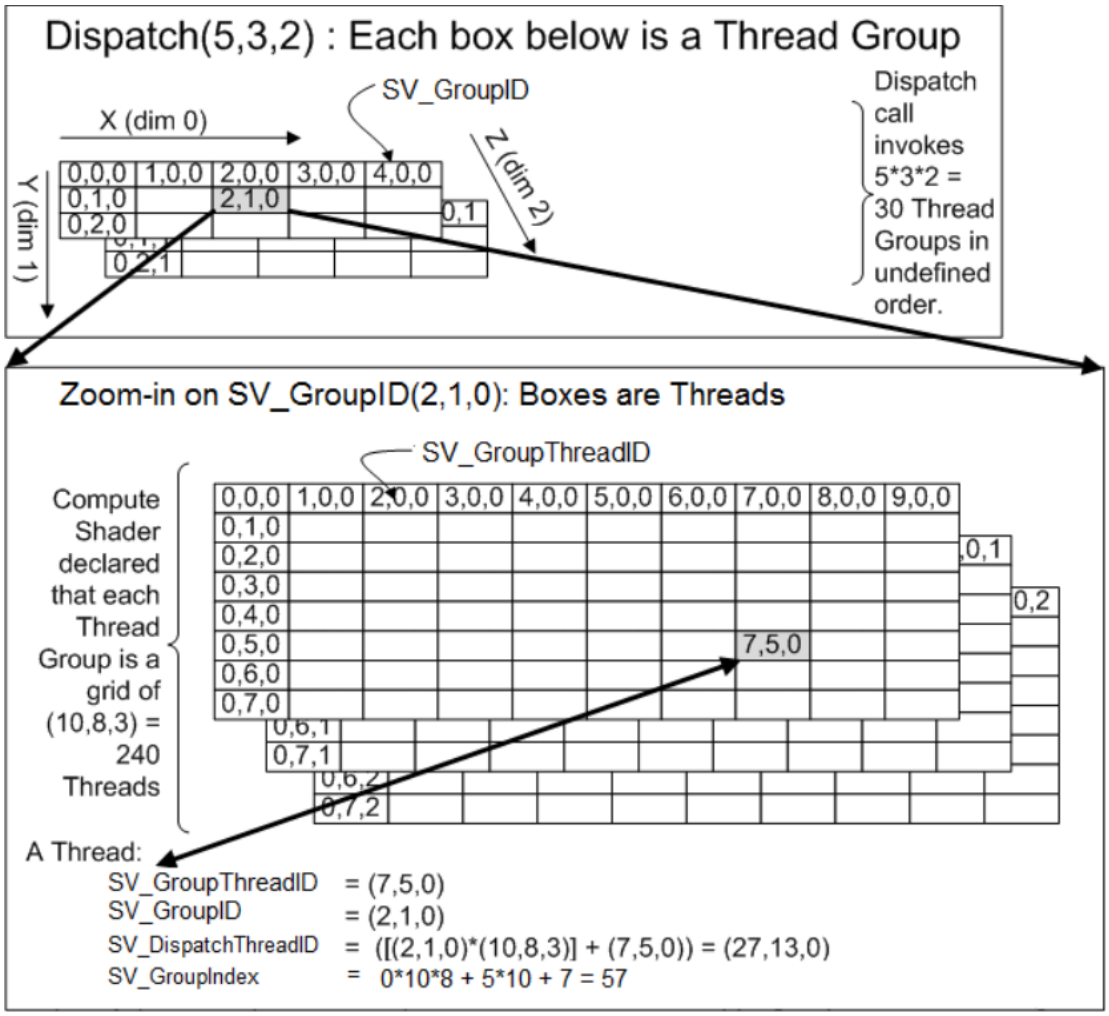


Figura 23: Exemple que mostra la relació entre els paràmetres del mètode *Dispatch* i de l'atribut *numthreads* d'un *kernel* d'un *compute shader*. Font: Microsoft, 2018.

tipus definit pel desenvolupador. Es va provar la solució que presentem a 3.3.2 amb l'ús d'aquests *búffers*, però no suportaven la mida per guardar la informació de tot el model volumètric.

Vam concloure que podem cridar a executar els nostres *compute shaders*, on el nombre de *threads* de cada dimensió sigui equivalent a la resolució de cada dimensió (respectivament) del model volumètric que volem segmentar. Al *compute shader* calcularem l'etiqueta del vòxel que determina el paràmetre $SV\_DispatchThreadID$. Un cop sabem quin vòxel estem visitant, guardarem el resultat a la textura de segmentació, que té reservat un índex per emmagatzemar la informació de segmentació del vòxel.[6].

3.3.1.3 Region Growing diffusion-based

El criteri de segmentació per llindar vist fins ara consisteix en una decisió binària que dificulta la interacció de l'usuari amb l'evolució de la segmentació de l'algorisme basat en regió. L'usuari no pot corregir aquesta evolució fins que no veu que algun determinat vòxel s'ha etiquetat. La mètrica de difusió no lineal de Perona i Malik (Perona i Malik, 1990) que proposa Sherbondy et al. (2003) consisteix en expandir les llavors per regions d'intensitat similar, mentre es redueix l'expansió de la difusió davant de gradients alts (Sherbondy, Houston i Napel, 2003).

Representem $S(t,x,y,z)$ com el valor de la difusió d'un determinat vòxel a la posició (x,y,z) a un determinat instant de temps t, on $t=0$ és l'estat inicial (amb les llavors col · locades per l'usuari) i $V(t,x,y,z)$ com el valor d'intensitat d'aquest vòxel. L'equació 2 mostra la difusió que governa l'expansió de les llavors:

$$\frac{\delta S(t,x,y,z)}{\delta t} = div(g(|\nabla V(t,x,y,z)|)\nabla S(t,x,y,z)) \quad where \quad g(s) = \nu \times exp^{\frac{-s^2}{K^2}} \tag{2}$$

Veiem que la difusió obté un valor escalar del camp vectorial que forma els valors d'intensitat del model volumètric. S'afegeixen les constants K i ν per controlar la velocitat de $g(s)$ a tendir a 0 davant de gradients alts i regular la sensibilitat davant del soroll i la velocitat d'agrupar objectes propers a les llavors inicials, respectivament.

Es discretitza l'equació 2 al *fragment shader* seguint el model eulerià. L'equació 3 mostra la discretització de la dimensió x:

$$S_i^{n+1} = S_i^n + h \times ((S_{i+1}^n - S_i^n) \times g(\frac{V_{i+1}-V_i}{\Delta x}) - (S_i^n - S_{i-1}^n) \times g(\frac{V_i - V_{i-1}}{\Delta x})$$
$$where \quad S_i^n \approx S(n \times \Delta t, i \times \Delta x), V_i = V(i \times \Delta x), 0 \leq n \leq N, 0 \leq i \leq W, \Delta x = 1/W, i \quad h = \Delta t/\Delta x^2. \tag{3}$$

[6]Com s'ha vist a 3.1, l'aplicació de visualització de models volumètrics utilitza una segona textura (de la mateixa resolució) per guardar la informació de segmentació de cada vòxel.

La discretització compta amb els sis veïns del vòxel que s'està avaluant. N és el nombre total d'iteracions de l'expansió de la segmentació, i W és la resolució de la dimensió x de la textura d'intensitats del volum. Explicitar la discretització a la GPU també redueix el temps de comunicació amb la CPU per l'intercanvi de paràmetres.

Presentem a l'algorisme 4 un pseudo-codi del funcionament de la segmentació per creixement de regió aplicant el criteri de difusió. Veiem que aquest mètode comportarà un creixement molt més lent de la segmentació, és per això que es sol afegir un criteri de parada per donar control del nombre d'iteracions a l'usuari, però també es pot implementar un mecansime per fer saber a la CPU que la difusió porta un nombre considerat d'iteracions sense canviar l'etiqueta de cap vòxel. A la secció 3.3.2 veurem com ho fem amb la nostra aplicació.

Algorithm 4 Region Growing basat en difusió.

```
1: V ← VolumeTexture
2: S                          ▷ Diffusion values to 0, except for ones that are labelled as seeds (1).
3: threshold_min, threshold_MAX, τ
4: Q ← seeds
5: while !Q.empty()  ||  !stoppingCreiteria do
6:     voxel ← Q.head()
7:     for neighbour in voxel.neighbours() do
8:         S[neighbour] ← S[neighbour] + h × diffusion()   ▷ diffusion() is a function similar to equation 3.
9:         if S[neighbour] == 1 then
10:            Q.enqueue(neighbour)
11:        end if
12:    end for
13:    if !voxel.AllNeighboursSeeded() then
14:        Q.enqueue(voxel)
15:    end if
16: end while
```

3.3.2 Integració amb Unity3D

Hem pogut implementar els algorismes de segmentació mitjançant les dues tècniques descrites a la secció 3.3.1 gràcies al suport de l'API de Unity3D per treballar amb programes escrits per ser executats a la GPU, ja sigui per la configuració d'un *fragment shader* com per la configuració i execució de *compute shaders*. L'etiquetatge de la textura de segmentació es fa a la GPU abans d'executar l'algorisme de visualització que també la llegeix, de manera que es treu del *pipline* el pas de la textura actualitzada a la GPU (crida *setPixels*), que penalitzava la velocitat de refresc de l'aplicació. La figura 24 mostra el *pipeline* de l'aplicació amb la funcionalitat de visualitzar la

informació de segmentació modificada a la GPU.

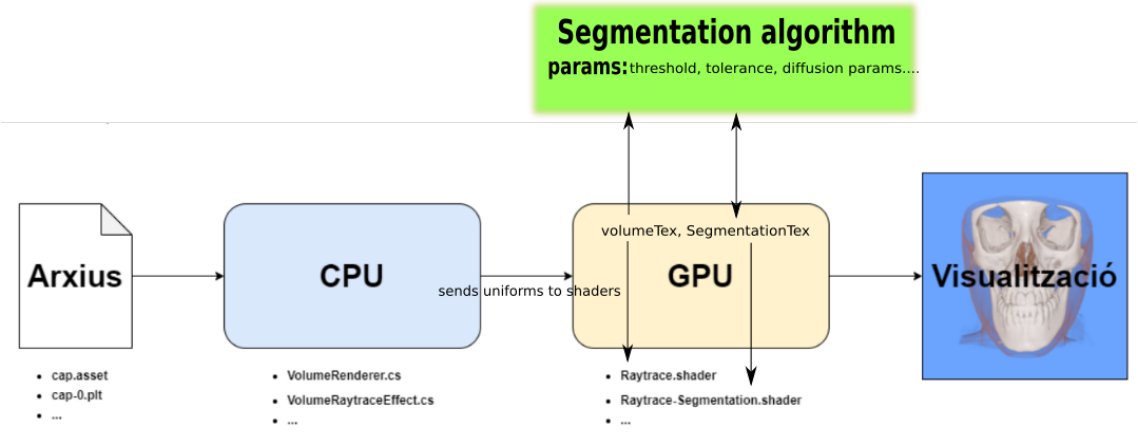


Figura 24: *Pipeline* de la funcionalitat de segmentar regions d'interès d'un model volumètric. Podem veure que tant el *shader* de visualització (*Raytrace-Segmentation.shader*) com els *shaders* que implementen els algorismes de segmentació utilitzen la mateixa textura de selecció.

L'API de Unity3D ofereix la classe *RenderTexture* per treballar amb textures que poden ser escrites. El seu ús és molt comú per crear efectes sobre la visualització que capta la càmera de l'escena. Podem associar a una *RenderTexture* un format vàlid pel nostre controlador de la targeta gràfica i declarar-la com a UAV. La recerca de com fer un ús pertinent de l'API per implementar les tècniques descrites va ser un procés amb una corba d'aprenentatge lenta, ja que a la documentació es presenta com a ús avançat de l'API del motor gràfic sense entrar en molt detall del seu funcionament intern, i que la comunitat ha trobat que hi ha poc suport d'errors i millores entre les versions. Això s'acumulava amb la dificultat inicial del projecte de treballar amb conceptes que en la major part han quedat fora de l'abast de les assignatures de gràfics per computador del grau. Aquesta situació va fer difícil implementar la solució per *render to texture*, i es va optar primer per oferir la funcionalitat mitjançant *compute shaders*, ja que la seva condició de ser *shaders* de propòsit general fa que el seu funcionament es presenti més intuïtiu per començar a portar les implementacions a la GPU.

3.3.2.1 Implementació amb Compute Shaders

La classe *ComputeShader* permet carregar, configurar i executar els algorismes de segmentació implementats amb aquesta metodologia. Concretament, podem enviar *uniforms*, saber el nombre de *threads* de cada grup definit pel *kernel* (*ComputeShader::GetKernelThreadGroupSizes*), i cridar a executar un nombre de grups per cada dimensió (*ComputeShader::Dispatch()*).

Com s'ha comentat a la secció 3.3.1, podem concebre cada *thread* com al còmput de l'etiqueta de cada vòxel. La figura 25 és un exemple de com implementar aquesta tècnica per aplicar la segmentació per llindar, on podem veure que declarem la textura de segmentació amb el tipus

RWTexture3D<float> per indicar que aquesta textura és un UAV. Podem declarar aquesta textura amb varis formats, però s'ha escollit utilitzar un sol valor per coma flotant per permetre l'accés als valors per la implementació de la segmentació per regió, ja que l'evolució de la difusió requereix aquest valor a cada iteració (vés a 3.3.1.3), i Direct3D limita el format de l'UAV si es vol fer ús d'aquest recurs.

```
#pragma kernel CSMain

uniform Texture3D<float4> _VolumeTex;
SamplerState sampler_VolumeTex; // "sampler + _VolumeTex"

uniform float _ThreshMin;
uniform float _ThreshMax;

uniform float3 _Resolution;

uniform float _SelectedLabel;
uniform float _DiscardedLabel;

uniform RWTexture3D<float> _SegmentationTexture;

float3 Normalize(float3 p)
{
    return float3(p.x / (_Resolution.x - 1), p.y / (_Resolution.y - 1), p.z / (_Resolution.z - 1));
}

bool CheckRange(float3 voxel)
{
    float value = _VolumeTex.SampleLevel(sampler_VolumeTex, Normalize(voxel), 0).a;
    return value >= _ThreshMin && value <= _ThreshMax;
}

[numthreads(8,8,1)]
void CSMain (uint3 id : SV_DispatchThreadID)
{
    if (CheckRange(id))
    {
        _SegmentationTexture[id] = _SelectedLabel;
    }
    else
    {
        _SegmentationTexture[id] = _DiscardedLabel;
    }
}
```

Figura 25: *Compute shader* que implementa la segmentació per llindar d'un vòxel marcat pel paràmetre *uint3 id : SV_DispatchThreadID*. Noteu que normalitzem l'índex del vòxel per llegir el valor d'intensitat de la textura de volum. S'envien els valors de *threshold* des de la CPU amb la tolerància aplicada per reduir el nombre d'operacions.

La figura 26 mostra un exemple de com hem implementat el mètode per configurar i executar aquest *compute shader*, que mostra el pas de *uniforms*, la lectura de la mida dels grups definida pel *kernel*, i la creació dels grups de *threads* per a executar-ne tants com vòxels té el model volumètric.

Volem destacar que l'aplicació original treballa amb textures de segmentació amb quatre canals. Com que l'ús de textures com a UAVs en limita el seu format, ha sigut necessari modificar el *shader Raytrace-Segmentation.shader* de l'aplicació original. Aquest canvi és transparent al *pipeline* original, ja que el *shader* modificat sap en tot moment amb quin format està la textura de segmentació amb la què està treballant, sense produir efectes secundaris a la visualització de la segmentació.

```
int kernelIndex = _algorithm.FindKernel("CSMain");
Volume volume = VolumeR.volume;

_algorithm.SetTexture(kernelIndex, "_VolumeTex", volume.volumeTexture);

_algorithm.SetFloats("_Resolution", new float[3] { volume.resolution.x, volume.resolution.y, volume.resolution.z });

_algorithm.SetFloat("_ThreshMin", ApplyTolerance(ThreshMin, max: false));
_algorithm.SetFloat("_ThreshMax", ApplyTolerance(ThreshMax));

_algorithm.SetFloat("_SelectedLabel", SelectedLabel);
_algorithm.SetFloat("_DiscardedLabel", DiscardedLabel);

_algorithm.SetTexture(kernelIndex, "_SegmentationTexture", SegmTexture);

uint x, y, z;
_algorithm.GetKernelThreadGroupSizes(kernelIndex, out x, out y, out z);
_algorithm.Dispatch(kernelIndex, (int)((int)VolumeR.volume.resolution.x / x), (int)((int)VolumeR.volume.resolution.y / y), (int)((int)VolumeR.volume.resolution.z / z));
```

Figura 26: Mètode per configurar i executar un *compute shader* mitjançant l'API de Unity3D (declarat amb el nom *_algorithm*). La textura *SegmentationTex* correspon a una *RenderTexture* 3D amb l'atribut *enableRandomWrite* activat per permetre tractar-la com a UAV al *compute shader*.

3.3.2.2 Implementació amb Render to texture

El *render to texture* requereix fer ús de la llibreria *GL*, que proporciona crides de més baix nivell per interactuar amb el controlador de gràfics, ja que la funció *Graphics.Blit* no permet *renderitzar* a textures de tipus *Texture3D*, que és com estan declarades al model de dades de l'aplicació existent. En aquest sentit, s'ha utilitzat la funcionalitat d'aquesta llibreria per reproduir l'esquema presentat a l'algorisme 3 un cop s'han definit els paràmetres del *shader*.

La figura 27 mostra la configuració dels paràmetres del *fragment shader* que conté la implementació de la segmentació per regió basada en el criteri de difusió. Es pot veure la creació d'una nova *RenderTexture* que emmagatzema valors amb coma flotant a la qual, mitjançant la funció *Blit3D* (explicada a l'algorisme 3), hi guardarem la sortida del *fragment*. Es pot observar el pas de l'actual textura de segmentació del model com a *uniform* (*material.SetTexture("_SegmentationTex", SegmTexture)*) i que, un cop s'han escrit les etiquetes noves a la *RenderTexture* (gràcies a la funció *Blit3D*), actualitzem la textura de segmentació (*SegmTexture* = *renderTexture*).

A partir d'aquí, l'única diferència entre el *compute* i el *fragment* que implementen els algorismes és que el primer acaba escribint explícitament a la textura de segmentació que rep com a recurs (com hem vist a la figura 25), mentre que el segon retorna el valor de l'etiqueta (mantenint el format de valor amb coma flotant que s'ha especificat anteriorment) que s'ha calculat, i gràcies a la configuració feta amb l'API de Unity3D, la sortida d'aquest *fragment* s'escriu a la textura. La figura 28 mostra l'exemple d'un *fragment shader* que implementa la segmentació per llindar, on es pot veure que afegim l'etiqueta del píxel al valor que retorna el programa.

3.3.2.3 Segmentació per creixement de regió basada en difusió

Com s'ha comentat a la secció 3.3.1.3, l'expansió de la segmentació consisteix en acumular iterativament el valor de la difusió de cada vòxel. Per aconseguir-ho, es crida a l'execució dels *shaders*

```
RenderTexture renderTexture = new RenderTexture(SegmTexture.width, SegmTexture.height, 0, RenderTextureFormat.RFloat, RenderTextureReadWrite.Linear)
{
    dimension = UnityEngine.Rendering.TextureDimension.Tex3D,
    volumeDepth = SegmTexture.volumeDepth,
    wrapMode = TextureWrapMode.Clamp,
    filterMode = FilterMode.Point,
    enableRandomWrite = true,
    useMipMap = false,
};

renderTexture.Create();

material = new Material(Shader.Find("VRMed/RegionGrowingSegm"));

material.SetTexture("_VolumeTex", VolumeR.volume.volumeTexture);
material.SetTexture("_SegmentationTex", SegmTexture);

material.SetFloat("_IncrX", 1.0f / VolumeR.volume.resolution.x );
material.SetFloat("_IncrY", 1.0f / VolumeR.volume.resolution.y);
material.SetFloat("_IncrZ", 1.0f / VolumeR.volume.resolution.z);

material.SetFloat("_HeatMapInit", SceneSetup.UserReservedSize() / 255.0f);
material.SetFloat("_HeatMapEnd", (SceneSetup.UserReservedSize() + SceneSetup.HeatMapSize() - 1) / 255.0f);

material.SetFloat("_SegmLabel", SelectedLabel);
material.SetFloat("_EmptyLabel", DiscardedLabel);
material.SetFloat("_BorderLabel", BorderLabel);

material.SetFloat("_K", K);

t++;

material.SetFloat("_hX", t * Mathf.Pow(VolumeR.volume.resolution.x, 2));
material.SetFloat("_hY", t * Mathf.Pow(VolumeR.volume.resolution.y, 2));
material.SetFloat("_hZ", t * Mathf.Pow(VolumeR.volume.resolution.z, 2));

Blit3D(renderTexture, (int)VolumeR.volume.resolution.z, material);

SegmTexture = renderTexture;
```

Figura 27: Pas de paràmetres al *fragment shader* contingut a una instància de la classe Material de Unity3D que conté la implementació d'una passada de la segmentació per regió basada en difusió.

amb les dues metodologies descrites, però a diferència de la segmentació per llindar, que només es fa la crida per tot el model volumètric una vegada, aquest mètode és crida amb el bucle principal del motor gràfic mentre l'usuari vol tenir activada la segmentació per regió, per no bloquejar la resta de funcionalitats de l'aplicació. La figura 29 mostra aquest esquema.

S'ha afegit una funcionalitat d'aturada automàtica per la segmentació per difusió alternativa a la parada condicionada per l'usuari a través de la metàfora d'interacció (vés a 3.4). Cada vegada que el valor de la difusió estableix un nou vòxel com a etiquetat es marca el valor d'un *RWStructuredBuffer*[7], i cada cert nombre d'iteracions[8], la CPU reclama el valor d'aquest *buffer*. Si en aquest interval d'iteracions no s'ha etiquetat cap vòxel, l'algorisme es para.

Finalment, s'ha implementat un mapa de calor per donar resposta visual a l'usuari de com evoluciona la difusió. S'ha fet ús de la textura que utilitza l'aplicació per associar etiquetes a colors (veure 3.1), on hem reservat un rang de posicions (etiquetes) que contenen un gradient de colors des del groc al vermell. D'aquesta manera, a cada iteració de l'algorisme, si el valor de la difusió resultant no s'ha mogut de la primera etiqueta d'aquest mapa de calor, el vòxel es considera buit. En canvi, quan el valor arriba al màxim, el vòxel es marca el vòxel amb l'etiqueta de sel · lecció. Si el valor de la difusió es troba dins del rang del mapa de calor, s'etiqueta el vòxel amb aquest valor, mostrant a l'usuari si aquest vòxel està prop o lluny de ser segmentat. Això permet veure en temps real l'expansió de la regió d'interès, i poder afegir noves llavors per accelerar el procés de

[7] S'ha fet ús d'operacions atòmiques que ofereix Direct3D per garantir les escriptures d'aquest recurs compartit.
[8] Fent proves amb la velocitat d'expansió de la difusió s'ha definit entre 1000 i 1500.

```
uniform sampler3D _VolumeTex;

uniform float _ThreshMin;
uniform float _ThreshMax;

uniform float _SelectedLabel;
uniform float _DiscardedLabel;

float and(float a, float b)
{
    return a * b;
}

float when_lt(float x, float y)
{
    return max(sign(y - x), 0.0);
}

float when_ge(float x, float y) {
  return 1.0 - when_lt(x, y);
}

float frag (v2f i) : SV_Target
{
    float value = tex3Dlod(_VolumeTex, float4(i.uv, 0)).a;
    float color = 0.0;
    color += lerp(0.0, _SelectedLabel, and(when_ge(value, _ThreshMin), when_ge(_ThreshMax, value)));
    color += lerp(0.0, _DiscardedLabel, 1.0 - and(when_ge(value, _ThreshMin), when_ge(_ThreshMax, value)));
    return color;
}
```

Figura 28: Implementació de la segmentació per llindar a un *fragment shader*. Es pot veure com les coordenades de la textura (*i.uv*) es tracten com a un vector de 3 posicions. Remarquem que hem implementat els nostres *shaders* intentant evitar l'ús d'estructures condicionals per evitar un nombre considerat de branques i optimitzar el rendiment.

selecció o marcar fronteres per evitar l'expansió (veure secció 3.4).

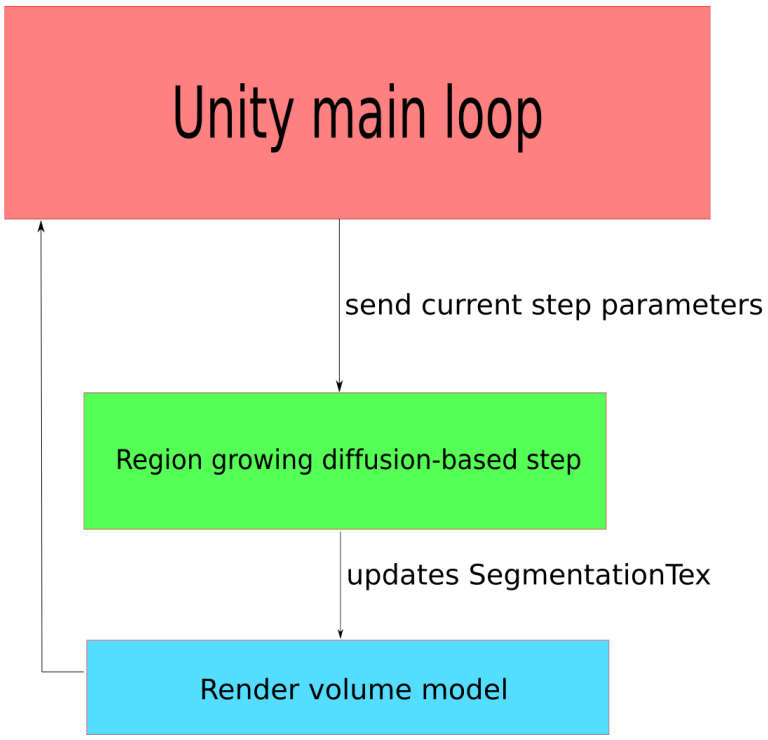


Figura 29: Esquema principal d'execució de la segmentació per creixement de regió basat en el criteri de difusió implementat a la GPU. Utilitzem el bucle principal de Unity3D per no bloquejar la resta de funcionalitats de l'aplicació.

3.3.3 Anàl · lisi de rendiment

Com era d'esperar a partir de l'estudi previ (vés a 1.4), la capacitat de paral · lelització de les targetes gràfiques i el fet d'evitar enviar la textura de segmentació modificada per la CPU ha reduït l'impacte al rendiment de l'aplicació comparades amb les versions per CPU. La figura 30 mostra els temps d'execució de la segmentació per llindar dels mateixos tres models volumètrics amb els que s'han fet les proves de les solucions per CPU fent ús de *compute shaders*. Es pot veure clarament la millora del rendiment respecte la implementació per CPU (veure figura 20).

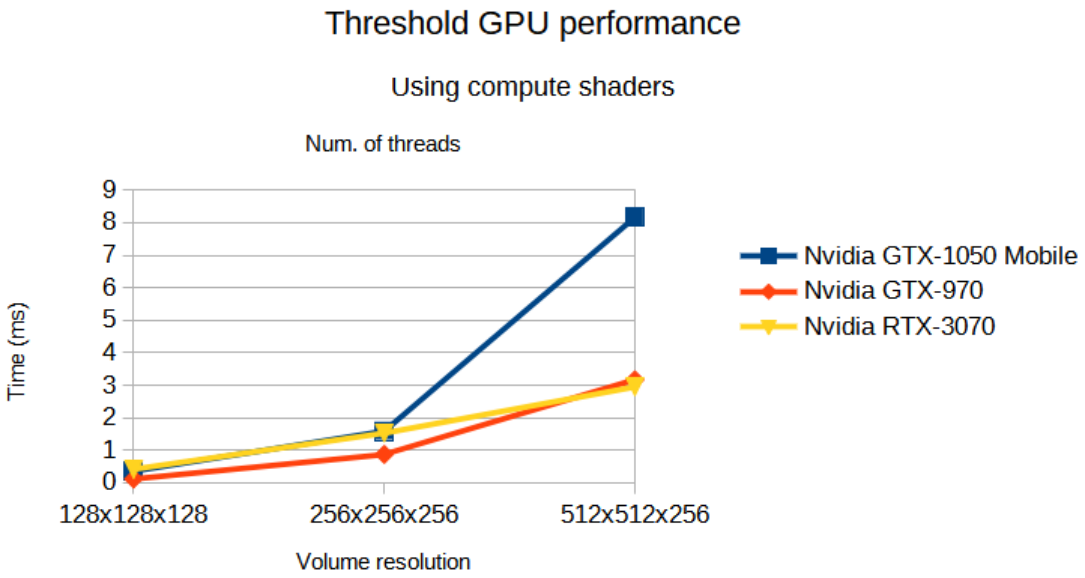


Figura 30: Rendiment del còmput de la segmentació per llindar implementada amb un *compute shader*. Es pot observar que a mesura que augmenta el nombre de *threads* (variable directament lligada a la resolució) el temps d'execució augmenta més ràpid. Elaboració pròpia.

La segmentació per regió que hem implementat a la GPU, és més costosa que la versió per regió basada en llindar que hem presentat a la secció 3.2.1, ja que amb el bucle principal es crida a l'execució d'un pas de la difusió on es visita tots els vòxels del volum, i el còmput d'aquest valor i el joc d'etiquetes amb el mapa de calor requereix un nombre més elevat d'instruccions. La figura 31 mostra els temps d'execució de la segmentació per regió amb el criteri de difusió implementada a la GPU. Veiem que fins que no ens enfrontem al model volumètric més pesant, el rendiment és molt superior a la versió per CPU, que ofereix resultats semblants visitant només 14.943 vòxels (veure figura 21).

Hem pogut comparar també el rendiment entre els dos mètodes explicats a 3.3.1. Amb les proves que s'han realitzat, fer un *render to texture* utilitzant *fragment shaders* és més ràpid que escriure a la textura de segmentació a través d'un *compute shader*. Les figures 32 i 33 mostren el rendiment de les implementacions mitjançant aquesta tècnica, on es pot observar una escalabilitat més amena

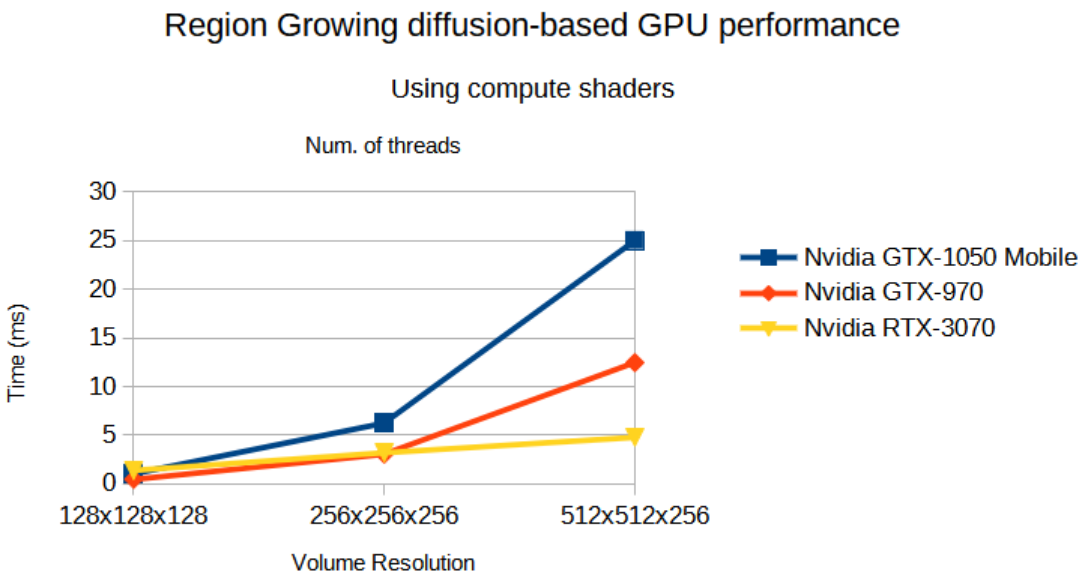


Figura 31: Rendiment de la segmentació per regió basada en el criteri de difusió, implementada fent ús d'un *compute shader*.

davant l'augment de la resolució del model volumètric.

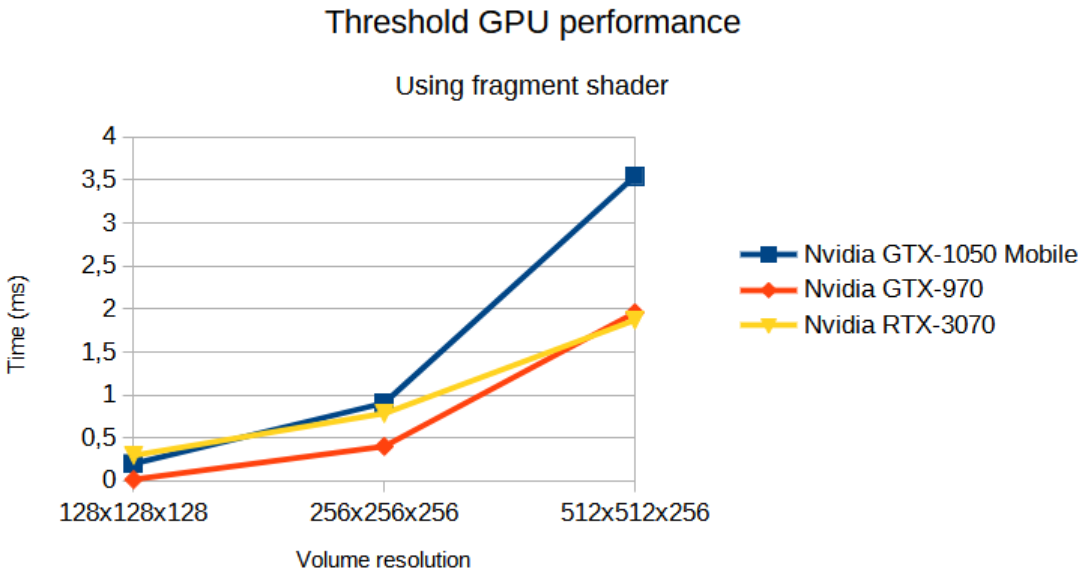


Figura 32: Temps d'execució de la segmentació per llindar utilitzant un *fragment shader*. Els temps d'execució són menors que els del mateix algorisme implementat amb *compute shaders*.

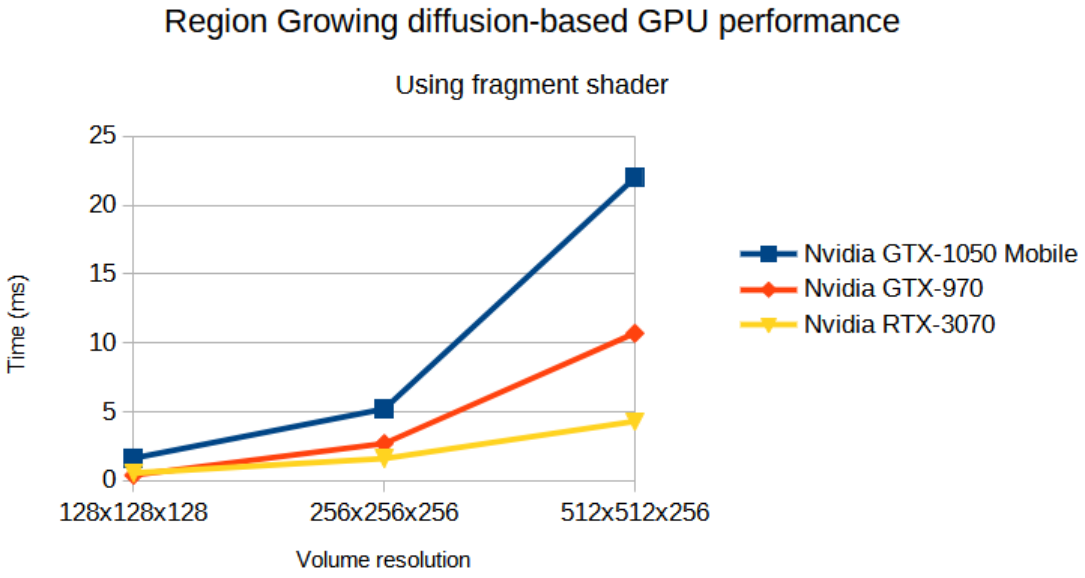


Figura 33: Temps d'execució de la segmentació per *region growing diffusion-based* utilitzant un *fragment shader*. Els temps d'execució són menors que els del mateix algorisme implementat amb *compute shaders*.

3.4 Interacció

La interacció juga un paper fonamental a una aplicació d'inspecció i diagnosi d'imatges mèdiques. Aquesta ha de permetre a un usuari accedir a les funcionalitats que ofereix. En el marc d'aquest projecte volem oferir mecanismes d'interacció per a poder etiquetar i visualitzar regions d'interès del model volumètric que s'està inspeccionant, tenint en compte que l'usuari es troba immers en una escena de realitat virtual.

L'aplicació de visualització de models volumètrics de la qual parteix aquest projecte ja compta amb elements que permeten desenvolupar i integrar noves metàfores d'interacció. La figura 34 mostra la inspecció d'un model volumètric en un entorn de RV immersiu que ens ofereix l'aplicació de visualització que utilitzem. Concretament, l'usuari pot moure i rotar el model volumètric mentre prem el gallet del controlador que ofereix el dispositiu de RV, sempre que el comandament estigui ubicat dins del model. Aquesta metàfora és útil perquè permet la inspecció del model des de qualsevol angle de visió, sense la necessitat que sigui l'usuari el que es mogui per veure el model des del punt de vista desitjat; amb aquesta metàfora es permet modificar el punt de vista aplicant al model el moviment que l'usuari està realitzant amb el comandament.

Figura 34: Inspecció d'un model volumètric en un entorn de RV immersiu. El raig que surt del controlador és una utilitat per millorar-ne la usabilitat. Permet saber a l'usuari cap a on està apuntant o dirigint el comandament. És una tècnica de *feedback* visual per l'usuari.

La figura 35 mostra els diferents botons que ens ofereix el controlador d'HTC Vive, els quals utilitzarem per al desenvolupament de les nostres metàfores.

3.4.1 Formulació del problema

Una metàfora d'interacció per a la funcionalitat de segmentar les regions d'interès d'un objecte ha de permetre a l'usuari afegir i treure llavors, i executar l'algorisme de segmentació. En el cas

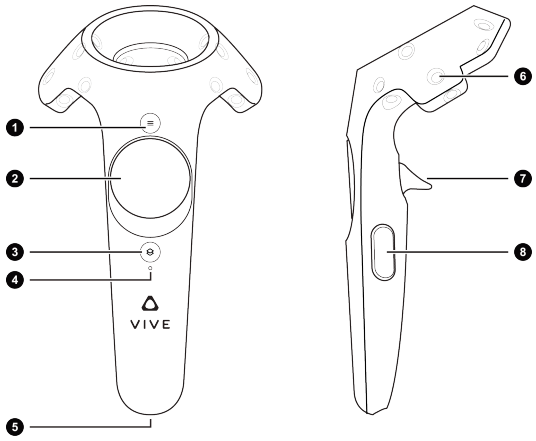


Figura 35: Mapa de botons del controlador del dispositiu de RV immersiu HTC Vive. 1: Menú. 2: Joystick. 3: Menú d'Steam. 4: Estat del controlador. 5: Port de càrrega. 7: Gallet. 8: Agafada (*Grip*).

de la segmentació per creixement de regió basat en la difusió, també interessa que l'usuari pugui decidir quan aturar l'expansió de la difusió, així pot beneficiar-se del caràcter interactiu d'aquest algorisme, a més de corregir per millorar el resultat de l'expansió.

La figura 36 mostra els passos d'interacció per a dur a terme la segmentació. Com que la funcionalitat de segmentació més sofisticada de la nostra aplicació és la segmentació per creixement de regió basat en difusió, sempre que parlem d'interacció ho farem per a l'execució d'aquesta funcionalitat concreta, ja que les altres només requereixen de passos d'interacció molt bàsics (els quals formen part de la interacció global d'aquesta metàfora).

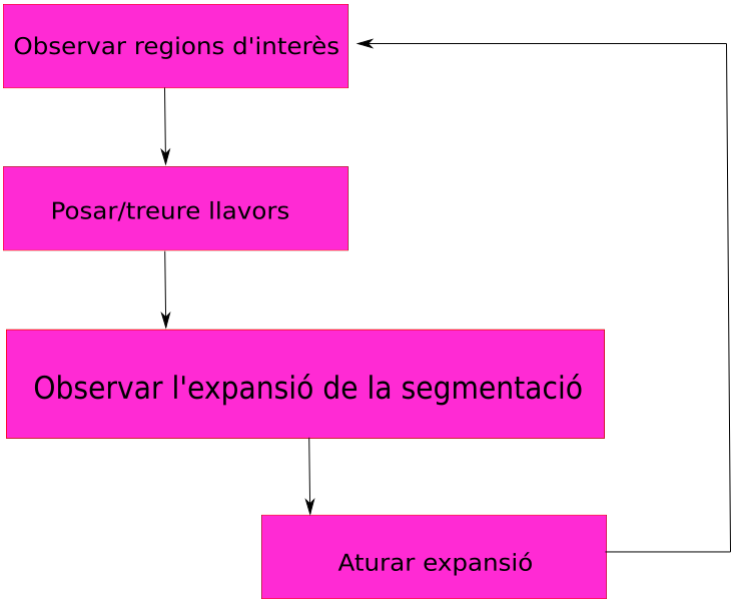


Figura 36: Esquema d'interacció de la nostra aplicació que volem ofereixi per facilitar a l'usuari la tasca de segmentació.

3.4.2 Integració amb Unity3D

Per reproduir els passos identificats a l'apartat anterior, s'ha definit el comportament de l'aplicació com una màquina d'estats que depèn de les accions de l'usuari. La figura 37 mostra aquesta màquina d'estats que canvia segons la configuració dels botons identificats a la figura 35, i permet estar en tots els estats descrits a la figura 36. Podem diferenciar dos tipus d'estats: els referents a l'estat d'execució de l'algorisme de segmentació, i els que fan referència amb l'addició o esborrat de llavors. Veiem doncs que l'usuari pot posar i treure llavors en tot moment (mentre tingui premut el gallet), i és decisió seva si vol fer-ho amb l'expansió de la difusió en funcionament o no.

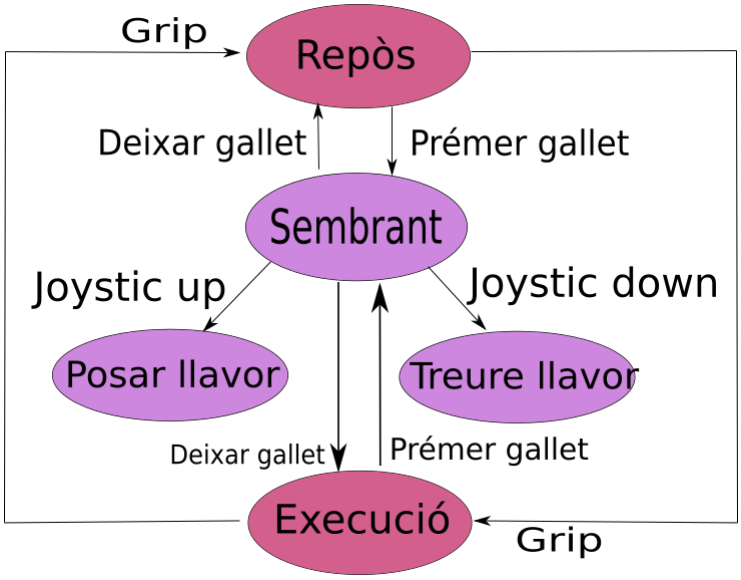


Figura 37: Esquema de la metàfora d'interacció que s'ha implementat per a la segmentació. Elaboració pròpia.

3.4.2.1 Metàfores d'interacció

Per la tasca de posar i treure llavors, es proposen dues metàfores: la metàfora d'afegir llavors amb *punter* i la metàfora d'afegir llavors amb *pla de clipping*. La primera consisteix en disposar una esfera del tamany d'un vòxel del model volumètric a una distància del controlador determinada per l'usuari (mitjançant un paràmetre que actuï com a *offset*). Aquesta metàfora d'interacció és molt senzilla, ja que no permet posar llavors amb comoditat a totes les regions del volum. Per exemple, ens podem trobar amb regions molt internes del model, o potser d'altres que es troben naturalment contingudes dins d'altres estructures o ocultes per altres estructures des del punt de vista que s'està inspeccionant el model volumètric. Per resoldre aquest problema, la metàfora del *pla de clipping* retalla del contingut visible tota regió del model volumètric que es troba a una banda del pla definit segons la orientació del propi comandament. La figura 38 mostra un exemple d'inspecció d'un model volumètric en un entorn de RV immersiu fent ús de la metàfora del *pla de clipping*. Aquesta metàfora crida a una versió modificada del *shader* de visualització original, el

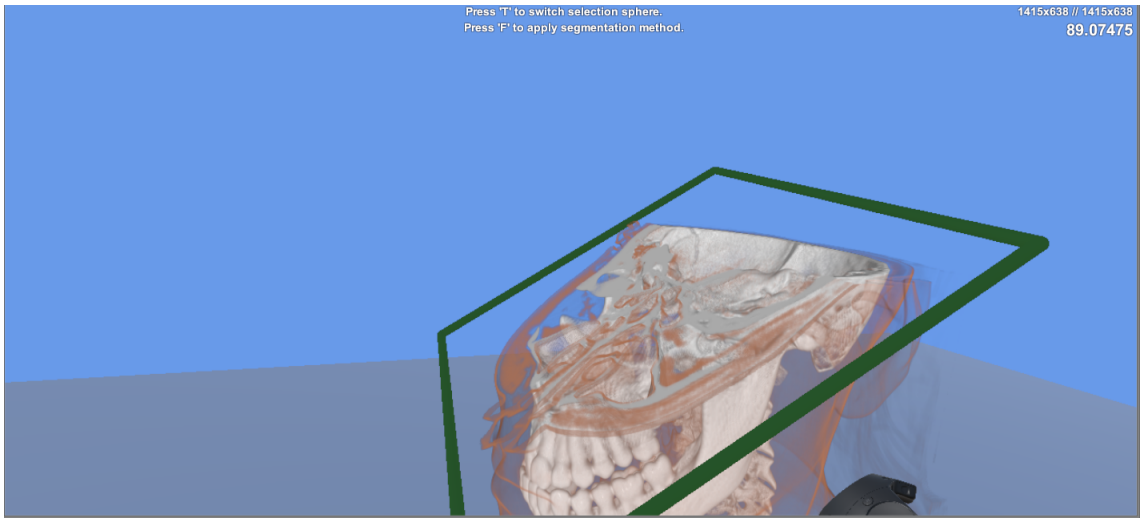


Figura 38: Inspecció d'un model volumètric en un entorn de RV immersiu fent ús de la metàfora del *pla de clipping*.

qual descarta tots els vòxels que es troben per sobre del pla, en direcció a la normal positiva[9].

Quan l'usuari entra a l'estat per col · locar llavors (veure figura 37), la metàfora del punter dóna la llibertat de posar les llavors a qualsevol posició. Aquesta característica afegeix errors de precisió a l'hora de col · locar la llavor, ja que un usuari que no té dominada la sensibilitat de controlador veurà que li costarà posar les llavors a la posició que vol. Amb la metàfora del pla treiem un grau de llibertat quan l'usuari vol realitzar aquesta tasca. Per fer-ho, quan s'entra a l'estat *Sembrant* es congela el pla i es projecta l'esfera al punt d'intersecció de la direcció on apunta el controlador i el pla.

Després de fer algunes proves, vam veure que podíem millorar la comoditat de treballar amb aquesta metàfora. Concretament, tenir la normal del pla amb la mateixa direcció que l'*up* del controlador comportava treballar amb l'amplada i profunditat com a graus de llibertat, mentre que és més fàcil interactuar amb l'amplada i alçada, com si l'usuari estigués pintant una paret vertical. És per això que es va afegir la funcionalitat de canviar la direcció de la normal del pla, per a què apuntés en la direcció (negada) amb la què apunta el controlador. Així, seguint la mateixa implementació, es descartarà tot píxel del model que es trobi entre l'usuari i el pla. D'aquesta manera, quan es volen posar llavors, es pot fer davant d'un pla vertical, millorant així la comoditat. La figura 39 mostra la visualització que es troba un usuari quan està posant llavors amb la metàfora del *pla de clipping*, on es pot apreciar que el *shader* tampoc evalua la funció de transferència pels píxels que es troben a la intersecció amb el pla, sinó que directament *renderitza* el valor llegint de la textura d'intensitats del volum, mostrant així una visualització de la imatge mèdica real que s'està utilitzant. Pensem que els pot ser de gran utilitat als professionals a qui va destinada aquesta aplicació, ja que estan més acostumats a treballar amb les imatges mèdiques originals.

[9]La normal del pla és el vector *up* del controlador

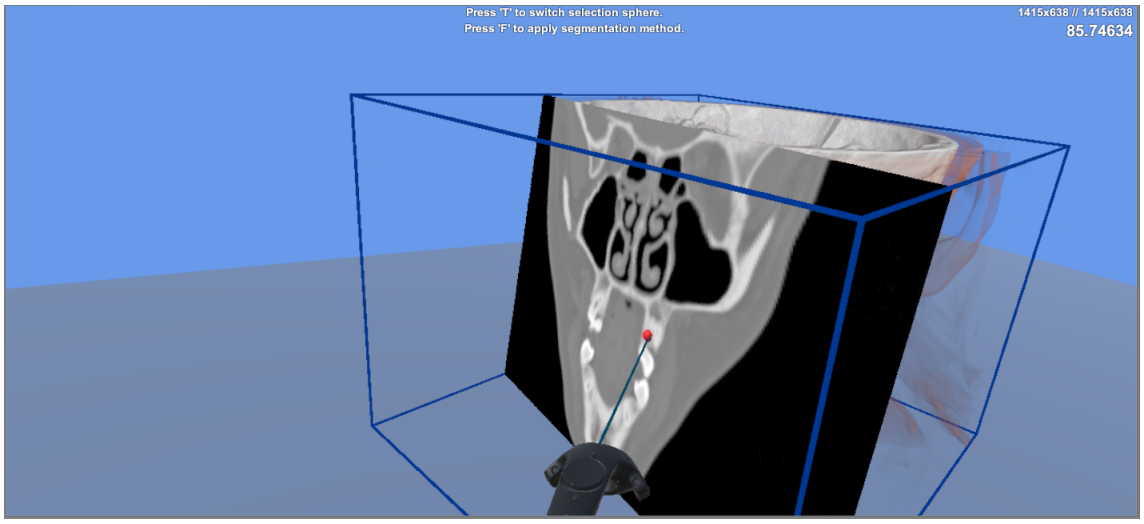


Figura 39: Inspecció d'un model volumètric en un entorn de RV immersiu, on s'ha entrat a l'estat per pintar/posar les llavors. El *pla de clipping* es troba fixat, de manera que l'usuari té dos graus de llibertat per decidir on vol col · locar les llavors. Noteu també que la normal del pla apunta a la direcció contrària al *forward* del raig del controlador.

La figura 40 mostra l'opció de posar llavors sobre un pla congelat on la normal apunta a la mateixa direcció de l'*up* del controlador. Veient la posició a la que es troba el controlador, ja es pot deduir que és molt més còmode l'alternativa presentada.

Finalment, vam haver voler sofisticar una mica més aquesta metàfora. La posició del pla fins ara respecte el controlador és sempre a la distància a la que es troba el centre de model volumètric. Quan la normal del pla apunta cap al controlador, no tenim forma de seleccionar vòxels de la tercera dimensió, ja que el pla es troba congelat al centre del volum. És per això que s'ha modificat la màquina d'estats que regeix la metàfora d'interacció amb *pla de clipping*, per poder afegir un desplaçament a la posició del pla calculada automàticament. La figura 41 mostra aquest canvi.

3.4.2.2 Altres

Com s'ha comentat a l'explicació de la segmentació per creixement de la regió, el procés d'assignar les llavors és clau per al funcionament d'aquest. Per oferir comoditat a l'usuari, l'aplicació mostra en temps real les llavors que va col · locant l'usuari. La inviabilitat d'actualitzar la textura per CPU i enviar-la a la GPU que s'ha remarcat al llarg de la secció 3.2 ens ha obligat a optar per les mateixes tècniques descrites a la secció 3.3. En aquest sentit, s'envia a un *shader* un *uniform* que indica a quin vòxel l'usuari vol posar la llavor, de manera que de nou estem mantenint actualitzada la textura de segmentació a la GPU. La figura 42 mostra una escena de RV immersiva on l'usuari ha posat llavors a un model volumètric fent ús de la metàfora del *pla de clipping*.

Finalment, s'ha volgut dotar d'una última utilitat a l'aplicació. Fent proves amb el creixament de

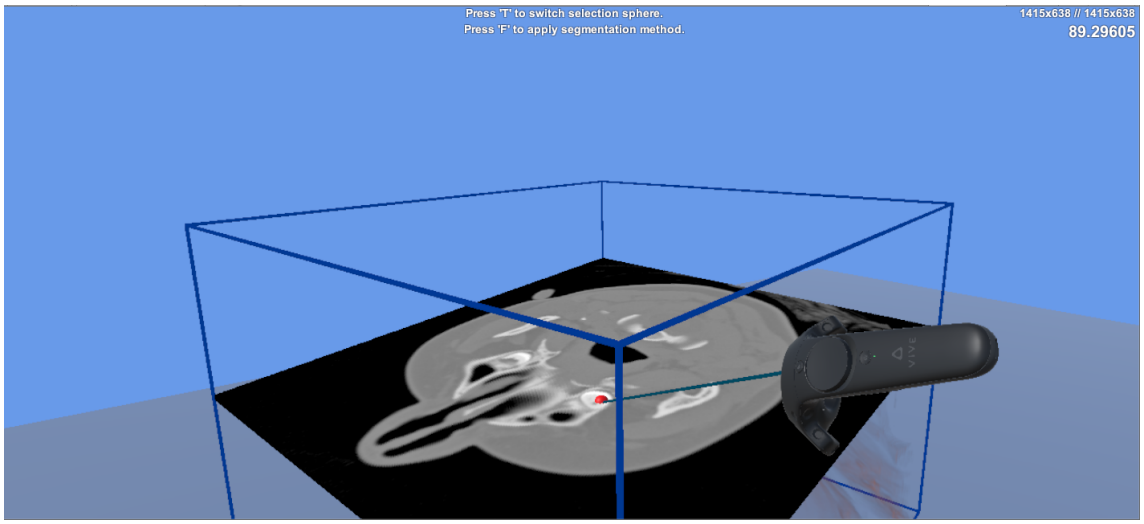


Figura 40: Interacció amb una metàfora de segmentació que implementa un *pla de clipping*. Es pot veure la rotació del controlador, que anuncia la incomoditat de treballar amb el pla fixat d'aquesta manera.

la segmentació, hi ha molts casos que l'expansió creix ràpidament a zones on l'usuari sap que són incorrectes. Hem afegit la funcionalitat de poder posar llavors que anomenem *tapes*, les quals la seva etiqueta no canviarà en funció del valor de la difusió, i per tant permeten crear fronteres entre diferents regions del model volumètric i augmentar així la precisió del *region growing*.

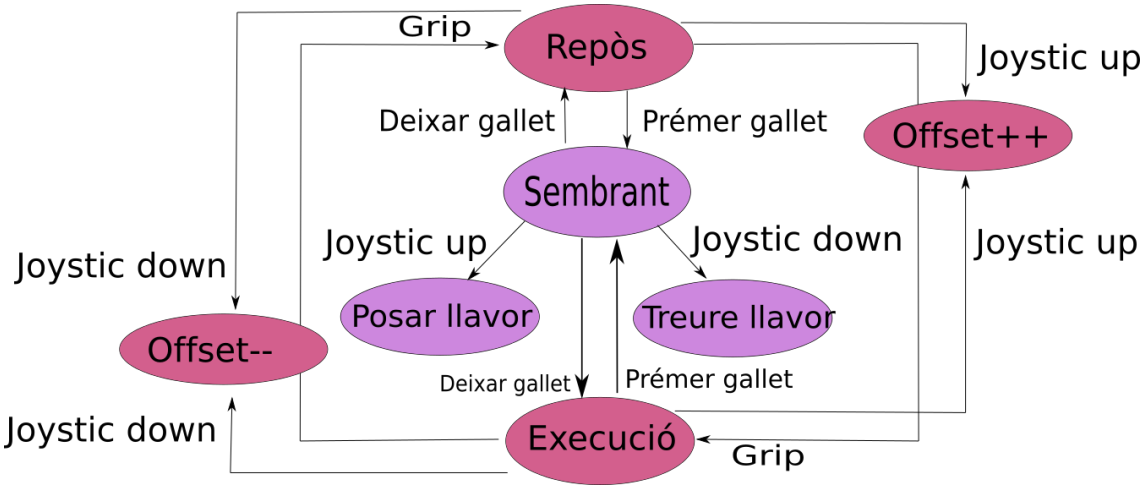


Figura 41: Màquina d'estats de la metàfora de segmentació que implementa un *pla de clipping*, que permet modificar la distància a la què es troba el pla. Noteu que es reaprofita l'utilitat del *Joystick*.

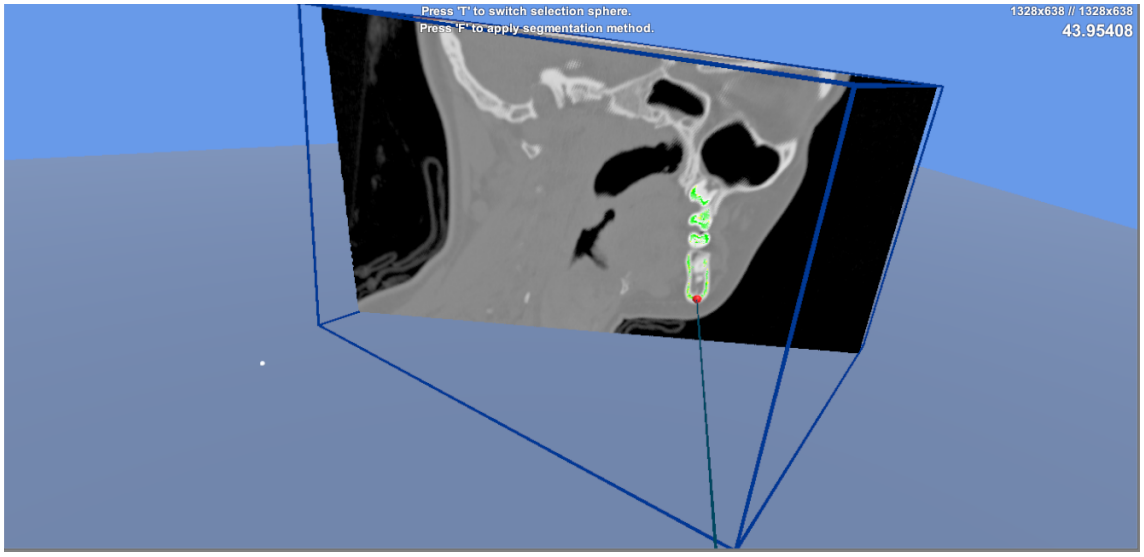


Figura 42: Posada de llavors a un model volumètric mitjançant la metàfora del *pla de clipping*. En aquesta imatge, s'està executant l'expansió de la segmentació amb el criteri de difusió. L'usuari segueix posant llavors mentre creix la regió de segmentació.

3.5 Resultats

Afegim aquesta secció per ensenyar gràficament un procés complet de segmentació mitjançant la nostra aplicació. Les fotografies mostren un usuari utilitzant l'aplicació. El fons d'aquestes mostren el mateix contingut que està veient l'usuari. Les fotografies prenen tot el procés d'inspeccionar el volum, utilitzar la metàfora d'interacció per posar les llavors i veure com s'expandeix la segmentació en temps real.

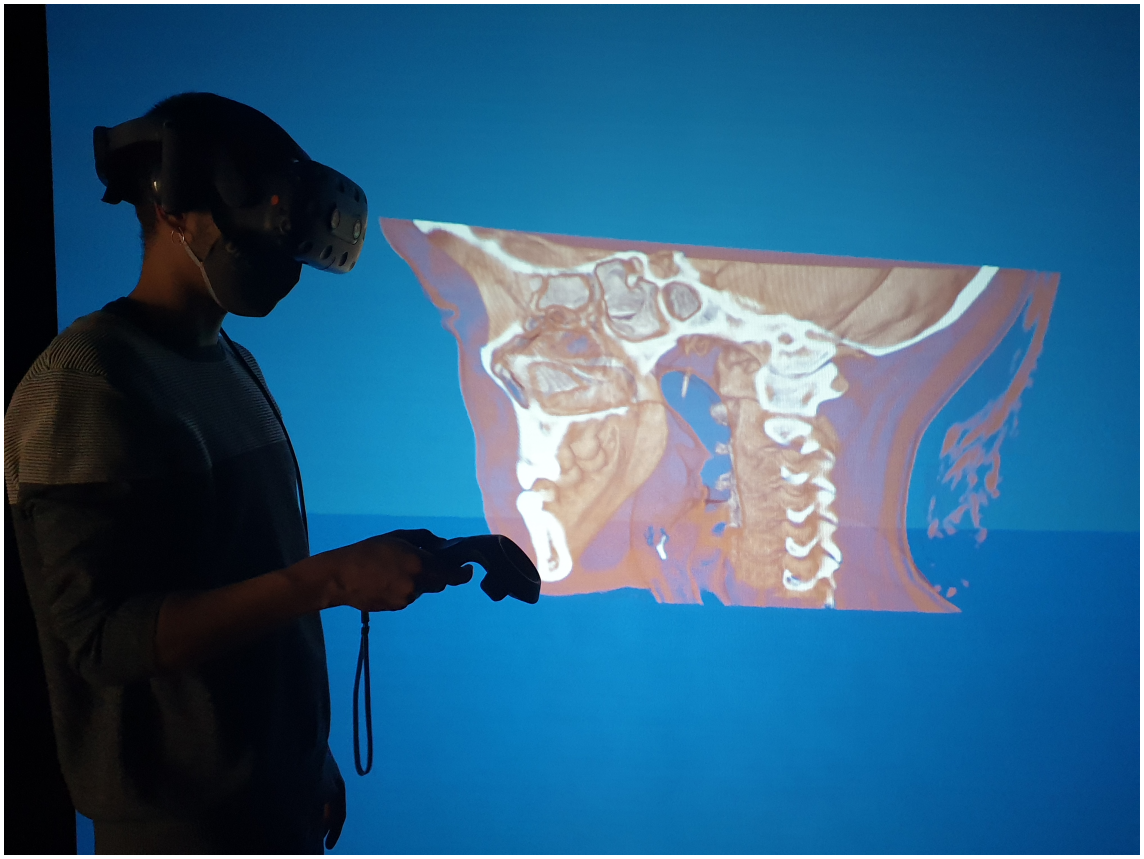


Figura 43: Test: Interacció amb el *pla de clipping.*

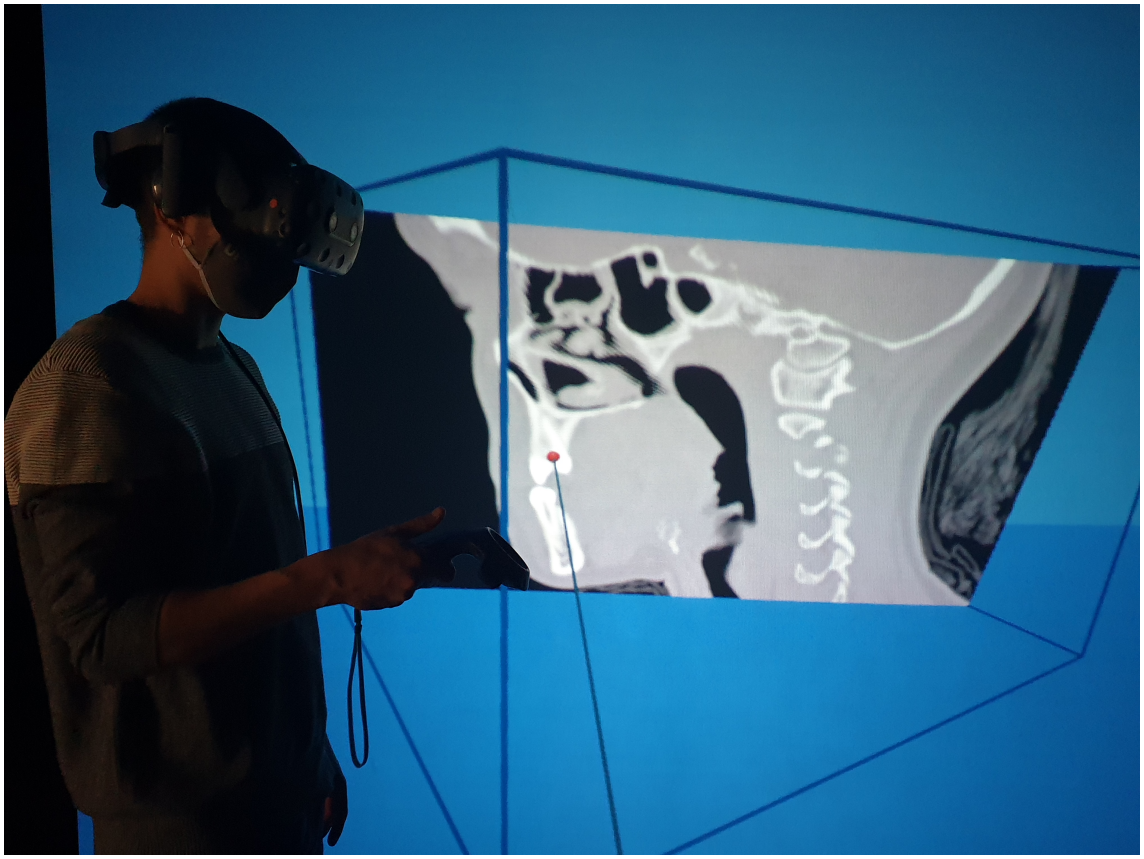


Figura 44: Test: Plantat de llavors.

Figura 45: Test: Creixement de la segmentació 1.

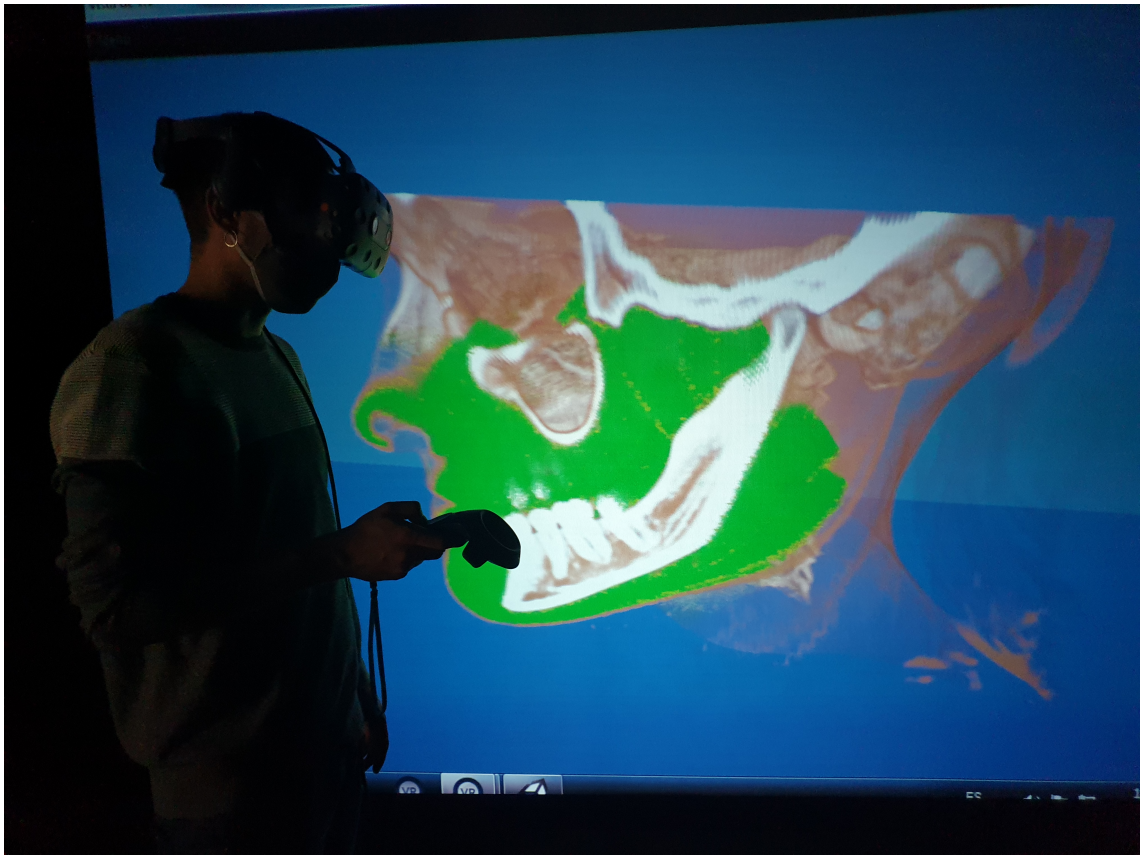


Figura 46: Test: Creixement de la segmentació 2.

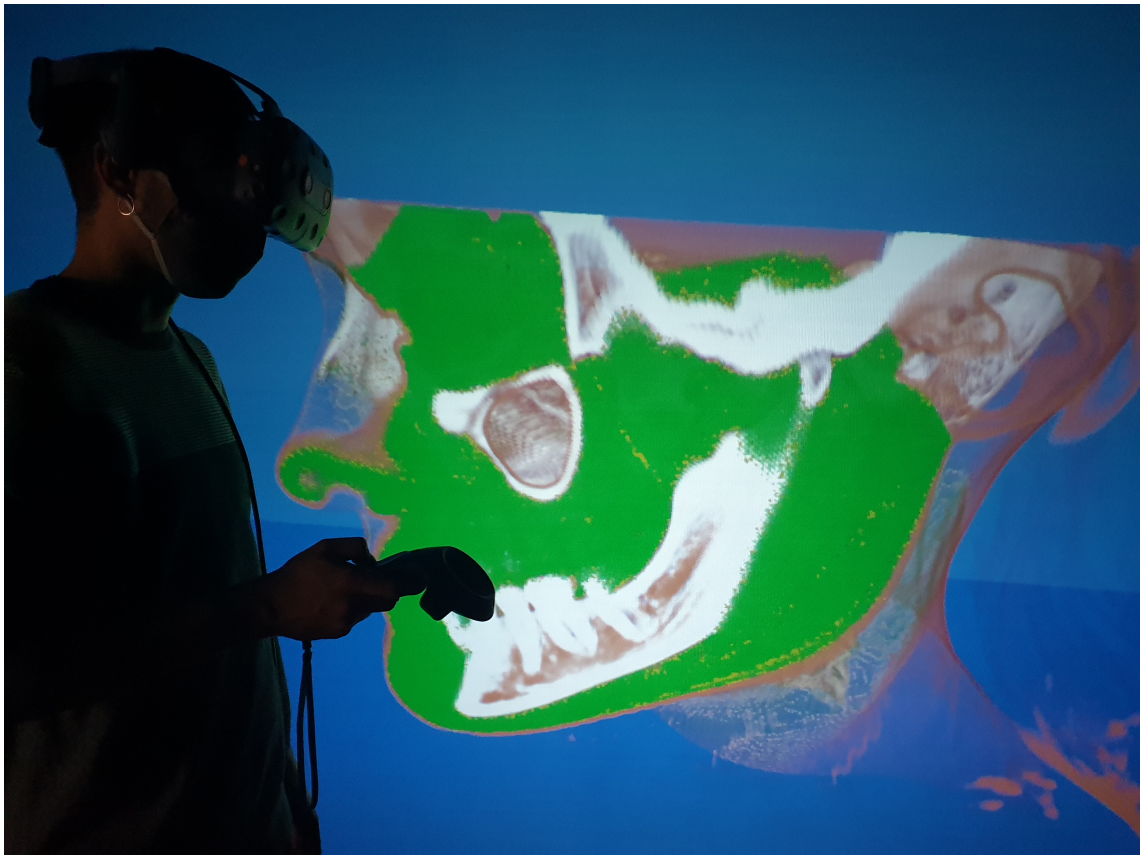


Figura 47: Test: Creixement de la segmentació 3.

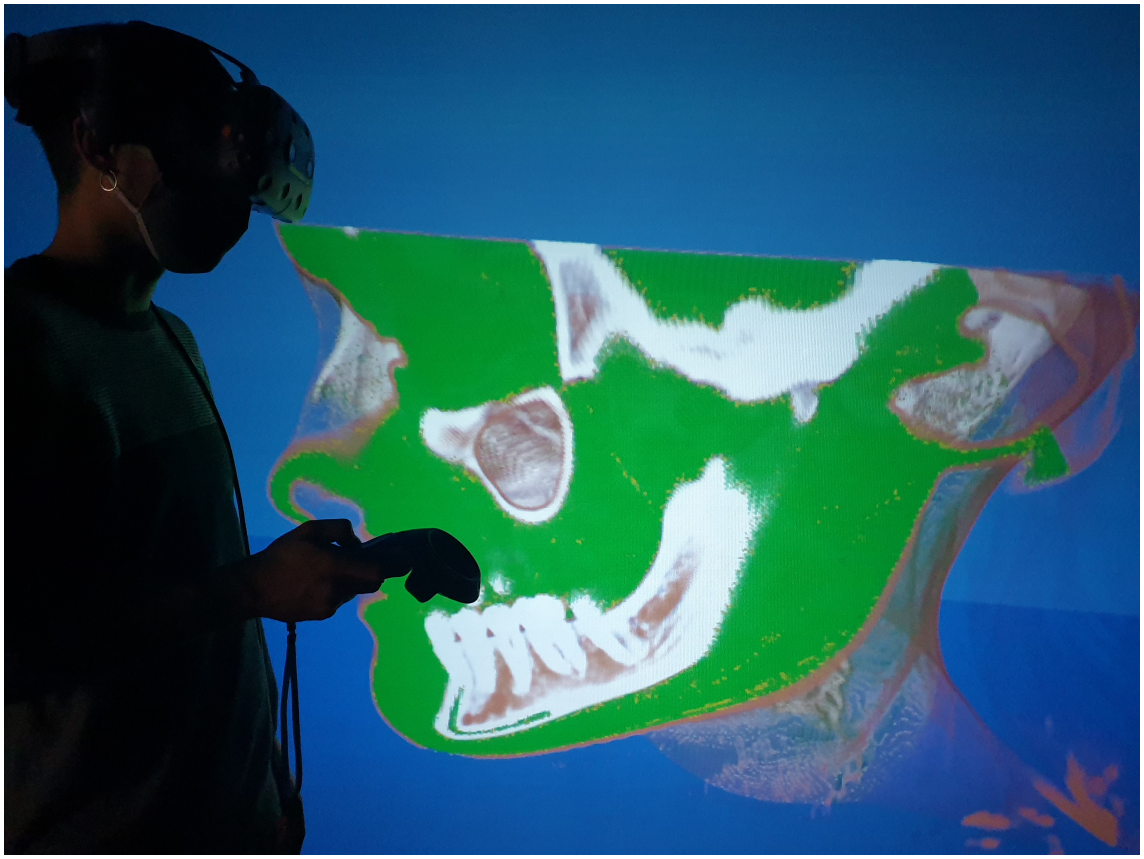


Figura 48: Test: Creixement de la segmentació 4.

3.6 Conclusions i treball futur

Hem aconseguit implementar dos algorismes de segmentació per a una aplicació de visualització i inspecció de models volumètrics en un entorn de realitat virtual suportat pel motor gràfic Unity3D. Davant de la senzillesa de la segmentació per llindar (*threshold*), hem vist la utilitat de la segmentació per creixement de la regió (*region growing*), que es presenta com un mètode més sofisticat que discrimina en funció de la ubicació a la que es troba un vòxel.

Hem vist de primera mà la inviabilitat d'usar la CPU per a implementar tècniques de gràfics per computador. La seva menor capacitat de paral · lelització davant les GPU i el pas d'informació entre elles ens ha obligat a portar les implementacions per a ser executades a la targeta gràfica, no només com a un objectiu de màxims del projecte, sinó com a tasca imprescindible per a poder oferir una aplicació interactiva amb resposta visual en temps real.

S'ha estudiat la tècnica *render to texture* com a eina clàssica per a emmagatzemar els resultats d'un *fragment shader* que calcula les etiquetes de segmentació a una textura 3D que posteriorment és utilitzada per un algorisme de visualització d'un model volumètric. Trobar la solució per implementar aquesta tècnica a Unity3D va costar més temps del previst, es va oferir una solució amb els mateixos resultats fent ús de *compute shaders*, que es presenten com a programes d'ús genèric per a ser executats aprofitant la capacitat de paral · lelització de les targetes gràfiques.

Com a treball de futur immediat d'aquest projecte, es proposa la implementació (amb les seves corresponents proves de reniment) de la versió híbrida de la segmentació per creixement de regió amb el criteri de difusió i el criteri de llindar (Schenke, Wünsche i Denzler, 2005) per aconseguir segmentar més ràpidament aquelles regions properes a les llavors inicials. També volem mencionar que als testos de la segmentació per GPU, la tècnica *render to texture* ens ha donat millors resultats, però els *compute shaders* permeten modificar explícitament el nombre de *threads* que s'executen en paral · lel, de manera que es podrien fer proves amb diferents configuracions d'aquests per veure si poden arribar a ser més ràpids.

A pesar que hem aconseguit l'objectiu marcat a l'inici del desenvolupament d'aquest projecte, existeixen optimitzacions de la segmentació per creixement de la regió (Sherbondy, Houston i Napel, 2003), fent ús d'una màscara computacional per minimitzar el nombre de píxels a ser evaluats. Aquesta solució s'aconsegueix jugant amb el *depth buffer* quan es treballa amb *fragment shaders*, però en aquest moment no hem fet recerca de com aconseguir-ho amb els *compute shaders*.

4 Gestió del projecte

4.1 Metodologia de treball i eines de seguiment

4.1.1 Metodologia

La metodologia de treball seguirà el model àgil de cicles curts. Cada setmana es realitzaran una o dues reunions presencials (si és necessari, es realitzaran reunions telemàticament) amb l'Eva Monclús Lahoya, membre del grup de recerca ViRVIG i tutora d'aquest TFG, que s'encarregarà de l'assessorament més tècnic. Setmanalment s'establirà una feina a realitzar en una setmana vista, de manera que les reunions han de servir per aclarir dubtes, millorar la proposta del desenvolupador (que seré jo) i plantejar el següent objectiu. A més, al Centre de Realitat Virtual de la UPC, centre d'operacions del ViRVIG (Department of Computer Science, 2020), es podran fer les proves d'immersió amb l'aplicació per verificar la qualitat de la solució en cada etapa de la realització del projecte.

La resta de feina es realitzarà des de casa, utilitzant l'eina *Git* (Conservancy, 2020) per al control de versions del software i per tenir el projecte sincronitzat amb l'ordinador connectat al dispositiu de RV que el centre de recerca posa a disposició.

El desenvolupament de l'aplicació començarà primer per la implementació d'aspectes bàsics de segmentació per tal familiaritzar-me amb el codi del projecte actual, l'estil de programació exigit i l'entorn de RV. Tot el desenvolupament es dividirà en subobjectius que permetin fer testos i avaluar amb major freqüència la correctesa de la solució proposada en aspectes d'eficiència i usabilitat. D'aquesta manera plantejarem avaluacions parcials amb diferents membres del grup de recerca per tal d'obtenir *feedback* a les etapes inicials de la nostra proposta. Si la situació de pandèmia ocasionada per la COVID-19 ho permet, no descartem fer un test d'usuari a escala més àmplia on siguin usuaris de la comunitat mèdica les persones que avaluin la nostra proposta.

4.1.2 Seguiment

Com s'acaba de comentar, la metodologia de treball emprada incentiva un contacte estret amb la tutora i permet un fàcil seguiment de l'avenç del projecte. A més el director del projecte, en Pere Pau Vázquez, s'encarregarà de la validació de l'assoliment dels objectius, de manera que amb ell també es realitzaran reunions presencials setmanalment. S'utilitzarà la plataforma web *Trello* (Altassian, 2020) per organitzar les tasques a realitzar, tenir un bon control del calendari i poder definir bé les fites a assolir (*deadlines*). Pel contacte telemàtic amb la tutora i director, tenim a disposició el servei de correu electrònic que ofereix la UPC, així com l'aplicació *Google Meet* (Google, 2020) per a les reunions i un canal d'*Slack* (Slack Technologies, 2020) per a converses que

requereixen un format de pregunta-resposta més ràpid.

4.2 Planificació temporal

4.2.1 Descripció de les tasques

Aquesta secció pretén formalitzar tota la planificació prevista amb l'objectiu de tenir bona claredat respecte al volum i la càrrega de treball que comporta aquest projecte. Una bona definició prèvia de les tasques que cal realitzar per arribar a complir l'objectiu final del projecte ens ha permès tenir una visió clara de com hem de gestionar el temps de treball.

El projecte es considera començat el dia 1 de setembre de 2020, i l'objectiu és considerar-lo acabat el dia 17 de desembre de 2020, que marca la data límit d'inscripció al torn de lectura del mes de gener, tot i que considerem que disposarem del temps entre la data límit d'inscripció i el dia del torn de lectura per si fos necessari acabar de lligar caps. Aquest interval consta de 108 dies, i es pretén dedicar entre 4 i 5 hores diàries. Els dies on es pot desenvolupar el projecte a l'espai de treball que ofereix el ViRVIG (Department of Computer Science, 2020) es podrà ampliar la jornada dedicada, ja que seran els dies que es dedicaran a les proves de correctesa i correccions especificades pel director o tutora. Els caps de setmana aniran enfocats a l'elaboració de la documentació requerida per la presentació del projecte. A continuació es donarà informació detallada de les tasques i com s'han organitzat, així com una explicació de la nomenclatura utilitzada.

S'han definit un total de 32 tasques, algunes de les quals són considerades *fites*, que ens han servit per identificar els punts clau i crítics de tot el desenvolupament. Destaquem dos grans blocs dins del conjunt de tasques: la *[GP] Gestió del projecte*, i el *[DP] Desenvolupament del projecte*. Totes les tasques que integren cada un d'aquests blocs s'etiqueten seguint el format *[xP.N]*, on *N* és un nombre enter que identifica cada tasca. Les *fites* seguiran el format *xP.F*. Amb aquesta nomenclatura ha sigut molt fàcil identificar i assignar dependències entre les tasques, i es detallaran més aquelles que no segueixin el model *Fi - Inici*. També volem mencionar que al llarg de la planificació s'han marcat les reunions ([R] Reunió de seguiment) coincidint amb els dies de treball presencial pactats amb els responsables del projecte, tal i com s'ha explicat a la secció 4.1.2.

[GP] Gestió del projecte

La gestió del projecte és la fase que permet contextualitzar i definir l'abast del projecte, justificar-ne la necessitat i identificar totes les eines que faran falta per dur-lo a terme. S'estima una dedicació d'aproximadament 110 hores.

També hem inclòs l'elaboració de la memòria final (*[MEM] Elaboració de la memòria final*), que es realitzarà durant l'etapa final del desenvolupament del projecte, però ens dóna més claredat

alhora de visualitzar el conjunt de tasques incloure-la dins de la gestió del projecte. S'estima una dedicació de 28 hores.

[GP.1] Contextualització i abast

Aquesta fase és el punt de partida del projecte. Permet la introducció de tots els elements que aquest requereix. En aquest punt es realitza una bona tasca de recerca per identificar el problema, entendre l'estat de l'art del camp que es tracta per justificar la solució que es proposa, i s'estableixen les pautes de metodologia i seguiment que s'utilitzaran al llarg del desenvolupament. La dedicació ha sigut d'aproximadament de 30 hores.

[GP.2] Planificació

Aquesta fase serveix per identificar totes les tasques, fites i punts crítics que intervenen en el desenvolupament del projecte. Aquesta tasca marca el camí per assolir l'abast definit a la fase anterior, i permetrà anticipar-se davant dels obstacles i riscos reconeguts a l'estudi previ. Una bona planificació des de l'inici ha de permetre poder-lo adaptar en fases del desenvolupament ja avançades, en cas que ens trobem davant d'un obstacle imprevist i sigui necessari optar per altres solucions. La dedicació ha sigut d'aproximadament 16 hores.

[GP.3] Pressupost i sostenibilitat

En el context d'aquest projecte, un cop se'n concep l'abast i la dedicació que requereix, haurem de quantificar-ne el cost tenint en compte el personal implicat i les eines i dispositius requerits, així com estimar el cost que puguin aportar els imprevistos que poden aprarèixer en fases més avançades. A més, es realitzarà un informe de sostenibilitat tenint en compte l'impacte medioambiental, econòmic i social del desenvolupament del projecte, així com de la producció i ús del material que utilitzarem. S'estima una dedicació de 14 hores a aquesta tasca.

[GP.4] Elaboració del document final

Aquesta tasca consistirà en fer un recull de les tasques anteriors, corregir els errors i afegir les millores suggerides pel tutor designat pel seguiment de la Gestió del projecte. En acabar aquesta tasca es tindrà un únic document amb tot el contingut mencionat anteriorment i es podrà complir amb la fita *[GP.F] Entrega de l'informe de Gestió del projecte.* Depenent dels canvis que calgui efectuar sobre la documentació, s'estima una dedicació màxima de 32 hores.

Durant la realització de la gestió del projecte es faran reunions setmanalment amb el director i la tutora. Aquestes reunions seran d'entre 1 i 3 hores, un o dos cops a la setmana durant tota la fase

de la gestió del projecte. S'estima un cost temporal de 15 hores.

[DP] Desenvolupament del projecte

Aquesta secció engloba tot el procediment relacionat amb la implementació de la nostra aplicació. Veurem que la metodologia emprada estableix tasques que permeten fer-ne un seguiment setmanal, i la línia evolutiva del desenvolupament anirà des de la implementació de parts o eines enfocades a establir un primer contacte amb l'aplicació fins a la introducció de les noves funcionalitats que pretenem afegir. Al moment de la presentació d'aquest informe de seguiment, es preveu una dedicació final de desenvolupament de 368 hores, envers les 300 hores que es van preveure durant la fase inicial de planificació. A continuació es detallarà quins aspectes han canviat.

Durant tot el procés de desenvolupament, es realitzaran reunions setmanalment amb el director del projecte, tal i com s'especifica a la secció de seguiment (vés a 4.1.2) d'acord amb la metodologia de treball escollida. Aquestes reunions generalment seran d'una hora.

[DP.1] Formació i preparació

Aquesta tasca ha consistit en la lectura de la memòria del projecte del qual partim (Sànchez, 2017), així com del codi font sobre el qual haurem d'implementar les nostres funcionalitats. Ha sigut molt important per entendre el model de dades de l'aplicació i com organitza les eines d'interacció per afegir-ne de noves. S'ha dedicat un temps d'unes 24 hores.

[DP.2] Threshold Segmentation per CPU

Aquesta tasca marca l'inici de la implementació de la nostra solució un cop s'ha assolit amb èxit l'objectiu de la tasca anterior. Aquest algorisme de segmentació és trivial (Schenke, Wünsche i Denzler, 2005), de manera que la seva implementació ha servit per poder jugar amb el model de dades, entendre com es guarda la informació dels vòxels i com accedir i modificar el seu etiquetatge. Encara que el temps dedicat a aquesta tasca no és superior a les 8 hores, el seu abast temporal ha estat major, ja que l'hem considerat acabada quan hem tingut implementat un sistema d'interacció bàsic (*[DP.3] Interacció bàsica*) per llegir la informació requerida dels vòxels i per aplicar l'algorisme i obtenir resultats visuals.

[DP.3] Interacció Bàsica

A l'inici del desenvolupament del projecte, es va planificar la implementació d'un sistema d'interacció bàsic per poder seleccionar els vòxels de les regions d'interès. Aquest procés és neccessari per obtenir les dades que necessiten els algorismes de segmentació. Aquesta interacció consistia en

disposar d'una esfera que servia per seleccionar els vòxels i obtenir-ne els valors d'intensitat per a llavors tenir el rang de *threshold* que utilitza la segmentació per a seleccionar o descartar la resta de vòxels.

Un cop es van fer les proves de correctesa, vam veure que aquest mètode oferia poca usabilitat a l'usuari alhora de voler seleccionar vòxels que es troben molt endins del model volumètric, encara que afegíssim un paràmetre a la metàfora d'interacció per a poder determinar la distància a la que es troba l'esfera del controlador de l'usuari. Així doncs es va afegir una nova mètafora que hem anomenat *pla de clipping*. Aquesta ofereix a l'usuari un pla perpendicular a la direcció del dispositiu que controla l'usuari, i descarta *renderitzar* tots aquells vòxels que es troben a la direcció positiva de la normal del pla. Així l'usuari pot accedir amb més facilitat a zones internes del model volumètric.

Aquesta nova metàfora també permet canviar la visualització del model volummètric, de manera que en comptes de visualitzar el model segons la funció de transferència, es pot visualitzar amb una escala de grisos pròpia d'una imatge obtinguda amb rajos X, oferint així a l'usuari una nova percepció de com és l'objecte d'interès. A més, l'esfera que marca els punts d'interès per l'usuari es troba sempre al punt d'intersecció entre la recta de direcció del controlador i el pla, de manera que l'usuari no ha de fer assajos per determinar a quina distància ha d'estar l'esfera per a poder seleccionar els vòxels de forma òptima, ja que simplement ha de posar el pla sobre la regió d'interès, i ja tindrà l'esfera preparada per marcar vòxels.

El temps total de dedicació a la interacció ha sigut d'aproximadament 64 hores, envers les 8 que es van preveure quan només teníem intenció d'implementar l'esfera. Aquesta diferència és deguda a una altra tasca que es va afegir més endavant (*[DP.3.4] Selecció i visualització de llavors per GPU*), la qual explicarem a la secció que va originar el problema que aquesta soluciona (vés a 4.2.1).

[DP.4] Region Growing per CPU

Aquesta tasca és el primer punt clau del desenvolupament del projecte. La implementació per CPU és més senzilla que per GPU, de manera que aquesta fase ens portarà a un punt mig on podrem entendre el funcionament de la segmentació basada en regió, fet que pot facilitar en el futur la implementació que volem oferir per GPU.

Aquesta tasca es desglossa en diferents fases de comprensió i disseny (*DP.4.1*, 10 hores) i d'implementació (*[DP.4.2]*, 4 hores; *[DP.4.3]*, 16 hores; *[DP.4.5]*, 8 hores). Es marquen també dues fites (*[DP.4.F.1]* i *[DP.4.F.2]*) on es pretén realitzar proves de correctesa, que es preveu que requeriran una dedicació d'unes 4 hores cadascuna. Finalitzarem la tasca amb la realització de tests d'usabilitat (*[DP.4.6]*) que es poden dur a terme entre disseny, implementació i avaluació per part dels usuaris, en 28 hores.

[DP.5] Region Growing per GPU

Aquesta tasca pretén adaptar l'algorisme dissenyat anteriorment per ser executat per la GPU. Aquest també es presenta com un punt clau del desenvolupament del projecte, ja que la implementació per GPU serà la base de l'estudi d'eficiència de la nostra aplicació. A la fase inicial del TFG aquesta tasca la vam desglossar en:

- ***[DP.5.1]*** **Lectura i comprensió** de les especificacions sobre la implementació per GPU que ofereixen els articles dels quals partim (Sherbondy, Houston i Napel, 2003; Schenke, Wünsche i Denzler, 2005; Chen et al., 2006) i altres conceptes relacionats (N. Corporation, 2020). Temps previst: 16 hores.
- ***[DP.5.2]*** **Implementació** de l'algorisme. Temps previst: 36 hores.
- ***[DP.5.3]*** **Possibles millores** d'eficiència, combinar l'algorisme amb un *threshold-based* (Schenke, Wünsche i Denzler, 2005), etc. Temps previst: 24 hores.
- ***[DP.5.F]*** **Prova de correctesa**.

Fins al moment de començar amb aquesta tasca, totes les implementacions feien ús sencer de la CPU, des de la selecció de vòxels de l'usuari fins a la selecció de vòxels dels algorismes. Sense entrar molt al detall de la implementació, que es deixa per la memòria final, la segmentació guarda les etiquetes dels vòxels seleccionats en una textura de la mateixa ressolució que el volum. Quan volem veure visualment el resultat de la segmentació, hem d'indicar al motor gràfic (Unity Software Inc., 2016) que envii aquesta textura actualitzada a la GPU per a que el *shader* que s'encarrega de processar aquesta informació computi la imatge actualitzada.

Aquest pas, tal i com s'indica a la literatura utilitzada (Smistad et al., 2015), és molt costós, i observàvem un augment de latència que bloquejava l'actualització (el *main loop*) de la nostra aplicació, fet problemàtic per vàries raons. Per una banda, l'usuari no pot veure en temps real quins vòxels està marcant, i cada cop que vulgui veure l'estat de la seva selecció ha de cridar a enviar la textura actualitzada i veure com l'aplicació es congela durant 1 o 2 segons. Per altra banda, *region growing* és un algorisme iteratiu, de manera que és interessant poder veure com evoluciona la segmentació per detectar si l'algorisme està etiquetant vòxels que no pertanyen a la regió d'interès i d'alguna manera limitar-ne l'expansió.

A partir d'aquí, abans de començar amb la implementació per GPU, es va decidir readaptar la metàfora d'interacció per a poder etiquetar els vòxels seleccionats per l'usuari escribint directament sobre la textura de segmentació carregada ja a la GPU. Aquest procediment es coneix usualment com a *render to texture*, i consisteix en redireccionar la sortida d'un *fragment shader* perquè s'escrigui en una textura que després serà llegida pel *shader* que renderitza la segmentació. Així doncs, vam començar a investigar com utilitzar l'API de Unity (Unity Software Inc., 2016) per

implementar aquesta tècnica (*[DP.5.2.1], 16 hores*). Malauradament ens vam trobar amb poc suport per part de la documentació oficial, i vam trobar sobretot que la comunitat tenia molts problemes alhora de dissenyar una seqüència de crides a l'API per a implementar aquesta tècnica.

Finalment, vam decidir abordar el problema fent ús de *compute shaders*. Aquests es defineixen com a *shaders* d'ús genèric que permeten fer càlculs per GPU fora de l'àmbit de gràfics per computador, aprofitant la capacitat de paral · lelització que ofereixen les targetes gràfiques (Microsoft, 2018). Aquests ofereixen tres grans avantatges: permeten a l'usuari definir el nombre de *threads* que executen un mateix *kernel*; permeten definir textures 3D com a estructures de dades amb accessos de lectura i escriptura; l'API de Unity ofereix la classe *ComputeShader* per poder-los configurar (Unity Software Inc., 2016). Quan es donin més detalls sobre la implementació, també veurem que són útils davant dels desavantatges de *region growing* analitzats a la secció 1.4.2.2.

D'aquesta manera, la textura de segmentació del model volumètric es troba referenciada a dos *compute shaders* amb accés d'escriptura, un per a poder pintar els vòxels d'interès per l'usuari, i un altre que tindrà el procés d'una iteració de *region growing*. Utilitzem el *main loop* de Unity per fer una crida al *shader* corresponent per cada situació, i el *fragment shader* inicial (el que s'encarregava de visualitzar la informació de segmentació) simplement llegirà d'aquesta textura actualitzada per la GPU, sense haver-la de transferir des de la CPU cada cop.

Al moment de redactar aquest informe de seguiment, l'estat del projecte es troba havent finalitzat aquesta tasca. Sí que és veritat que quan es va proposar la planificació durant la fase inicial, no vam preveure el problema real de treballar amb l'alta latència de passar informació de CPU a GPU, ni que haguéssim de readaptar la tècnica *render to texture* per a la implementació de l'algorisme. És per això que aquesta tasca ha ocupat 40 hores de més. Per altra banda, ja tenim una versió d'un *seeded region growing* per GPU que funciona, i el projecte es troba en una fase que anomenem de *refinament*, envers la fase de millores (*[DP.6] Millores*) que vam definir durant la planificació inicial sense cap especificació.

[DP.6] Refinament

Un cop tenim una primera versió funcional d'un algorisme de segmentació per GPU, hem realitzat un estudi de millores de la interacció i usabilitat (*[DP.6.A]*). Tenim previst acabar les següents tasques abans d'acabar l'any, i deixar les vacances per acabar de redactar la memòria.

- Afegir la funcionalitat de poder esborrar una *seed*.
- Permetre a l'algorisme que s'aturi automàticament si detecta que no s'han etiquetat nous vòxels.
- Poder guardar i recuperar la textura de segmentació.
- Poder plantar *seeds* que no permetin l'expansió de la segmentació.

- Tenim previst comprovar la performance de la solució oferta amb models volumètrics més grans (256x256x256, i 512x512x256).

[DP.7] Tests d'usabilitat

Un cop tinguem l'aplicació acabada, procedirem a realitzar tests d'usabilitat, per comprovar que podem oferir una solució que compleix els requisits definits durant la fase d'estudi de l'abast. Podrem dedicar unes 24 hores a fer testos d'usabilitat, ja que s'hauran de realitzar al Centre de Realitat Virtual (Department of Computer Science, 2020), amb la supervisió del director i la tutora.

4.3 Recursos

4.3.1 Recursos humans (RH)

Aquest projecte requereix de la intervenció de quatre rols ben definits:

- **Cap de projecte** S'encarrega del curs que ha de seguir tot el desenvolupament, des de la planificació, les reunions i la documentació.
- **Investigador** S'encarrega de fer tot el treball relacionat amb la recerca d'informació per assolir l'abast del projecte.
- **Programador** S'encarregarà d'implementar la solució definida per la gestió del projecte.
- **Avaluador** S'encarregarà de realitzar les proves de validació de l'aplicació, tan del seu correcte funcionament com dels tests d'usabilitat amb usuaris.

4.3.2 Recursos materials (RM)

Llistem el conjunt de RM que utilitzarem per a la realització del projecte:

- **PC** És important disposar d'ordinador propi pels dies que no es treballi al Centre de Realitat Virtual (Department of Computer Science, 2020).
- **Overleaf** Aplicació web per crear documents en format LaTeX(Overleaf, 2020).
- **Unity3D** Motor gràfic (Unity Software Inc., 2020).
- **HTC Vive** Dispositiu de RV immersiu (H. Corporation, 2020).
- **GanttProject** Eina de lliure distribució per elaborar el diagrama de Gantt i l'assignació de RH (BarD Software, 2020).

La taula 1 sintetitza tota la informació que s'ha donat fins ara. S'ha assignat totes les tasques a un o més membres de l'equip (veure secció 4.3.1), i per cada una s'indica el material requerit. Pel què fa a les reunions de seguiment (tasques amb etiqueta *[R]*), s'ha assumit que hi assistiran els membres que estiguin fent la tasca d'aquella setmana i el cap de projecte, el qual s'encarrega de la supervisió de tot el desenvolupament.

Al diagrama de Gantt adjuntat a l'apèndix (figura 49) es mostren els detalls de les tasques amb més profunditat. Concretament, per cada tasca es veu la seva data d'inici i finalització, la durada en dies, els recursos humans assignats i el risc que se li ha associat. El risc també es veu reflectit amb textures que es troben per sobre de les franges de colors que marquen el temps que ocupen les tasques.

4.4 Gestió del risc: Plans alternatius i obstacles

Al diagrama de Gantt (veure figura 49) podem veure algunes trames marcades amb una textura més fosca. Aquestes corresponen a les tasques que considerem que pertanyen al camí crític o de risc. Aquestes són tasques que estan relacionades amb els obstacles i riscos analitzats durant l'etapa de *[DP.1] Contextualització i abast* (veure seccions 2.2 i 2.3).

Per una banda, les tasques d'implementació de l'algorisme de segmentació basat en regió per CPU (*[DP.4]*) les considerem crítiques perquè serà molt important que aquesta implementació sigui correcta i eficient, ja que un endarreriment o una mala implementació pot fer-nos arrossegar problemes d'eficiència o de temps a dedicar quan vulguem passar la implementació a la GPU. És per això que s'ha volgut desglossar bé aquesta tasca i fer un seguiment a detall de les diferents fases de les que consta l'algorisme. Serà molt important obtenir bons resultats dels tests d'usabilitat associats a aquesta tasca (*[DP.4.6]*).

De la mateixa manera, la implementació per GPU (*[DP.5]*) és crítica perquè representa una part fonamental de l'aplicació que volem oferir, de manera que serà molt important tenir ben clar el disseny que farem abans de posar-nos-hi i realitzar una bona implementació amb el temps que li hem assignat, per a poder tenir marge més endavant per estudiar i aplicar les millores. Finalment, els tests d'usabilitat també els considerarem una tasca de risc, ja que si la nostra aplicació no compleix els requisits d'usabilitat desitjats, potser no la podem tenir llesta abans de la *[F] Inscripció al Torn de Lectura*.

Un dels majors obstacles que podem tenir és la impossibilitat de realitzar les proves d'usabilitat al llarg del desenvolupament degut a l'emergència sanitària, ja que no podem preveure si en algun moment ens és impossible poder accedir a l'espai de treball i fer que varis usuaris utilitzin el dispositiu de RV. Si veiem que fos complicat poder fer estudis amb la interacció, ens veurem obligats a optar per enfocar la tasca *[DP.6] Millores* a oferir millores relaciones amb l'algorisme de segmentació que com ja vam comentar, al presentar-se com un algorisme de segmentació clàssic,

Nom	Durada(h)	Relacions	RH	RM
[GP] Gestió del projecte	**135**	-	-	
[GP.1] Contextualització i abast	30	-	CP, I	PC, Overleaf
[GP.2] Planificació temporal	16	[GP.1]	CP	PC, Overleaf, GantProject
[GP.3] Pressupost i sostenibilitat	14	[GP.2]	CP	PC, Overleaf, GantProject
[GP.4] Integració del document final	32	[GP.3]	CP	PC, Overleaf
[GP.F] Entrega d'informe de Gestió del projecte + Reunions	15	-	CP	PC, Overleaf
[MEM] Elaboració de la memòria Final	28	-	CP	PC, Overleaf
[DP] Desenvolupament del projecte	**298**	-	-	
[DP.1] Formació i preparació.	24	-	I, P	PC, Unity3D, HTC Vive
[R] Reunió de seguiment	1	-	CP, I, P	PC
[DP.2] Threshold Segmentation per CPU	8	[DP.1]	P	PC, Unity3D, HTC Vive
[R] Reunió de seguiment	1	-	CP, P	PC
[R] Reunió de seguiment	1	-	CP, P	PC
[DP.3] Interacció bàsica	10	[DP.2] Fi-Fi	P	PC, Unity3D, HTC Vive
[DP.3.1] Esfera per llegir valors de voxels	7	-	P	-
[R] Reunió de seguiment	1	-	CP, P	-
[DP.3.2] Botó "aplica segmentació"	1	[DP.3.1]	P	-
[R] Reunió de seguiment	1	-	CP, P	-
[DP.4] Region Growing per CPU	79	[DP.2]	-	-
[DP.4.1] Comprensió de l'algorisme	10	-	I	PC
[R] Reunió de seguiment	1	-	CP, I	PC
[DP.4.2] Plantat de llavors	4	[DP.4.1]	P	PC, Unity3D
[DP.4.3] Evolució de la segmentació	16	[DP.4.2]	P	PC, Unity3D
[DP.4.4] Adaptar la interacció	2	[DP.4.3] Fi-Fi	P	PC, Unity3D, HTC Vive
[DP.4.F.1] Prova de correctesa	4	[DP.4.3]	P, A	PC, Unity3D, HTC Vive
[DP.4.5] Image Smoothing	8	[DP.4.3] Fi-Fi	P	PC, Unity3D
[R] Reunió de seguiment	1	-	CP, P, A	PC
[DP.4.F.2] Prova de correctesa	4	[DP.4.5]	P, A	PC, Unity3D, HTC Vive
[DP.4.6] Test d'usabilitat	28	-	A	PC, Unity3D, HTC Vive
[R] Reunió de seguiment	1	-	CP, P, A	PC
[DP.5] Region Growing versió híbrida (CPU - GPU)	82	[DP.4]	-	-
[DP.5.1] Lectura i comprensió	16	-	I	PC
[R] Reunió de seguiment	1	-	CP, I	PC
[DP.5.2] Implementació	36	[DP.5.1]	P	PC, Unity3D
[R] Reunió de seguiment	1	-	CP, P	PC
[DP.5.3] Possibles millores	24	[DP.5.2]	I, P	PC, Unity3D
[DP.5.F] Prova de correctesa	4	[DP.5.3]	P, A	PC, Unity3D, HTC Vive
[DP.6.A] Millores d'interacció	8	-	CP, I	PC
[DP.6.B] Millores de l'algorisme	8	-	CP, I	PC
[R] Reunió de seguiment	1	-	CP, I	PC
[DP.6] Aplicar millores	50	[DP.6.A];[DP.6.B]	P	PC, Unity3D, HTC Vive
[R] Reunió de seguiment	1	-	CP, P	PC
[DP.7] Tests d'usabilitat	24	-	A	PC, Unity3D, HTC Vive
Total	***433***	-	-	-

Taula 1: Taula amb la planificació del projecte. Per cada tasca s'inclou la durada en hores, les relacions de dependència i els recursos humans (RH) i materials requerits (RM). Llegenda: CP - Cap de projecte, I - Investigador, P - Programador, A - Avaluador.

al llarg dels anys han anat aprareixent-ne adaptacions i millores.

4.5 Gestió econòmica

4.5.1 Pressupost

4.5.1.1 Recursos humans (RH)

S'ha calculat la despesa que generen els RH identificats a l'etapa de *Planificació temporal* (vegeu la secció 4.3) a partir de l'assignació de personal que es va fer de totes les tasques. Per poder calcular les partides per tasca primer calculem el salari per hora dels diferents components de l'equip que s'encarreguen del projecte. A la taula 2 es detalla el salari dels quatre rols que vam identificar, basant-nos amb el salari mitjà anual a Espanya d'aquests perfils professionals, publicat a un sondeig de la població del sector TIC (CCOO, 2020). Concretament, el salari mitjà pel càrrec de cap de projecte és de 44000 euros bruts anuals, per l'investigador 33550, per un programador (sènior) 23800 i per l'avaluador 29000. El salari per hora s'ha calculat suposant que l'any té 253 dies laborables (Laborables, 2020) i la jornada laboral és de 8 hores diàries. A la taula 2 hi ha el salari per hora resultant arrodonit.

Nom	Tarifa estàndard (euros/hora)
Cap de projecte	22
Investigador	17
Programador	12
Avaluador	14

Taula 2: Taula de costos del personal. Elaboració pròpia. Font: CCOO, 2020

La taula 3 mostra les partides per tasca, tenint en compte el cost del personal de la taula 2 i les tasques definides durant l'etapa de planificació de la gestió del projecte. El cost total de personal -incloent la quota de la Seguretat Social[10]- és de 11626 euros.

4.5.1.2 Cost genèric (CG)

Hem identificat el cost genèric del projecte a partir de les eines que vam concretar a l'etapa de planificació. S'ha dividit l'estimació del càlcul discriminant entre els recursos de *software* (veure taula 4) i els dispositius de *hardware* (veure taula 5).

Les empreses que proveeixen el *software* privatiu normalment ofereixen plans de pagamament anuals o mensuals. Com que estimem una durada de cinc mesos per la total finalització del

[10] $salari \times 1.3$

Nom	Durada(h)	RH	Cost(euros)	Cost S.S (euros)
[GP] Gestió del projecte	**135**	-	**3446**	**4479**
[GP.1] Contextualització i abast	30	CP, I	1154	1500
[GP.2] Planificació temporal	16	CP	349	454
[GP.3] Pressupost i sostenibilitat	14	CP	306	397
[GP.4] Integració del document final	32	CP	698	908
[GP.F] Entrega d'informe de Gestió del projecte + Reunions	15	CP	327	426
[MEM] Elaboració de la memòria Final	28	CP	611	794
[DP] Desenvolupament del projecte	**298**	-	**5498**	**7147**
[DP.1] Formació i preparació.	24	I, P	683	888
[R] Reunió de seguiment	1	CP, I, P	50	65
[DP.2] Threshold Segmentation per CPU	8	P	94	123
[R] Reunió de seguiment	1	CP, P	34	44
[R] Reunió de seguiment	1	CP, P	34	44
[DP.3] Interacció bàsica	10	P	162	210
[DP.3.1] Esfera per llegir valors de voxels	7	P	83	107
[R] Reunió de seguiment	1	CP, P	34	44
[DP.3.2] Botó "aplica segmentació"	1	P	12	15
[R] Reunió de seguiment	1	CP, P	34	44
[DP.4] Region Growing per CPU	79	-	1267	1648
[DP.4.1] Comprensió de l'algorisme	10	I	166	216
[R] Reunió de seguiment	1	CP, I	38	50
[DP.4.2] Plantat de llavors	4	P	47	61
[DP.4.3] Evolució de la segmentació	16	P	189	246
[DP.4.4] Adaptar la interacció	2	P	24	31
[DP.4.F.1] Prova de correctesa	4	P, A	105	136
[DP.4.5] Image Smoothing	8	P	94	123
[R] Reunió de seguiment	1	CP, P, A	48	62
[DP.4.F.2] Prova de correctesa	4	P, A	105	136
[DP.4.6] Test d'usabilitat	28	A	403	524
[R] Reunió de seguiment	1	CP, P, A	48	62
[DP.5] Region Growing versió híbrida (CPU - GPU)	82	-	1551	2016
[DP.5.1] Lectura i comprensió	16	I	266	346
[R] Reunió de seguiment	1	CP, I	38	50
[DP.5.2] Implementació	36	P	425	553
[R] Reunió de seguiment	1	CP, P	34	44
[DP.5.3] Possibles millores	24	I, P	683	888
[DP.5.F] Prova de correctesa	4	P, A	105	136
[DP.6.A] Millores d'interacció	8	CP, I	308	400
[DP.6.B] Millores de l'algorisme	8	CP, I	308	400
[R] Reunió de seguiment	1	CP, I	38	50
[DP.6] Aplicar millores	50	P	590	767
[R] Reunió de seguiment	1	CP, P	34	44
[DP.7] Tests d'usabilitat	24	A	345	449
Total	***433***	-	***8943***	***11626***

Taula 3: Taula de partides per cada tasca. Llegenda: CP - Cap de projecte, I - Investigador, P - Programador, A - Avaluador.

projecte, optarem per les subscripcions mensuals. Per la llicència d'Overleaf (Overleaf, 2020) només comptarem dos mesos, ja que l'utilitzarem només durant la gestió del projecte (des de 22/09 fins el 19/10) i l'elaboració i entrega de la memòria final (mes de desembre), sent el cost total de l'ús dels recursos *software* (SW) de 778 euros.

Recurs	**Mesos d'ús**	**Preu mensual (euros)**	**Cost total (euros)**
Overleaf	2	14	28
Unity3D	5	150	750
GanttProject	5	0	0
Total	-	***164***	***778***

Taula 4: Taula de costos dels recursos de *software*. Les caselles que marquen el preu mensual per cada producte n'indiquen les fonts.

Per al càlcul del cost del *hardware* s'ha estimat l'amortització[11] de cada element suposant els mateixos dies laborables i hores diàries de l'apartat anterior (vés a 4.5.1.1). Hem assumit una vida útil de quatre anys dels ordinadors i de dos anys del dispositiu de RV (H. Corporation, 2020), ja que es tracta d'un component més delicat i relacionat amb un sector en una fase de creixement i innovació molt ràpida, fet que en redueix el temps d'obsolescència. El cost dels ordinadors s'ha estimat segons Logical Increments (Increments, 2020), un portal web que publica diferents configuracions basant-se amb el rendiment que ofereixen davant del processament de gràfics per a usos a nivell d'usuari. Hem comptat que cada membre de l'equip disposarà d'un ordinador portàtil per realitzar totes i cada una de les tasques que té assignades (vegeu la taula 1). El cost total de l'ús dels recursos *hardware* (HW) és de 358 euros, i es mostra amb detall a la taula 5.

Recurs	**Preu (euros)**	**Unitats**	**Vida útil (mesos)**	**Hores**	**Amortització (euros)**
PC sobretaula	1500	1	4	290	62
HTC Vive	800	1	4	290	33
PC treball remot	800	4	2	290	264
Total	*3100*	-	-	-	***358***

Taula 5: Taula de costos dels recursos de *hardware*.

4.5.1.3 Contingència

A continuació es formalitza un sobre-cost del projecte per cobrir els possibles obstacles i imprevistos. Com que es tracta d'un projecte amb diferents etapes que requereixen investigació i és molt important la precisió i eficiència de la solució que pretén oferir, s'ha fixat un sobre-cost del 15%. La taula 6 mostra els detalls del cost total de la contingència, que és de 1914 euros.

[11] $\frac{cost\text{-}producte}{vida\text{-}útil \times hores\text{-}hàbils\text{-}anuals}$

Tipus	Cost (euros)	Contingència (euros)
RH	11626	1744
SW	778	117
HW	358	54
Total	***12763***	***1914***

Taula 6: Taula de contingència del 15% pel tipus de despesa.

4.5.1.4 Imprevistos

Finalment estimem el cost produït pels obtacles i imprevistos que poden sorgir al llarg del projecte, els quals s'han identificat a partir dels obstacles i riscos definits a l'etapa de planificació temporal de la gestió del projecte, afegint a més el risc d'haver de tornar a comprar els dispositius *hardware* per alguna avaria. A continuació es justifica el cost per risc degut a la demora del temps de realització del projecte.

Quan vam identificar els obstacles i riscos, vam veure que els punts crítics del desenvolupament del projecte estaven relacionats amb la qualitat de la solució. És molt important que el nostre producte ofereixi un resultat correcte, de forma eficient i que sigui usable. En el cas que a les diferents fases de la implementació ens trobem que no podem avançar per corregir errors, augmentarà el cost dels RH assignats a aquella tasca. Concretament, per les tasques crítiques d'implementació (*DP.4* i *DP.5*) estimem que pot augmentar la càrrega de treball a una setmana per cada una, essent en total un augment de 40 hores pel programador, i un augment de 8 hores per l'avaluador (587 euros). També assumim una setmana de càrrega extra (20 hores pel programador, 4 hores per l'avaluador: 294 euros) per la correcció i reavaluació del producte en cas de suspendre els tests d'usabilitat (*[DP.7]*).

Comptem un risc del 20%, ja que els punts crítics es troben entre la metiat i la fase final del desenvolupament. El cost total dels imprevistos pels obstacles i riscos del projecte és de 176 euros.

Pel què fa al deteriorament dels dispositius, assumim només un risc del 5% pels ordinadors, i un 10% pels dispositius de RV. La taula 7 mostra el resum del cost per imprevistos, que és de 491 euros.

4.5.1.5 Pressupost inicial del projecte

Amb tota la informació obtinguda dels apartats anteriors, podem estimar que el cost total que tindrà el projecte és de 15168 euros. A la taula 8 es motren els detalls.

Imprevist	Cost (euros)	Risc (%)	Cost total (euros)
Demora del temps d'implementació	881	20	176
PC treball remot (x4)	3200	5	160
PC sobretaula	1500	5	75
HTC Vive	800	10	80
Total	*6381*	-	***491***

Taula 7: Taula de despeses degudes a imprevistos.

Concepte	Cost (euros)
RH	11626
SW	778
HW	358
Contingència	1914
Imprevistos	491
Total	***15168***

Taula 8: Pressupost inicial del projecte.

4.5.2 Control de gestió

L'objectiu del control de la gestió és comparar i avaluar les desviacions que pot haver entre el pressupost i els costos reals de la realització del projecte. Durant la fase de desenvolupament, a les reunions de supervisió i seguiment es podran identificar d'aprop les hores reals de cada tasca, fet que permetrà calcular les desviacions del cost de personal i de material, tant de l'ús de *hardware* com les subscripcions de les llicències del *software*.

A continuació es defineixen els indicadors de control per supervisar les desviacions dels costos durant l'execució del projecte.

- **Desviació del preu del personal per tasca** (*cost_estimat* - *cost_real*) × *hores_reals*
- **Desviació del preu per l'ús del *hardware*** (*cost_estimat* - *cost_real*) × *dies_amort_reals*
- **Desviació del preu per l'ús del *software*** (*cost_estimat* - *cost_real*) × *mesos_ús_reals*
- **Desviació en consum del personal per tasca** (*hores_estimades* - *hores_reals*) × *cost_estimat*
- **Desviació en consum (*hardware*)** (*dies_amort_estimats* - *dies_amort_reals*) × *cost_estimat*
- **Desviació en consum (*software*)** (*mesos_ús_estimats* - *mesos_ús_reals*) × *cost_estimat*
- **Desviació total pels RH** *desviació_preu_personal* + *desviació_consum_personal*

- **Desviació total pels recursos *hardware*** *desviació_preu_hw + desviació_consum_hw*
- **Desviació total pels recursos *software*** *desviació_preu_sw + desviació_consum_sw*

4.6 Informe de sostenibilitat

4.6.1 Autoavaluació

A partir de les respostes a l'enquesta sobre el coneixement que tenim les estudiants de plantejar i enfocar un projecte TIC amb una perspectiva sostenible, em disposo a detallar les conclusions extretes.

Durant tota la realització del grau, moltes activitats relacionades amb l'assoliment de *Competències Transversals* han tractat aspectes econòmics, socials i mediambientals dels productes i serveis de les TIC al món, més enllà de l'entorn acadèmic on ens formem. Algunes d'aquestes activitats m'han donat nocions més pròximes sobre problemes reals com l'obsolescència programada o la part més fosca del mercat de matèries primeres per a la fabricació i ús de components i eines utilitzades tant pel sector de les empreses TIC com pels seus usuaris. Com a conseqüència d'això, he respòs satisfactòriament la majoria de preguntes relacionades amb saber valorar els impactes econòmics, socials i mediambientals, tant de les solucions tecnològiques amb les que treballo dia a dia com de les que són usades per un conjunt ampli de la societat.

Per altra banda, he vist que tinc poc coneixement relacionat amb quantificar aquests aspectes. La realitat que veig és que la tecnologia sempre ha servit per facilitar i millorar la vida de les persones, però ja fa molt de temps que generalment tant les solucions TIC com les eines d'investigació i innovació no són accessibles pel conjunt global de la humanitat. Això ha facilitat al sector empresarial a augmentar el seu rendiment i a reduir i precaritzar molts de treball. A més, en el marc del sector TIC, moltes solucions limiten l'ús que en poden fer els seus usuaris, per raons no més enllà de les econòmiques. També veiem que durant els últims anys alguns sectors de la població han posat al punt de mira les grans empreses tecnològiques per construir monopolis (BBC, 2020) o per fer un ús deontològicament reprobable de les dades que extreuen dels seus usuaris (Clavell, 2015).

És per tot això i més que no era conscient de la feina relacionada amb quantificar la qualitat de les solucions envers la sostenibilitat que es fa des del sector TIC. Veient que em vull desmarcar d'aquesta tendència, espero que aquest TFG m'ajudi a trobar les eines necessàries per poder plantejar, crear i oferir solucions tecnològiques que assumeixin necessitats reals de les persones, tenint en compte l'impacte en l'explotació que fan aquestes solucions dels recursos naturals.

4.6.2 Impacte econòmic

Estimació del cost de realització del projecte

El pressupost d'aquest projecte posa molta èmfasi als recursos humans que requereix, essent l'element que marca la tendència del preu per a la realització del projecte. Inclús així, el desglossament de les tasques ens ha permès filar prim amb l'estimació del cost de la mà d'obra. Creiem que la selecció i ús del maquinari i *software* és molt concret i compleix amb els requisits mínims per poder elaborar una solució de qualitat. A més, aquest projecte es construirà sobre una aplicació ja existent, de manera que la nostra solució pretén abordar una petita part d'una que ja tenim disponible. D'aquesta manera valorem el producte que ens disposem a desenvolupar com a un recurs competitiu i a l'abast econòmic d'un òrgan o institució que el demani.

Solucions actuals: On es troba aquest projecte?

La gran diferència que veiem de les aplicacions actuals de segmentació de models volumètrics en 3D amb la nostra és la interacció, ja que tant metges com estudiants interactuen amb els models volumètrics a través d'una pantalla que ofereix eines d'interacció en 2D. És llavors mèrit de cada aplicació oferir eines que en faciliten la interacció. Sí que és veritat que a la nostra solució la interacció es basarà en un entorn de RV immersiu, amb tot el potencial que això suposa. Per altra banda, això requereix als usuaris de la nostra aplicació tenir més recursos, com un ordinador de gamma alta i el dispositiu de RV, comportant un cost extra al que s'estima només amb l'elaboració i obtenció de l'aplicació. També volem destacar que amb la ràpida evolució que estan patint els sistemes de RV actuals, una hipotètica obsolescència de la nostra tecnologia pot ser perjudicial pels seus usuaris.

4.6.3 Impacte ambiental

Impacte ambiental de la realització del projecte

Creiem que el consum relacionat amb el desenvolupament del projecte entra dins dels estàndards de consum habituals de la vida quotidiana de les persones, ja que el projecte es desenvoluparà en un espai de treball compartit (Department of Computer Science, 2020), on es coordinen i es realitzen varis projectes de forma simultània. També hem definit el material mínim requerit, que encara que l'ús que en donarem no comporti un gran consum d'energia, no hem tingut en compte l'impacte ambiental del seu procés de fabricació.

Problemes i solucions actuals: On es troba aquest projecte?

De la mateixa manera que quan s'ha comentat l'impacte econòmic, els requeriments mínims que es necessiten per al bon funcionament del nostre producte poden suposar un consum superior que les solucions actuals que no demanen tants recursos. També és important destacar que si l'impacte de la producció del nostre dispositiu de RV és qüestionable, estarem contribuint per partida doble a aquesta producció, ja que requerim als usuaris l'ús de dispositius com el nostre.

4.6.4 Impacte social

Aportació personal d'aquest projecte

A nivell personal, la realització d'aquest projecte em pot aportar tenir una visió més clara de l'ús i les aplicacions del processament de gràfics per computador, més enllà dels coneixements adquirits al respecte durant la realització del grau, que m'han servit per adquirir els coneixements fonamentals sobre les eines que hi ha avui en dia. El fet d'oferir una solució destinada a la comunitat mèdica em fa veure la importància de la informàtica i l'impacte que suposa per a altres sectors fins ara aliens durant el meu desenvolupament en la disciplina que vaig escollir estudiar.

Solucions actuals: Necessitat real del projecte

La segmentació de models volumètrics en un entorn 3D no és una eina encara molt utilitzada, i segueix formant part del camp de la recerca. Però hi ha la possibilitat real que poder segmentar en un entorn 3D ofereixi més precisió a l'usuari que fent-ho en 2D, ja que en un entorn de RV immersiu l'usuari pot veure tot el model amb percepció 3D. Caldria fer una avaluació de temps per veure si és més ràpid que emprant eines d'interacció en 2D, però com que actualment no s'han desenvolupat tècniques d'interacció específiques per a segmentació en entorns 3D, no disposem de suficient material per realitzar aquesta comparació.

D'aquesta manera, creiem que oferir una eina de segmentació de models volumètrics en un entorn de RV, encara que no sigui l'única, servirà per a incentivar la recerca i la innvocació dels sistemes de RV, amb l'objectiu de trobar i oferir solucions estables per a l'ús quotidià de les persones, més enllà de potenciar el sector de l'entreteniment. La recerca i innovació d'aquest sector pot portar solucions també més eficients, així com perfeccionar l'impacte a la salut -marejos, problemes de visió, etc.- de les persones que en fan ús. Si aconseguim concebre els sistemes de RV com a eines que faciliten i milloren la qualitat de vida de les persones, en un futur podria créixer encara més la producció d'aquestes tecnologies, podent estar a l'abast de tothom.

Referències

[1] Bernhard Preim i Charl P. Botha. *Visual Computing for Medicine: Theory, Algorithms, and Applications*. 2a ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013. ISBN: 9780124159792.

[2] Markus Hadwiger. Joe Michael. Kniss Christof. Rezk Rezk-salama. Daniel Weiskopf. Klaus J Engel. *Real-time Volume Graphics*. A. K. Peters, Ltd., 2006.

[3] Universitat Politècnica de Catalunya: Department of Computer Science. *ViRVIG: Visualització, Realitat Virtual i Interacció Gràfica*. URL: https://www.virvig.eu/ (cons. 26 de set. de 2020).

[4] Unity Technologies Unity Software Inc. *Unity Real-Time Development Platform | 3D, 2D VR & AR Engine*. URL: https://unity.com/ (cons. 26 de set. de 2020).

[5] Joan Fons Sànchez. *Inspecció interactiva i immersiva de models volumètrics. Aplicació diagnosi mèdica*. Inf. tèc. Universitat Politècnica de Catalunya, 2017.

[6] HTC Corporation. *HTC Vive*. URL: https://www.vive.com/eu/ (cons. 26 de set. de 2020).

[7] Hung-Li Chen et al. "Sketch-based Volumetric Seeded Region Growing". A: 2006, pàg. 123 - 130. ISBN: 3905673398. DOI: 10.2312/SBM/SBM06/123-129.

[8] Josep Ciurana Herrera. *Visualització avançada de models mèdics*. Inf. tèc. Universitat Politècnica de Catalunya, 2020.

[9] Stefan Schenke, Burkhard Wünsche i Joachim Denzler. "GPU-based volume segmentation". A: (2005).

[10] Erik Smistad et al. "Medical image segmentation on GPUs - A comprehensive review". A: *Medical Image Analysis* 20 (febr. de 2015), pàg. 1 - 18. DOI: 10.1016/j.media.2014.10.012.

[11] A. Sherbondy, M. Houston i S. Napel. "Fast volume segmentation with simultaneous visualization using programmable graphics hardware". A: *IEEE Visualization, 2003. VIS 2003.* 2003, pàg. 171 - 176.

[12] Unity Technologies Unity Software Inc. *Unity - Manual*. URL: https://docs.unity3d.com/540/Documentation/Manual/index.html (cons. 2016).

[13] Joey de Vries. *Learn OpenGL - Framebuffers*. 15 de gen. de 2021. URL: https://learnopengl.com/Advanced-OpenGL/Framebuffers.

[14] Braynzar Soft. *Render to texture*. URL: https://www.braynzarsoft.net/viewtutorial/q16390-35-render-to-texture (cons. 2 de nov. de 2015).

[15] Aassif Benassarou et al. "MC Slicing for Volume Rendering Applications". A: vol. 3515. Abr. de 2005, pàg. 47 - 84. ISBN: 978-3-540-26043-1. DOI: 10.1007/11428848_39.

[16] Microsoft. *Compute Shader Overview - Win32 apps | Microsoft Docs*. URL: https://docs.microsoft.com/en-us/windows/win32/direct3d11/direct3d-11-advanced-stages-compute-shader (cons. 31 de maig de 2018).

[17] Khronos. *Compute Shader - OpenGL Wiki*. 22 d'abr. de 2019. URL: https://www.khronos.org/opengl/wiki/Compute_Shader.

[18] P. Perona i J. Malik. "Scale-space and edge detection using anisotropic diffusion". A: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12.7 (1990), pàg. 629 - 639. DOI: 10.1109/34.56205.

[19] Software Freedom Conservancy. *Git –fast-version-control*. URL: https://git-scm.com/ (cons. 27 de set. de 2020).

[20] Altassian. *Trello*. URL: https://trello.com/ (cons. 27 de set. de 2020).

[21] Google. *Google Meet*. URL: https://gsuite.google.com/products/meet/ (cons. 27 de set. de 2020).

[22] Inc. Slack Technologies. *Slack*. URL: https://slack.com (cons. 27 de set. de 2020).

[23] NVIDIA Corporation. *NVIDIA Developer Zone: Cg Language Specification*. URL: https://developer.download.nvidia.com/cg/index_language.html (cons. 4 d'oct. de 2020).

[24] Overleaf. *Online Latex Editor Overleaf*. URL: https://www.overleaf.com/ (cons. 5 d'oct. de 2020).

[25] s.r.o. BarD Software. *GanttProject: Free desktop project scheduling app*. URL: https://www.ganttproject.biz/ (cons. 2 d'oct. de 2020).

[26] Federación de Servicios CCOO. "Sondeo a la población del sector TIC". A: (12 de febr. de 2020).

[27] Sr. Días Laborables. *Días laborables*. URL: https://www.dias-laborables.es/ (cons. 11 d'oct. de 2020).

[28] Logical Increments. *Logical Increments*. URL: https://www.logicalincrements.com/ (cons. 10 d'oct. de 2020).

[29] Redacció BBC. "La ofensiva en EE.UU. para desmembrar las grandes empresas tecnológicas acusadas de monopolio". A: (7 d'oct. de 2020). Ed. de BBC News. URL: https://www.bbc.com/mundo/noticias-54458049.

[30] Gemma Galdon Clavell. "¿Qué hacen con nuestros datos en internet?" A: (12 de juny de 2015). Ed. d'El País. URL: https://elpais.com/tecnologia/2015/06/12/actualidad/1434103095_932305.html.

Apèndix

A Diagrama de Gantt

Per les tasques relacionades amb la gestió del projecte s'utilitza el color blau, per les fites (proves de correctesa) i les reunions el color rosa, per les tasques relacionades amb la implementació dels algorismes el color vermell, i per les tasques relacionades amb implementar eines d'interacció s'utilitza el color verd. El color groc s'ha reservat pel període que encara queda per determinar s'hi s'aprofundirà amb segmentació o interacció.

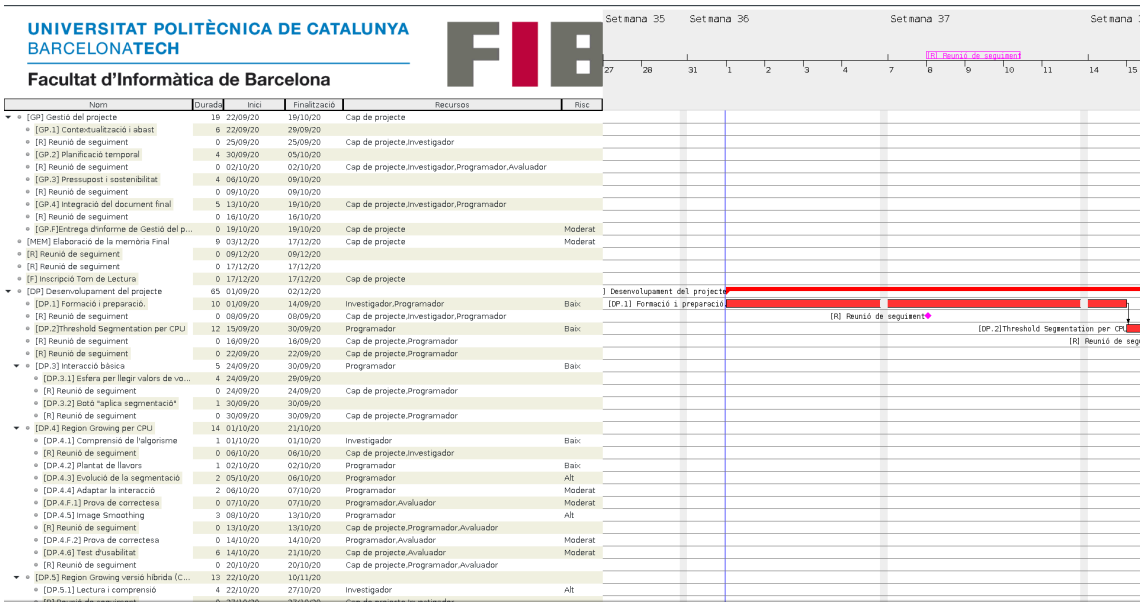


Figura 49: Diagrama de Gantt, part 1.

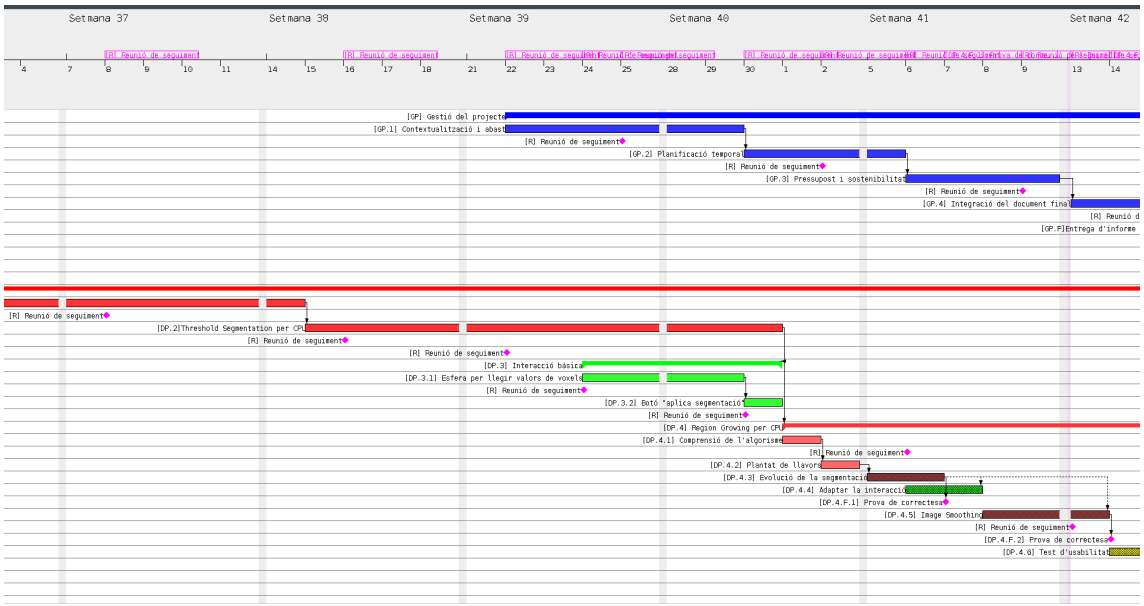


Figura 50: Diagrama de Gantt, part 2.

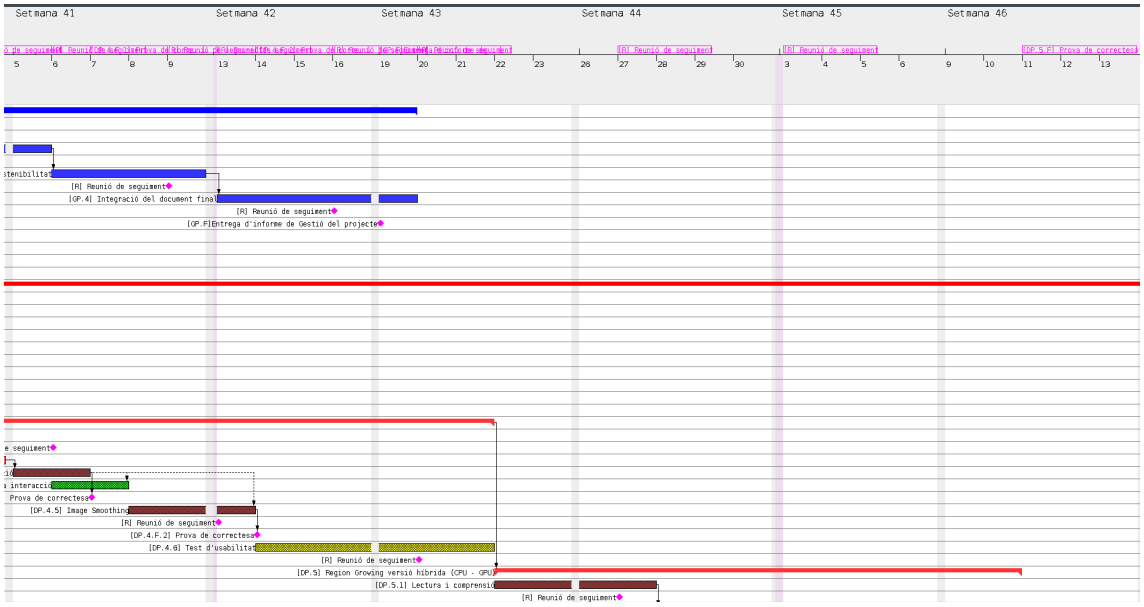


Figura 51: Diagrama de Gantt, part 3.

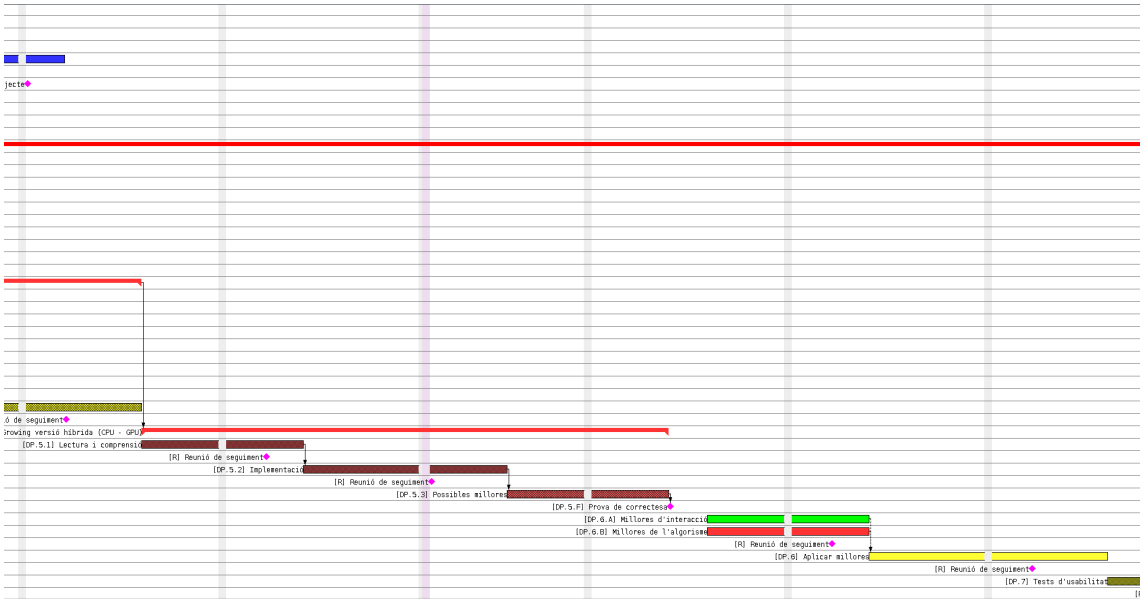


Figura 52: Diagrama de Gantt, part 4.

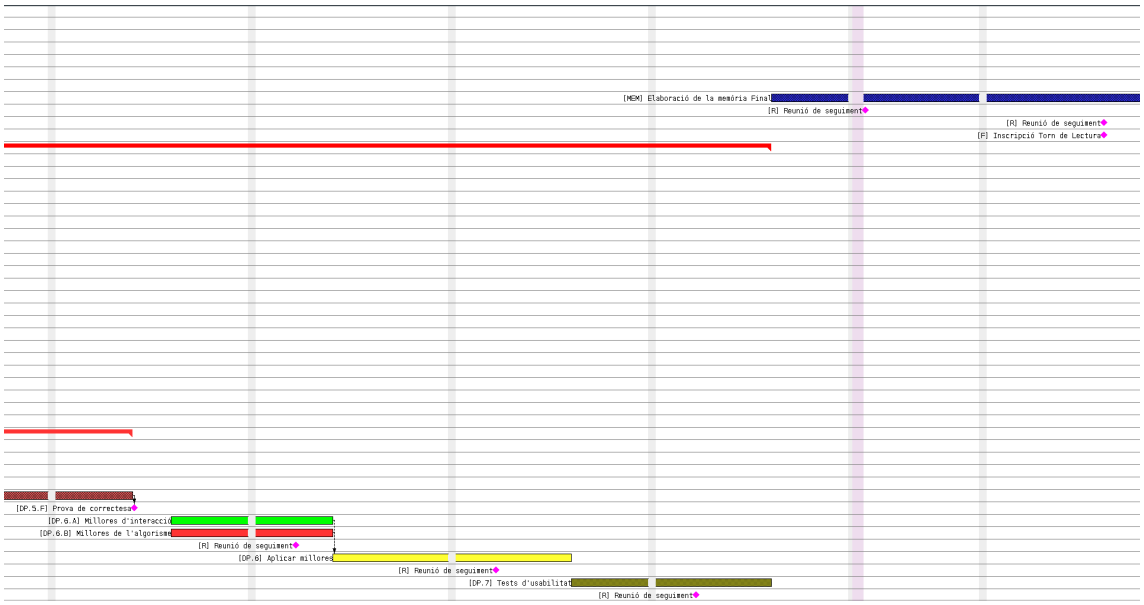


Figura 53: Diagrama de Gantt, part 5.