Títol: Disseny i implementació d'una aplicació de visualització de dades mèdiques per a un dispositiu Windows Phone

Alumne: Eric Antonio Venti Crespo

Director: Pere Pau Vázquez Alcocer

Departament del director: Llenguatges i Sistemes Informàtics

Data: Barcelona, 25 d'octubre de 2013

Centre: Facultat d'Informàtica de Barcelona (FIB)

Universitat: Universitat Politècnica de Catalunya (UPC) BarcelonaTech

Índex

Índex de Figures

1 Introducció

1.1 Motivació del projecte

A les últimes dècades s'han fet grans avenços en el camp de la medicina, sobretot amb la fabricació de material informàtic cada cop més potent. Aquests avenços han permès que es puguin visualitzar dades mèdiques de forma cada vegada més realista i precisa.

Amb el descobriment dels rajos X a finals del segle XIX pel científic alemà Wilhem Conrad Röntgen, es va obrir la porta a la possibilitat de poder veure el que passava dintre del nostre organisme sense la necessitat de cirurgia. Primerament van ser les imatges sobre plaques fotogràfiques en dues dimensions les que van permetre que ràpidament es fessin grans avenços sobre el diagnòstic de dolències al cos humà a la primera meitat del segle XX. No va ser fins la dècada dels 70 quan es van construir els primers aparells per a aconseguir tomografies axials computeritzades i imatges de ressonància magnètica, que són moltes imatges en dues dimensions que permeten reconstruir un model en tres dimensions per a saber què passa a l'interior del cos humà. A partir de la dècada dels 80, gràcies als grans avenços al camp de la informàtica, es van anar apropant cada cop més els camps de la informàtica i de la medicina, ja que els equips informàtics permetien processar aquestes imatges i aconseguir un diagnòstic molt més acurat. A causa del gran abaratiment i miniaturització de l'equip informàtic que s'ha esdevingut en aquests últims anys, i gràcies a l'aparició de telèfons mòbils cada cop més petits i potents a partir del segle XXI, ara és possible visualitzar aquests models de tres dimensions fins i tot amb un *smartphone* (mòbil intel·ligent) amb la gran comoditat que això suposa.

Així doncs, aquest projecte es centrarà en el disseny, implementació i prova d'una aplicació per a dispositius mòbils capaç de visualitzar dades mèdiques en tres dimensions a partir de tomografies computeritzades obtingudes directament del pacient.

1.2 Objectius

Aquests són els objectius que el projecte ha de complir per a que l'aplicació implementada realitzi correctament la seva funció, visualitzar dades mèdiques.

1.2.1 Descripció dels requeriments de l'aplicació

L'aplicació mòbil desenvolupada haurà de ser capaç de dur a terme les funcionalitats que ha de disposar una aplicació d'aquest tipus, això és, moviment i interacció amb el model de volum, manipulació de funcions de transferència, emmagatzemament d'aquestes i visualització realista. La descripció d'aquestes funcionalitats es veurà més endavant en aquesta memòria.

1.2.2 Rendiment

Donat que l'aplicació serà interactiva, l'usuari haurà de veure els canvis que realitza pràcticament de forma instantània mentre treballa. S'haurà de posar especial èmfasi en la fluïdesa i la rapidesa de resposta de l'aplicació a l'hora de mostrar els canvis realitzats. S'intentarà minimitzar sempre que es pugui els moments de càrrega o treball intens de l'aplicació fent saber a l'usuari que l'aplicació està realitzant la tasca assignada. Per tal de mantenir aquesta fluïdesa, s'incorporarà a l'aplicació una opció per a poder canviar la qualitat de la visualització mentre es manipula el model. D'aquesta forma l'usuari sempre podrà triar si prefereix rapidesa o qualitat mentre es treballa abans d'obtenir les imatges amb il·luminació finals.

1.2.3 Visualització amb il·luminació

Amb l'objectiu de visualitzar les dades mèdiques amb més detall i qualitat, s'implementarà a més a més, un algoritme de *raycasting* que permetrà obtenir una imatge estàtica incorporant il·luminació i especularitat al model. Amb la incorporació d'aquest mètode es disposarà del mètode simple de visualització sense il·luminació mentre es manipula el model i el complex i

amb il·luminació quan es vulgui obtenir una imatge estàtica per tenir una millor qualitat i detall a la visualització final del model.

1.3 Breu història dels *smartphones*

En aquesta introducció es veurà com han evolucionat els *smartphones* des de la seva aparició. Es centrarà només en els *smartphones* ja que els telèfons mòbils anteriors no tenien la potència ni la pantalla adequada per a visualitzar dades mèdiques.

S'anomena *smartphone* a un telèfon mòbil que permet dur a terme accions pròpies d'una PDA (Personal Digital Assistant), més enllà de trucades de veu i SMS. La potència de càlcul d'un *smartphone* és comparable a la d'un ordinador de sobretaula o portàtil de fa uns anys, a més són capaços d'executar un sistema operatiu mòbil amb les seves pròpies aplicacions.

1.3.1 Sistemes Operatius per a mòbils

Gairebé sempre a l'historia dels telèfons mòbils el sistema operatiu ha estat propietari (desenvolupat per la mateixa marca que fabricava el telèfon mòbil). Aquí hi ha una petita descripció de l'historia de cadascun dels sistemes operatius més importants que s'han incorporat als *smartphones.*

- **Symbian**

  És un sistema operatiu que va ser producte de l'aliança de varies empreses de telefonia mòbil, entre les quals es troben Nokia, Sony, Ericsson, Samsung, Siemens, Benq, Fujitsu, Lenovo, LG, Motorola, Mitsubishi Electric, Panasonic, Sharp, etc. La primera versió de Symbian destinada a telèfons mòbils data de l'any 2001. De totes formes la versió Symbian per a *smartphones* data de l'any 2006 amb el Nokia 3250. Aquest sistema operatiu va ser el més utilitzat fins que els sistemes operatius Android i iOS van entrar a mercat.

Figura 1.1: Nokia 3250, primer model a incorporar Symbian per a smartphones

- **Android**

  És un sistema operatiu mòbil basat en Linux, que juntament amb aplicacions middleware (software que assisteix a una aplicació per a interactuar o comunicar-se amb altres aplicacions) està orientat per a ser utilitzat a dispositius mòbils com *smartphones*, tablets, Google TV i altres dispositius. Inicialment va ser desenvolupat per Android Inc, però aquesta empresa va ser comprada per Google al 2005. El sistema operatiu Android és posat al mercat al mercat a novembre del 2008, i el HTC Dream es converteix en el primer dispositiu mòbil en dur-lo incorporat. Des de la seva sortida a mercat, Android no ha parat de créixer i, juntament amb iOS, és un dels sistemes operatius mòbils que més s'utilitzen actualment.

Figura 1.2: HTC Dream, primer model a incorporar sistema operatiu Android

- **iOS**

  És un sistema operatiu mòbil d'Apple. Va ser originàriament desenvolupat per a l'*smartphone* iPhone, tot i que després va ser utilitzat en dispotius tals com l'iPod Touch, iPad i Apple TV. Apple, Inc. no permet la instal·lació de iOS a *hardware* que no sigui propietari, és a dir, en telèfons que no siguin Apple. iOs deriva de Mac OS X, que a la vegada esta basat en Darwin BSD (plataforma de codi obert), que és un sistema operatiu Unix. Apple va revelar la existència del iPhone OS a la Macworld Conference & Expo al gener de 2007, tot i que el sistema no va tenir un nom oficial fins que va sortir la primera beta del SDK un any més tard, quan es va decidir canviar-li el nom de iPhone OS a iOS. El llançament de l'iPhone OS va ser al juny de 2007. Des de llavors la quantitat d' iPhone al mercat no ha deixat de créixer, i juntament amb Android és un dels sistemes operatius mòbils que més s'utilitzen actualment.

Figura 1.3: Iphone, primera versió del dispositiu d'Apple

- **BlackBerry OS**

  És un sistema operatiu mòbil desenvolupat per RIM (Research In Motion) per als seus dispositius BlackBerry. El sistema permet multitasca i té suport per a diferents mètodes d'entrada adoptats per RIM per al seu ús en ordinadors portàtils, particularment la trackwheel, trackball, touchpad i pantalles tàctils. El primer dispositiu BlackBerry, el 850, amb sistema operatiu BlackBerry OS, va ser presentat l'any 1999 a Alemanya com a un buscapersones de doble sentit (amb el qual també es podia contestar). Aquest sistema operatiu va evolucionar fins que al 2003 va ser presentada la primera BlackBerry com a *smartphone*. La peculiaritat de les BlackBerry és que pràcticament la majoria han mantingut sempre

la característica de disposar d'un teclat QWERTY. El fet que Android i iOS hagin guanyat tant mercat juntament amb una gran fallada de la xarxa mòbil dels dispositius BlackBerry al 2011 han ocasionat que BlackBerry OS hagi perdut molt mercat en aquestes ultimes dates. L'1 de maig de 2012 es va presentar BlackBerry 10, el sistema operatiu més recent per a dispositius BlackBerry. De fet, encara que van anunciar el BB10 també per a *tablets*, al final l'han descartat, de forma que el sistema operatiu per a *tablets* s'anomena BB OS 2.1.

Figura 1.4: Blackberry pager 850, primer model en incorporar Blackberry OS

- **Windows Mobile**

  És un sistema operatiu mòbil compacte desenvolupat per Microsoft dissenyat per al seu ús a *smartphones* i altres dispositius mòbils. Es basa en el nucli del sistema operatiu Windows CE (Embedded Compact), i originàriament va aparèixer sota el nom de Pocket PC, com a una ramificació del desenvolupament de Windows CE per a equips mòbils amb capacitats limitades. La seva primera versió com a Pocket PC data de l'any 2002, canviant després al 2003 el seu nom a Windows Mobile.

- **Windows Phone**

  És un sistema operatiu mòbil desenvolupat per Microsoft, com a successor de la plataforma Windows Mobile. Esta pensat per al mercat de consum generalista en lloc del mercat empresarial, pel qual manca de moltes funcionalitats que proporciona la versió anterior. Microsoft ha decidit no fer compatible Windows Phone amb Windows Mobile, ocasionant que les aplicacions de Windows Mobile no funcionen a Windows Phone i s'hagin de desenvolupar de noves. La versió final de Windows Phone 7 va sortir al mercat al setembre de 2010. Aquest

sistema operatiu no ha aconseguit convèncer als usuaris de telèfons mòbils i no ha aconseguit situar-se entre els més utilitzats.

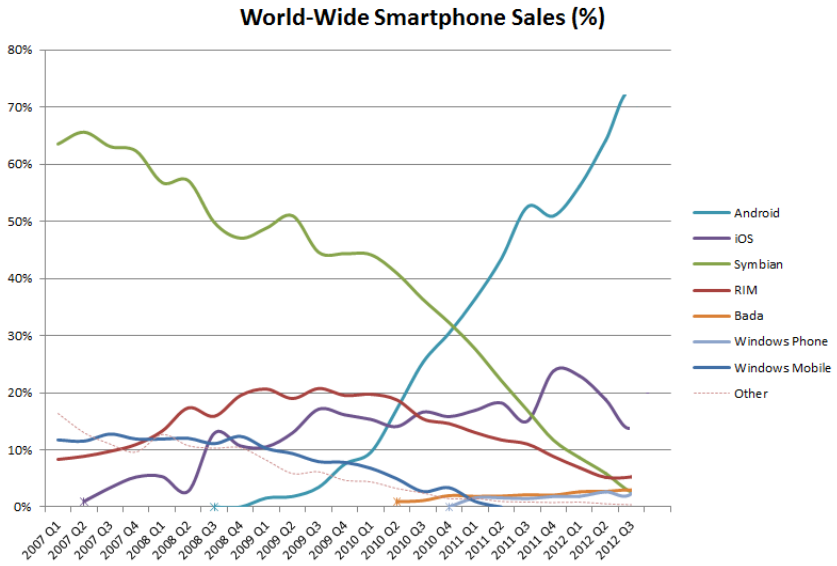


Figura 1.5: Gràfic amb l'evolució de ventes de telèfons segons el seu sistema operatiu (http://blog.gigavoice.com/a-brief-history-of-smartphones/)

1.3.2 Hardware

A l'hora de poder fabricar aquests *smartphones*, el *hardware* ha tingut moltíssima rellevància. El fet de disposar de components electrònics cada vegada més petits, més eficients i més potents ha permès que els *smartphones* siguin capaços de realitzar multitud de diferents tasques.

El *hardware* dels *smartphones* es basa en el SoC (System-on-a-Chip o sistema dintre d'un xip), que permet incorporar dintre d'un mateix circuit integrat o xip diversos processadors on cadascun d'aquests es dedica a una tasca específica. Aquests processadors poden dedicar-se a tasques tals com el nucli del sistema, el sistema de gràfics, la memòria RAM, controladors d'interfície USB, tecnologia inalàmbrica, reguladors de voltatge, etc. L'idea del SoC és que tots els components crítics d'un dispositiu es trobin dintre d'una àrea relativament petita.

Inicialment aquests *smartphones* disposaven d'un sol processador a 400Mhz, 128Mb de RAM, 96Mb de capacitat i pantalles de 240x320 píxels de resolució. Aquest *hardware* ha anat evolucionant fins arribar als vuit processadors a 2.3GHz amb 3Gb de memòria RAM, 64Gb de capacitat i pantalles que arriben fins als 1920x1080 píxels de resolució als mòbils més nous d'última generació tals com el Galaxy Note 3.

El fet que aquests dispositius disposin de sistemes operatius més orientats al multimèdia i que els usuaris cada cop demandin més contingut audiovisual i videojocs, ha generat que cada cop sigui necessari disposar de GPU's més potents.

La unitat de processament gràfic o GPU (Graphics Processing Unit) és un coprocessador dedicat al processament de gràfics o operacions de coma flotant. Existeix bàsicament per alleujar la càrrega de treball al processador central durant l'execució de videojocs o aplicacions 3D interactives. D'aquesta forma, mentre gran part de les tasques relacionades amb els gràfics es processen a la GPU, la unitat central de procés (CPU) pot dedicar-se a altre tipus de càlculs.

Els primers *smartphones* no disposaven d'unitat de GPU, ja que les necessitats gràfiques de llavors no necessitaven d'una unitat de procés dedicada. Va ser Apple amb el seu primer model d'iPhone qui va incorporar la primera GPU model PowerVR MBX Lite 3D. Serien des de llavors 2 empreses, Qualcomm amb la seva sèrie Adreno i Nvidia amb la seva sèrie Geforce ULP, les que també proporcionarien xips gràfics per a la gran majoria de *smartphones* que es troben actualment. Cal remarcar que actualment també es troben al mercat dispositius que no incorporen unitat de GPU.

1.3.3 Plataformes de distribució d'aplicacions

Inicialment els telèfons mòbils només disposaven d'aplicacions programades pel fabricant incorporades a la memòria del sistema i no era possible afegir-hi de noves. Aquesta situació va canviar amb l'aparició dels *smartphones*. Va ser llavors quan les empreses propietàries dels sistemes operatius van decidir permetre que empreses externes poguessin programar aplicacions per als sistemes operatius dels dispositius mòbils.

A causa que aquests nous terminals disposaven de connexió a internet, les aplicacions podien arribar als *smartphones* dels usuaris mitjançant plataformes de distribució d'aplicacions mòbils o tendes d'aplicacions mòbils. La primera empresa en crear la seva pròpia plataforma de distribució d'aplicacions va ser Apple. Inicialment les aplicacions per a iPhone s'aconseguien mitjançant la seva tenda de música digital iTunes. No obstant, Apple va implementar i obrir el seu propi portal exclusiu d'aplicacions, l'App Store al juliol de 2007 d'on es podien descarregar aplicacions per al sistema operatiu iOS. Tal va ser l'èxit d'aquest portal que Google va decidir obrir ràpidament la seva plataforma d'aplicacions anomenada Google Play per al sistema operatiu Android a octubre de 2007. Més endavant es van obrir altres portals tals com App World per a Blackberry OS a abril del 2009, Nokia Store per a Symbian al maig de 2009, Windows Phone Store per a Windows Phone a octubre del 2010 i Amazon Appstore al març de 2011 per a Android. De totes formes l'App Store i Google Play són les dos tendes d'aplicacions més importants actualment, amb més de 700.000 aplicacions cadascuna i més de 25 mil milions d'aplicacions descarregades cadascuna en total.

1.4 Adquisició de dades volumètriques

Més endavant en aquesta memòria es veurà què són els models de volum, però en aquesta secció es veuran quins són els mètodes més empleats actualment per a aconseguir aquestes dades de volum necessàries per al projecte.

Els mètodes que més s'utilitzen actualment per a aconseguir dades de volum són dos, les tomografies axials computeritzades (TAC) i les ressonàncies magnètiques (MRI), tot i que també existeixen els PET, que són tomografies per emissió de positrons, a més, les MRI tenen diferents varietats.

1.4.1 Tomografia axial computeritzada o TAC

Aquest mètode utilitza els raigs X per a aconseguir imatges radiogràfiques a seccions progressives de la zona estudiada de l'organisme i imatges tridimensionals dels òrgans o estructures orgàniques internes. Les primeres imatges aconseguides per aquest tipus de mètodes daten de 1971.

Dintre dels seus avantatges es té que el temps d'adquisició de les dades és curt, s'està parlant de l'ordre dels 30 segons als 5 minuts. També té l'avantatge que a les seves imatges resultants es poden apreciar a la vegada tant òssos com teixits tous i vasos sanguinis. Aquest és el mètode que ofereix més qualitat a l'hora de obtenir imatges d'ossos.

Aquest mètode, però, ofereix també desavantatges, tals com que s'irradia sobre el pacient una quantitat petita de rajos X, existint una lleugera possibilitat d'irradiació. Això ocasiona que normalment per precaució les dones embarassades o els nens no puguin obtenir una tomografia a no ser que sigui absolutament necessari. També té el desavantatge que a les imatges resultants és complicat diferenciar entre diferents teixits tous.

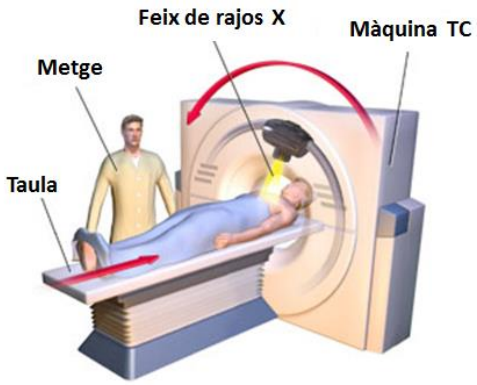


Figura 1.6: Esquema de l'escàner que efectua tomografies axials computeritzades

1.4.2 Imatge de ressonància magnètica o MRI

Aquest mètode utilitza la propietat de la ressonància magnètica nuclear (Magnetic Resonance Imaging), és a dir, la vibració dels àtoms d'aigua a l'interior dels teixits del cos humà mitjançant l'ús de grans imants, per a aconseguir imatges. Les primeres imatges aconseguides amb aquest mètode daten de l'any 1976.

Els avantatges que ofereix una ressonància magnètica són les següents: una ressonància magnètica utilitza camps magnètics en lloc de raigs X, evitant així exposar al pacient a una radiació, a més a més, les imatges resultants de les MRI són molt més detallades a l'hora de mostrar teixits tous, aconseguint major diferenciació entre diferents òrgans interns.

Per contra, aquesta tècnica també té alguns desavantatges. Aconseguir una MRI pren des dels 20 fins als 45 minuts, a més a més, el fet que l'escàner sigui un tub tancat, fa que els pacients amb claustrofòbia no puguin obtenir-ne una. Un altre desavantatge és que les persones amb fragments metàl·lics, monitors cardíacs, marcapassos, o qualsevol element de metall dintre del cos no se'ls hi pot fer una MRI.

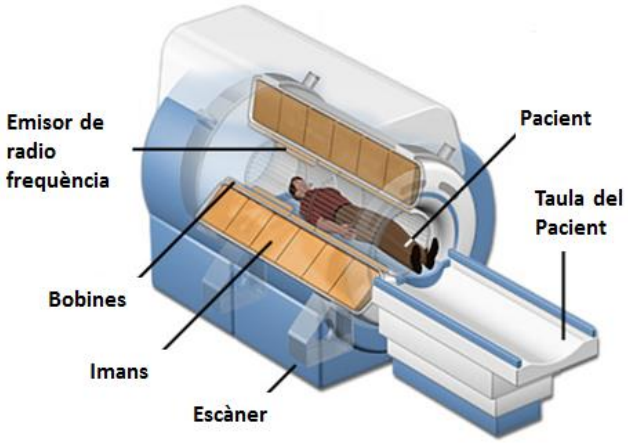


Figura 1.7: Esquema de l'escàner que realitza ressonàncies magnètiques

2 Anàlisi d'antecedents i factibilitat.

2.1 Altres aplicacions mèdiques al mercat per a plataformes mòbils

Abans de començar a dissenyar l'aplicació, s'ha dut a terme una recerca d'aplicacions al mercat actual de dispositius mòbils que realitzin les mateixes o semblants tasques que l'aplicació que s'ha d'implementar. Així doncs, s'ha descobert que al mercat actual de dispositius mòbils, ja existeixen aplicacions com la que es pretén implementar en aquest projecte. El fet que ja hi hagin aplicacions de visualització de dades mèdiques al mercat ajudarà a inspirar-se per a saber de quina forma es poden implementar i millorar algunes funcionalitats que aquestes ja disposen.

2.1.1 Android

- **ResolutionMD Mobile**

Aquesta aplicació permet connectar-se a un servidor remot i descarregar al telèfon dades mèdiques reals de pacients d'Estats Units. Es poden visualitzar cadascuna de les llesques que formen el volum en mode 2D o una reconstrucció en 3D del model. Aquest últim proporciona una alta qualitat de visualització però per contra, l'aplicació es torna molt lenta. La manipulació del model es torna molt lenta i farragosa quan el 3D està activat, deixant de ser doncs una aplicació interactiva. Per a canviar el color de les diferents parts del model que s'està veient a pantalla, disposa d'uns *presets* que ja venen incorporats per a canviar el color segons el tipus de teixit. Aquesta aplicació és gratuïta al Google Play.

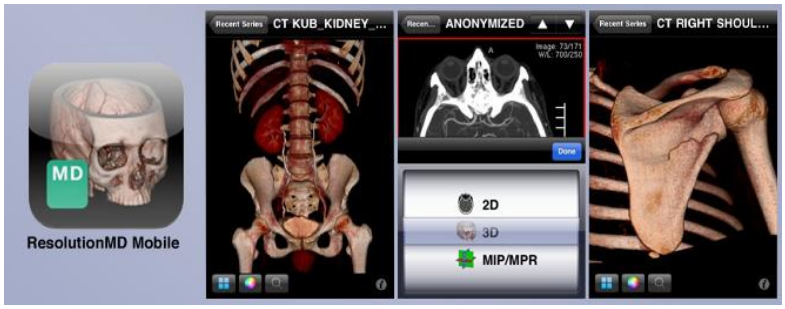


Figura 2.1: Diferents captures de l'aplicació ResolutionMD Mobile (http://www.calgaryscientific.com/resolutionmd/mobile-resmd/)

- **Aplicació de final de màster d'en Marcos Balsa**

En Marcos Balsa Rodríguez, un alumne d'aquesta universitat, va implementar una aplicació de visualització de dades mèdiques l'any 2011 com a projecte de final de màster. Es pot aconseguir la memòria final d'aquest projecte i diferents articles que ha publicat tant al portal de CRS4 Visual Computing (http://www.crs4.it/vic/) com a la web de la Universitat Politècnica de Catalunya (http://www.upc.edu/). Aquesta aplicació inclou moltes de les funcionalitats que es volen implementar en aquest projecte. Inclou la possibilitat de manipular el model i les seves funcions de transferència de forma fàcil i ràpida. El mètode de manipulació de funcions de transferència de l'aplicació a implementar es basarà en el mètode que empra aquesta, si més no, s'intentarà millorar per a poder tenir encara més control sobre aquestes funcions.

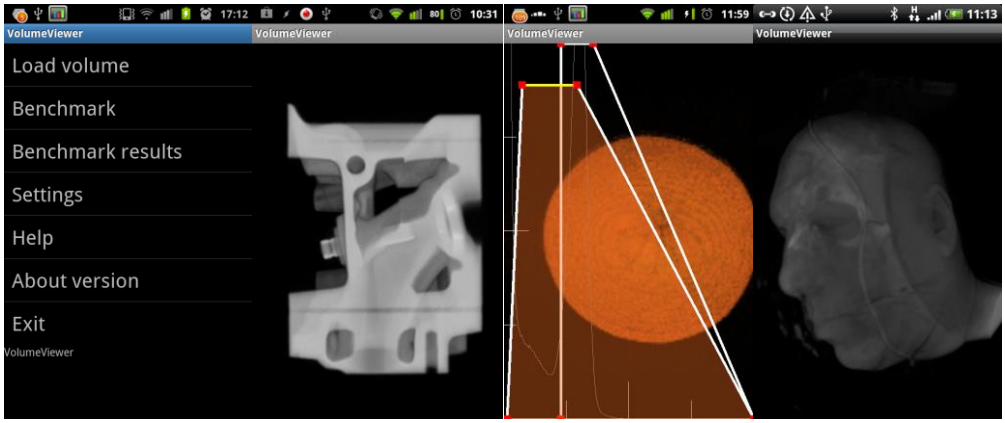


Figura 2.2: Diferents captures de l'aplicació d'en Marcos Balsa Rodríguez

- **Centricity Rad Mobile Access**

Aquesta aplicació és pràcticament igual a ResolutionMD vista anteriorment. Té algunes diferències a la seva interfície gràfica, però en quant a funcionalitats, ofereix les mateixes. Entre d'altres, permet connectar-se a un servidor remot i descarregar al telèfon dades mèdiques reals de pacients d'Estats Units. Es poden visualitzar cadascuna de les llesques que formen el volum en mode 2D o una reconstrucció en 3D del model. Aquest últim proporciona una alta qualitat de visualització però per contra, l'aplicació es torna molt lenta. La manipulació del model es torna molt lenta i farragosa quan el 3D està activat, deixant de ser doncs una aplicació interactiva. Per a canviar el color de les diferents parts del model que s'està veient a pantalla, disposa d'uns *presets* que ja venen incorporats per a canviar el color segons el tipus de teixit. Aquesta aplicació és gratuïta al Google Play.

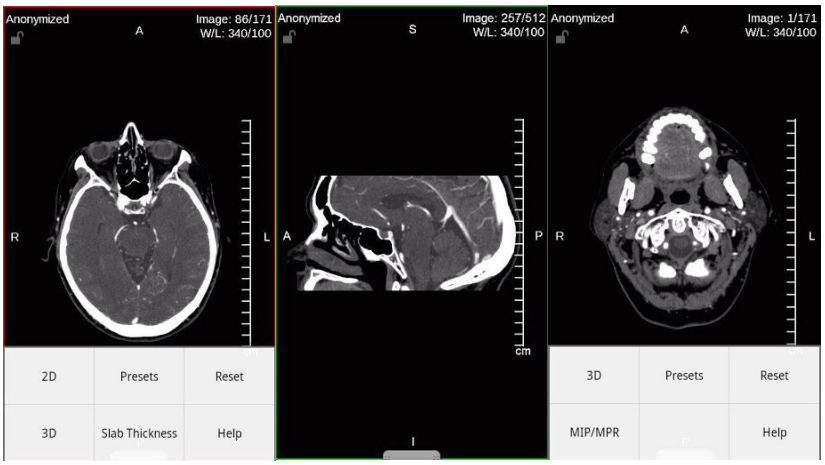


Figura 2.3: Diferents captures de l'aplicació Centricity Rad Mobile Access (https://play.google.com/store/apps/details?id=com.ge.centricitymobile&hl=es)

2.1.2 iOS

- **OsiriX HD**

Aquesta és la versió per a dispositius mòbils amb sistema operatiu iOS del programa amb mateix nom per a Mac. Aquesta aplicació es podria dir que és una extensió mòbil per a l'aplicació de l'ordinador de sobretaula. Permet visualitzar a dispositius mòbils arxius DICOM (Digital Imaging and Communication in Medicine), que és l'estàndard mundialment conegut per a l'intercanvi d'imatges mèdiques. Permet visualitzar cadascuna de les llesques i també permet prendre mesures de les imatges. També es pot realitzar una manipulació bàsica de les imatges utilitzant els gestos típics d'iOS, podent tocar el contrast i la intensitat de les imatges. El preu d'aquesta aplicació a l'App Store és de 29.99$.

Figura 2.4: Diferents captures de l'aplicació Osirix HD (https://itunes.apple.com/app/id419227089?mt=8)

- **ImageVis3d Mobile**

Aquesta és la versió per a dispositius mòbils de ImageVis3d, aplicació per a PC. Aquesta és una bona aplicació ja que permet manipular el model amb gran fluïdesa i de forma fàcil i intuïtiva. Destaca per la seva facilitat a l'hora de manipular el model a l'espai 3d, els moviments són suaus i utilitza un mode de menor resolució per a fer fluida la visualització mentre es manipula el model a la pantalla. Disposa d'una bona quantitat de models per a visualitzar i permet incorporar models de volum externs. També permet canviar-li la qualitat que es veurà el model.

La seva part negativa, però, és la manera en que s'editen les funcions de transferència. Aquesta manipulació pot servir per a buscar aproximacions de funcions de transferència de forma general, però deixa molt que desitjar quan el que es necessita es precisió sobre aquestes per a poder visualitzar zones molt concretes i petits detalls del model de volum.

Aquesta aplicació és gratuïta i es pot descarregar des de l'App Store.

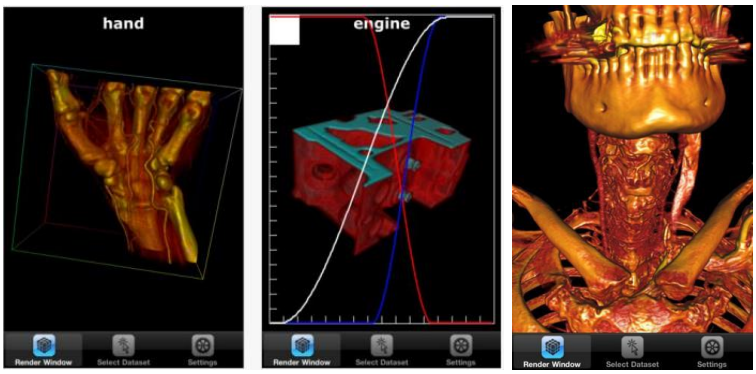


Figura 2.5: Diferents captures de l'aplicació ImageVis3d Mobile (https://itunes.apple.com/us/app/imagevis3d-mobile-universal/id378071694?mt=8)

2.2 Per què s'escull Windows Phone

Així doncs, si ja hi ha aplicacions al mercat, per què fer-ne una altra per a Windows Phone? Hi ha diversos motius pels quals es desitja implementar una nova aplicació per a aquest sistema operatiu:

2.2.1 Única aplicació disponible de visualització 3D de dades mèdiques

S'ha fet una recerca intensiva sobre l'oferta d'aplicacions de visualització de dades mèdiques a la plataforma Windows Phone i tot i que s'han trobat aplicacions capaces de visualitzar imatges 2D aconseguides mitjançant un escàner, fins al moment no hi ha cap aplicació disponible que sigui capaç de mostrar un model de volum en 3 dimensions i poder modificar les corbes de les funcions de transferència. Això suposa una gran motivació ja que aquesta serà la primera aplicació per a Windows Phone que permeti visualitzar i manipular dades mèdiques en 3 dimensions.

2.2.2 Poc temps al mercat

El fet que Windows Phone dugui relativament poc temps al mercat ocasiona que s'hagi d'investigar com s'hauran d'implementar les tècniques que s'hauran de fer servir tenint en compte les limitacions que té aquest sistema operatiu. Windows Phone és terreny verge sobre el qual es poden implementar tot tipus de noves aplicacions, amb la motivació que comporta.

2.2.3 Facilitat d'implementació

El llenguatge de programació que utilitza Windows Phone es C#, un llenguatge orientat a objectes que fa molt fàcil la programació d'aplicacions. A més, Windows ofereix la plataforma Visual Studio 2010 de forma gratuïta per a que els programadors puguin programar aplicacions de forma més fàcil. De totes formes, a la web de desenvolupadors d'aplicacions hi ha molts exemples i tutorials de com programar per a Windows Phone, fent que el fet de començar a implementar aplicacions sigui molt fàcil i ràpid.

2.2.4 Facilitat de distribució

Ja s'ha vist anteriorment que Windows Phone disposa igual que els principals sistemes operatius, de la seva pròpia plataforma d'aplicacions. D'aquesta manera és molt fàcil el pas de posar l'aplicació al mercat amb l'objectiu d'arribar a la major quantitat possible d'usuaris que desitgin adquirir-la.

Windows Phone, igualment que Apple, té una política per a garantir la qualitat de les aplicacions que es troben a la tenda. D'aquesta manera, l'aplicació que es vulgui publicar a la tenda haurà de complir una sèrie de requisits i aquesta serà examinada pels tècnics de Microsoft per a verificar que es compleixen. Una vegada es compleixen aquests requisits i l'aplicació ha estat verificada pel personal de Microsoft, l'aplicació es publica a la plataforma de distribució d'aplicacions per a que els usuaris que es connectin puguin descarregar-la.

3 Desenvolupament tècnic (especificacions, disseny, etc.), resultats experimentals.

3.1 Conceptes teòrics necessaris

En aquesta secció s'explicaran els conceptes teòrics de les parts més importants de l'aplicació i que són necessaris per a la comprensió del projecte.

3.1.1 Vòxels i models de volum

Amb l'objectiu de poder visualitzar el que passa dintre del cos humà, es necessitarà una representació d'aquest volum en tres dimensions. Aquesta representació se li anomena model de volum, ja que conté un conjunt de valors discrets que són interpretats com a valors de densitat de l'espai. Aquests models no s'obtenen només mitjançant tomografies computeritzades o escàners de ressonància, també poden ser resultats sintètics de simulacions aconseguides per software. El volum que ocupa el model és discretitzat en petits cubs anomenats vòxels, que són la unitat mínima de volum que un model pot tenir. La quantitat de vòxels que disposi un model de volum definirà la seva resolució. A vòxels més petits, caldran més vòxels per a representar el mateix volum i es tindrà més resolució. Donat que aquests volums són en tres dimensions, cadascun tindrà la seva resolució d'amplada, alçada i profunditat. Cadascun d'aquests vòxels representa un valor discret de densitat d'aquell punt de l'espai.

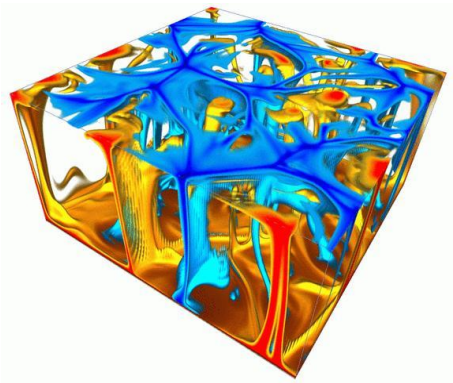


Figura 3.1: Model de volum d'una simulació de corrents de convecció de fluids de diferent densitat

3.1.2 Funcions de transferència

Una vegada es disposa del model de volum, aquest model s'ha de poder visualitzar, i per a aconseguir-ho es necessiten les anomenades funcions de transferència i les corbes que tenen cadascuna d'aquestes. En aquesta aplicació només es disposarà d'una única funció de transferència. El model de volum s'ha vist anteriorment que són un conjunt de valors de densitat. Una corba d'una funció de transferència fa correspondre valors de densitat dels vòxels amb un color i una opacitat determinada per a que la densitat dels mateixos pugui ser visualitzada.

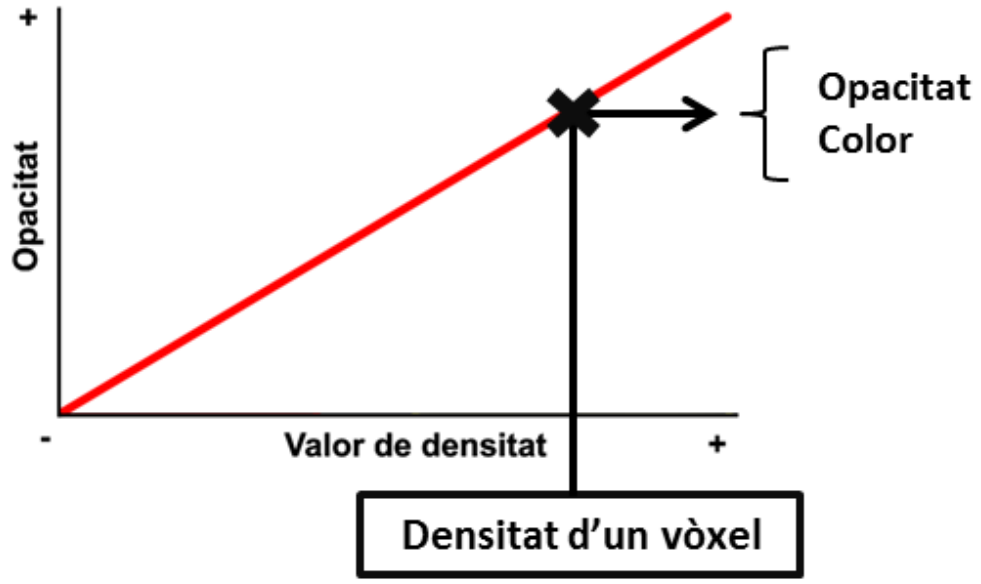


Figura 3.2: Correspondència entre densitat i opacitat i color

Aquesta funció de transferència tindrà al seu eix horitzontal el valor de densitat d'un vòxel i al seu eix vertical tindrà quina és la opacitat per a aquell valor de densitat. D'aquesta forma els vòxels més densos estaran situats sobre la zona dreta de l'eix horitzontal, mentre que els vòxels menys densos estaran situats a la part esquerra de l'eix horitzontal. Aquesta distribució pot variar si el model té com a densitat màxima un valor de teixit tou, per exemple. D'aquesta forma la visualització es centrarà només en un determinat rang de densitats que pot no trobar-se dintre del rang que mostra teixits tous i ossos a la mateixa vegada.

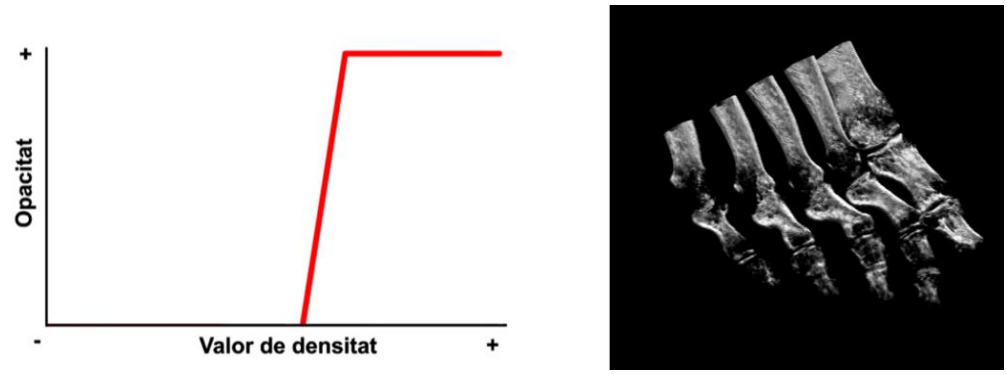


Figura 3.3: Visualització amb més opacitat dels elements amb més densitat del model d'un peu

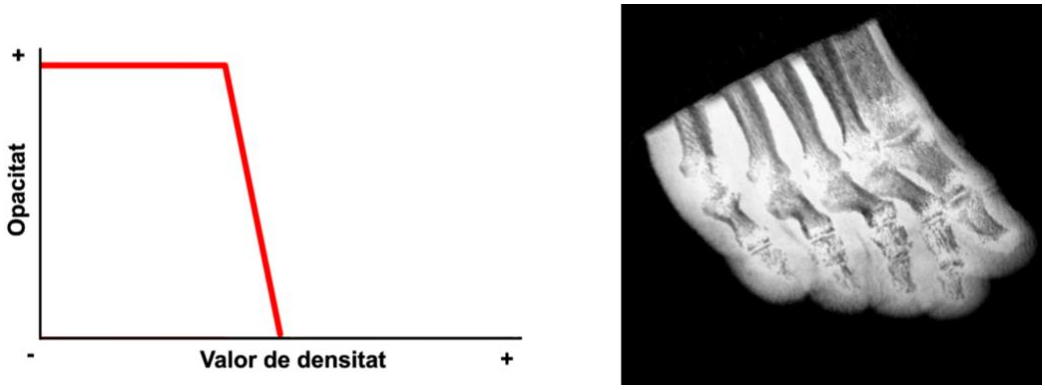


Figura 3.4: Visualització amb més opacitat dels elements amb menys densitat del model d'un peu

Una funció de transferència pot disposar de més d'una corba de mapat, però mai de menys d'una. Cadascuna de les corbes de la funció de transferència disposarà del seu propi color, de forma que es podran tintar de diferent color les diferents zones de densitat. Les corbes de les funcions de transferència estan fetes per a que per a un valor de densitat li correspongui únicament un valor d'opacitat. D'aquesta forma tots els possibles valors de densitat dintre d'un rang tindran correspondència amb un únic valor d'opacitat. En cas que es disposi de més d'una corba dintre de la funció de transferència, el color i opacitat per a les zones que hi hagi solapament entre corbes serà la suma resultant tant de colors com d'opacitats.

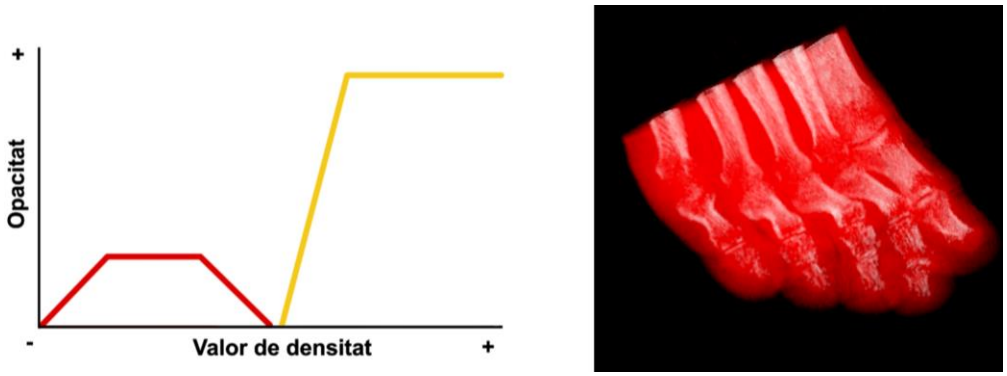


Figura 3.5: Utilització de dos corbes de mapat diferents per a colorar diferents densitats

3.1.3 Equació de visualització de volum

La visualització de models de volum és una simulació aproximada de la propagació de la llum a traves d'un medi representat pel volum. Simular mitjançant una equació el comportament de la llum implica que s'hagin d'aplicar models físics de radiació amb l'objectiu d'intentar aproximar el màxim possible a la realitat el resultat. D'igual manera també es tindrà en compte que la llum viatja en línia recta d'un punt a un altre a no ser que tingui interacció amb el medi pel qual es mou. A la visualització de volum el medi és participatiu, ja que té efectes sobre el comportament de la llum que viatja a través d'ell. En aquesta aproximació teòrica no es tindran en compte altres propietats físiques de la llum tals com la difracció, les interferències, les longituds d'ona o la polarització dels rajos. Així doncs les propietats físiques quan un raig de llum travessa un medi són les següents:

- Emissió o *emission*

  Ocorre quan al raig de llum se li afegeix o crea certa quantitat de llum.

- Absorció o *absortion*

  Ocorre quan el raig de llum es va atenuant o extingint a causa de la pèrdua d'energia gastada en travessar el medi.

- Dispersió cap enfora o *outscattering*

  Ocorre quan part de la llum d'un raig es veu dispersada cap a altres direccions dintre del medi. Cal dir que la radiància (quantitat d'energia d'un raig de llum) no disminueix, només canvia de direcció.

- Dispersió cap endins o *inscattering*

  Ocorre quan a un raig de llum se li afegeix la dispersió cap enfora ocasionada per altres rajos de llum propers.

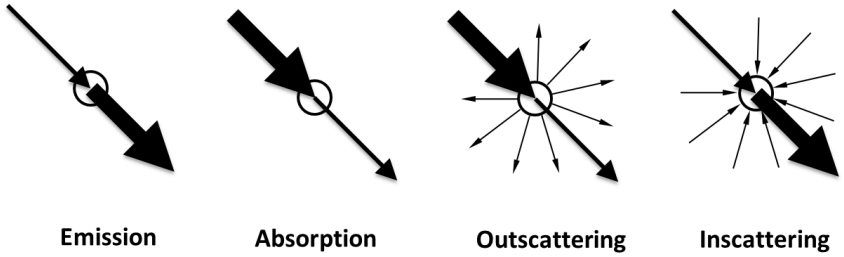


Figura 3.6: Representació gràfica de les propietats físiques de la llum mentre travessa un medi

A l'hora de visualitzar un volum, el model teòric de visualització més utilitzat és el de Emission-Absortion, que només té en compte les propietats d'emissió i absorció. Aquest és el mètode més utilitzat ja que és fàcil d'implementar i proporciona una qualitat i rendiment raonables. Cal dir que hi han models més complexes tals com *simple scattering and shadows* o *multiscattering* i menys complexes tals com només absorció o només emissió.

A continuació es presenta l'equació contínua de visualització de volum que representa el model Emission-Absortion. Amb aquesta equació es calcula la intensitat o quantitat d'energia, també anomenada radiància, en forma de llum al punt final s des del punt inicial de sortida s_0 havent passat a traves d'un medi participatiu.

$$I(s) = I(s_0)\, e^{-\tau(s_0,s)} + \int_{s_0}^{s} q(s')\, e^{-\tau(s',s)} ds'$$

Figura 3.7: Equació contínua de visualització de volum on q representa emissió i I radiància

Dintre de l'equació 3.8 el terme τ representa l'absorció o extinció a causa de la pèrdua d'energia que pateix la llum a mesura que viatja a través del medi entre els punts s_1 i s_2. K representa la quantitat d'absorció d'energia del medi.

$$\tau(s_1, s_2) = \int_{s_1}^{s_2} \mathrm{K}(s') ds'$$

Figura 3.8: Integral d'absorció de llum entre els punts s1 i s2

Un cop arribat fins aquí, per a poder calcular-la, es discretitzarà l'equació agafant petits passos equidistants s_k.

$$I(s_k) = I(s_{k-1}) e^{-\tau(s_{k-1},s_k)} + \int_{s_{k-1}}^{s_k} q(s) e^{-\tau(s,s_k)} ds$$

Figura 3.9: Discretització de l'equació de visualització de volum

D'aquesta forma es tindran dos components a l'equació, b_k que és la component d'emissió i θ_k que és la component de transparència, absorció o opacitat.

$$b_k = \int_{s_{k-1}}^{s_k} q(s) e^{-\tau(s,s_k)} ds \quad \theta_k = e^{-\tau(s_{k-1},s_k)}$$

Figura 3.10: Components a l'equació de visualització de volum

Integrant mitjançant la suma de Riemann s'aconsegueix finalment la següent equació, que aproxima la radiància total mitjançant les sumes dels intervals amb emissió i absorció entre el punt inicial i final, on b representa color i α representa opacitat.

$$I(s_n) = I(s_{n-1})\theta_n + b_n = \sum_{k=0}^{n}\left(b_k \prod_{j=k+1}^{n}\theta_j\right)$$
$$= I(s_{n-1})(1-\alpha_n) + b_n$$

Figura 3.11: Equació de visualització de volum aproximada mitjançant sumes

Una vegada aconseguida aquesta equació, s'ha de procedir amb el *compositing* que és la computació iterativa d'aquesta suma. Hi han dos formes de fer-la, de darrere cap endavant o *back-to-front* i de davant cap endarrere o *front-to-back*. En aquests dos mètodes el color i l'opacitat s'assignaran mitjançant les corbes de la funció de transferència vistes anteriorment.

L'equació del mètode *back-to-front* que està directament extreta de la integral discretitzada però canviant l'emissió per color i absorció per opacitat, té la forma següent: $C^{out} = C^{in}(1-\alpha) + C$, on α és la opacitat i C el color del punt travessat.

Al mètode de *front-to-back* s'han d'anar acumulant les opacitats dels punts travessats, de forma que es tindran el color i la opacitat per separat. Les equacions resultants doncs són les següents: $C^{out} = C^{in}(1-\alpha^{in}) + C$, $\alpha^{out} = \alpha^{in} + (1-\alpha^{in})\alpha$, on α és l'opacitat i C el color del punt travessat. S'ha de tenir en compte que el α^{in} de cada iteració serà el α^{out} de la iteració anterior. Aquest mètode serà el que s'implementarà amb el *raycasting* que es veurà en profunditat més endavant en aquesta memòria.

3.2 Descripció de tècniques per a implementar un sistema de visualització de models de volum

El model de volum serà el punt de partida, ja que s'implementaran dos mètodes de visualització d'aquests vòxels, l'*slicing* i el *raycasting*, al qual se li incorporarà un model d'il·luminació per a poder veure els models il·luminats. S'ha de mencionar que aquestes tècniques per a visualitzar models de volum s'anomenen tècniques de *direct volume rendering* o de visualització directa de volum, ja que les imatges es generen directament des del model de volum sense haver de passar per la extracció i generació d'una geometria intermèdia.

Qualsevol dels mètodes de visualització que s'explicaran a continuació utilitzen les corbes de la funció de transferència abans explicades per a decidir amb quin color i amb quina opacitat es visualitzen els vòxels del volum que es troben a pantalla. Així doncs, quan s'hagin de visualitzar aquests vòxels, s'hauran de consultar les corbes de mapat de la funció de transferència per a saber amb quina opacitat i quin color s'han de visualitzar els vòxels al dispositiu.

Figura 3.12: Diversos exemples de direct volume rendering

3.2.1 Slicing

Aquesta és una tècnica bàsica de visualitzar un model de volum. És un mètode que, tot i no ser el més avançat o punter, dóna uns resultats força bons. A causa d'això, no és necessari que les targetes gràfiques disposin de mètodes per a manipular textures de tres dimensions mitjançant *hardware,* tot i que també es pot dur a terme si la targeta disposa d'aquesta funcionalitat. Cal dir que aquest mètode es segueix utilitzant actualment als nostres dies a causa de les seves característiques que tot seguit s'expliquen.

Aquesta tècnica consisteix en emmagatzemar tres conjunts de dades de volum i utilitzar-los com tres conjunts perpendiculars de llesques alineades amb el sistema de coordenades local del model. Aquestes llesques són extretes del volum ortogonalment segons els tres eixos (x, y i z) i la informació de cadascuna d'elles es guarda en forma de textura 2d (imatge) de forma que cadascuna d'aquests tres conjunts d'imatges es maparan sobre un conjunt de polígons alineats amb els tres eixos. La visualització es realitza situant els polígons alineats amb els eixos i visualitzant-los des de darrere cap endavant a causa que els últims polígons dibuixats sempre queden per sobre els que ja s'havien pintat prèviament. Depèn de com sigui la composició, les llesques del darrere podrien tapar parts de llesques del davant ocasionant una visualització errònia.

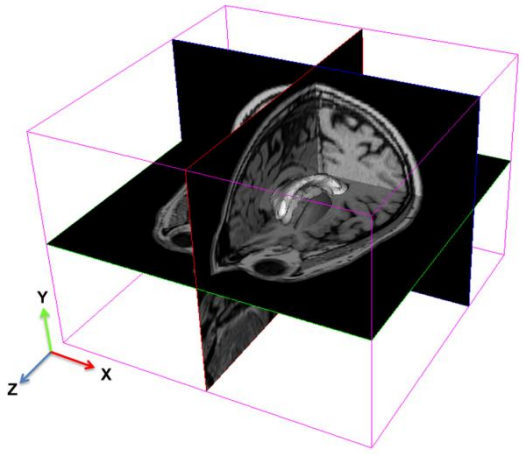


Figura 3.13: Representació de la orientació de les llesques

En tot moment mentre es manipuli el volum, es visualitzarà només el conjunt de llesques que estigui més perpendicular a la direcció de visió de l'usuari. Cada vegada que es faci un canvi a les corbes de la funció de transferència,

s'han de tornar a recórrer tots els vòxels del model de volum per a tornar a generar els conjunts d'imatges que hauran de ser mapades sobre els polígons. Així doncs la gran càrrega de procés vindrà cada cop que s'editin o manipulin les corbes de la funció de transferència. Una vegada aquestes corbes estiguin definides i les imatges recalculades, mostrar-les a la pantalla del dispositiu no requerirà més procés intens de càlcul. Aquesta tècnica té l'avantatge que és fàcil d'implementar i els requisits de *hardware* del dispositiu no han de ser molt elevats per a dur-la a terme.

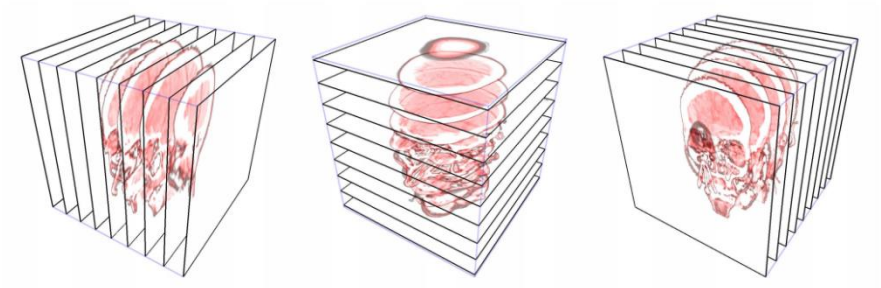


Figura 3.14: Visualització dels 3 conjunts de llesques necessaris per a l'*slicing*

De totes formes, també presenta alguns inconvenients. El fet d'haver de tenir 3 conjunts de llesques implica que s'hagi d'emmagatzemar a memòria tres vegades tot el conjunt d'imatges de cada eix necessàries per visualitzar el volum. Aquesta característica pot ocasionar que el dispositiu es quedi sense memòria disponible i hi hagin baixades de rendiment sobretot si el model a visualitzar té una resolució alta i el dispositiu no disposa de molta memòria. D'altra banda, aquest mètode ocasiona el nomenat *popping*, que és el petit salt que ocorre al model quan es canvia el conjunt de llesques a visualitzar. Aquest *popping* es pot arribar a atenuar una mica si s'incrementa el nombre de llesques per eix. D'altra banda, el nombre de llesques necessàries a cada eix per a la visualització correcta del volum, hauria de ser igual o superior a la resolució en vòxels de cadascun d'aquests eixos. Si el nombre de llesques es igual a la resolució no hi haurà pèrdua de resolució a l'hora de visualitzar el volum, però en cas de haver-hi menys no es visualitzarà el model amb tota la seva resolució. En cas d'haver-hi més llesques per eix que la seva resolució, s'hauran d'interpolar bilinearment els valors entre vòxels per a obtenir les imatges intermèdies que no es tenen. Aquesta tècnica de *slicing* origina que si el nombre de llesques no és molt alt, es pugui arribar a veure l'espai que hi ha entre elles sobretot a les posicions just abans de canviar el conjunt de llesques a visualitzar tal com es pot apreciar a la figura 3.15.

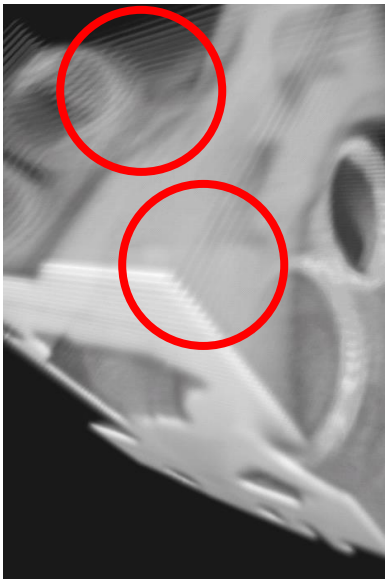


Figura 3.15: Imatge on es pot apreciar la separació entre llesques

3.2.2 Raycasting

El mètode de *raycasting* és un dels mètodes més utilitzats a l'hora de fer visualització de volum. Aquest mètode es basa en traçar raigs des de la càmera en direcció al volum, permetent aplicar i resoldre l'equació discreta de renderització de volum vista anteriorment. Aquest mètode de visualització ofereix una major qualitat que l'*slicing*, ja que en aquest mètode no hi han llesques i per tant no hi han espais buits entremitjos a la imatge resultant. Inicialment aquest mètode es duia a terme completament mitjançant la CPU, a causa que no hi havia suport *hardware* per part de les GPU per a manipular textures en 3 dimensions. S'ha de tenir en compte que les CPU eren molt més lentes que les actuals, això ocasionava que el *raycasting* sigues un mètode molt lent i computacionalment molt costós. La sortida al mercat de CPU's i targetes gràfiques més potents i amb suport *hardware* per a textures 3D ha permès que aquest mètode es dugui a terme en gran part a la GPU mitjançant shaders, fent el *raycasting* molt més ràpid.

El codi per a fer un *raycasting* molt simple mitjançant CPU seria el seguent:

```
Per cada pixel de la imatge resultant fer {
        color_acumulat <- 0
        opacitat_acumulada <- 0
        calcular direcció del raig()
        posició_mostra <- calcular_mostra_inicial()
                mentre ( posició_mostra < distancia màxima a la càmara ) fer {
                        si( posició_mostra està dintre del volum ) {
                                valor <- interpolació_trilinear( posició_mostra )
                                avalua_curves_FT( valor )
                                color_acumulat += color_evaluat
                                opacitat_acumulada += opacitat_evaluada
                        }
                avança_posició( posició_mostra, increment )
                }
        pinta_pixel( color_acumulat, opacitat_acumulada )
}
```

Amb aquesta tècnica, un raig és generat i llençat per a cada píxel de la imatge resultant. L'angle de visió de la càmera és utilitzat per generar un pla d'imatge (a aquest pla també se li sol anomenar pla de projecció) que servirà per a saber amb quina direcció s'han de projectar els raigs de visió que poden travessar el model de volum. Aquests rajos es llencen des de la posició de la càmera amb direcció del píxel que s'hagi de calcular en aquell moment. Aquest raig pot travessar o no el model de volum. En cas que el raig no travessi, el píxel es queda de color negre. En cas que el raig travessi el model de volum, s'haurà de calcular el color final resultant per aquell píxel. Cadascuna de les posicions de mostreig de l'espai es calcula aplicant un increment a cadascun d'aquests rajos fent que la posició "avanci" a través de l'espai. Aquest increment s'ha d'escollir de forma no arbitrària ja que si l'increment és massa alt pot causar que es tinguin poques mostres a l'interior del model i si és massa petit pot ocasionar que el temps necessari per a visualitzar el model sigui molt alt. Una vegada el raig es troba a dintre del model, s'haurà d'interpolar el valor de densitat entre els vòxels del voltant en aquella posició de l'espai. Aquesta interpolació s' anomena interpolació trilinear ja que ha de calcular el valor interpolat entre 8 valors de densitat. Una vegada obtingut el valor de densitat interpolat s'aplica la funció de transferència per aquest valor i el color i opacitat resultant s'acumulen per a aquest píxel. Aquest procés es repeteix fins que el raig surt fora del volum. Finalment, el color resultant s'aplica al píxel per al qual s'estava fent el càlcul del raig. Aquest procés es repeteix per a tots els píxels de la imatge resultant que es desitja obtenir. Tal com es pot observar, és un mètode intensiu de càlcul, sobretot quan la majoria de rajos travessen el model de volum.

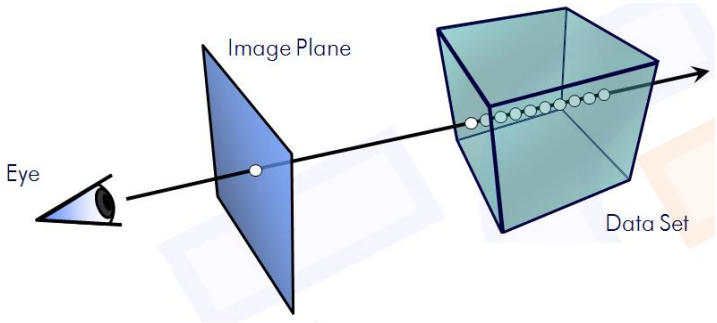


Figura 3.16: Imatge on es pot veure quin és el camí que segueix el raig dintre del volum

Les equacions que donen el color i la opacitat final per a un píxel són les que es poden veure a la figura 3.17. Tal com es pot veure, tant per al color com per a l'opacitat (lletra A a l'equació), el resultant és l'acumulació de cadascun dels colors previs mostrejats multiplicats per la seva opacitat. Aquesta equació és

la resultant discretitzada de l'equació contínua de *render* de volum vista anteriorment.

$$C = \sum_{i=1}^{n} C_i \prod_{j=1}^{i-1} (1 - A_j)$$

$$A = 1 - \prod_{j=1}^{n} (1 - A_j)$$

Figura 3.17: Equacions per al color i opacitat del *raycasting*

Aquesta tècnica només necessita emmagatzemar a memòria una sola vegada el model de volum, no com el *slicing* que necessitava disposar de tres conjunts d'imatges. Això ocasiona que la memòria requerida sigui menor, però per contra, aquest mètode és molt més lent a causa de la quantitat d'operacions per píxel que ha de realitzar. Una de les operacions més costoses que ha de realitzar el *raycasting* és la interpolació trilineal. Aquesta interpolació aconsegueix el valor de densitat d'un punt a l'espai dintre del model de volum mitjançant la interpolació linear entre la densitat dels 8 vòxels que el rodegen. S'ha de recordar que aquesta operació s'ha de realitzar a cadascuna de les posicions del raig de llum dintre del model de volum i que s'han de consultar 8 valors de densitat emmagatzemats a memòria cada cop. Això és el que ocasiona que el *raycasting* sigui un mètode tant costós computacionalment sobretot si l'increment de la posició del raig és molt petita.

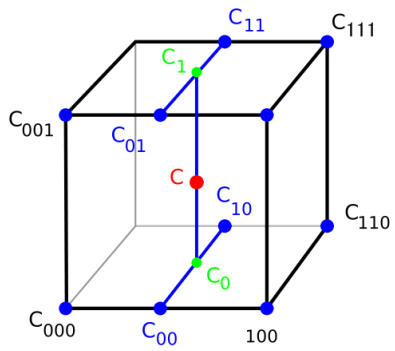


Figura 3.18: Interpolació lineal dels valors de densitat dos a dos començant per Cxxx, seguint per Cxx i Cx fins obtenir finalment C

Aquest mètode de visualització permet incorporar millores per a accelerar el procés. Les millores que s'implementaran són les següents:

- Finalització primerenca de raigs:

Es basa en el fet que una vegada la opacitat del píxel ha superat un màxim les posteriors mostres del raig no tindrien mes impacte sobre el color final d'aquest, de forma que no és necessari seguir calculant-les.

- Col·lisió de rajos amb esfera englobant del model de volum:

Amb l'objectiu d'accelerar encara més el càlcul de la imatge resultant, s'implementarà aquesta millora que permet calcular de forma ràpida si un raig travessarà l'esfera que engloba el model de volum. Amb aquesta millora un raig que no travessa l'esfera englobant directament no es calcularà i posarà el píxel de color negre. En el cas que el model ocupi poc espai a la pantalla aquest mètode proporcionarà grans millores de velocitat.

- Utilització de textura per al càlcul de la direcció i posició d'entrada i sortida de la capsa englobant del volum:

Aquest mètode realitza 3 passos per a aconseguir tant el punts d'entrada i sortida del volum com la direcció dels raigs que l'han de travessar.

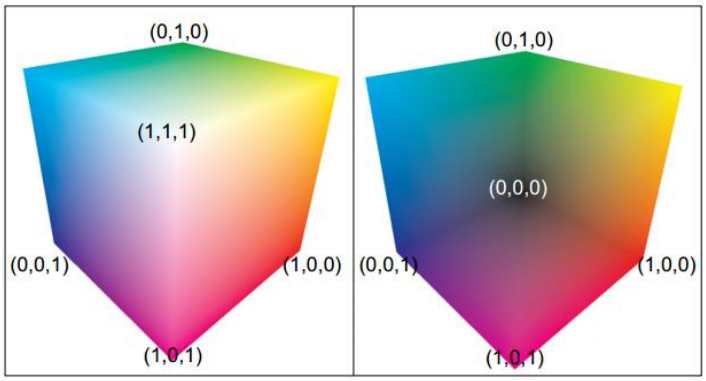


Figura 3.19: Visualització de les textures amb les cares frontals i posteriors de la capsa englobant del model de volum

Amb aquest mètode el primer pas consisteix a renderitzar les cares frontals de la capsa englobant del volum a una textura 2d que posteriorment es podrà

consultar (part esquerra de la figura 3.19) . La posició en 3d de cadascun dels vèrtexs es codifica amb el color dels mateixos, de forma que el color al canal vermell R de la textura és la posició a l'eix X, el color al canal verd G és la posició a l'eix Y i el color al canal blau B és la posició a l'eix Z. D'aquesta forma queda emmagatzemada a textura quina és per a cada píxel la posició d'entrada del raig al volum.

El segon pas es renderitzar a una segona textura les cares posteriors de la capsa englobant, d'aquesta forma s'emmagatzema per a cada píxel la posició de sortida del raig (part dreta de la figura 3.19). Una vegada es tenen aquestes dues textures es pot esbrinar quin és la direcció del raig restant la posició d'entrada de la posició de sortida per a cadascun dels píxels.

El tercer pas és situar la posició d'entrada del raig a la posició emmagatzemada al píxel mitjançant la primera textura i fer-lo avançar per dintre del volum fins que arribi a la posició de sortida, tal com s'ha explicat prèviament. D'aquesta forma amb aquest mètode només es llençaran rajos al volum als píxels on s'hagin rasteritzat fragments amb la capsa englobant.

3.2.3 Models d'il·luminació

Qualsevol volum visualitzat amb alguna de les tècniques anteriors només disposarà d'un color constant donat per les corbes de la funció de transferència. Es millorarà aquest cas incorporant al mètode de *raycasting* un model d'il·luminació amb l'objectiu que els models visualitzats siguin més realistes. Hi ha models d'il·luminació extremadament complicats que permeten visualitzar les ombres o la dispersió de la llum dintre del volum, però per a aquesta aplicació s'implementarà una il·luminació de Lambert (per a la component difusa) i Phong (per a la component especular). Amb aquests dos mètodes s'obtindran imatges molt millors de les que s'obtindrien sense models d'il·luminació.

$$I = k_a + I_L k_d\left(\vec{L} \cdot \vec{N}\right) + I_L k_s(\overrightarrow{H} \cdot \vec{N})^n$$

Figura 3.20: Equació d'il·luminació d'un punt que incorpora els models d'ambient, Lambert i Phong

A l'equació d'il·luminació de la figura 3.20 el primer sumand és la component ambient. Aquesta component és una il·luminació constant i igual per a totes les mostres avaluades que serveix per a il·luminar per igual totes les zones del model. En aquest projecte aquesta constant només es tindrà en compte si es desactiven els models de Lambert i Phong.

Figura 3.21: Visualització del model peu només amb component ambient

El model de Lambert és un mètode que modela la il·luminació en una escena on totes les superfícies són difuses, això és, que tenen color i poden ser il·luminades per una font de llum. Aquest mètode no és físicament correcte, ja que només serveix per a simular i aproximar visualment alguns aspectes de la llum. Aquesta il·luminació es durà a terme tenint en compte quin és l'angle que forma la normal de cada vòxel amb el vector de que apunta cap al punt de llum. Així es podrà comprovar en quin grau un vòxel està orientat cap a la font de llum per a il·luminar-lo més o menys. El model de Lambert és el segon sumand a l'equació d'il·luminació de la figura 3.20. Es pot veure com la intensitat de la llum I_L i la component difusa k_d tenen en compte com està d'alineat el vector de direcció a la llum $\vec{L}$ amb la normal $\vec{N}$ mitjançant el producte escalar. D'aquesta manera els vòxels que tinguin la seva normal orientada en direcció a la llum estaran més il·luminats que els que tenen la normal en la direcció oposada.

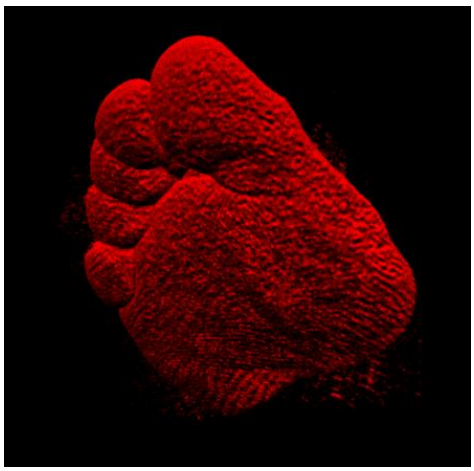


Figura 3.22: Visualització del model peu mitjançant el model de Lambert

El model de Phong serveix per a simular una superfície que incorpora reflexió especular, això és, una taca blanca o taca de brillantor provinent d'una suposada reflexió del punt de llum. Aquesta brillantor depèn de la posició de l'observador respecte a la normal del punt, la posició de la llum i les propietats del material, de forma que només es podrà veure si l'observador es troba a una determinada orientació respecte la reflexió de la llum. Cal dir que el model de Phong tampoc és físicament correcte, ja que serveix per a aproximar visualment aquest tipus de superfície. D'aquesta manera el mètode de Phong utilitza una aproximació d'aquest vector per a dur a terme els càlculs. L'especularitat es calcularà mitjançant l'obtenció del vector H, que és el vector resultant de sumar el vector de visió V i el vector L orientat a la font de llum. Aquest vector és el que es farà servir per a aproximar la reflexió, ja que es

calcularà quant d'orientat està amb la normal del punt. Quant més orientat respecte la normal, més reflexió especular tindrà aquell punt.

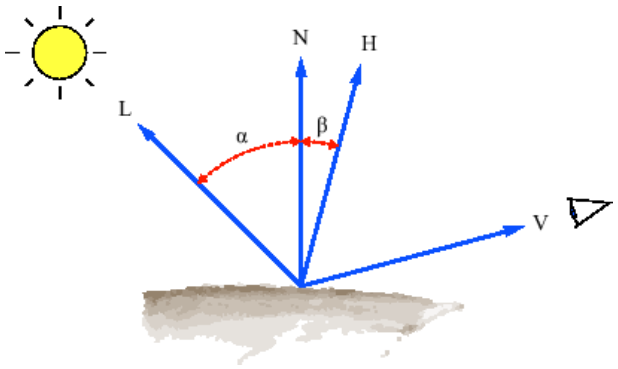


Figura 3.23: Representació del vector H,N,L i V

El model de Phong és el tercer sumand dintre de l'equació d'il·luminació de la figura 3.20. Es pot veure com en aquest model la intensitat de la llum I_L i la component especular k_s, que indica si es vol més o menys especularitat, tenen en compte com d'alineat està el vector $\vec{H}$ amb la normal $\vec{N}$ i el resultat elevat a la component de *shininess* n que engrandeix o disminueix la mida de la taca especular.

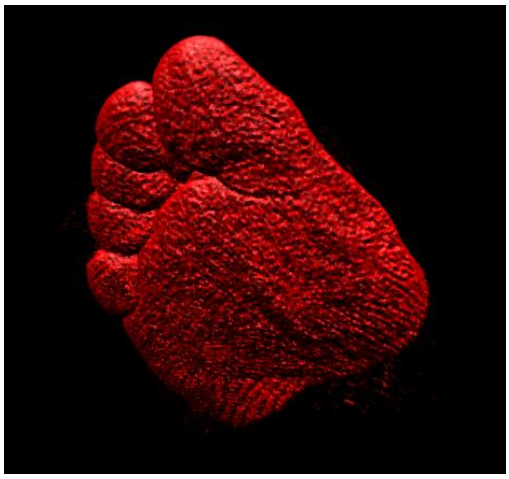


Figura 3.24: Visualització del model peu amb el model Lambert i Phong

Aquests models d'il·luminació es basen en la informació de la normal a cadascun dels vòxels del model de volum. Aquesta normal o gradient es pot pre-calcular gràcies a la utilització del mètode de les diferències centrals, ja que la normal a un vòxel és un valor constant. Aquest mètode calcula la normal a cadascun del vòxels fent la mitja dels valors de densitat dels vòxels que el rodegen per a cadascun dels tres eixos tal com es pot veure a la figura 3.25.

$$g_x\left(\vec{P}_{(i,j,k)}\right) = \frac{v\left(\vec{P}_{(i+1,j,k)}\right) - v\left(\vec{P}_{(i-1,j,k)}\right)}{2h}$$

Figura 3.25: Equació de càlcul de les diferències centrals

Calcular la normal, però, ocasiona que s'hagi d'emmagatzemar a memòria una quantitat de dades 4 vegades superior a la inicial, ja que per cada vòxel es disposa del valor de densitat i de 3 valors més que representen cadascuna de les components x, y i z de la normal o gradient. Això ocasiona que si no es disposa de molta memòria pot emplenar-se de seguida. A l'aplicació implementada aquestes normals es calcularan quan s'iniciï l'aplicació o quan es canviï el model a visualitzar, ja que es pressuposa que l'usuari voldrà realitzar una visualització amb il·luminació de Lambert i Phong.

El fet de disposar de la normal a cada vòxel implica que aquesta també s'hagi d'interpolar trilinialment a cadascuna de les posicions del raig dintre del volum, igual que es fa amb el valor de densitat. La interpolació del valor de densitat ja s'ha vist que és un procés costós, però la interpolació de la normal ho és més, ja que s'han d'interpolar de forma trilinear els tres valors del vector, no només un. D'aquesta forma el fet de disposar d'il·luminació de Lambert i Phong implicarà que el temps de càlcul per a cada imatge visualitzada es multipliqui com a mínim per 4.

En quant a la posició de la font de llum, sempre estarà situada a sobre de l'observador, de forma que el model de volum sempre estarà il·luminat des de la mateixa posició. S'ha triat aquesta posició per a la llum ja que és la que més contrast proporciona sobre les zones més i menys il·luminades dels models, a més de ser una bona posició per a poder observar la reflexió especular.

3.3 Estudi de característiques i limitacions de l'entorn Windows Phone

Una vegada estudiats el mètodes a implementar, s'ha d'investigar com és l'arquitectura de la plataforma Windows Phone i quines són les seves limitacions.

3.3.1 Arquitectura de Windows Phone

L'arquitectura interna com Windows Phone està implementat es basa en funcionalitats o serveis que són classificats en cinc grans grups:

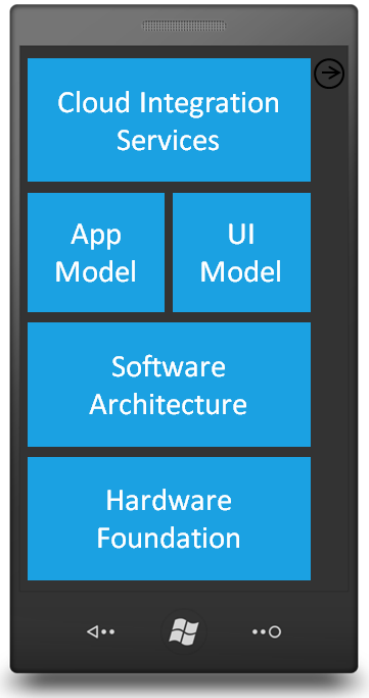


Figura 3.26: Grups que formen l'arquitectura de Windows Phone (http://lazure2.wordpress.com/2012/06/22/windows-phone-8-software-architecture-vs-that-of-windows-phone-7-7-5-and-the-upcoming-7-8/)

- Hardware Foundation

  Com a fabricant del sistema, Microsoft requereix que tot telèfon que desitgi executar Windows Phone disposi d'una sèrie de característiques mínimes, per assegurar la consistència de tots els usuaris del sistema. Els requisits són els següents:

- Una resolució de 800 x 480 o de 480 x 320 píxels
- Pantalla capacitiva amb 4 o més punts de contacte simultanis
- Disposar com a mínim dels sensors següents: A-GPS, acceleròmetre, brúixola, llum i sensor de proximitat
- Càmera de fotos de 5 Megapíxels o més
- Reproducció multimèdia accelerada mitjançant còdecs
- Mínim de 256 MB de memòria RAM i 8GB de memòria Flash
- GPU compatible amb DirectX 9
- CPU de 800Mhz o més
- Botons físics d'endarrere, menú principal i cerca

- Software Architecture

  L'arquitectura de software indica com el sistema gestiona la memòria tant per a aplicacions com per al nucli del sistema, el Kernel. En aquest cas, la plataforma d'aplicacions resideix a memòria d'usuari mentre que el Kernel, els divers, el sistema d'arxius, el networking, el sistema de visualització i gràfics i el sistema d'actualitzacions resideixen a l'espai de Kernel. Aquest sistema és de 32 bits, per tant només pot manipular 4 GB de memòria, 2 per a processos i 2 per a Kernel.

- App Model

  A Windows Phone les aplicacions es despleguen en forma de paquet XAP. Bàsicament es tracta d'un arxiu comprimit dintre del qual es poden trobar els arxius compilats i recursos originals de l'aplicació. L'única forma d'instal·lar una aplicació a Windows Phone és mitjançant la tenda oficial de Microsoft, la Marketplace, on el desenvolupador s'haurà de registrar per a poder vendre les seves aplicacions. Per a garantir la seguretat del sistema i evitar la pirateria el malware o virus, a cada aplicació li és assignada un ID únic i un certificat de seguretat emès quan l'aplicació ha estat aprovada al Marketplace de Windows Phone.

- UI Model

  El model d'Interface d'usuari de Windows Phone es composa d'elements, pàgines i sessions. Un element és tot control que es mostra a l'usuari, una pàgina és una agrupació lògica d'elements i una sessió és el conjunt d'interaccions que realitza un usuari sobre una aplicació, podent arribar inclús a invocar altres aplicacions.

- Cloud Integration Services

  Windows Phone inclou la possibilitat d'integració amb la *cloud computing,* un paradigma que permet oferir serveis de computació a través d'internet. Per defecte s'inclou la integració amb serveis com Exchange, Google Mail, Hotmail, Xbox Live, Skydrive, Facebook, Twitter o Bing. A més d'incorporar connexions a web-services o mètodes de notificació *push*.

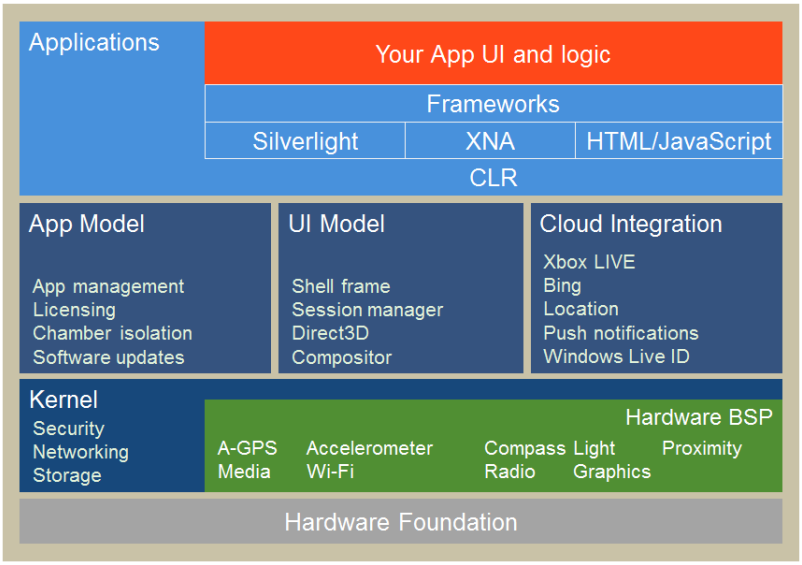


Figura 3.27: Grups en que esta dividida l'estructura de Windows Phone amb els seus principals serveis oferts (http://lazure2.wordpress.com/2012/06/22/windows-phone-8-software-architecture-vs-that-of-windows-phone-7-7-5-and-the-upcoming-7-8/)

Així doncs cadascuna de les aplicacions que es desenvolupin podran utilitzar els serveis oferts per cadascun d'aquests grups. A la figura 3.27 es pot veure quins són els principals serveis oferts per cadascun dels grups en que està dividit Windows Phone. Per exemple una aplicació dedicada a gràfics farà servir principalment determinats serveis oferts per diferents grups, tal com es pot veure a la figura 3.28.

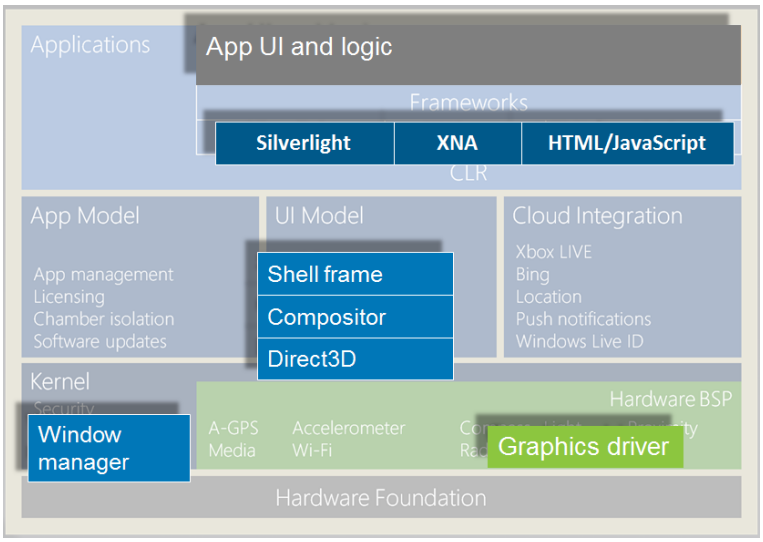


Figura 3.28: Exemple dels serveis utilitzats per una aplicació dedicada a gràfics (http://ecn.channel9.msdn.com/o9/te/NorthAmerica/2010/pptx/WSV313.pptx)

3.3.2 C#, Silverlight i XNA

Windows Phone disposa de diverses eines i elements per a implementar aplicacions.

El llenguatge que Windows Phone utilitza es C#, un llenguatge de programació orientat a objectes desenvolupat i estandarditzat per Windows com a part de la seva plataforma .Net. La seva sintaxi deriva de C/C++ i utilitza el model d'objectes de la plataforma .Net.

Amb l'objectiu de fer més fàcil la programació d'aplicacions, Windows ofereix de forma gratuïta l'editor Windows Visual Studio 2010 per a Windows Phone. Aquest editor es una potent eina ja que disposa de totes les funcionalitats necessàries per a implementar, desenvolupar i testejar aplicacions. Inclús disposa d'un emulador de dispositiu mòbil per a veure com es comportarà l'aplicació al dispositiu físic.

Windows Phone disposa de Silverlight per a implementar les interfícies gràfiques de les seves aplicacions. Microsoft Silverlight es una estructura per a aplicacions web i mòbils que afegeix noves funcions multimèdia com la reproducció de vídeo, gràfics vectorials, animacions i interactivitat de forma similar a la que ho fa Adobe Flash. L'editor que es disposa per a dissenyar i animar la interfície d'usuari és Microsoft Expression Blend. Aquesta és una aplicació de pagament amb la que es pot dissenyar, implementar i animar des de zero la interfície gràfica de les aplicacions.

No obstant, Silverlight no ofereix la possibilitat de treballar amb operacions gràfiques 3D i segons quines operacions 2D. Windows Phone li deixa totes aquestes possibilitats a XNA, un conjunt d'eines amb un entorn d'execució administrat i proporcionat per Microsoft que facilita la creació de jocs per a les plataformes Windows, Windows Phone i Xbox. XNA també funciona dintre de Visual Studio i amb C#, de forma que es fa transparent la utilització de XNA amb Silverlight.

El fet que s'utilitzi XNA per als gràfics i Silverlight per a la interfície gràfica té els seus pros i els seus contres. Per començar, Silverlight i XNA estan integrats dintre de Visual Studio, cosa que permet fer servir un o l'altre de forma pràcticament transparent per al dissenyador. Tant XNA com Silverlight disposen de mètodes que permeten mostrar contingut de l'altre entorn. A més, La plataforma XNA està dissenyada per a la implementació de jocs, de forma que ja incorpora totes les funcions necessàries per a manipular i tractar gràfics de forma molt fàcil i senzilla. Per contra, ja que en aquest cas es fa servir la versió per a mòbils de XNA, aquesta versió no té disponibles totes les funcionalitats que ofereix a d'altres plataformes. Per exemple, no permet disposar de textures en tres dimensions o *shaders* propis. Això impossibilita poder implementar mitjançant *shaders* alguns dels mètodes explicats prèviament, tal com el *raycasting* amb il·luminació. XNA si que disposa d'uns *shaders* predefinits, però cap d'aquests serveix per a implementar les funcionalitats que necessita l'aplicació. A la versió de Windows Phone 7.5 que es la que es disposa actualment, aquestes funcionalitats no hi son disponibles de moment, tot i que sembla que seran incorporades en futures versions.

Aquest estudi revela que es podrà implementar tant el mètode de *slicing* com el *raycasting* amb il·luminació. De totes formes el *raycasting* s'haurà d'implementar sense l'ús de *shaders* i fent totes les operacions d'interpolació i manipulació de vectors a la CPU. El fet de no disposar de *shaders* ocasionarà que el *raycasting* sigui molt més lent ja que no hi hauran operacions que es realitzin mitjançant la GPU.

3.4 Disseny funcional de l'aplicació

Aquestes seran les operacions que l'usuari haurà de poder realitzar a l'aplicació una vegada finalitzada:

- Selecció del model a visualitzar
- Edició de les diferents corbes de la funció de transferència
    - Definició de color
    - Edició ràpida de forma
    - Edició precisa de forma
    - Selecció precisa de corba
- Emmagatzematge de corbes de la funció de transferència
- Visualització del model mitjançant *raycasting*

Amb aquest conjunt d'operacions l'usuari podrà controlar què i com es visualitzen els diferents models de l'aplicació. Tot seguit s'explica què fa cadascuna de les operacions.

3.4.1 Selecció de model

L'aplicació disposarà d'un petit conjunt de models de prova ja incorporats per a que l'usuari pugi experimentar amb ells. En aquesta implementació no serà possible que l'usuari incorpori els seus propis models de volum. S'intentarà que aquests models siguin variats per a que l'usuari tingui diversos models per a investigar. A aquests models se'ls hi haurà disminuït la resolució per a que la execució de l'aplicació sigui més fluida i la mida de l'arxiu final sigui menor.

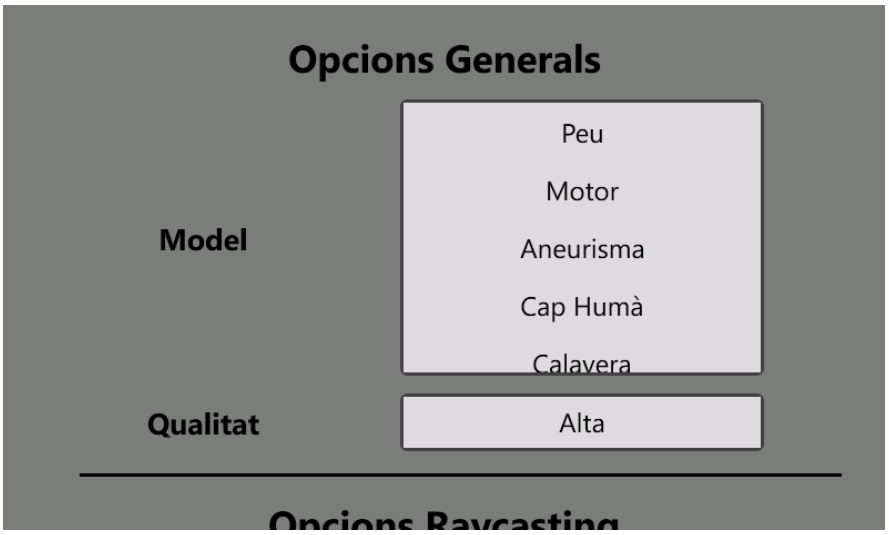


Figura 3.29: Captura amb la selecció de model de l'aplicació on hi ha *scroll* de pantalla per a mostrar tots els models disponibles

3.4.2 Navegació 3D del model

Una vegada el model de volum es mostri a la pantalla, serà possible manipular-lo per a poder observar les parts que requereixin atenció. El model es podrà rotar desplaçant un dit per sobre de la pantalla. Aquesta rotació serà implementada mitjançant la tècnica del *trackball* o *arcball* virtual. Amb aquesta tècnica el model sempre girarà respecte el seu centre, fent molt més còmoda la manipulació. Les propietats i característiques d'aquesta tècnica s'explicaran més endavant.

També es permetrà fer zoom sobre el model situant dos dits sobre la pantalla i separant-los o ajuntant-los es podrà controlar el nivell de zoom que es desitgi aconseguir. Aquesta funcionalitat estarà limitada de forma que no es pugui fer zoom fins massa prop o fins massa lluny. D'aquesta forma s'eviten les possibles situacions en que l'usuari perd de vista el model.

S'incorporarà també la possibilitat de fer un desplaçament lateral o *pan* del model per a poder veure altres parts del mateix. Es podrà moure el model desplaçant dos dits sobre la pantalla mantenint sempre la mateixa distància entre ells. Aquesta funcionalitat estarà limitada de forma que el model de volum sempre es mantindrà dintre dels límits de la pantalla per a evitar possibles situacions en que l'usuari perd de vista el model.

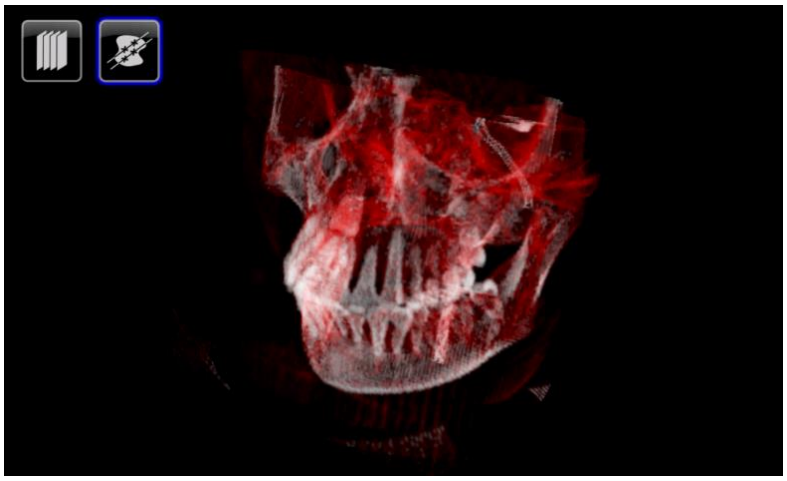


Figura 3.30: Captura de l'aplicació amb la navegació 3d del model

3.4.3 Edició de corbes de la funció de transferència

En aquestes seccions s'explica com es manipularan les diferents corbes de la funció de transferència. Es recorda que a la secció 3.1.2 d'aquesta memòria s'ha explicat com funcionen aquestes corbes i com fan possible que el model pugui visualitzar-se. Es recorda però que cada corba tindrà la seva forma que definirà quines zones es volen veure més opaques o transparents, i el seu color, que definirà de quin color es veuen el vòxels que es visualitzin amb la corba. Cal dir que a l'aplicació, cadascuna de les corbes estarà definida mitjançant una sèrie de vèrtexs on el primer i l'últim sempre estaran posicionats a la part més inferior de la pantalla on l'opacitat es 0, tal com es pot veure a la figura 3.31. En el cas que hi hagin dos vèrtexs que tinguin el mateix valor de densitat com per exemple els dos situats més a la dreta a la figura 3.31, el sistema tindrà en compte només el de menys densitat.

Cada cop que l'aplicació s'inicïi, la visualització del model inclourà una corba inicial de la funció de transferència genèrica idèntica per a tots els models. Per a entrar al mode d'edició de corbes, es farà un doble clic amb el dit a sobre de la pantalla de navegació 3d del model. D'aquesta forma es podran veure i editar les corbes actives. Aquest mode d'edició de corbes tindrà en compte que mentre es manipulin s'ha de veure el model, així, aquestes corbes seran sempre una mica transparents per a deixar veure el model que hi ha a sota.

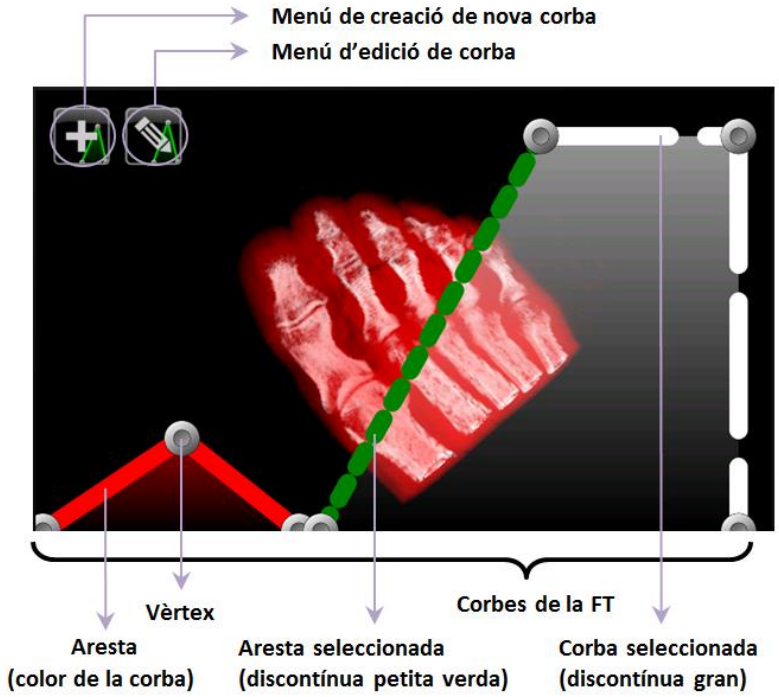


Figura 3.31: Explicació dels diferents elements del mode d'edició de corbes

3.4.3.1 Definició de color i forma

En tot moment durant l'edició de corbes de la funció de transferència, sempre hi haurà una d'aquestes seleccionada. La corba que en aquell moment estigui seleccionada serà la que es podrà editar. Es podrà saber quina és la corba seleccionada perquè les seves arestes seran discontínues.

En qualsevol moment es podrà afegir una nova corba al model polsant sobre el botó amb un signe de més que hi ha disponible a pantalla en mode d'edició de corbes, veure figura 3.31. Aquesta acció mostrarà el menú de creació de nova corba on es podran definir els paràmetres inicials d'aquesta, tals com el color o la forma.

En el cas que es vulgui editar el color o la forma de la corba seleccionada sense haver de crear-ne una de nova, s'haurà de prémer el botó d'edició de corba per a entrar al menú d'edició. Aquesta acció ensenyarà la pantalla amb el menú d'edició on es podrà seleccionar el nou color per a la corba o la seva nova forma. Les diferents opcions que proporciona aquest menú es poden veure a la figura 3.32. Una vegada s'està d'acord amb els canvis, es premerà el botó amb un *tick* per a acceptar les modificacions. Després d'això, es tornarà a mostrar el mode d'edició de corbes.

Figura 3.32: Explicació dels diferents elements del menú d'edició de corbes

Des del menú d'edició també es podrà canviar la forma que té la corba seleccionada. S'inclouran 4 tipus de formes que l'usuari podrà decidir quina és la que desitja per a la corba. S'ha de tenir en compte que seleccionant una forma de corba predefinida s'eliminarà la forma que hagués tingut aquella corba anteriorment.

En cas que es desitgi eliminar la corba seleccionada, es podrà eliminar entrant al menú d'edició i prement el botó amb la icona d'un cubell d'escombraries. Si aquest botó no està actiu és a causa que en aquell moment només hi ha una sola corba. En tot moment hi haurà com a mínim una corba de la funció de transferència que li donarà color i opacitat al model. Per a poder eliminar una corba s'han de tenir com a mínim dues.

3.4.3.2 Edició ràpida

Els dos modes d'edició que es presenten a continuació serà possible efectuar-los sobre 3 elements: corbes senceres, arestes i punts.

- **Edició ràpida de corbes**

  Quan una corba es troba seleccionada, es pot editar la corba sencera, incloent tots els seus punts i totes les seves arestes. D'aquesta forma, si el que es desitja és realitzar una edició ràpida sobre una corba, es situarà el dit al polígon interior de la corba i es mourà per la pantalla, ajustant-se als límits de la pantalla en el cas que s'arribi a qualsevol dels quatre cantons d'aquesta. Aquesta edició s'anomena edició ràpida perquè no es busca precisió al seu resultat. Aquesta és una bona forma de començar la edició ja que permet veure ràpidament per on es troben les diferents zones de densitat del model.

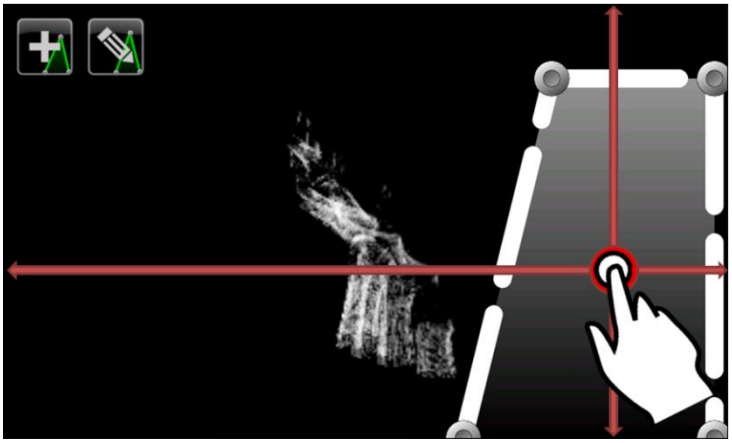


Figura 3.33: Posició de selecció i límits de moviment d'una corba seleccionada

- **Selecció d'arestes**

  Quan es seleccioni una aresta, aquesta passarà a tenir una discontinuïtat més petita per a diferenciar-la de les demés arestes de la corba de la funció de transferència. Per a seleccionar una aresta s'ha de polsar sobre ella a la pantalla. Quan es seleccioni una aresta i es comenci a bellugar el dit per la pantalla, només els dos punts que pertanyen a l'aresta es mouran. Aquest mètode s'ha implementat de forma que l'aresta no es pugui bellugar més enllà dels límits imposats per les arestes que pot tenir al costat. Es recorda que en el cas que hi hagin dos vèrtexs que tinguin el mateix valor de densitat com per exemple els dos

situats més a la dreta a la figura 3.33, el sistema tindrà en compte només el de menys densitat.

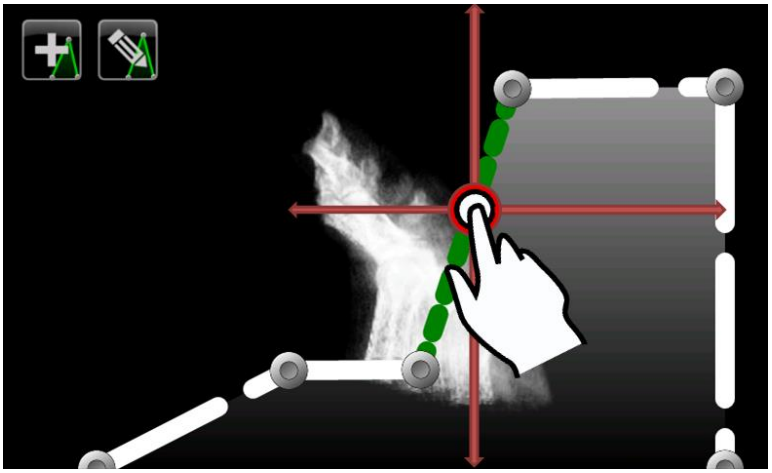


Figura 3.34: Posició de selecció i límits de moviment d'una aresta seleccionada

- **Selecció de punts**

  Quan es seleccioni un punt, aquest passarà a ser de color verd per a diferenciar-lo dels demés punts de la corba. L'edició ràpida de punts és exactament igual a l'edició d'arestes. També s'ha implementat el mètode per a que un punt no es pugui moure més enllà dels dos punts de les puntes de les arestes a les quals pot pertànyer. S'ha d'observar que els punts de les puntes de la corba no es podran moure en vertical.

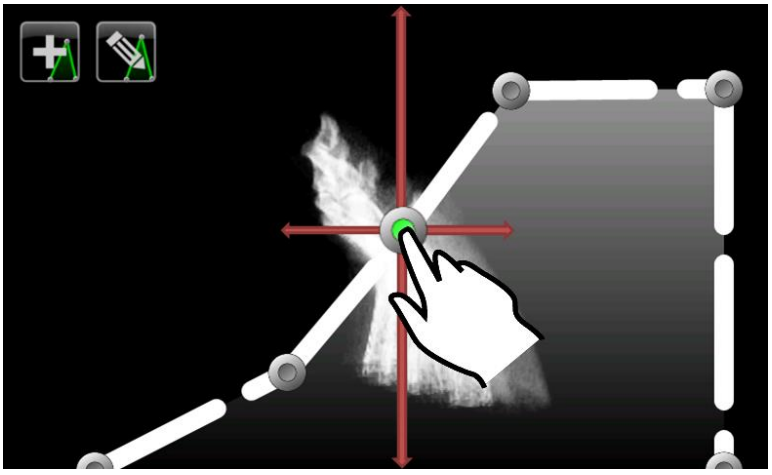


Figura 3.35: Posició de selecció i límits de moviment d'un punt seleccionat

Es recorda que la selecció tant de punts com d'arestes d'una corba deixa aquesta seleccionada per a poder canviar-li el color o la forma amb el menú d'edició de corbes abans esmentat.

- **Selecció i eliminació de punts**

  Serà possible també afegir o esborrar punts d'una corba. Per a afegir punts, s'haurà de fer un doble clic sobre qualsevol de les arestes per a que aparegui un nou punt de control al lloc on s'ha realitzat el gest.

  Per a esborrar un punt de control, s'haurà de fer un doble clic sobre un punt de control. S'ha de tenir en compte que no es podran esborrar els punts de control inicials que té una corba, això és, els de les puntes.

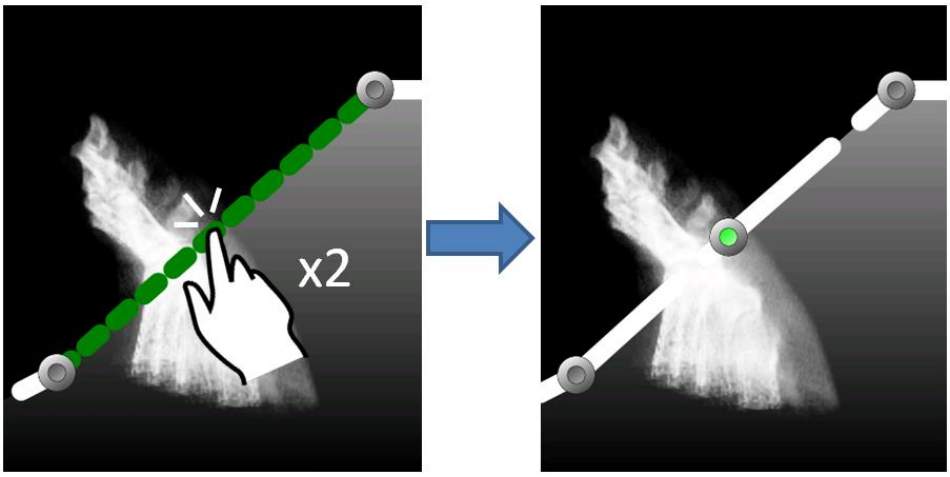


Figura 3.36: Creació d'un punt a sobre d'una aresta

3.4.3.3 Edició precisa

Tant si es té seleccionada una corba o un punt o una aresta, es realitzarà un moviment precís del mateix situant 2 dits sobre la pantalla. Aquest moviment permetrà posicionar i moure de forma precisa l'element seleccionat. A més, no serà necessari tenir el dit justament sobre l'element que es vulgui moure, podent realitzar aquest moviment des de qualsevol lloc de la pantalla.

Aquesta edició està pensada per a quan es desitja fer una edició molt acurada d'algun dels elements de la funció de transferència. Aquest mètode permetrà definir funcions de transferència amb una gran precisió permetent representar a pantalla parts molt específiques del model de volum.

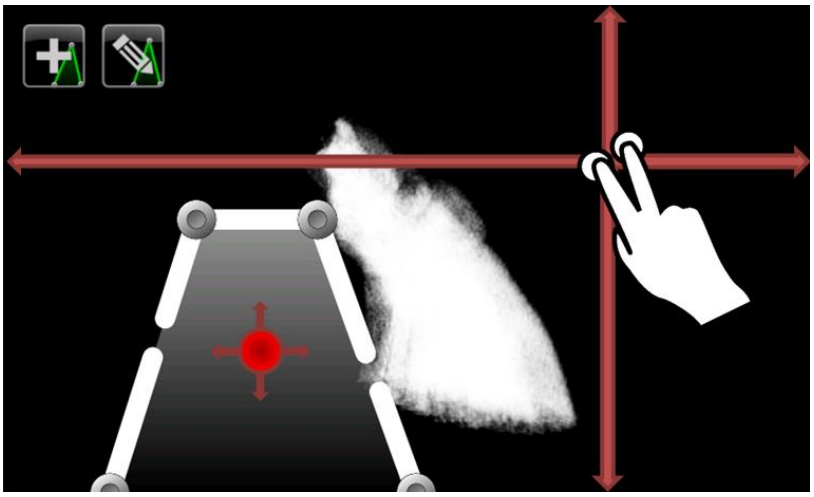


Figura 3.37: Edició precisa de qualsevol element seleccionat

3.4.3.4 Selecció precisa

Aquest mode de selecció està pensat per a quan es tinguin a pantalla moltes corbes a la vegada i sigui complicat haver de seleccionar alguns dels elements a causa de la densitat d'aquests a pantalla, sobretot punts o arestes. D'aquesta forma es podrà disposar de totes les funcions de transferència desitjades sense que sigui un problema haver de seleccionar els diferents elements que les composen per a la seva edició.

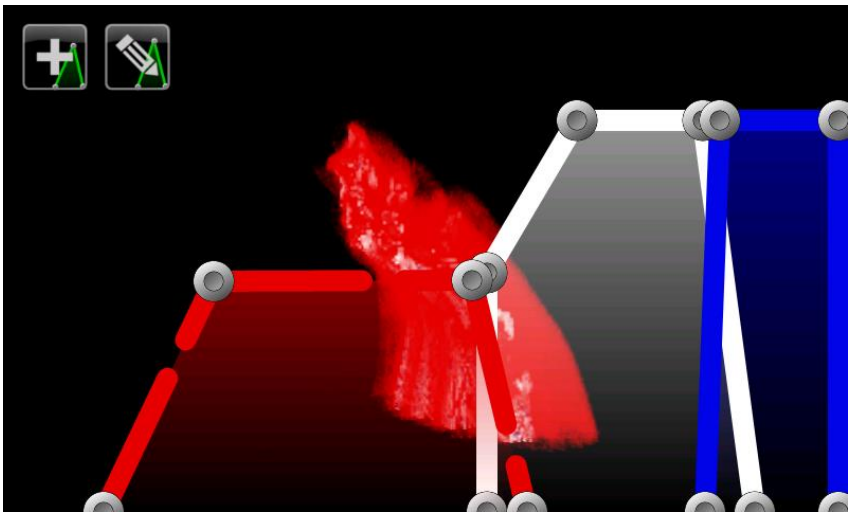


Figura 3.38: Moltes corbes i difícil selecció de punts i arestes concretes

Amb qualsevol dels 3 elements seleccionats, es deixarà el dit a qualsevol lloc de la pantalla durant un breu període de temps per a realitzar una selecció precisa d'un element. Quan es realitzi una pulsació d'una breu duració apareixerà al ben mig de la corba seleccionada un petit punt de color vermell que es podrà bellugar sense aixecar el dit de la pantalla amb el qual es seleccionarà l'element que quedi sota aquest punt. D'aquesta manera es podrà seleccionar un element de forma precisa i sense haver de tenir el dit sobre l'element que es desitja moure. Una vegada seleccionat un element, quedarà també seleccionada la corba a la qual pertanyi l'element que s'ha seleccionat.

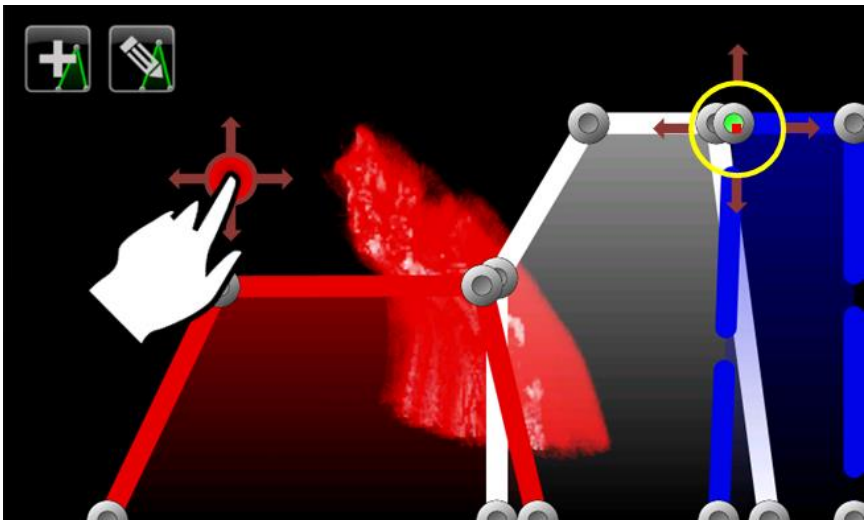


Figura 3.39: Selecció precisa d'un punt molt proper a d'altres

3.4.3.5 Emmagatzematge de corbes

Dintre del menú d'edició i de creació de corbes també s'inclourà la possibilitat de desar a fitxer el conjunt de corbes per a poder utilitzar-les més endavant. Aquesta funcionalitat està pensada per a que les corbes es puguin emmagatzemar una vegada acabada la seva edició per no perdre-les o per a continuar la seva edició més endavant.

Dintre del menú d'edició i de creació de funcions també hi haurà la possibilitat de carregar un conjunt de corbes guardades prèviament. S'ha d'anar amb compte perquè el fet de carregar corbes des de fitxer substituirà les corbes que hi hagi prèviament al mode d'edició, es a dir, les corbes prèvies es perdran.

Dintre d'aquest menú cadascun dels conjunts de corbes disposa d'un botó per a eliminar-los del dispositiu mòbil. Una vegada eliminades les corbes ja no es podran tornar a recuperar.

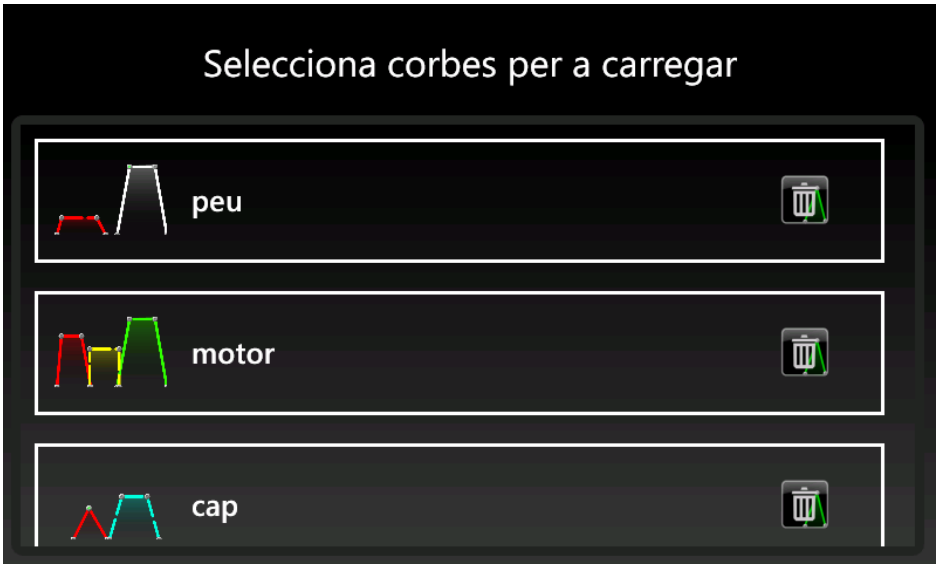


Figura 3.40: Captura de l'aplicació amb el menú de càrrega de corbes des de fitxer

3.4.4 Visualització del model mitjançant *raycasting*

Quan l'usuari entri a mode de manipulació del model, podrà decidir si visualitzar el model mitjançant *slices* (llesques) o mitjançant el mètode de *raycasting*. Per defecte la visualització del model es farà amb el mètode de *slicing* a causa que es pot manipular el model en temps real. Si en algun moment l'usuari vol canviar de mètode de visualització haurà de prémer el botó de *raycasting* visible en tot moment durant la manipulació del model. A causa que el *raycasting* és lent, primerament es visualitzarà una versió amb molt baixa resolució d'imatge, però aquesta s'anirà refinant a mesura que passi el temps. Fins que la imatge no estigui acabada es mostrarà un indicador d'espera per a indicar que la imatge encara no està totalment calculada. Una vegada la imatge estigui completa aquest indicador desapareixerà.

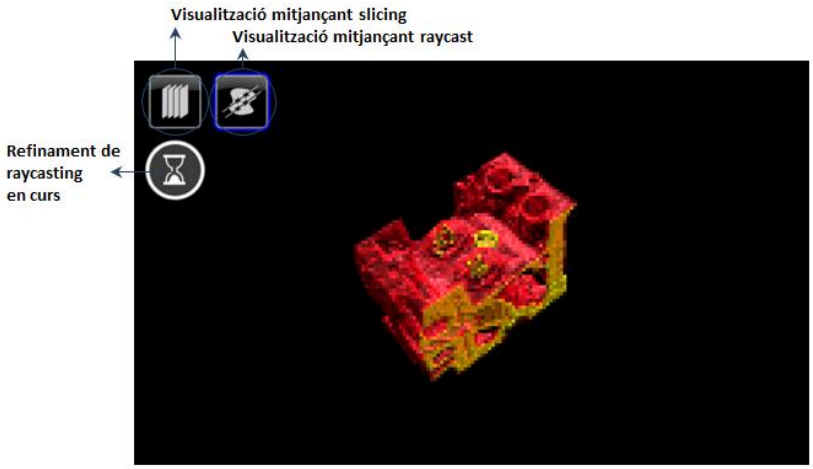


Figura 3.41: Explicació dels diferents elements del mode de manipulació 3d mentre es refina la imatge amb el mètode de *raycasting*

El mètode de *raycasting* està principalment pensat per a poder visualitzar el model amb una gran qualitat. El seu ús principal doncs serà veure la imatge del model una vegada acabades de modificar les diferents corbes de la funció de transferència. A continuació es mostra una comparativa d'imatges del mateix model aconseguides mitjançant els mètodes de *slicing* i *raycasting*. Es pot observar com la qualitat obtinguda pel mètode de *raycasting* és superior ja que ofereix una millor percepció del volum a causa de l'enfosquiment de les zones orientades en direcció contraria a la llum. A la part de conclusions d'aquesta memòria es profunditza més en els temps emprats per aconseguir aquestes imatges així com d'altres resultats obtinguts.

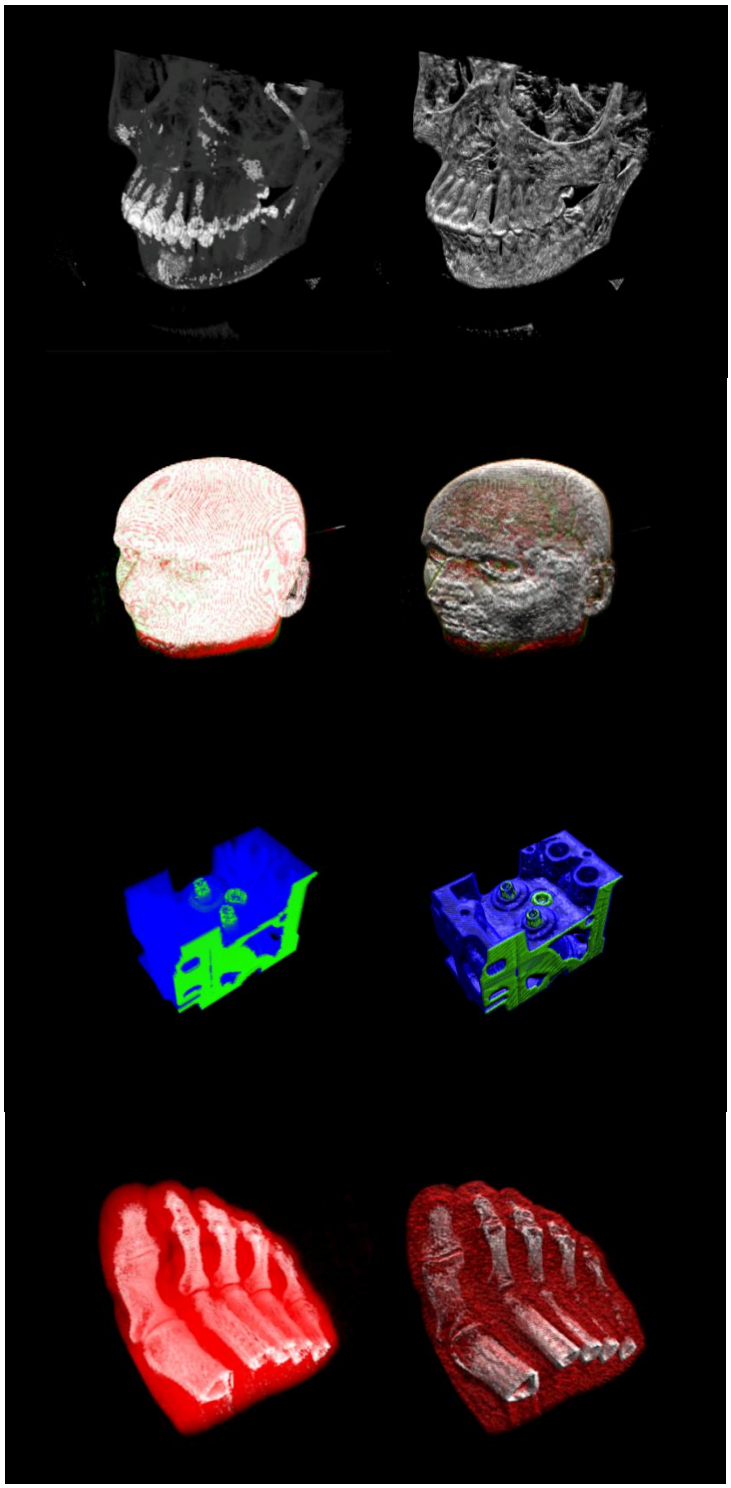


Figura 3.42: Comparació de la visualització de diferents models amb la tècnica de *slicing* (esquerra) i *raycasting* (dreta)

3.4.5 *Workflow* habitual

En aquesta secció es mostra quin és el *workflow* habitual d'aquesta aplicació per a aconseguir visualitzar els models de volum de forma correcta. Així doncs un esquema inicial de les operacions habituals que s'hauran de fer quan s'executi l'aplicació seran els següents:

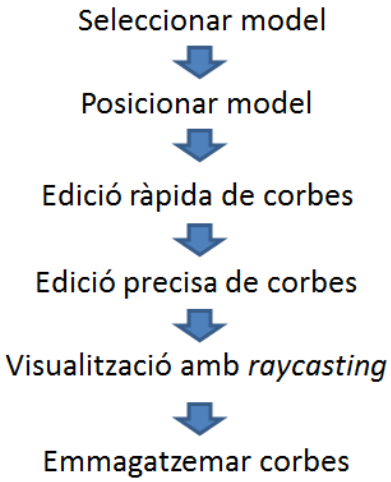


Figura 3.43: Esquema de *workflow* habitual

Figura 3.44: Menú inicial de l'aplicació

- Primerament s'accedirà al menú d'opcions on es seleccionarà el model que l'usuari vulgui visualitzar, igualment amb la qualitat i les propietats del *raycasting* segons es desitgi.

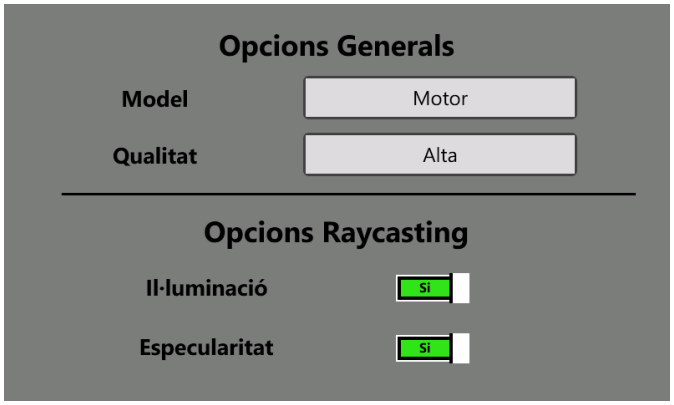


Figura 3.45: Menú d'opcions de l'aplicació

- Una vegada seleccionat el model i les diferents opcions de visualització, s'accedirà al mode de manipulació 3d del volum des del primer botó del menú principal i es manipularà el model fins posicionar-lo a una posició on es pugui veure sencer i totes les seves zones.

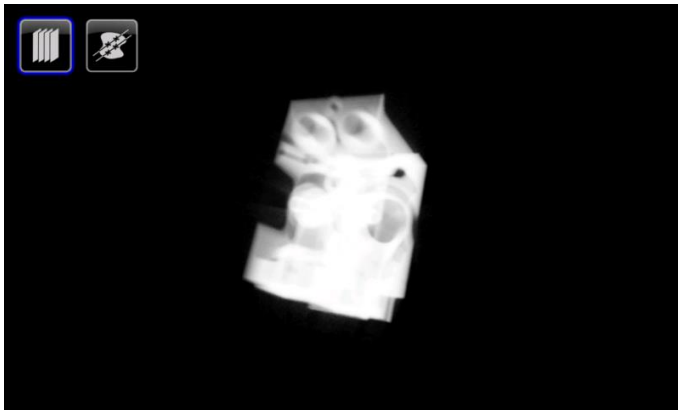


Figura 3.46: Model manipulat fins obtenir una posició correcta

- Generalment la corba per defecte de la funció de transferència donarà una idea de la forma general del model de volum, no obstant, aquesta corba no serà suficient per a visualitzar de diferents colors i amb diferents opacitats les diferents parts del model.

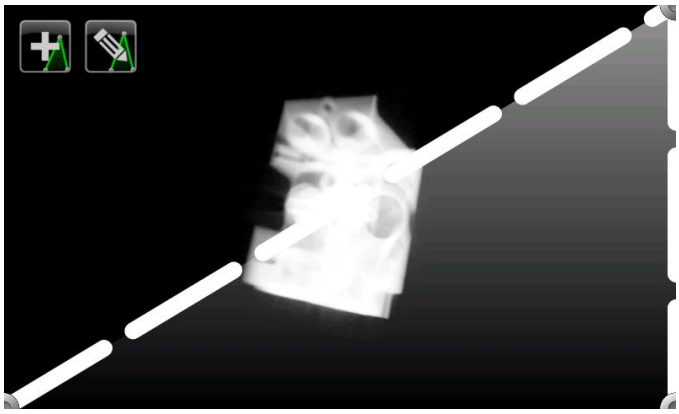


Figura 3.47: Mode d'edició de corbes amb la corba inicial per defecte

- Així doncs primerament s'haurà d'accedir al mode d'edició de corbes fent doble clic sobre la pantalla i una vegada a aquest mode es procedirà a afegir o modificar corbes canviant la forma o el color segons es desitgi accedint al menú d'edició o creació de corba amb el botó corresponent. La manipulació de les corbes en aquesta etapa serà amb l'edició ràpida ja que només interessa visualitzar de diferents colors les parts de diferent densitat del model.

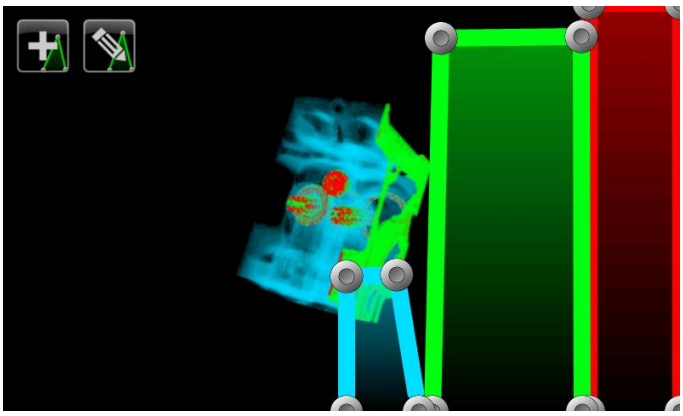


Figura 3.48: Zones de densitat del model localitzades després d'una edició ràpida de corbes

- Una vegada s'han localitzat les diferents zones de densitat que es volen localitzar es procedirà a l'edició precisa de corbes per acabar de refinar cadascuna d'aquestes corbes i poder visualitzar correctament cadascuna de les parts del model que requereixin atenció.

- Un cop s'han acabat d'editar de forma precisa les corbes es tornarà al mode de manipulació del model per a poder veure el model des de diferents angles segons es necessiti. Cal recordar que si la manipulació del model es torna molt lenta a causa que s'ha fet zoom sobre el model sempre es pot accedir al menú d'opcions i canviar la qualitat de visualització del model per a que la manipulació sigui més fluida.

- Després d'haver comprovat que les zones del model que es visualitzen són correctes es procedirà a visualitzar el model amb el mètode de *raycasting* per a poder observar tots els detalls del model amb una major qualitat.

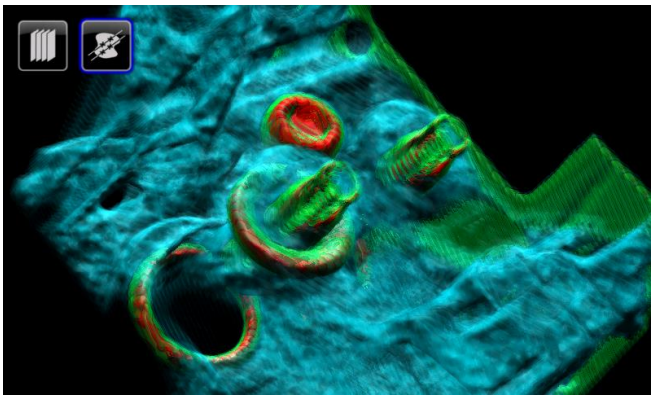


Figura 3.49: Visualització de les zones d'importància de model amb *raycasting*

- Finalment, les corbes utilitzades per a aconseguir la visualització del model es desaran a fitxer de forma que posteriorment es pugin tornar a fer servir o editar si és necessari.

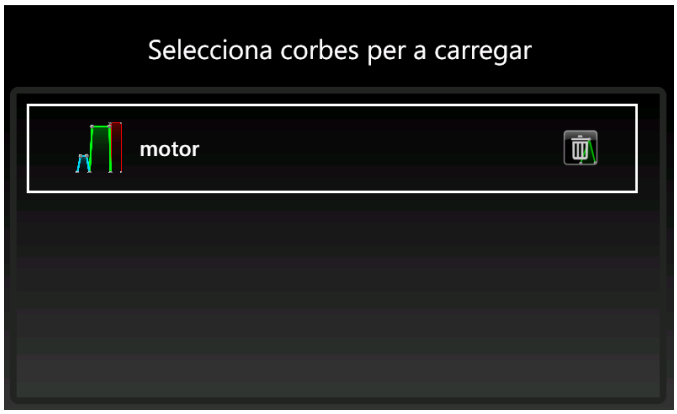


Figura 3.50: Corbes desades a fitxer per a la seva posterior utilització

3.5 Implementació

En aquesta secció s'explicaran quines han sigut les decisions més rellevants preses en quant a implementació de l'aplicació.

3.5.1 Bucle principal

Després de veure quines són les diferents característiques que ofereix Windows Phone per a implementar aplicacions, s'ha decidit implementar l'aplicació com si d'un joc de XNA es tractés, però amb Silverlight per als menús i el mode d'edició de corbes de la funció de transferència. Aquesta decisió ha estat presa per a poder utilitzar tots els mètodes gràfics que XNA inclou, amb la versatilitat del Silverlight per a la creació i manipulació de menús. Així doncs aquesta serà una aplicació hibrida entre XNA i Silverlight.

El fet que l'aplicació tingui components de XNA i Silverlight fa que aquesta s'executi en forma de joc. Això vol dir que l'aplicació seguirà el gràfic de la figura 3.50, on es pot veure que primerament s'iniciaran i es carregaran tots els materials necessaris i després s'executarà el bucle principal entre les funcions d'Update i Draw. Aquestes dos funcions principals s'encarreguen d'actualitzar els components de l'aplicació i de dibuixar-los a pantalla. D'aquesta forma tota la gestió d'interacció amb l'usuari, tractament de dades del model de volum i càlcul del *raycasting* quedaria dintre de la funció Update, mentre que el desplaçament, rotació i pintat de polígons de l'*slicing* es realitzaria dintre de la funció Draw del bucle.

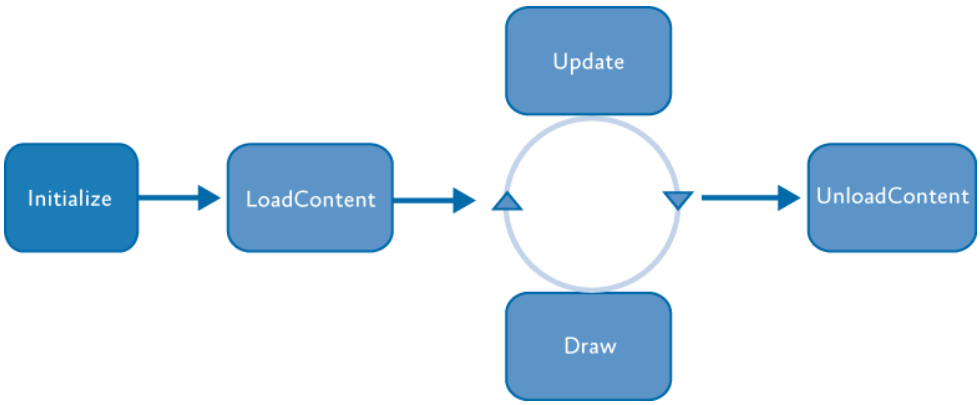


Figura 3.51: Esquema amb el bucle principal de l'aplicació (http://iloveshaders.blogspot.com.es/2011/04/anatomy-of-xna-project.html)

3.5.2 Arquitectura de l'aplicació

En aquesta secció s'explica quines son les diferents parts com està estructurada l'aplicació. La figura 3.51 mostra un diagrama simplificat de classes de les classes principals que formen l'aplicació.

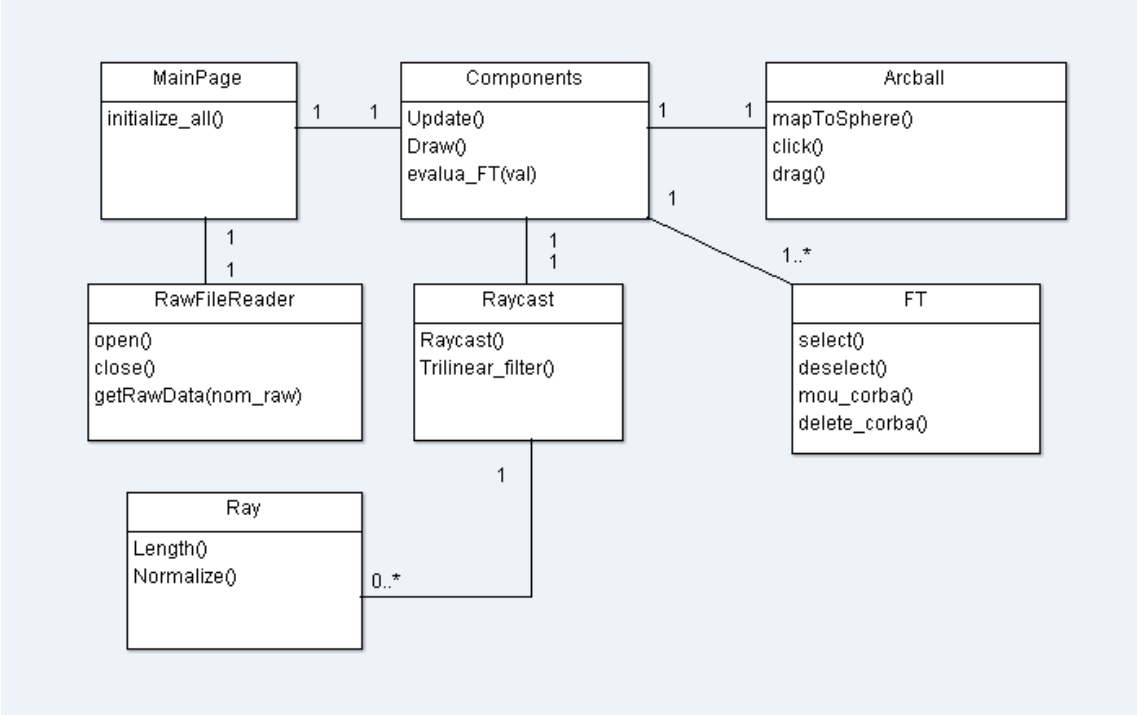


Figura 3.52: Diagrama UML simplificat de l'arquitectura de l'aplicació

- MainPage
    - Aquesta classe definirà la pàgina principal de l'aplicació, des d'on s'iniciaran totes les corbes de la funció de transferència i la plataforma XNA. Aquesta serà la classe que incorporarà també les funcionalitats dels diferents elements que formen els menús i la interacció amb l'usuari que proporciona Silverlight. Donat que aquesta classe és la principal, serà la que s'encarregarà d'enviar les peticions a la classe RawFileReader per a obtenir els models de volum des de fitxer i enviar-los-hi als components XNA que és on s'emmagatzemaran durant l'execució de l'aplicació.

- Components_XNA
    - Aquesta serà la classe que inclourà totes les operacions de gestió de gràfics proporcionats per XNA. Dins d'aquesta classe serà on estaran els mètodes importants de Draw i Update explicats a la secció següent i que són els que fan capaç mostrar el model de volum a pantalla. Donat que incorpora les funcionalitats d'XNA, aquí serà on s'implementarà el mètode de *slicing* abans explicat i on s'emmagatzemaran totes les llesques que formen el model de volum. D'igual manera aquí serà on es definiran tots els elements necessaris per a visualitzar l'escena tals com paràmetres de la càmera, matrius de món, de view i de projecció, etc.

- FT
    - Aquesta classe definirà la funció de transferència amb totes les corbes que aquesta pot disposar. Els components que aquesta classe haurà de tenir són el color de les corbes i els seus vèrtexs i arestes. De la mateixa forma també incorporarà totes les funcions necessàries per a que l'usuari pugui interactuar amb tots aquests elements i les funcions necessàries per a poder avaluar la funció de transferència mitjançant totes les seves corbes per a obtenir un color i una opacitat per a una densitat donada.

- Raycast
    - En aquesta classe serà on s'implementarà el mètode de visualització mitjançant *raycasting* vist anteriorment. D'igual manera també incorporarà totes les funcions necessàries per a dur a termes les operacions tals com la interpolació trilinear o el càlcul de normals del volum per a aconseguir il·luminació de Lambert i Phong al model.

- Ray
    - Aquesta classe defineix els raigs necessaris per a dur a terme el *raycasting*, amb totes les seves operacions necessàries tals com el seu mòdul o la seva normalització. S'hagués pogut fer servir el propi vector de 3 components que ofereix XNA, però es va decidir implementar-ne un de nou ja que els constructors que oferia el vector de XNA no oferien tanta flexibilitat com els que s'han implementat en aquesta classe.

- Arcball
    - Aquesta classe tindrà disposarà de totes les funcions necessàries per a poder dur a terme la rotació del model mitjançant el mètode de *virtual trackball* o *arcball*. Aquest mètode s'explicarà més endavant en aquesta memòria.

- RawfileReader
    - Aquesta classe tindrà totes les funcions necessàries per a poder llegir de fitxer les dades de densitat dels diferents models de volum emmagatzemats al dispositiu mòbil. Aquests fitxers tindran format .raw, o sigui un fitxer de dades cru.

3.5.3 Eines externes utilitzades

- S'ha utilitzat el software per a manipulació de models de volum Voreen, que permet realitzar tot tipus de manipulació als models tant com visualitzar-los i editar les funcions de transferència. Inicialment els models de prova tenien una resolució més alta que la que la memòria del dispositiu mòbil era capaç d'emmagatzemar. Així doncs s'ha utilitzat Voreen per a reduir de mida els arxius dels models de volum, així com la seva resolució per a que el dispositiu mòbil sigui capaç d'emmagatzemar-los a memòria. També s'ha aconseguit que tots els models tinguin la mateixa resolució per a que la incorporació de paràmetres per a la seva visualització sigui més fàcil.

- Per a la creació de les imatges dels diferents botons i elements que s'utilitzen a tota l'aplicació s'ha fet servir el software d'edició i creació d'imatges Eyeon Fusion.

- S'ha fet servir el software Expression Blend 4.0 que ofereix Windows per a canviar les imatges dels diferents elements que composen els menús oferts per Silverlight tals com botons, imatges de fons etc. D'igual manera també s'ha utilitzat per a canviar el comportament d'alguns botons de forma que s'adaptessin a les necessitats requerides. Gràcies a aquest programa s'han pogut dissenyar i organitzar de forma clara i ordenada els diferents menús de forma visual i interactiva, ja que l'editor que ofereix Microsoft Visual Studio 2010 és molt limitat en aquest aspecte.

- S'ha utilitzat la Toolkit proporcionada per coding4fun per a Windows Phone per a incorporar a l'aplicació un selector de color per a les corbes de la funció de transferència. Aquesta llibreria proporciona diverses funcionalitats ja implementades, entre d'altres, el selector de color que s'utilitza a l'aplicació.

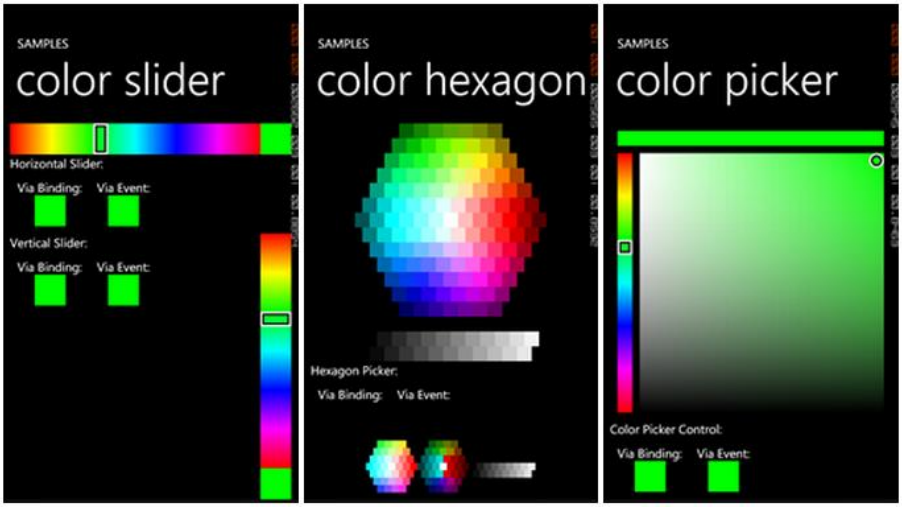


Figura 3.53: Diversos models de selector de color que ofereix coding4fun (http://coding4fun.codeplex.com/)

Per tal de poder utilitzar el selector de color d'aquesta llibreria, només ha calgut incorporar el fitxer de llibreria al conjunt d'arxius necessaris per executar l'aplicació i incorporar-la al conjunt de llibreries de referència que inclou el projecte.

Una vegada copiat el fitxer i referenciada la llibreria, es construeix una instància del selector:

```
Coding4Fun.Phone.Controls.ColorPicker col_picker = new
Coding4Fun.Phone.Controls.ColorPicker();
```

Una vegada fet això, es podrà accedir a les funcions que incorpora aquesta classe per a poder utilitzar el color que l'usuari hagi seleccionat.

3.5.4 Problemàtica

En aquesta secció s'explicarà quins han sigut els mètodes implementats per a aconseguir les funcionalitats de l'aplicació i quins han sigut els problemes sorgits durant la implementació.

3.5.4.1 Corbes de funcions de transferència

El fet d'implementar les corbes de la funció de transferència va ser tot un repte ja que les corbes havien de ser interactives, s'havien de poder modificar sense pràcticament cap limitació. S'havia de poder afegir punts, s'havia de poder esborrar punts, moure només arestes, moure només punts, moure la corba sencera... Tal com es pot apreciar s'havia de dissenyar una estructura de dades que fos capaç de gestionar totes aquestes operacions de forma fàcil i ràpida.

La forma que es va decidir implementar doncs les corbes va ser que una corba estaria composada per dues o més components. Cadascuna d'aquestes components seria cadascuna de les arestes que defineixen la corba. D'aquesta forma, cada aresta hauria de tenir la informació de la posició dels seus dos punts, el tipus de línia que es dibuixaria entre ells, i el tipus de punt de la seva dreta. Aquestes arestes estarien connectades entre elles mitjançant una llista encadenada, ja que aquesta estructura de dades permet incorporar o extreure elements de la llista tenint en compte la seva ordenació.

Així doncs, amb aquesta estructura de dades es podia gestionar de forma fàcil i ràpida tant el moviment de punts i arestes, com l'eliminació o adició de punts.

Cadascun d'aquests components (des d'ara se'ls anomenarà arestes) pertanyen només a una única corba. El fet d'implementar aquestes arestes dintre d'una única classe permet accedir molt fàcilment a tota la informació de la corba general, tal com el color.

L'elecció d'aquesta estructura de dades permet generar de forma molt fàcil el vector d'opacitats de cadascuna de les corbes de la funció de transferència. El vector d'opacitats de la corba guarda quin és el valor d'opacitat que li correspon a un valor de densitat donat, d'aquesta forma, per a esbrinar quina opacitat li pertany a un vòxel, només s'ha d'accedir a la posició del vector que conté la densitat d'aquell vòxel.

Per a generar aquest vector, s'ha de recórrer la llista encadenada d'arestes i interpolar els valors d'opacitat entremitjos dels punts de les arestes. Una

vegada recorreguda tota la llista, el vector contindrà el valor corresponent d'opacitat per a tots el valors de densitat.

En cas de voler desplaçar tota la corba, només s'ha de recórrer la llista encadenada d'arestes i desplaçar-les la quantitat necessària cadascuna d'aquestes.

El fet de tenir aquestes arestes permet accedir de forma fàcil tant a l'aresta anterior com a la següent, podent implementar amb uns senzills càlculs totes les operacions de limitació en quant al moviment que poden realitzar aquestes arestes. És molt fàcil saber quins són els límits per l'esquerra i per la dreta que tindrà un punt ja que només s'ha d'accedir a la informació dels punts que tenen les arestes anterior i posterior.

3.5.4.2 Rotacions

Per a realitzar les rotacions necessàries al model de volum durant la seva visualització, s'ha decidit implementar el mètode de rotació anomenat de *trackball*, *arcball* o rotació en coordenades d'ull.

El mètode de *trackball* trasllada els moviments en 2D del dit sobre la pantalla a rotacions en 3D. Això s'aconsegueix projectant la posició del dit sobre una esfera imaginaria situada al viewport. A mesura que el dit es mou, la càmera o l'escena roten per a mantenir el mateix punt a l'esfera sota la posició del dit.

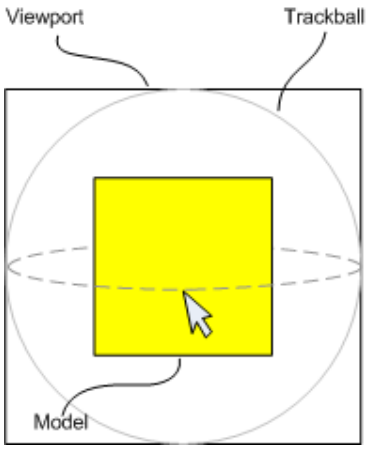


Figura 3.54: El visor en 3D amb l'esfera de *trackball* des de la prespectiva de l'usuari

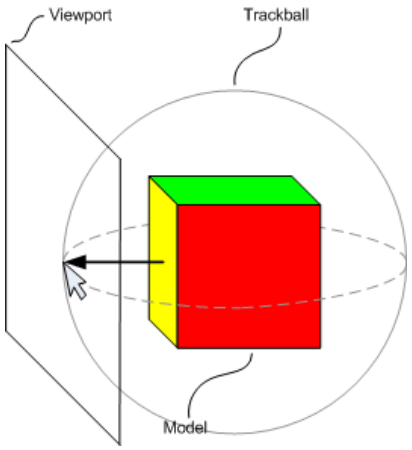


Figura 3.55: Vista de costat il·lustrant el punt de l'esfera que es correspon amb la posició del dit

Quan el dit es mou horitzontalment és necessari una rotació sobre l'eix Y per a mantenir el mateix punt de l'esfera sota el dit.

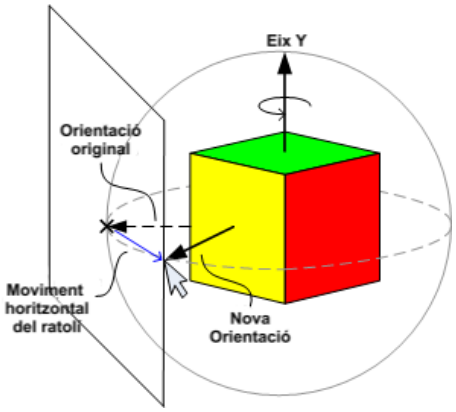


Figura 3.56: Un moviment del dit en horitzontal ocasiona una rotació de l'escena sobre l'eix Y en coordenades d'ull

De la mateixa manera, un desplaçament del dit en vertical ocasiona una rotació al voltant de l'eix X.

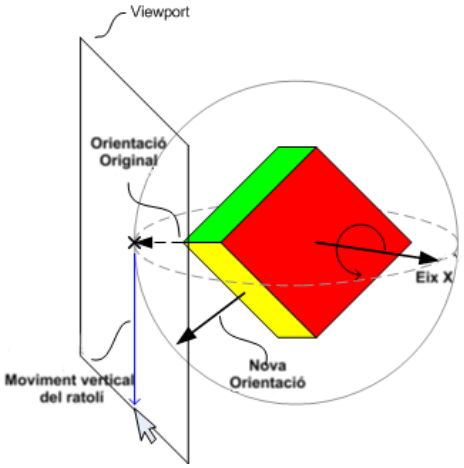


Figura 3.57: Un moviment del dit en vertical ocasiona una rotació de l'escena sobre l'eix X en coordenades d'ull

Aquesta tècnica proporciona un mètode intuïtiu pel qual un model pot ser dut a qualsevol orientació aplicant una combinació de rotacions sobre els eixos X i Y.

A cada moviment del dit es necessita calcular la rotació amb l'objectiu de mantenir el mateix punt de l'esfera sobre el cursor. Són dos els passos que s'han de dur a terme per tal de aconseguir-ho. El primer pas és esbrinar quin punt de l'esfera es troba sota la posició del dit. El segon és computar la rotació necessària per a moure el punt vell al nou punt de l'esfera.

Amb l'objectiu de trobar el punt a l'esfera sota la posició del cursor s'ha de projectar la posició en 2D en sistema de coordenades de la pantalla sobre l'esfera situada al *viewport*.

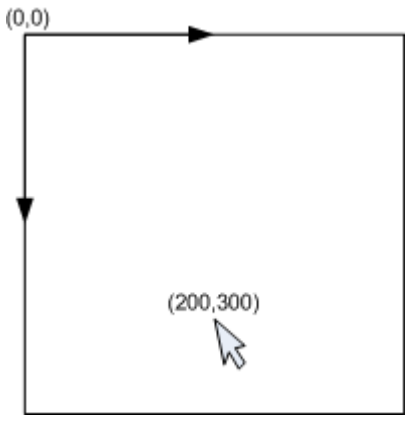


Figura 3.58: El cursor proporciona la seva posició en sistema de coordenades de la pantalla

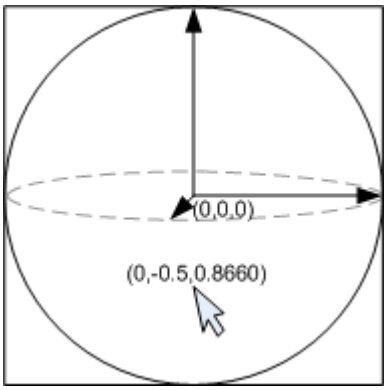


Figura 3.59: Es projecta aquest punt 2D a l'esfera inscrita al *viewport*. El resultat és un punt amb coordenades 3D

Com només es té interès en calcular la rotació, es pot triar el sistema de coordenades que sigui més convenient. El cas més simple és utilitzar una esfera de radi 1 centrada a l'origen (0,0,0). Això ocasiona que per trobar les components X i Y s'hagin de convertir les coordenades 2D entre el sistema de coordenades del *viewport* i el sistema de coordenades de l'esfera del *trackball*.

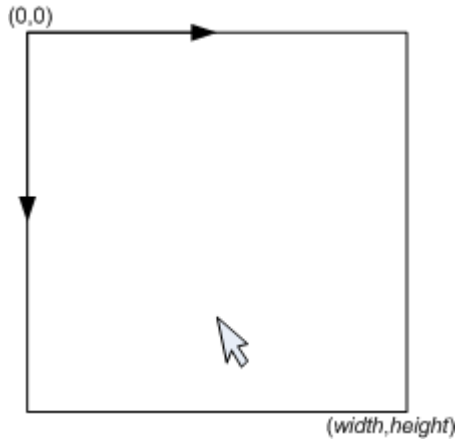


Figura 3.60: Sistema de coordenades del *viewport*

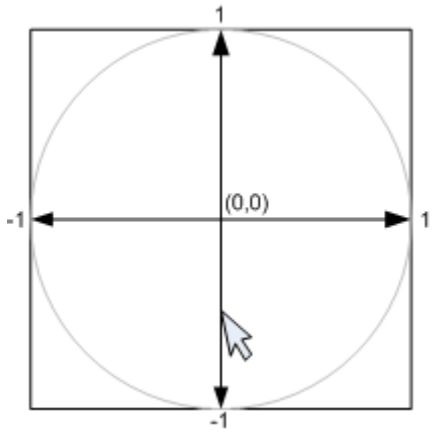


Figura 3.61: Sistema de coordenades de l'esfera

Per a passar d'un sistema de coordenades a l'altre s'han de fer correspondre les coordenades entre [0,width] [0,height] i [-1,1][1,-1]. D'aquesta manera la X i Y de l'esfera seran:

```
// Es passa del viewport a [0,0] - [2,2]
double x = 2 * p.x / width;
double y = 2 * p.y / height;

// Es desplaça el 0,0 al centre
x = x - 1;

// Es gira la Y per a que apunti cap a dalt
y = 1 - y;
```

Una vegada trobades les posicions X i Y a l'esfera es pot trobar z. Donat que l'esfera es de radi 1, es pot saber que $\sqrt{x^2+y^2+z^2}=1$. Si es resol per a z s'obté:

$$z = \sqrt{1-x^2+y^2}$$

$$p = (x,y,z)$$

```
double z2 = 1 - x * x - y * y;
double z = z2 > 0 ? Math.Sqrt(z2) : 0;

Vector3D p = new Vector3D(x, y, z);
p.Normalize();
```

D'aquesta manera ara es tenen les coordenades (x,y,z) del punt a l'esfera sota la posició del cursor.

A cada moviment del dit es vol construir una rotació que mantingui el mateix punt de l'esfera sota el cursor. Això es pot aconseguir emmagatzemant el punt anterior a l'esfera des d'on es va fer l'anterior moviment de cursor i posteriorment construir la rotació que desplaça el punt anterior fins a l'actual punt del cursor.

Per a computar aquesta rotació es necessiten dues coses, l'eix de rotació i l'angle de rotació θ.

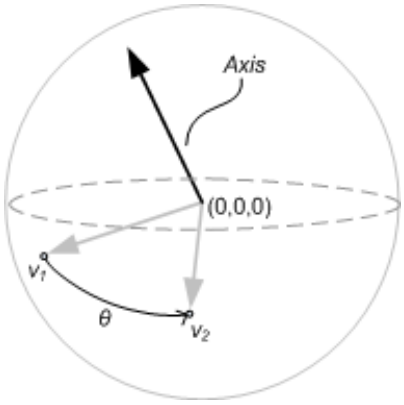


Figura 3.62: S'ha de trobar l'eix de rotació amb angle θ que desplaça v1 fins v2

A causa que l'esfera està centrada a l'origen es poden interpretar aquests punts com a vectors. Trobar així l'eix de gir és trivial ja que només s'ha de fer el producte vectorial d'aquests dos vectors. Una vegada aconseguit, el mòdul d'aquest vector donarà quin és l'angle de gir per fer arribar el punt 1 al punt 2.

```
Vector3D axis = Vector3D.CrossProduct(v1,v2);
```

Una vegada s'ha aconseguit l'eix de rotació només queda crear la matriu de transformació resultant mitjançant la funció oferta per XNA que ja té en compte el mòdul del vector per a crear l'angle de gir:

```
world *= Microsoft.Xna.Framework.
Matrix.CreateFromYawPitchRoll(axis.y, axis.x, axis.z);
```

En cas que el dit es posi a sobre d'una part on el punt no es projecta sobre l'esfera, es posa el valor de Z a 0 i X i Y es deixa la posició tal qual. Això ocasiona que l'eix de rotació quan la posició està fora de la esfera sigui perpendicular al pla de visió de l'usuari.

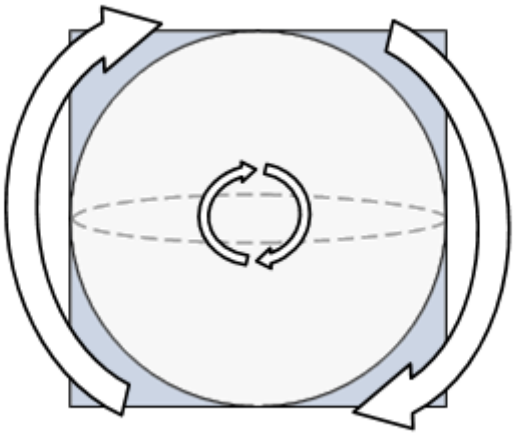


Figura 3.63: A la zona grisa no es projecta cap punt sobre l'esfera, ocasionant un moviment com el que indiquen les fletxes

3.5.4.3 Gestió de gestos i interrupcions

Per a implementar la interacció amb l'usuari s'ha decidit utilitzar les funcions de gestió de gestos que ofereix Windows Phone. Aquestes funcions permeten detectar de forma fàcil i ràpida tota una varietat de gestos que pot realitzar l'usuari, tals com desplaçar un dit sobre la pantalla, desplaçar-ne dos, fer clic, mantenir un dit sobre la pantalla, etc... Aquestes funcions estan pensades per a que el programador no hagi d'entrar a implementar esdeveniments relacionats directament amb la pantalla, simplement disposa d'ells i els tracta una vegada han estat realitzats.

De totes formes, aquestes funcions donen alguns problemes, sobretot a l'hora de detectar gestos similars. La manera que té Windows Phone de detectar gestos és mitjançant interrupcions. Quan es realitza un gest, Windows Phone llença una interrupció i s'executa la funció associada a aquell gest.

El major problema ha vingut donat a l'hora d'haver de detectar una pulsació llarga sobre alguna de les corbes de la funció de transferència per a poder realitzar una selecció precisa d'algun component. Windows Phone sempre abans de detectar una pulsació llarga ha de detectar una pulsació normal. Així, sempre abans de poder fer una selecció precisa amb pulsació llarga, s'executarà la funció associada a la interrupció de la pulsació normal. Aquest problema s'ha resolt havent de guardar en tot moment quina era la funció de transferència prèviament seleccionada, per a poder restablir-la una vegada realitzat una pulsació que havia de ser una pulsació llarga.

3.5.4.4 Lectura de volum de dades

Els models de volum que s'han de visualitzar a l'aplicació estan emmagatzemats en arxius amb format .raw, o també anomenat arxiu binari o arxiu "cru". Aquest fitxer ve inclòs dintre de l'executable de l'aplicació, però per a que l'aplicació pugui llegir el seu contingut, s'ha de compilar de forma especial.

L'aplicació podrà accedir de forma fàcil a tots els arxius que s'hagin compilat en mode *content*. Aquests arxius es pot dir que son arxius que s'han compilat juntament amb l'aplicació i que estaran incorporats dintre del fitxer .xap final de l'aplicació. Però als arxius disponibles dintre de la secció *content* s'ha d'accedir de forma diferent a com es faria a una aplicació dissenyada per a PC.

En ser una aplicació per a telèfons mòbils, la gestió de les rutes dels arxius funciona de forma diferent. Cada aplicació només pot accedir a determinades zones de la memòria del telèfon, això es fa per a que les aplicacions tinguin

molta més seguretat. Així doncs, cada vegada que es vulgui accedir a un arxiu de dintre del telèfon, se li ha de demanar al sistema operatiu que retorni la ruta a aquest arxiu, de forma que queda ocult per al programador. Una vegada obtinguda la ruta de l'arxiu, es llegirà cadascuna de les dades de densitat i es guardaran a un vector de la mida del model de volum per a poder treballar amb elles posteriorment.

L'aplicació implementada només serà capaç de llegir fitxers .raw codificats amb 8 bits d'informació, això vol dir que els models de volum tindran 256 valors possibles de densitat. Així doncs l'eix horitzontal de les funcions de transferència anirà de 0 a 255. També existeixen models de volum que tenen codificada la densistat amb més de 8 bits d'informació, però s'ha decidit no implementar aquesta funcionalitat a causa que tots els models de prova que s'han incorporat a l'aplicació estaven codificats amb 8 bits.

```
StreamResourceInfo resource =
System.Windows.Application.GetResourceStream(new Uri("foot.raw",
UriKind.Relative));

StreamReader sreader = new StreamReader(resource.Stream);

Stream filestream = resource.Stream;

BinaryReader binaryReader = new BinaryReader(filestream);

byte[] dataBuffer = new byte[width * height * depth];

byte[] output = new byte[filestream.Length];

filestream.Read(dataBuffer, 0, dataBuffer.Length);

filestream.Dispose();
```

3.5.4.5 Compilació de l'aplicació

Durant molta part de la implementació de l'aplicació s'ha utilitzat l'emulador de Windows Phone del que disposa Microsoft Visual Studio per a comprovar que l'aplicació funciona correctament. No obstant, una vegada arribat el moment de testejar el funcionament de la gestió de gestos amb més d'un sol dit s'ha hagut de traspassar l'aplicació al dispositiu mòbil físic.

Per a que una aplicació pugui funcionar a un dispositiu mòbil, és necessari registrar el dispositiu mòbil a la web de desenvolupadors de Windows Phone. Aquesta és una operació amb cost econòmic si el desenvolupador no és estudiant. En aquest cas, es va haver de registrar al web utilitzant el comprovador de identitat que proporciona la UPC per a aquestes tasques. Una

vegada registrat i acceptat, és possible descarregar l'aplicació implementada directament al telèfon des de Microsoft Visual Studio per a la seva execució.

El fet que el processador del dispositiu mòbil sigui més lent que el de l'ordinador de sobretaula també ha donat més d'un mal de cap. Windows té la política que qualsevol aplicació que s'executi al telèfon ha de trigar menys de cinc segons en mostrar un menú o una pantalla inicial. Aquesta limitació no està present a l'emulador de Windows Phone, de forma que sempre havia funcionat perfectament si l'aplicació s'executava amb l'emulador. Una vegada es va decidir començar a testejar l'aplicació al dispositiu mòbil em vaig adonar que l'aplicació ni tan sols s'executava. Això era a causa que el processador del telèfon no era capaç d'executar les funcions d'inici tan ràpid com ho feia l'ordinador, ocasionant que tardés mes de cinc segons en mostrar la pantalla inicial. Al començament va ser bastant frustrant perquè no sabia d'on provenia l'error, fins que després de buscar molt a fòrums de desenvolupadors vaig donar amb la solució. Una vegada trobada la solució, es va haver de reorganitzar la forma en què l'aplicació executava les funcions d'inici, de forma que primer mostrés el menú inicial i després realitzés totes aquestes operacions.

3.5.4.6 Execució d'operacions en segon pla

Donat que hi ha operacions molt costoses de dur a terme, s'ha decidit aprofitar l'execució de funcions en segon pla que ofereix Windows Phone. S'ha explicat anteriorment que l'aplicació segueix l'esquema de funcionament de XNA, això és, un bucle que fa Update i Draw. A causa d'aquest esquema es va trobar el problema que aquestes operacions costoses tals com actualitzar les llesques o executar el *raycasting* bloquejaven totalment l'aplicació si s'executaven directament dintre de l'Update, deixant a l'usuari sense feedback per part de l'aplicació. L'usuari podria inclús no saber si l'aplicació està realitzant alguna tasca o quant temps li queda per esperar a que la tasca acabi. Així doncs es va haver de reimplementar la forma en què aquestes funcions eren cridades, de forma que s'executessin en segon pla i l'aplicació no quedés bloquejada. Aquesta execució en segon pla, però, impedeix que la funció executada canviï segons quines variables de la funció i classe principal. D'aquesta forma es va haver de buscar una solució de forma que la funció executant-se en segon pla avisa a la funció principal una vegada ha acabat i aquesta utilitza les dades que la funció ha deixat en variables auxiliars.

A continuació hi ha la part del codi que executa en segon pla el *raycasting* del model de volum.

```
IScheduler scheduler = Scheduler.NewThread;

            taula_bits = new int[width * height];//taula que indica si s'ha
            llençat prèviament un raig per cada píxel

            components_xna.mainpage.render_acabat = false;//indicador de
            finalització per al fil d'execució principal.

            w = new Action(() =>
            {

                Raycast(0, 1, incr, 16, true);//Raycast amb mínima qualitat
                Raycast(0, 1, incr, 8, true);
                Raycast(0, 1, incr, 4, true);
                Raycast(0, 1, incr, 2, true);
                Raycast(0, 1, incr, 1, true);//Raycast amb màxima qualitat

                components_xna.mainpage.render_acabat = true;

            });

            scheduler.Schedule(w, TimeSpan.FromSeconds(0));
```

Aquesta execució d'operacions en segon pla normalment tardava bastant en dur-se a terme, ocasionant que la imatge resultant tardés un temps considerable en mostrar-se al dispositiu. Per evitar-ho es va decidir afegir a la funció de *raycast* un paràmetre amb el que es podia modificar la qualitat desitjada de la imatge resultant. El paràmetre en qüestió (el quart de la trucada a la funció Raycast()) indica al *raycast* cada quants píxels de la imatge resultant ha de llençar un raig al model de volum, d'aquesta forma el nombre de raigs que es llencen es menor i una imatge de baixa qualitat apareix en molt poc temps al dispositiu.

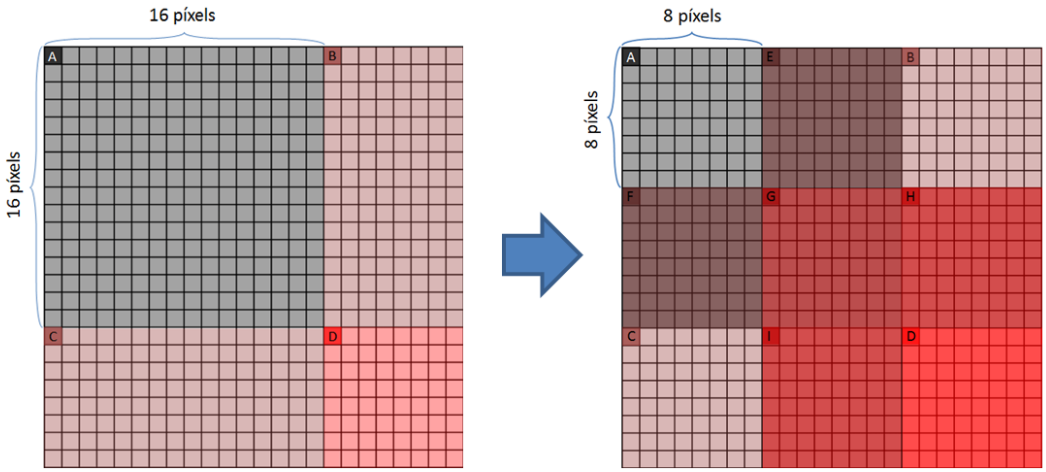


Figura 3.64: Qualitat incremental de la imatge del *raycasting*

Tal com es pot veure a la figura 3.64 i al codi, es comença llençant un raig cada 16 píxels, i es posa aquest color a un quadrat de 16x16 píxels. Cada vegada es dobla el nombre de rajos que es llencen al volum, d'aquesta manera l'usuari pot fer-se una idea de com quedarà la imatge final. Aquesta quantitat de rajos es va augmentant fins que finalment es té un raig llençat per a cadascun dels píxels de la imatge resultant. Cal dir que tal com es pot veure al codi, s'ha creat una taula de bits que emmagatzema si ja s'ha llençat un raig per a cadascun dels píxels, de forma que mai es llença un raig a un píxel que ja s'hagi calculat el seu color prèviament.

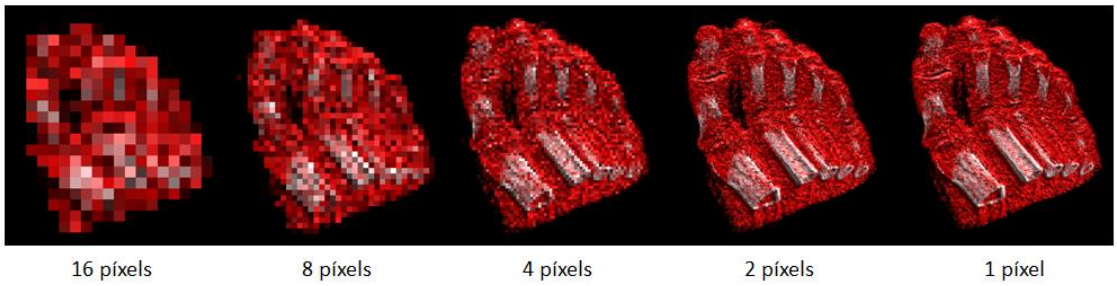


Figura 3.65: Nivells de qualitat de la imatge final

Una vegada finalitzat el càlcul per al nivell màxim de qualitat de la imatge, aquest fil d'execució avisa al fil principal que ha acabat mitjançant l'ús d'una variable que es comprova en cada iteració del Draw, mostrant la imatge final acabada.

3.5.4.7 Qualitat de visualització de l'*slicing*

Després d'haver baixat la resolució dels models amb Voreen tal com s'ha explicat prèviament, es va veure que inclús així la fluïdesa de manipulació dels models mitjançant l'*slicing* no era bona. La manipulació es tornava lenta, baixant molt els fotogrames per segon que mostrava l'aplicació sobretot quan es feia zoom sobre el model per a poder veure amb més detall parts concretes dels models. A causa d'això, es va implementar un paràmetre de qualitat de visualització de l'*slicing*, de forma que l'usuari pogués seleccionar quina qualitat de visualització desitja per a que la fluïdesa a la manipulació sigui millor.

Es va decidir implementar 3 modes de qualitat:

- **Qualitat Alta**.

  En aquest mode es pinten les 128 llesques del model, totes amb una resolució de 128x128 píxels cadascuna. Això vol dir que es veu tot el model amb la màxima resolució.

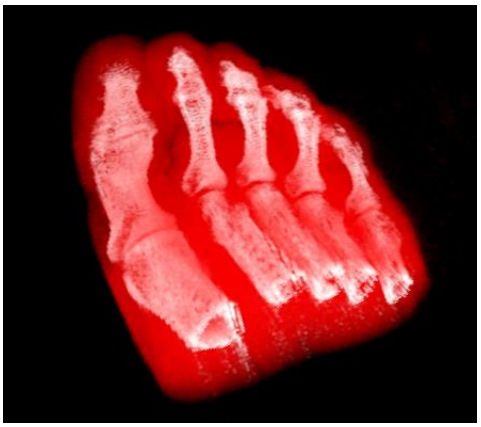


Figura 3.66: Visualització del model peu amb qualitat alta

- **Qualitat Mitja**

  En aquest mode es pinten només la meitat de les llesques, 64 amb la meitat de resolució cadascuna 64x64 píxels. Donat que només s'han de pintar la meitat de llesques, l'aplicació anirà més fluïda perdent només la meitat de resolució disponible del model.

Figura 3.67: Visualització del model peu amb qualitat mitja

- **Qualitat Baixa**

  En aquest mode es pinta només un quart de les llesques, 32 amb un quart de la seva resolució cadascuna 32x32 píxels. La pèrdua de qualitat a la visualització del model d'aquest mode és molt evident, tot i que la fluïdesa de l'aplicació és excel·lent.

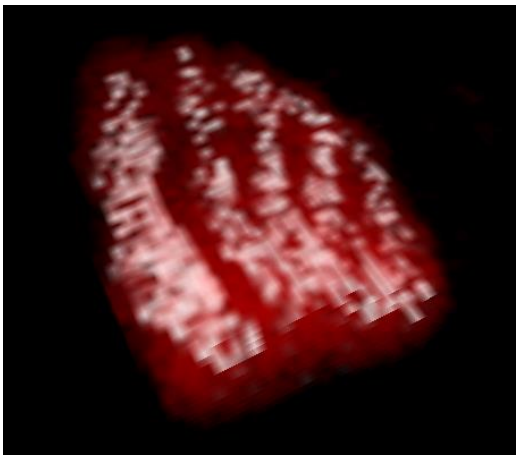


Figura 3.68: Visualització del model peu amb qualitat baixa

3.5.4.8 Utilització de textures per a emmagatzematge de posicions d'entrada i sortida del raig a *raycasting*

Inicialment s'havia implementat el mètode de l'esfera englobant del model de volum per a saber quina era la posició d'entrada del raig que havia de fer el mostreig per a cadascun dels píxels de la imatge resultant. Posteriorment aquest mètode es va canviar pel mètode que més s'acostuma a utilitzar i que també pot ser implementat mitjançant *shaders*.

Aquest mètode utilitza dues textures per a emmagatzemar la posició d'entrada i sortida dels rajos de la capsa englobant del volum i només projecta rajos on s'han rasteritzat fragments. De totes formes, es va poder observar que la imatge resultant obtinguda tenia menys qualitat que l'obtinguda mitjançant el mètode de l'esfera englobant.

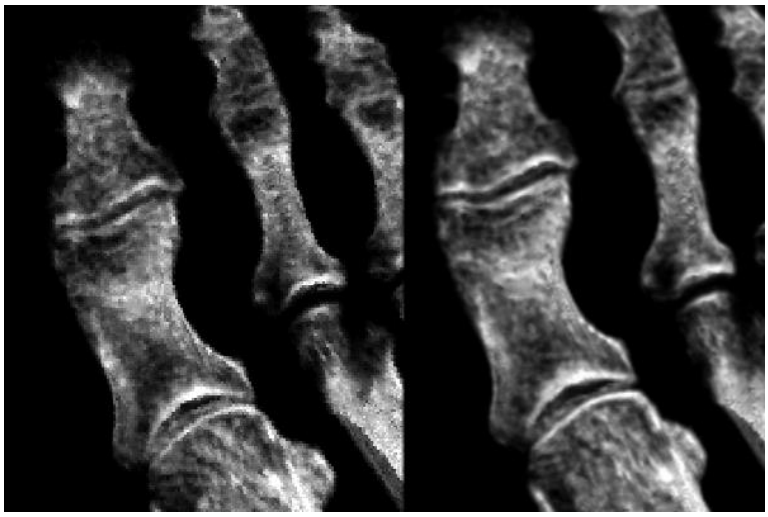


Figura 3.69: Imatge resultant amb el mètode de textura (esquerra) i mètode de l'esfera englobant (dreta)

Després de cercar quin era el problema que ocasionava aquesta diferència d'imatges (l'única part de codi que es canviava era la del posicionament i direcció del raig) es va esbrinar que el problema era a causa de l'anomenat *banding*. El *banding* és un problema de visualització incorrecta dels colors, a causa de la limitació del nombre de bits amb els que es pot representar un color que presenten alguns formats d'imatge. Normalment les imatges tenen una profunditat de color de 24 bits, això és, 8 bits per cadascun dels tres canals vermell verd i blau que la formen. Aquesta profunditat de color normalment és suficient per a representar imatges a tot l'espectre visible. No obstant, en algunes ocasions aquesta profunditat de color no és suficient i es produeixen canvis abruptes en tonalitats de color similars, com per exemple en imatges on es mostren gradients naturals així com postes de sol o cels blaus.

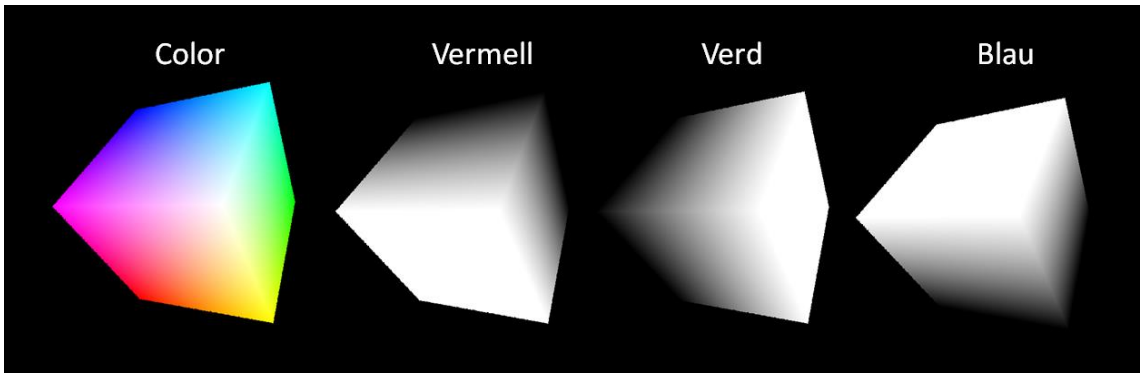


Figura 3.70: Color de cadascun dels vèrtexs de la capsa englobant del volum i representació dels 3 canals que formen la imatge

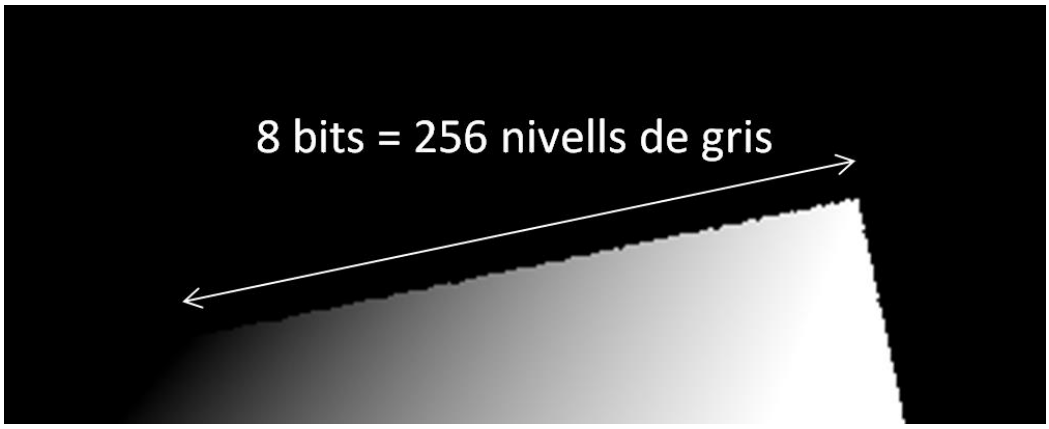


Figura 3.71: Quantitat de nivells de gris que es poden representar amb una imatge d'una profunditat de 8 bits per canal

En aquest cas, s'utilitza la textura per a emmagatzemar una posició a l'espai als tres eixos, no per a ser visualitzada. Això ocasiona que els 8 bits que es disposen per cada canal no siguin suficients per a representar totes les possibles posicions que hi ha entre els vèrtexs de la capsa englobant del volum, ja que només es disposen de 256 "llesques" entre el punt negre del canal (vèrtex a la coordenada 0 de l'eix) i el punt més blanc del mateix canal (vèrtex a la coordenada 1 de l'eix). És a dir, hi han més píxels a la pantalla del dispositiu que llesques possibles entre diferents tonalitats de gris a la textura.

La manera de solucionar aquest problema seria disposar de més bits per canal, ja que es podrien representar més "llesques" entre el negre i el blanc dels canals. A Windows Phone això no és possible ja que per impossibilitats de *hardware* no hi ha manera d'utilitzar més bits per canal. Això ocasiona que a la textura resultant, a causa del *banding*, hi hagin píxels propers que tenen exactament el mateix color, de forma que per a píxels diferents el raig entra al volum per la mateixa posició, ocasionant petites desviacions als rajos i ocasionant la pèrdua de qualitat abans esmentada.

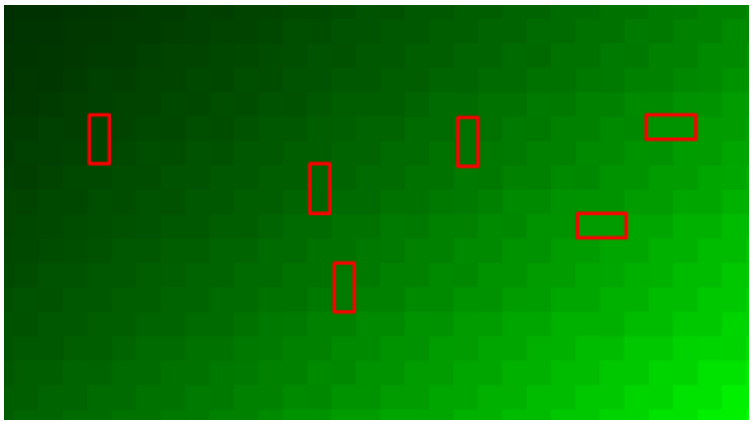


Figura 3.72: Detall de la textura on es pot apreciar *banding* i píxels amb exactament el mateix color

La solució que s'ha trobat per a arreglar aquest problema ha sigut arribar a una solució intermèdia entre els dos mètodes, de forma que només es calcula la posició d'entrada a l'esfera englobant als píxels on s'han rasteritzat fragments. D'aquesta forma només es calcula la posició de col·lisió amb l'esfera englobant als píxels que contenen el model, i no a tots i cadascuns dels píxels que formen la imatge resultant tal com es feia només amb el mètode de l'esfera englobant.

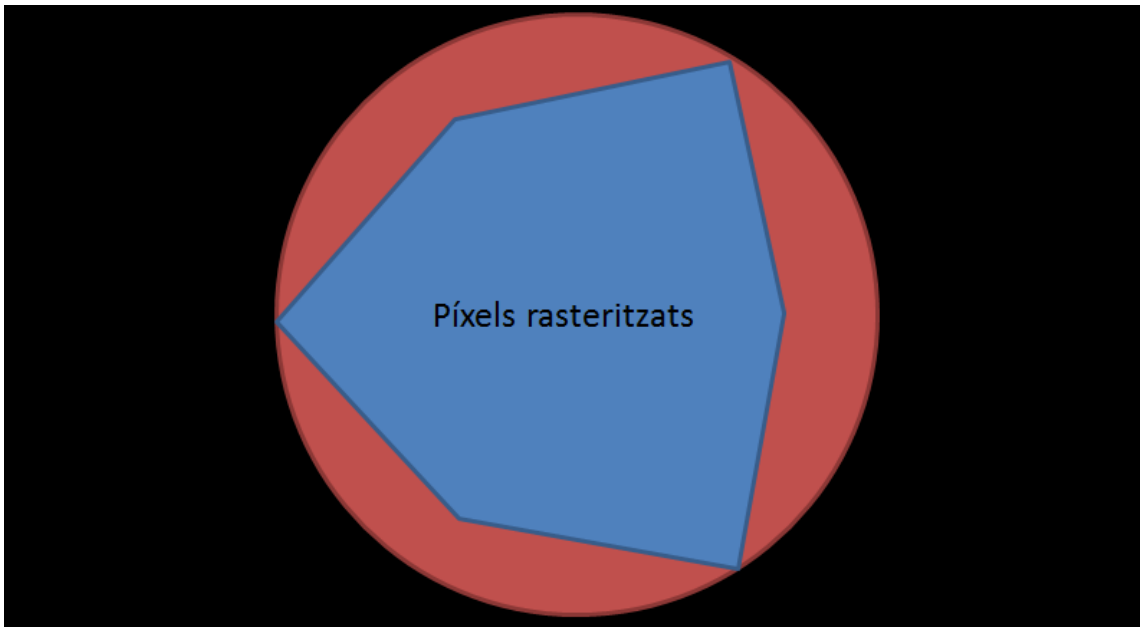


Figura 3.73: Els rajos que travessaran el volum només es llençaran a la zona blava on s'han rasteritzat píxels

3.5.4.9 Codi de les principals parts del *raycasting*

Donada la importància que té el mètode de *raycasting* en aquesta aplicació i donada la seva complexitat, s'ha decidit incorporar en aquesta secció les principals parts del codi implementat per a dur-lo a terme.

El primer pas és saber la direcció en que s'hauran de llençar els rajos per a cadascun dels píxels de la imatge resultant. El que es fa es calcular les posicions de 3 vèrtexs d'aquest pla imaginari situat en la direcció de visió de l'usuari, de forma que ocupi tota la pantalla del dispositiu. Una vegada es disposen d'aquests tres punts es construeix una base a l'espai tridimensional tant per a l'eix X com per a l'eix Y d'aquest pla.

```
//S'aplica la translació a l'escena
world = world *
Microsoft.Xna.Framework.Matrix.CreateTranslation(world.Translation);

//Es calcula la matriu inversa de la matriu de transformació de l'escena per a
poder aplicar-la als diferents elements de l'escena
Microsoft.Xna.Framework.Matrix world_invert =
Microsoft.Xna.Framework.Matrix.Invert(world);

//es calcula la posició dels 3 punts que defineixen el pla sobre el que es
projectaran el raigs al volum, el de dalt a l'esquerra, el de baix a l'esquerra i
el de dalt a la dreta
transf_up_left_proj_plane = Vector3.Transform(up_left_corner_proj_plane,
                                              world_invert);
transf_down_left_proj_plane = Vector3.Transform(down_left_corner_proj_plane,
                                               world_invert);
transf_up_right_proj_plane = Vector3.Transform(up_right_corner_proj_plane,
                                               world_invert);

//es calcula la posició de l'observador i de la llum a l'escena
pos_obs_trans = Vector3.Transform(pos_obs, world_invert);
pos_llum_trans = Vector3.Transform(pos_llum, world_invert);


//es calcula una base per a poder desplaçar-se pel pla de projecció
nova_base_x = (transf_up_right_proj_plane - transf_up_left_proj_plane) / width;
nova_base_y = (transf_down_left_proj_plane - transf_up_left_proj_plane) / height;
```

Una vegada s'ha obtingut aquesta base per al pla de visió, és molt fàcil construir el bucle que iterarà per tots aquests píxels de la imatge resultant. Quedant el bucle principal de la següent manera:

```
for (int j = 0; (j < height) && !manip_started; j += div)
     {
      for (int i = 0; (i < width) && !manip_started; i += div)
          {
           punt_calc_vect = transf_up_left_proj_plane + nova_base_x * i +
                            nova_base_y * j;
```

Aquesta iteració només llançarà rajos als píxels on s'hagin rasteritzat fragments. En cas positiu, es calcula quina és la distància fins al punt de col·lisió de l'esfera i es situa en aquella posició el raig per a que comenci a desplaçar-se a l'interior del volum.

```
  if (colors_depth_raycast_entrada[c] !=
              Microsoft.Xna.Framework.Color.Black){// si s'ha rasteritzat pixels

             Ray V = Ray.Ray_from_2_points(pos_obs_trans, punt_calc_vect);

             Sphere_hit(V, out dist);

              Ray startpos = new Ray(pos_obs_trans.X + V.nX * dist,
                                     pos_obs_trans.Y + V.nY * dist,
                                     pos_obs_trans.Z + V.nZ * dist);
```

Una vegada el raig es troba dintre del volum s'efectuen els càlculs necessaris per a obtenir la densitat d'aquell punt de l'espai i es converteix a un color per a afegir-lo al color emmagatzemat fins llavors a aquell píxel.

```
int opac_255 = trilinear_filter(startpos);

double comp_difusa = 0;
float comp_especular = 0;

int r_255;
int g_255;
int bl_255;
byte a;

Microsoft.Xna.Framework.Color color_def;

float opac_01;

components_xna.evalua_opac(opac_255, out color_def, out opac_01);//opac_01 entre
0 i 1 i color ja multiplicat per opac

opac_255 = (int)(opac_01*255); // opac_def ve entre 0 y 1

r_255 = (color_def.R);
g_255 = (color_def.G);
bl_255 = (color_def.B);

Vector3 mostra = new Vector3();

if (iluminacio_habilitada)
   {

   grad_trilinear_filter(startpos, out mostra);

   comp_difusa = component_difusa(mostra, startpos); // retorna entre -1 i 1
   comp_difusa = (comp_difusa + 1) * 0.5;//es posa entre 0 i 1

   if (especular_habilitada)
      {

      comp_especular = component_especular(mostra, startpos, V);// retorna entre
0 i 1

  }
}
```

```
else{

    comp_difusa = 1;

    }

byte pixel_r;
byte pixel_g;
byte pixel_b;

convert_color(out pixel_r, out pixel_g, out pixel_b, out a, pixel);//passo els
colors de pixel a rgbaux

int raux2 = pixel_r;
int gaux2 = pixel_g;
int baux2 = pixel_b;

double Id = comp_difusa * opac_01 ;// *0.003921568627450980; // opac entre 0 i
255. Es divideix entre 255 per a tindre el rang entre 0 i 1
double Is = kspecular * comp_especular * opac_01 ;// *0.003921568627450980;
//1/255 = 0.003921etc

int Rd = (int)(Id * r_255 );
int Gd = (int)(Id * g_255 );
int Bd = (int)(Id * bl_255 );


int Is_tot = (int)(Is * 255);//color de la especular blanc pur

raux2 = (raux2 + Rd + Is_tot);
gaux2 = (gaux2 + Gd + Is_tot);
baux2 = (baux2 + Bd + Is_tot);

acum_opac += opac_255;

if (acum_opac > 255) earlytermination = true;


if (raux2 > 255) { raux2 = 255;}
if (gaux2 > 255) { gaux2 = 255;}
if (baux2 > 255) { baux2 = 255;}


pixel = convert_color(raux2, gaux2, baux2);
```

Després d'això es fa avançar el raig per l'interior del volum i es surt del bucle quan el raig surt fora de la capsa englobant del volum. Una vegada fora, el valor de color del píxel es guarda a la textura que finalment es mostrarà a l'usuari.

4 Concordança de resultats i objectius.

4.1 Comparació aplicació planificada i aplicació finalitzada

En aquest apartat s'analitzaran els resultats esperats i obtinguts de les diferents funcionalitats de l'aplicació implementada.

Primerament, s'ha de destacar que els resultats esperats en quant a la visualització dels models ha sigut l'esperada. El resultat principal que s'esperava tenir una aplicació que fos capaç de visualitzar un model de volum. Aquest resultat s'ha dut a terme, ja que s'han implementat dos mètodes capaços d'oferir una visualització, el *slicing* i el *raycasting*. De totes formes amb l'*slicing* s'esperava que la manipulació sigués fluida en tot moment, però s'ha pogut comprovar que si les llesques ocupen molta part de la pantalla la manipulació del model no és tot el fluida que hauria de ser. Això és a causa del nombre de llesques que s'han de pintar a pantalla, de forma que si el nombre de llesques a pintar es disminueix mitjançant la disminució de qualitat que ofereix la aplicació, aquesta manipulació es torna més fluida a costa de baixar la qualitat de visualització.

En quant al *raycasting*, el resultat obtingut ha sigut també l'esperat. Ja es sabia que el *raycasting* seria lent a causa de la impossibilitat d'utilitzar la GPU. De totes formes la incorporació del mètode de visualització del *raycasting* amb refinament de píxels va significar un gran avenç ja que gracies a aquest mètode es possible que l'usuari es faci una idea del resultat final havent d'esperar no molt temps. Una vegada l'usuari ha vist una aproximació del resultat final, pot decidir si esperar o no fins a la visualització totalment refinada del model.

L'edició de corbes de la funció de transferència ha estat tot un èxit. Editar funcions de transferència és fàcil i ràpid, tal com s'havia pensat inicialment. Tant la selecció general com la selecció precisa funciona perfectament i les diferents corbes s'autoseleccionen correctament mentre el punter assenyala les diferents parts. No obstant, a causa de les característiques del sistema operatiu i com aquest gestiona les interrupcions, cal una mica de pràctica per a realitzar aquestes operacions i de vegades es poden produir errors. Si es desitja crear un punt nou a una corba s'ha de fer doble clic sobre una aresta, però de vegades aquest doble clic no es fa bé sobre l'aresta i el programa passa a mode de manipulació del model de volum. Aquest problema podria solucionar-se fent l'amplada de les arestes major, però si es fes les arestes ocuparien molta part de pantalla i podrien fins i tot molestar a l'usuari. En quant a la manipulació precisa de elements d'una corba (amb dos dits sobre la pantalla) s'ha de dir que tampoc funciona tot el bé que hauria. Si quan es posen els dos dits sobre la pantalla no es fa absolutament a l'hora, Windows Phone no detecta bé que s'han posat els dos dits, detecta només el primer dit que s'ha situat. Això ocasiona que l'aplicació seleccioni la corba que té en aquell

moment a sota d'algun dels dos dits, el que normalment faria si es posés només un dit a sobre la pantalla.

La resta de funcionalitats de l'aplicació s'han implementat correctament i sense moltes complicacions. Quan es va implementar la funcionalitat de poder emmagatzemar les corbes de la funció de transferència es va veure que era molt fàcil guardar també una imatge de les corbes de forma que sigués molt més fàcil identificar-les a l'hora de carregar-les. Es podria dir que el resultat final d'aquesta funcionalitat ha estat inclús millor del que s'havia planejat.

Finalment s'han incorporat 5 models de prova diferents per a que l'usuari pugui experimentar amb ells. Inicialment s'havia pensat en posar-hi un parell, però gracies a la rapidesa i facilitat amb les que Voreen era capaç de baixar de resolució els models de volum es va decidir incorporar-hi més.

En quant a les opcions que s'han incorporat a l'aplicació també s'han obtingut els resultats esperats, ja que realitzen correctament la seva funció. El fet que es puguin canviar els paràmetres del *raycasting*, així com canviar la qualitat de *slicing* ha permès que es pugui treballar més ràpid amb l'aplicació i s'hagin pogut dur a terme moltes més proves.

Figura 4.1: Captura del menú principal de l'aplicació

4.2 Anàlisi dels resultats obtinguts

En aquesta secció es mostren els resultats de la visualització dels diferents models de volum amb diferents tècniques i amb diferents qualitats.

Es recorda que els 5 models incorporats a l'aplicació se'ls ha baixat la seva resolució original fins als 128x128x128 vòxels ja que sinó aquests models no entraven a la memòria del dispositiu i l'aplicació no s'executava. Aquesta resolució ofereix una bona relació qualitat-espai a disc, fent que l'aplicació pugui emmagatzemar els models a memòria sense cap problema.

A la taula 1 es poden veure quants fotogrames per segon és capaç de mostrar l'aplicació segons quin nivell de zoom s'està fent sobre el model i segons quin nivell de qualitat s'ha triat per visualitzar el model en temps real amb *slicing*. Cal dir que no s'han apreciat canvis significatius segons quin model s'hagi triat per visualitzar.

Nivell de zoom	**Qualitat Alta**	**Qualitat Mitja**	**Qualitat Baixa**
Nivell mínim de zoom	13 fps	23.2 fps	29.5 fps
Nivell inicial de zoom	5.5 fps	8 fps	11.5 fps
Nivell màxim de zoom	1.5 fps	2.7 fps	5.3 fps

Taula 1

Impacte del zoom sobre la fluïdesa amb visualització de les diferents qualitats d'*slicing*

Estudiant aquesta taula es pot veure com l'aplicació no és capaç de visualitzar amb fluïdesa els models amb qualitat alta ni amb qualitat mitja. Es pot dir que el nivell inicial de zoom és un nivell de zoom molt bo per a visualitzar els models, així que aquest nivell seria el recomanable quan l'usuari estigui visualitzant el model i vulgui una manipulació fluida.

A la taula següent es poden veure els diferents temps que ha tardat l'aplicació en processar tot el model passant tots els seus vòxels per les corbes de la funció de transferència i generant les llesques necessàries per a la visualització amb *slicing*. S'han agafat els temps fins a 3 corbes, ja que normalment no es tindran més de tres.

Nombre de corbes	Qualitat Alta	Qualitat Mitja	Qualitat Baixa
1 corba	22 segons	3 segons	< 1 segon
2 corbes	27 segons	4 segons	< 1 segon
3 corbes	30 segons	5 segons	1 segon

Taula 2

Temps en segons de processat del model per a realitzar les llesques de l'*slicing* segons el nombre de corbes de FT

Tal com es pot veure a la taula hi ha un augment molt significatiu del temps necessari per a processar el model segons la qualitat escollida. Una alta qualitat multiplica quasi per 8 el temps necessari per a processar el model. D'altra banda, triar una qualitat baixa no té cap impacte sobre el processament del volum. D'aquí es pot arribar a la conclusió que la qualitat Mitja és la més adient per a realitzar molts canvis a les corbes de la funció de transferència i haver d'esperar poc fins que aquests canvis es visualitzen al model, ja que ofereix una bona relació entre qualitat de visualització i temps d'espera de l'usuari. Amb una baixa qualitat l'usuari pràcticament no s'ha d'esperar gens però la qualitat de visualització és molt dolenta tal com es pot apreciar a la figura 3.66.

A la taula següent es mostren el temps emprats per l'aplicació en mostrar la imatge complerta i amb el màxim grau de refinament mitjançant el mètode *raycasting* amb els diferents models d'il·luminació implementats segons el nivell de zoom sobre el model. S'ha de tenir en compte que la corba utilitzada en totes les proves d'aquesta taula ha sigut la inicial que ve per defecte quan s'inicia l'aplicació. S'ha triat aquesta corba ja que permet visualitzar amb menys opacitat els nivells de densitat menors i amb més opacitat les densitats majors. Segons es pot veure a les taules següents es podria dir que aquesta corba és un dels casos pitjors, ja que els casos on hi ha molts píxels on no s'arriba a la opacitat màxima per a poder realitzar la finalització primerenca de raigs són on la visualització final triga més.

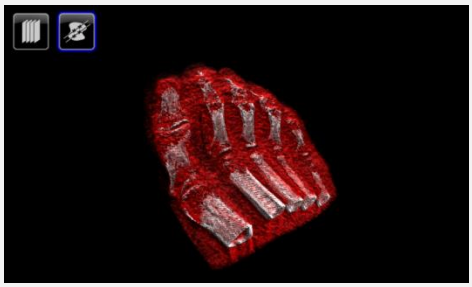
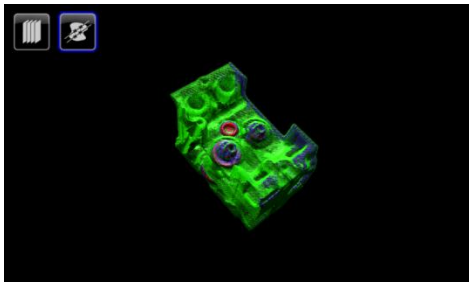
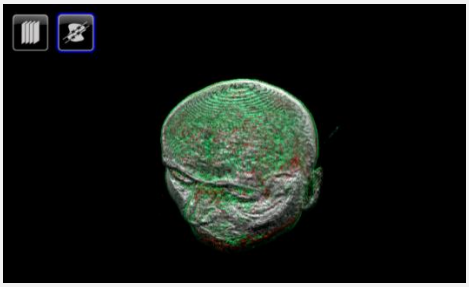
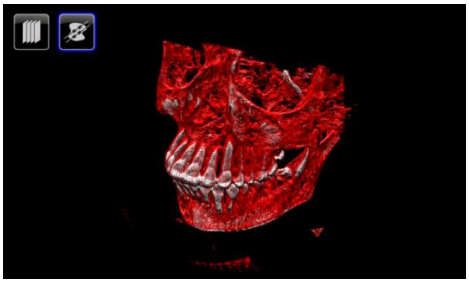
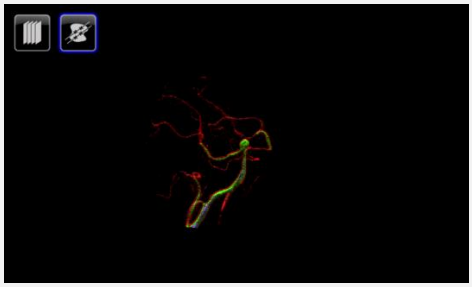
Nivell de zoom	***Raycasting* + Lambert + Phong**	***Raycasting* + Lambert**	***Raycasting* sense il·luminació**
Nivell mínim de zoom	1 min 20 segons	1 min	35 segons
Nivell inicial de zoom	6 min 20 segons	3 min 30 segons	2 min 10 segons
Nivell màxim de zoom	15 min 20 segons	13 min 10 segons	7 min 30 segons

Taula 3

Temps emprat per l'aplicació en visualitzar els models amb *raycasting* segons el nivell de zoom i mètodes d'il·luminació

Segons s'aprecia a la taula el fet de visualitzar amb *raycasting* els models amb un alt nivell de zoom pot arribar inclús a ser el doble o el triple de visualitzar-los amb un nivell de zoom normal com és l'inicial. Es pot apreciar com tant el model de Lambert com de Phong suposen un gran impacte sobre el temps, ja que les imatges visualitzades tarden prop del doble més que les mateixes sense il·luminació. D'aquesta taula es pot extreure que si l'usuari necessita veure amb *raycasting* una zona sobre la que ha de fer molt de zoom haurà d'esperar-se una bona estona per a poder visualitzar-la.

A la taula següent es mostren els temps emprats per l'aplicació per a visualitzar cadascun dels models que incorpora l'aplicació amb *raycasting* amb Lambert i Phong. Aquests models es visualitzen amb el nombre de corbes habitual i necessari per cadascun d'ells. Es pot dir que aquests temps són més ajustats al que l'usuari haurà d'esperar normalment, ja que no s'han visualitzat amb la corba inicial que ja s'ha vist que era un dels casos pitjors. Es pot veure que aquests temps són més raonables, havent d'esperar sobre els 5 minuts per a tenir la imatge amb refinament màxim.

Model	Corbes	temps	Imatge Final *Raycasting*
Peu	2	4 min 30 sec	
Motor	3	6 min 5 sec	
Cap humà	3	4 min 30 sec	
Calavera	2	4 min 45 sec	
Aneurisma	3	7 min	

Taula 4

Temps emprat per l'aplicació i imatge resultant dels diferents models de prova

4.3 Possibles vies de millora

Durant la implementació d'aquesta aplicació han anat apareixent millores que en el futur serien possibles vies per a que aquesta aplicació sigués molt millor.

- Es podria canviar el mètode de visualització de llesques orientades segons els eixos cartesians pel mètode de llesques orientades a l'eix de visió de l'espectador. Aquest mètode eliminaria el *popping* molest ocasionat pel canvi de conjunt de llesques. No obstant, aquest mètode necessita que el dispositiu permeti manipular textures 3D, així que s'hauria d'esperar fins a que futures actualitzacions incloguin aquesta funcionalitat.

- Un dels majors inconvenients que s'han tingut quan s'ha implementat aquesta aplicació ha sigut la impossibilitat d'utilitzar *shaders* propis. Si en el futur Windows Phone permet això, es podria aprofitar tota la potència de la GPU per a implementar el mètode de *raycasting* que tan lent s'executa en aquesta aplicació.

- En el futur es podria implementar algun altre mètode d'il·luminació més complex com per exemple el mètode per a obtenir ombres o per aconseguir la difusió de la llum dintre d'un cos.

5 Anàlisi econòmica global i comparada amb alternatives.

5.1 Estudi econòmic de la realització de l'aplicació

La implementació d'aquesta aplicació ha durat pràcticament un any ja que a causa de problemes amb el treball que tinc no he pogut dedicar tot el temps que m'hagués agradat, per això s'ha allargat més d'un quadrimestre.

Per tal d'implementar aquesta aplicació s'han hagut de passar per tota una sèrie de fases.

- Fase de documentació inicial i disseny. Aquesta fase va durar prop de tres mesos, ja que es va haver de buscar informació sobre com i quins serien els mètodes que s'haurien d'implementar, així com començar a dissenyar i implementar versions bàsiques de l'*slicing* per esbrinar com funcionen XNA i Silverlight. També es va buscar com s'havia de fer per a registrar el telèfon físic per a poder instal·lar-hi aplicacions.
- Fase d'implementació d'aplicació hibrida. Una vegada complerta la familiarització amb XNA i Silverlight, es va decidir començar a implementar l'aplicació final. Aquesta fase va durar 6 mesos, ja que és la que té la part principal de l'aplicació. Aquí es on es van implementar el mètode de *slicing* i tots els menús. Una vegada acabada aquesta fase es disposava d'una aplicació totalment funcional però que no incorporava la visualització dels models mitjançant *raycasting*.
- Fase d'implementació del mètode de *raycasting.* Una vegada es tenia l'aplicació funcionant de forma pràcticament completa, es va decidir implementar el mètode de *raycasting*. Aquesta fase va durar 3 mesos ja que es va haver d'implementar i incorporar a l'aplicació aquest nou mètode de visualització.
- Fase de testing. Una vegada finalitzada la implementació de tota l'aplicació, es va procedir a testejar-la per a descobrir possibles mal funcionaments i aconseguir que l'aplicació sigui el més usable possible.
- Fase de documentació del projecte. La documentació del projecte s'ha dut a terme a mesura que s'implementaven i testejaven les diferents parts. Es va començar a documentar sobre la meitat de la fase d'implementació de l'aplicació hibrida, i ha durat fins a la

finalització del projecte. Aquesta fase l'hauria de dur a terme un programador que fes també un paper d'analista, ja que ha de saber com funcionen i com estan implementades totes les parts de l'aplicació per a poder documentar-les correctament.

La realització d'aquestes fases es podrien dur a terme per un dissenyador, un programador i un tester. El cost per hora d'aquests treballadors serien de 45€ l'hora per a un dissenyador, 30€ l'hora per a un programador i 20€ l'hora per al tester. Per a realitzar la comptabilitat d'hores s'ha fet una mitja de 3 hores treballades per dia durant tot el període que han durat totes aquestes fases.

Així el cost total dels recursos humans per a dur a terme aquesta aplicació quedaria de la següent forma:

Fase	**Dut a terme**	**Hores**	**Cost**
Documentació inicial i disseny	Dissenyador	180	8100€
Implementació aplicació final	Programador	360	10800€
Implementació *Raycasting*	Programador	180	5400€
Fase de testing	Tester	60	1200€
Documentació del projecte	Programador/Analista	100	3000€
Total		**880**	**28500€**

5.2 Comparació de l'aplicació finalitzada amb altres aplicacions

Una vegada finalitzada l'aplicació es pot comparar amb les altres aplicacions i veure en quines coses és millor i en quines coses és pitjor. Es compararan amb altres aplicacions les tres principals funcionalitats que se li va demanar al començament del projecte.

- Visualització i manipulació en temps real

  En quant a visualització mentre es manipula el model es pot dir que l'aplicació implementada està al nivell de totes les altres. Potser l'aplicació implementada es torna més lenta a l'hora de visualitzar els models en qualitat alta, però això es pot solucionar baixant el nivell de qualitat a mitja o baixa. La manipulació és suau i funciona correctament igual que la resta d'aplicacions, així que en aquest apartat l'aplicació implementada està al mateix nivell.

- Visualització realista

  En quant a la visualització realista es podria dir que aquí es on l'aplicació implementada hauria de millorar-se. En comparació amb Imagevis3D o ResolutionMD, aquesta és una aplicació molt més lenta, tenint en compte que la qualitat de la imatge finalitzada no és millor que la que ofereixen aquestes dues.

  No obstant, l'aplicació implementada és la única que permet modificar les propietats del *raycasting*, tals com si es vol il·luminació o si es vol especularitat.

- Edició de corbes de la funció de transferència

  En aquest aspecte es podria dir que l'aplicació implementada permet realitzar accions que la majoria d'altres aplicacions no permet. L'aplicació del Marcos és l'única que permet editar les corbes de la funció de transferència d'una forma fàcil precisa i ràpida. Per això, l'aplicació implementada s'ha basat en la forma d'editar corbes que tenia l'aplicació d'en Marcos. Imagevis3D té edició de corbes però no tan precisa com l'aplicació que s'ha implementat. La resta d'aplicacions no disposa d'edició de corbes, de forma que l'aplicació implementada inclou aquesta funcionalitat que les altres no posseeixen. A més, s'ha de dir que cap de les altres aplicacions té la funcionalitat d'emmagatzemar corbes de funcions de transferència.