Universitat de Girona
Escola Politècnica Superior

Projecte/Treball Final de Carrera

Estudi: Eng. Tèc. Informàtica de Gestió

Títol:

Visualització de volums usant Miralls Màgics

Document: Memòria

Alumne: Marc Ruiz Altisent

Director/Tutor: Imma Boada Oliveras
Departament: Informàtica i Matemàtica Aplicada
Àrea: Llenguatges i Sistemes Informàtics

Convocatòria: Setembre 2004

Índex

1 Introducció

La Visualització Científica és una àrea de la Informàtica Gràfica que té com a objectiu la representació i interpretació de dades científiques obtingudes a través de simulacions i/o dispositius de captació, és a dir, pretén fer comprensibles conjunts de dades a partir d'interpretacions gràfiques. Els camps d'aplicació de la Visualització Científica són molts i diversos: geologia, medicina, química, meteorologia, ciències de la terra, etc. Aquesta diversitat d'àrees fa que els algorismes i les tècniques que es desenvolupin depenguin directament de les característiques de les dades de les quals inicialment es disposa. Podem dir per tant que la majoria de tècniques de visualització científica s'han desenvolupat per resoldre problemes concrets.

Una de les línies d'investigació del grup d'Informàtica Gràfica de la Universitat de Girona és l'aplicació de la Visualització Científica en medicina. Per aquesta raó s'ha establert un conveni de col·laboració entre el Grup d'Informàtica Gràfica i el grup de neuro-radiologia de l'Institut de Diagnòstic per la Imatge de l'Hospital Universitari Dr. Josep Trueta de Girona. En el marc d'aquest conveni la primera fita que s'ha marcat és la de desenvolupar una plataforma que incorpori les tècniques bàsiques de visualització científica. Aquesta plataforma pretén complementar la visualització bidimensional (2D) majoritària en l'entorn mèdic amb una visualització tridimensional (3D) que permeti inspeccionar la informació captada del pacient de forma més eficient i facilitant-ne el seu diagnòstic.

El projecte que es presenta en aquest document forma part del desenvolupament d'aquesta plataforma de visualització d'imatges mèdiques. Abans d'entrar en detall en el treball realitzat en el projecte començarem donant una descripció de la plataforma de visualització per tal de situar de forma exacta l'abast de la feina feta.

1.1 La Plataforma de Visualització d'Imatges Mèdiques

Desenvolupar una plataforma de visualització implica desenvolupar un conjunt de tècniques que permetin obtenir a partir d'un conjunt de dades obtingudes del pacient una imatge en el computador que ens permeti i ens ajudi a interpretar-les. Independentment de les tècniques que s'implementin hi ha sempre una part que es comuna a totes elles: l'adquisició de dades del pacient i la segona la definició del model que s'usarà per representar-les.

1.1.1 Adquisició de Dades

La captació de dades del pacient es realitza mitjançant algun dispositiu físic que permet mesurar un o més paràmetres determinats en diferents punts del cos del pacient. Les dades obtingudes dependran de les prestacions de l'aparell: la seva resolució, fiabilitat, precisió, etc.

Els dispositius de captació més utilitzats són:

> **Computer Tomography (CT):** A cada punt de l'espai s'hi mesura el coeficient d'atenuació del raig X. Aquest dispositiu s'utilitza principalment per la detecció de sòlids d'alta densitat, com ara els ossos.

Magnetic Resonance Imaging (MRI): A cada punt es mesura la densitat i la mobilitat dels protons detectant l'energia que emeten en desplaçar-se. Aquest dispositiu permet la detecció de teixits.

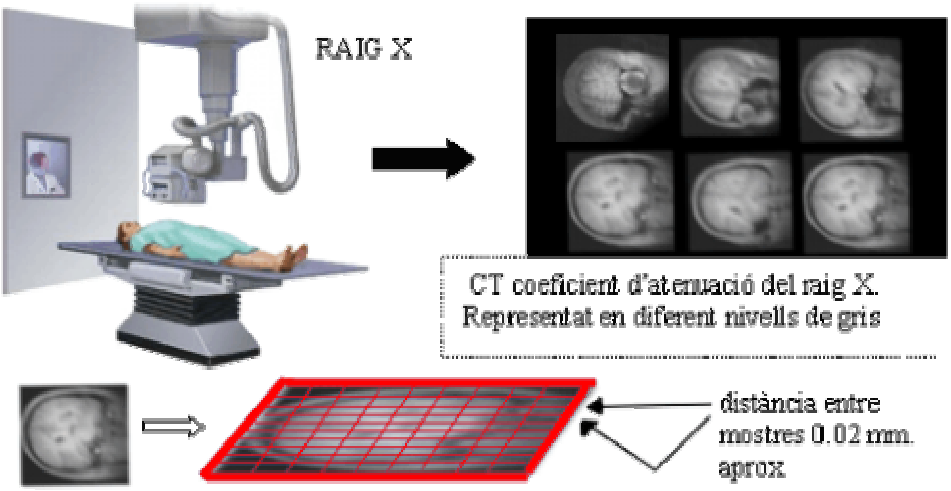
Positron Emission Tomography (PET): Mesura el número de raigs gamma que s'emeten en cada punt. Té una resolució molt baixa i s'utilitza per detectar la dispersió de fluids, molt apropiat per l'estudi de funcions metabòliques.

Single Photon Emission Tomography (SPECT): Igual que l'anterior capta el número de raigs gamma que s'emeten en cada punt. S'aplica en l'estudi de fluids, principalment el flux de sang.

Ultrasound Image (UI): Detecta les reflexions de les ones acústiques permetent calcular profunditats. Una de les seves aplicacions és la visualització de fetus.

Magnetic Source Image (MSI) / Magneto EncephaloGraphy (MEG): Detecta els canvis magnètics que es produeixen en el cervell humà.

Les dades que s'obtenen del pacient a traves d'aquests dispositius es mantenen en un fitxer amb un format propi del fabricant de l'aparell o bé algun dels estàndards predefinits existents. Tot i que hi ha diversos estàndards sembla que el DICOM és el que actualment està prenent més força. Malgrat la diversitat de formats que existeixen una de les característiques que tenen en comú el tipus de dades amb els quals nosaltres treballem és que sempre segueixen una distribució espacial regular i que sempre ens venen distribuïdes sobre plans.



En el cas que les dades del pacient s'obtinguin a través d'un CT s'obtenen un conjunt de mostres distribuïdes sobre plans. Cada pla representa el coeficient d'atenuació del raig X en un determinat punt del cos. La distància entre mostres és de mil·límetres.

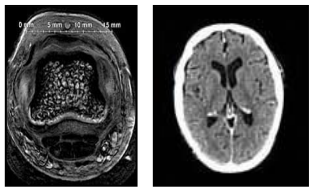
Figura 1.1. Exemple d'obtenció de dades a través d'un CT.

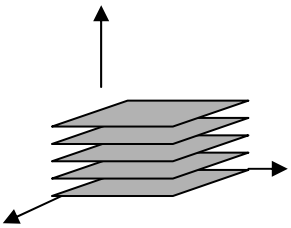
1.1.2 Definició del Model de Voxels

Si volem tractar les dades del pacient al computador cal definir un esquema de representació que ens permeti accedir fàcilment a aquestes dades.

El model més usat en el camp de la medicina és el model de voxels proposat per Kaufman. Aquest model subdivideix l'espai en un conjunt de cubs o paral·lelepípedes de les mateixes dimensions, seguint una malla regular. Cadascun d'aquests cubs, o paral·lelepípedes, rep el nom de voxel. Sobre cada voxel s'hi representaran les dades que s'han obtingut del pacient. En l'esquema següent podem veure una descripció gràfica de com es realitza aquest procés:

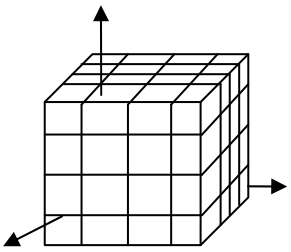
Partirem de les dades que s'han obtingut del pacient. Observem que segueixen una distribució regular sobre un pla cadascun dels quals es pot interpretar com un tall de la part del cos que s'està examinant.





Els diferents plans o talls s'apilaran un sobre l'altre intentant reconstruir de nou la part 3D del cos.

Finalment es definirà una malla regular 3D formada per un conjunt de cel·les sobre els vèrtexs de les quals es representarà la informació de cada tall.



El model de voxels es caracteritza per:

- **Ser un model aproximat:** Només emmagatzema les propietats d'un número determinat de punts del volum.
- **Requerir una gran quantitat de memòria:** L'espai ocupat pel model té ordre $O(n^3)$ respecte a la resolució (n).

A partir de les dades captades i distribuïdes sobre una malla regular es poden considerar dos models de voxels: el model tipus cel·la i el model tipus voxel. Quan les mostres estan distribuïdes als vèrtexs dels paral·lelepípedes que formen el model s'anomena model de cel·la. Si pel contrari, es considera que l'interior del paral·lelepípede és homogeni és quan es parla de model de voxels.

Per tal de poder desenvolupar el meu projecte i integrar-lo en aquesta plataforma se'm va facilitar una interfície inicial que em permetia obrir un model de voxels. Evidentment el model de voxels s'havia creat amb les dades que s'havien obtingut d'un determinat pacient. A les imatges de les figures Figura 1.2, Figura 1.3, Figura 1.4 i Figura 1.5 es pot veure aquesta interfície.

Pel desenvolupament de la plataforma només s'ha usat software lliure (llibreries Qt, ITK, VTK, OpenGL).

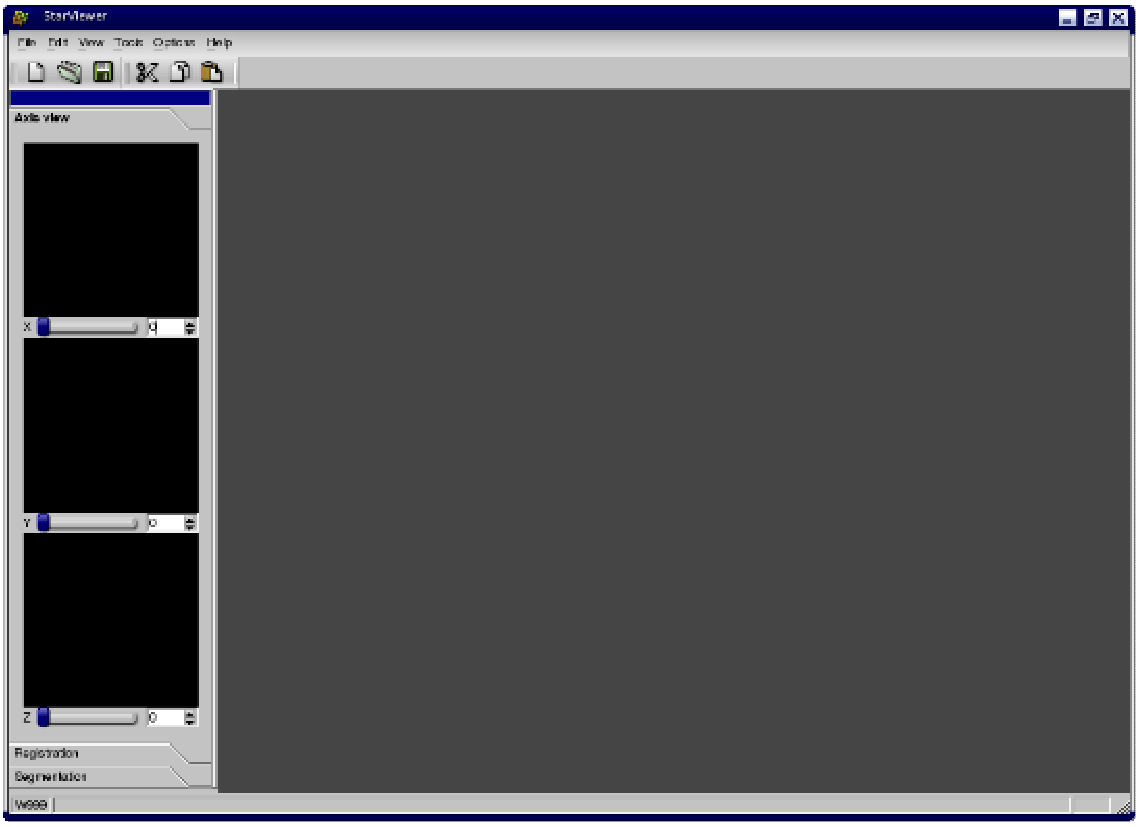


Figura 1.2. Interfície de la plataforma de visualització d'imatges mèdiques.

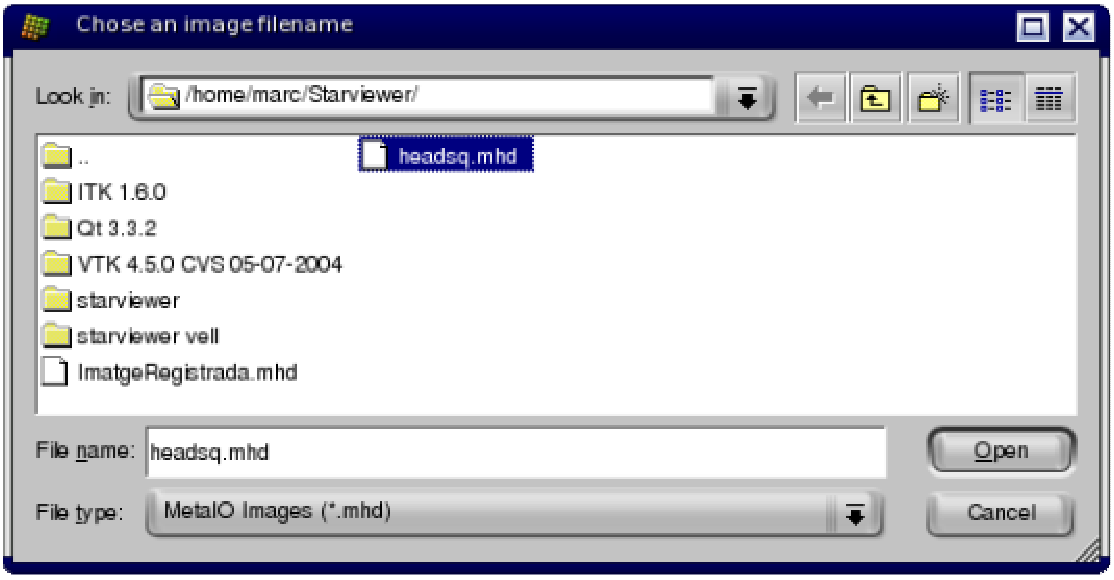


Figura 1.3. Quadre de diàleg per obrir un model.

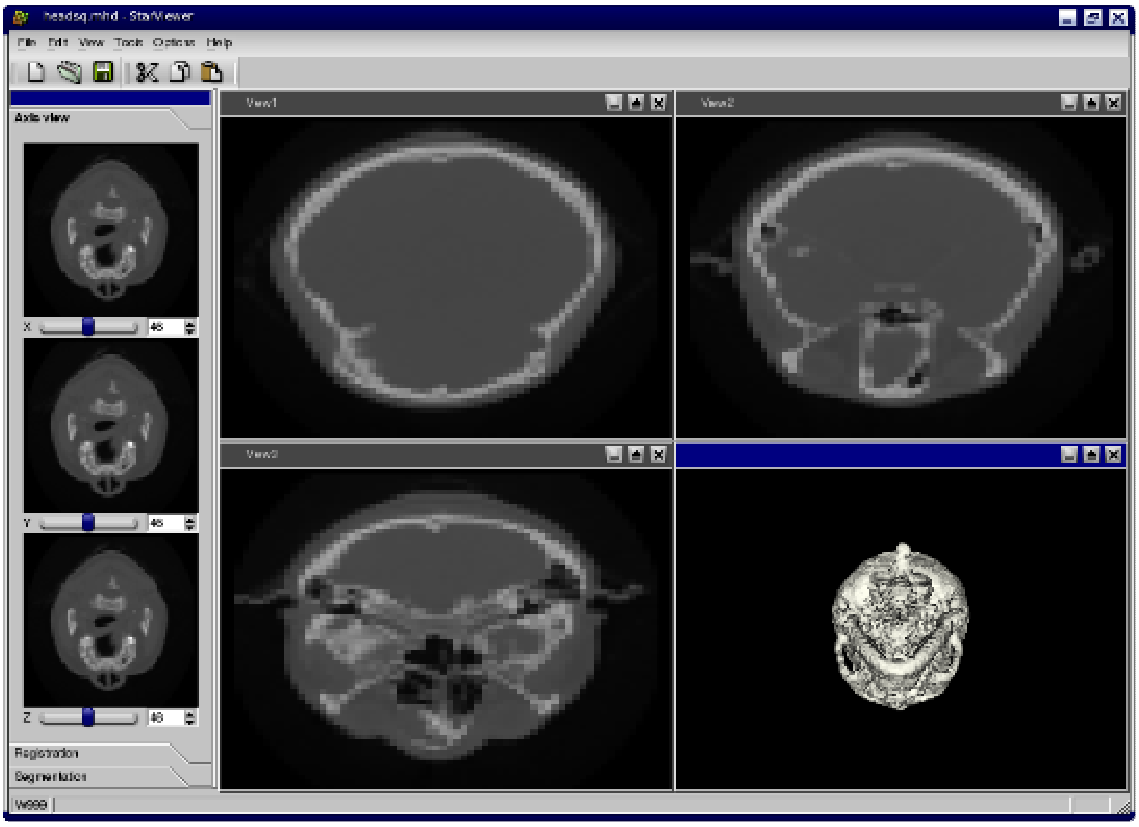


Figura 1.4. Plataforma amb el model obert.

Figura 1.5. Aquests quadrats permeten veure diferents llesques del model.

2 Objectius

L'objectiu principal d'aquest projecte és implementar un mètode per facilitar la visualització i interpretació de models de voxels simples i models de voxels registrats (models que tenen més d'un valor de propietat a cada voxel). La implementació realitzada s'haurà d'integrar dins d'una plataforma de visualització d'imatges mèdiques.

Per tal d'assolir aquest objectiu caldrà:

- Implementar la tècnica bàsica dels Miralls Màgics o Magic Mirrors. Aquesta tècnica, explicada de forma senzilla, consisteix en fer la visualització del model de voxels al centre de la pantalla i situar al seu voltant uns plans semblants a uns miralls on hi ha la imatge del mateix model vist des de la posició del mirall. Els miralls faciliten la interpretació de qualsevol model de voxels ja que permeten a l'usuari visualitzar-los des de diferents punts de vista de forma simultània.

- Estendre la tècnica dels miralls màgics per tal de suportar models registrats. Donat que els models registrats tenen més d'una propietat a cada voxel els Miralls Màgics han de ser capaços de visualitzar diferents tipus de propietats en cadascun dels miralls.

- Donar màxima flexibilitat a l'usuari. L'usuari ha de poder triar el nombre de miralls i alterar les seves posicions. L'usuari ha de poder decidir quines propietats vol veure en cada mirall i de quina manera (de quin color).

Deixant una mica de banda els Miralls Màgics, un altre objectiu important que ha de complir l'aplicació és que ha de tenir una interfície gràfica d'usuari que ha de permetre a l'usuari controlar totes les opcions de l'aplicació, ja siguin opcions dels Miralls Màgics o altres opcions que pugui tenir l'aplicació. La interfície ha de ser senzilla d'utilitzar, tant intuïtiva com es pugui.

Aquesta interfície gràfica s'ha de poder integrar a una plataforma de visualització d'imatges mèdiques que ja té una interfície pròpia, per tant la interfície dissenyada ha de ser compatible amb la que ja té la plataforma.

Aquesta integració amb la plataforma de visualització d'imatges mèdiques també implica que el disseny ha de ser modular, perquè es pugui integrar sense problemes.

Altres objectius referents a l'aplicació són que ha de ser ràpida per permetre una alta interactivitat i hauria de ser multiplataforma.

L'aplicació s'ha de poder construir fent servir eines de domini públic, per tal que la seva modificació o ampliació no impliqui costos elevats.

3 Estudi Previ

3.1 Fonaments Teòrics

3.1.1 Introducció

Els avenços tecnològics que s'han produït en aquests darrers anys han transformat considerablement les característiques del tractament de les imatges mèdiques. L'aparició de dispositius de captura sofisticats com són els equips de tomografia computeritzada (CT), ressonància magnètica (MRI), etc. han marcat un canvi revolucionari sobre les pràctiques de diagnòstic en permetre la reconstrucció de talls interns del cos humà amb alta precisió. D'altra banda les millores en el camp de la informàtica gràfica i la reducció de costos dels equips personals ha permès desenvolupar aplicacions que abans estaven restringides a grans equipaments. D'aquesta forma es factible actualment la implementació d'entorns de visualització i anàlisi d'imatges tridimensionals generades a partir de dispositius de captació, proporcionant eines de caràcter no invasiu per a l'assistència dels professionals en diferents tipus de procediments mèdics.

Malgrat tots aquests avenços hi ha molts problemes que encara queden per resoldre. Entre aquests problemes hi ha la integració i visualització de la informació obtinguda d'un pacient a través de diferents dispositius de captació en una mateixa imatge. En general la informació que s'obté del pacient a través dels diferents dispositius de captació es informació complementària, això ha fet que la possibilitat d'integrar tota aquesta informació en un model de representació únic hagi esdevingut una eina de gran valor pels especialistes.

El problema és que integrar informació en un mateix model no es fàcil. Hem de tenir en compte que ***els conjunts de dades proporcionats pels diferents dispositius de captació poden tenir resolucions diferents***. Cada volum de dades tindrà la seva pròpia resolució, depenent de la prova que s'hagi fet i la màquina que s'hagi fet servir. Per exemple, un fMRI genera imatges de menys resolució que un MRI. D'altra banda cal tenir en compte que **l'extensió espacial dels conjunts de dades també poden ser diferents**. Podem trobar-nos per exemple, que en un model hi ha una exploració de tot el cap i en l'altre només una part. A la Figura 3.1 hi ha un diagrama amb 2 dispositius de captació diferents i les imatges que generen. Com es pot veure, la mida de les imatges és diferent, i també la direcció d'exploració. Per resoldre tots aquests problemes s'han proposat diferents tècniques de registre. *El registre* consisteix bàsicament en alinear espacialment conjunts de dades, és a dir, que hi hagi una correspondència entre les dades obtingudes pels diferents dispositius de captació. El model resultat d'aquest procés és el que es coneix com a *model registrat*. El model registrat es pot interpretar com un model de voxels en el que cada voxel manté *n* valors de propietat que poden provenir de diferents dispositius de captació.

Una vegada s'ha realitzat el registre cal aplicar alguna tècnica de visualització que permeti obtenir la imatge en la qual es representi la informació d'aquest model. Tot i que s'han proposat moltes tècniques per poder visualitzar models de voxels en els quals només s'ha representat un sol valor de propietat, la visualització de models registrats resulta bastant complicada. Per fer més entenedora tota la problemàtica que es planteja començarem donant una descripció del procés de visualització d'un model simple, entenent per model simple el que només té representat en cada

voxel un valor de propietat. A continuació presentarem una tècnica per visualitzar models fusionats anomenada *Magic Mirrors*. Aquesta tècnica és la que s'ha implementat en aquest projecte.

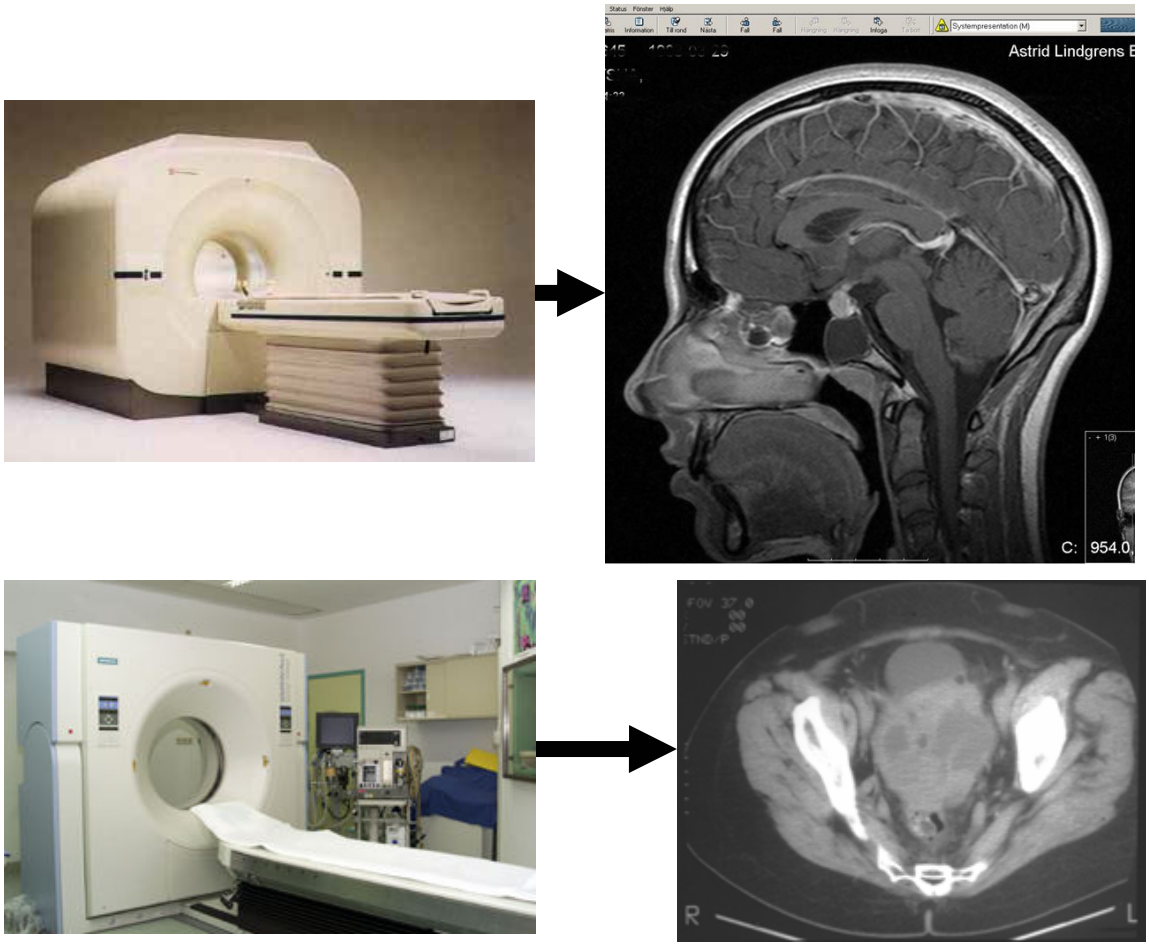


Figura 3.1. Exemples de dispositius de captació i les imatges obtingudes. A dalt, dispositiu i imatge de MRI. A baix, dispositiu i imatge de CT.

3.1.2 Visualització de Volums

En aquest apartat donarem una breu descripció de les diferents etapes que formen el procés de visualització i donarem una classificació dels diferents algorismes de visualització de volums que existeixen.

3.1.2.1 El procés de visualització

El procés de visualització segons el model de Haber-McNabb és un procés format per cinc etapes:

1. **Adquisició de les dades.** Aquesta etapa consisteix en l'obtenció de les dades que es volen visualitzar. En el nostre cas són les dades que ens proporciona el dispositiu de captació a través d'un fitxer en format DICOM.

2. **Definició d'un model.** Es dóna un format a les dades per tal de poder ser processades. En el nostre cas les representarem en el model de voxels, que

com hem dit abans consisteix en una malla rectangular 3D formada per un conjunt de cel·les sobre els vèrtexs de les quals representem els valors de propietat.

3. ***Mapping*.** A partir del model de voxels seleccionem la informació que volem usar per obtenir la imatge final. Aquesta informació anirà associada amb un tipus de primitiva concret. En particular podem seleccionar com a primitives punts, contorns, superfícies o voxels. En els tres primers casos el procés de *mapping* el que farà serà crear un model simplificat del model de voxels del que partíem inicialment. En el darrer cas donat que la primitiva usada per representar la informació és el voxel el procés de *mapping* no fa res, simplement ens quedem amb tot el model inicial. Evidentment aquesta opció és la que permet representar més informació en la pantalla, i serà també la que resultarà més complicada de tractar.

4. **Visualització.** Aquesta etapa consisteix en assignar atributs gràfics a les primitives seleccionades i aplicar alguna tècnica que permeti obtenir la imatge final. La tècnica de visualització que s'apliqui dependrà del procés de *mapping* aplicat. No és el mateix visualitzar punts que visualitzar contorns, superfícies o voxels.

5. **Imatge.** Obtenció de la imatge final.

El següent diagrama mostra el procés de visualització més gràficament:

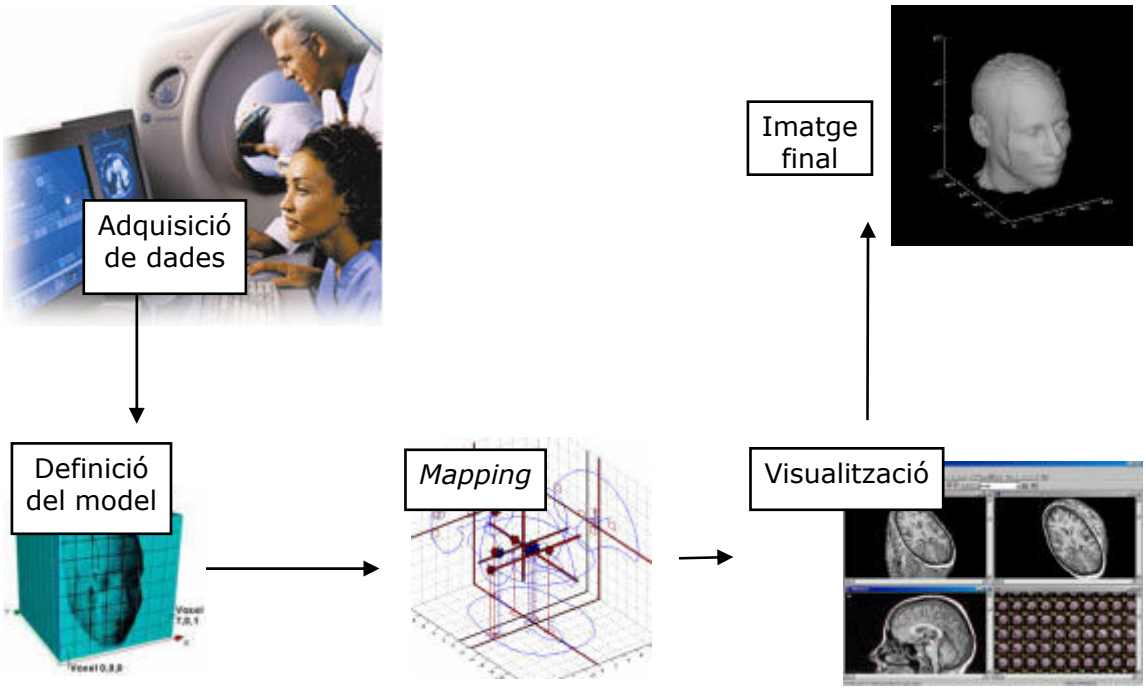


Figura 3.2. Procés de visualització.

3.1.2.2 Classificació dels algorismes de visualització

En funció de la tècnica de *mapping* que s'apliqui durant el procés de visualització els algorismes de visualització es classifiquen en dues grans famílies:

- **Algorismes de visualització de models reconstruïts**

En els cas de la visualització de models reconstruïts es poden considerar tres grups diferents.

- Algorismes que visualitzen punts
- Algorismes que visualitzen contorns
- Algorismes que visualitzen superfícies

De tots aquests algorismes els més populars són els que visualitzen superfícies i dins d'aquest grup el més conegut és l'algorisme de *Marching Cubes*.

- **Algorismes de visualització directa de volums**

Aquests algorismes generen una imatge tenint en compte tota la informació que hi ha representada en el model de voxels. Dins d'aquesta família es distingeixen dos grups diferents:

- Algorismes d'ordre imatge o tipus *ray-casting*

  Per generar la imatge es llença un raig per cada pixel de la pantalla. Aquest raig intersecta amb diversos voxels del volum. Segons com sigui la incidència del raig i les propietats gràfiques assignades als voxels, es calcula el color que cal assignar al pixel de la imatge final. A la Figura 3.3 a l'esquerra hi ha una esquema d'aquesta tècnica de visualització.

- Algorismes d'ordre objecte o tipus *splatting*

  Per generar la imatge final es projecten directament els voxels sobre la pantalla. Es fa un recorregut del model de voxels seguint una determinada direcció. El procés es similar al de llençar boles de neu sobre una paret.

En aquest projecte ens hem centrat en els algorismes de visualització directa de volums que segueixen la tècnica de *ray-casting*. Hem descartat els algorismes de visualització de models reconstruïts perquè tractaven amb models simplificats del model de voxels inicial i per tant resultaven una mala opció per visualitzar models fusionats. Una vegada decidits a aplicar algorismes de visualització directa de volums hem escollit els algorismes de tipus *ray-casting* ja que són més ràpids i obtenen més qualitat d'imatge que els basats en tècniques de *splatting*.

3.1.3 Visualització Directa de Volums

En aquest apartat analitzarem els conceptes previs comuns a totes les tècniques de visualització directa de volums.

3.1.3.1 Conceptes previs

El nostre objectiu és obtenir a la pantalla del computador una imatge que representi tota la informació que hi ha en el model de voxels. Sabem que el model de voxels manté un conjunt de valors de propietats que s'ha obtingut d'un pacient a través d'un dispositiu de captació. D'aquestes propietats només sabem que estan definides dins d'un rang de valors fixat pel dispositiu de captació i sabem que segueixen una distribució regular.

D'altra banda sabem que la imatge final estarà formada per un conjunt de pixels i que cada pixel porta associat un determinat color, que és el color que nosaltres veurem a la pantalla. El nostre problema és:

Com passar dels valors de propietat representats en el model de voxels als colors dels pixels a la pantalla?

Per resoldre aquest problema el que es fa és considerar el volum com si fos un gel format per una sèrie de partícules minúscules. Aquestes partícules corresponen als valors de propietat que tenim representats en el model de voxels. Es considera que cadascuna d'aquestes partícules és capaç d'emetre llum i també d'absorbir-ne. Les quantitats de llum que pot emetre i absorbir estaran en funció de les propietats que tinguin assignades les partícules. Per determinar quines són aquestes propietats es defineix una funció de transferència.

3.1.3.1.1 La funció de transferència

La funció de transferència determina el color i l'opacitat (o grau de transparència) que li correspon a un determinat valor de propietat. Per fer-ho més entenedor la funció de transferència determina el color del qual volem pintar una determinada propietat. Aquest color el determina l'usuari. Nosaltres podem dir per exemple que volem veure l'os d'un color blanquinós i totalment opac, o bé que el volem veure blanquinós però semi-transparent per tal de poder determinar les seves estructures internes. Podem dir que la pell la volem d'un color rosat, les venes vermelles...

Per definir la funció de transferència cal assignar un color i una opacitat a cada valor de propietat. El color i l'opacitat es representen com un valor RGBA on R representa la component vermella del color final, G la component verda, B la component blava i A el grau d'opacitat. La component A està definida entre 0 i 1 representant 0 transparent i 1 opac.

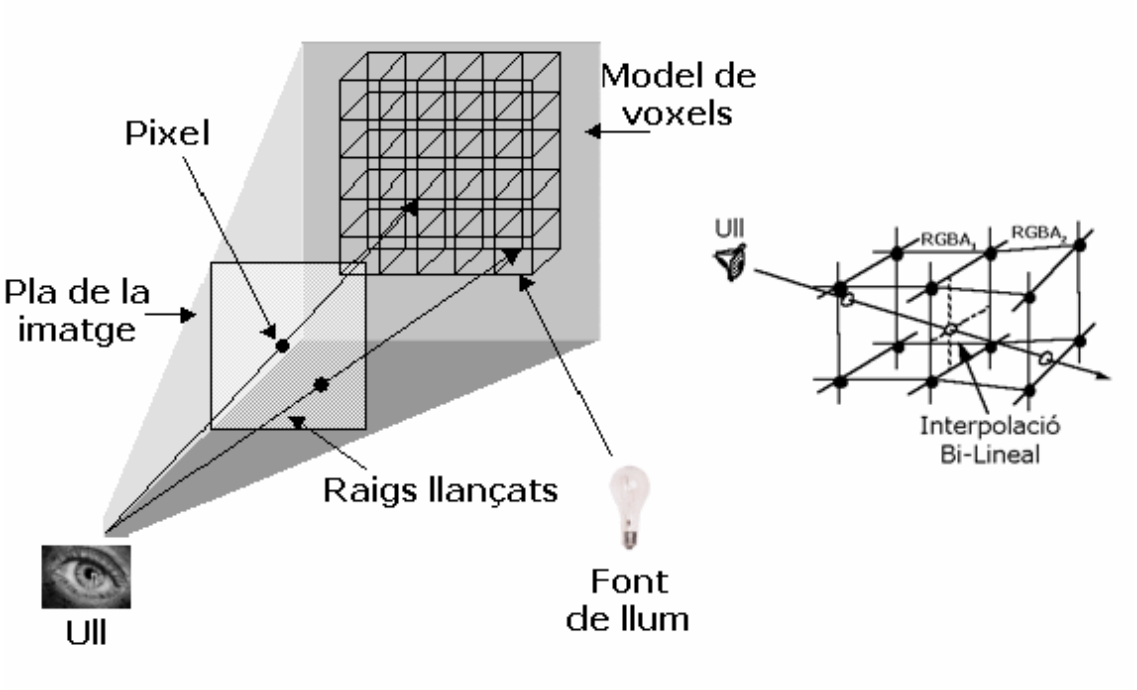


Figura 3.3. Tècnica de *ray-casting*. Esquerra: per cada pixel de la pantalla es llança un raig que intersecta amb el model de voxels. Dreta: el raig travessa una sèrie de voxels que tenen un valor RGBA assignat.

El valor RGBA assignat a una propietat representa la quantitat de llum que es capaç d'absorbir i emetre aquella propietat. A la Figura 3.3 a la dreta es pot veure que quan un raig travessa els voxels es pot fer una interpolació entre els valors per decidir el color que s'agafa per aquell punt, i que després servirà per fer la composició de colors. A l'esquema es fa una interpolació bi-lineal entre els 4 valors del voltant. Aquesta és l'opció quan es considera que els valors estan als vèrtexs. Si es considera que els valors estan al centre dels voxels s''agafa el més proper.

3.1.3.1.2 Composició de colors

Una vegada definida la funció de transferència i per tant assignats els valors RGBA als diferents valors de propietat representats en el model de voxels per generar la imatge final el que hem de fer és per cada pixel de la pantalla llançar un raig de llum. Aquest raig és una línia recta que travessarà algunes de les partícules que composen el volum. El problema que ara es planteja és com determinar el color final que s'obté tenint en compte la quantitat de llum que absorbeix i emet cadascuna de les partícules intersectades, és a dir, tenint en compte el valor RGBA que té assignada cadascuna de les propietats.

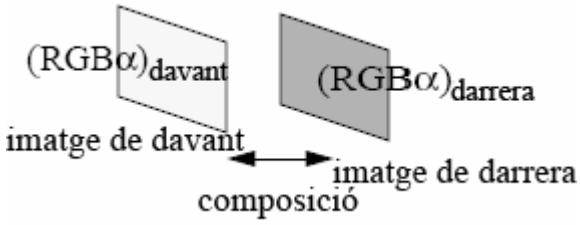
Per resoldre aquest problema s'aplica una equació simplificada de la teoria de transport de la llum. La base d'aquesta simplificació és no considerar el raig de llum com una línia continua sinó considerar-lo com una línia discreta.

- Els colors i opacitats dels pixels del darrera són atenuats per les opacitats dels pixels del davant:

$$rgb = RGB_{darrera} \cdot \alpha_{darrera}(1 - \alpha_{davant}) + RGB_{davant} \cdot \alpha_{davant}$$
$$\alpha = \alpha_{darrera}(1 - \alpha_{davant}) + \alpha_{davant}$$

- Fent servir:

$$rgb_{darrera} = RGB_{darrera} \cdot \alpha_{darrera}$$
$$rgb_{davant} = RGB_{davant} \cdot \alpha_{davant}$$



obtenim 2 equacions recursives que es poden fer servir per compondre qualsevol nombre d'objectes de davant cap endarrera:

$$rgb_{davant} = rgb_{darrera}(1 - \alpha_{davant}) + rgb_{davant}$$
$$\alpha_{davant} = \alpha_{darrera}(1 - \alpha_{davant}) + \alpha_{davant}$$

- La visualització directa de volums utilitza aquesta expressió recursiva per combinar (= compondre) les mostres preses al llarg del raig

Aplicant aquesta equació de forma recursiva obtenim la imatge final.

Aquesta equació és la base de tots els algorismes de visualització directa de volums. La diferència que hi ha entre els diferents algorismes rau en la forma de prendre les mostres sobre el raig de llum que es llança, en l'ordre en que es prenen les mostres o en l'ordre en que es fa l'assignació de colors i opacitats.

3.1.4 Ray-Casting

Aquí explicarem la tècnica de *ray-casting* de forma general, tant per visualitzar polígons com per fer visualització directa de volums. L'explicació general i l'específica del ray-casting geomètric està extreta de la pàgina següent, que conté una explicació dels algorismes de *ray-tracing* i *radiositat*:

http://dmi.uib.es/~ramon/Docencia/ig2/IGII_tema4.doc

Podem definir un raig com una línia recta amb un punt de partida fix a partir del qual s'estén en una direcció donada cap a l'infinit. Quan encenem una llum, els raigs es posen en moviment rebotant en tots els objectes. Si un raig toca un objecte poden passar vàries coses: es pot reflectir, es pot absorbir o es pot transmetre. Fins i tot hi ha casos en què es pot absorbir i després transmetre (objectes fluorescents).

Si tenim en compte que els raigs es poden representar com a rectes i que els objectes de l'escena es poden modelar matemàticament, llavors afegint els coneixements de la física que ens diuen com es calculen les reflexions i les refraccions d'un raig podríem modelar el comportament de la llum únicament coneixent la posició de totes les fonts.

El que es pretén fer és seguir el camí dels raigs procedents de les fonts de llum i veure la seva influència en cadascun dels objectes de l'escena. Com que és molt difícil (i costós) seguir el camí de cada raig (dels quals d'altra banda n'hi ha infinits) a partir de la font de llum, es realitza el procés invers: es selecciona un centre de projecció (el punt de vista) i una finestra en un pla arbitrari (el que correspondria a la pantalla), es divideix la finestra en pixels depenent de la resolució que volem tenir i després per a cadascun d'aquests pixels es traça (o llança) un raig amb origen al centre de projecció, cap a l'escena (raig de visió). Quan el raig col·lisiona amb un objecte llavors es calcula el color del pixel corresponent. Llavors es tracen altres raigs, de manera recursiva, amb l'origen al punt d'intersecció i en la direcció de cada una de les fonts de llum. D'aquesta manera es calcula de quina manera la font contribueix a la il·luminació del pixel.

D'aquesta manera, s'aconsegueix considerar únicament aquells raigs que arriben directament als nostres ulls i que contribueixen a la il·luminació de l'escena.

Això és per a un *ray-casting* geomètric (els objectes de l'escena són polígons). En el cas del *ray-casting* volumètric no hi ha fonts de llum i el raig travessa una sèrie de voxels i calcula el color del pixel fent una composició dels valors RGBA que retorna la funció de transferència per cada valor de propietat. No existeixen "col·lisions".

3.1.4.1 Elements d'un traçador de raigs

Fonamentalment, per modelar el comportament de la llum en aquest cas necessitem tres elements: l'observador, els objectes de l'escena i les fonts d'il·luminació. Ara veurem breument cadascun d'aquests elements.

3.1.4.1.1 L'observador

En el nostre cas, l'observador es pot considerar com una camera amb la qual es pren la foto de l'escena. A partir d'un punt d'observació, i a una distància determinada, es col·loca un pla quadriculat (la pantalla) perpendicular a l'escena

que volem representar. Per cada un dels quadradets del pla (pixels) es veurà un tros d'escena, que serà el que representarem al pixel corresponent de la pantalla. Aquest procés no és nou, ja que és exactament el que s'utilitzava durant el Renaixement per la representació en perspectiva[1]. Per exemple, l'artista alemany Alebrecht Dürer utilitzava una quadrícula transparent (de fet era un cristall) i mirava a través d'ella per dibuixar les seves pintures. La tela del quadre estava també dividida en el mateix número de quadrícules i l'artista mirava per un petit forat i així reduïa el punt de vista a una única posició.

3.1.4.1.2 Els objectes

Per aquest tipus d'algorismes, els objectes més adequats són aquells que es poden representar fàcilment amb expressions matemàtiques (això és fonamentalment la causa que la majoria d'imatges d'exemple del *ray-casting* representin esferes, cilindres o cubs). Encara que de fet es pot utilitzar qualsevol representació amb la qual es pugui calcular (més o menys ràpidament) la seva intersecció amb un raig (per exemple, un model de voxels).

3.1.4.1.3 Les fonts de llum (només al ray-casting geomètric)

El tipus de fonts de llum és molt variat (llum solar, bombetes, fluorescents, foc, etc.). D'elles ens interessen les seves principals propietats: el color, la posició, la intensitat, la grandària i la forma.

Les dues simplificacions més habituals al *ray-casting* són:

1. Suposar que la llum que s'emet és d'un únic color (en realitat es composa d'una combinació d'un ampli interval).
2. Suposar que la llum prové d'un punt infinitament petit.

La segona simplificació fa que els objectes no mostrin penombres, sinó que únicament mostraran ombres ben definides (la qual cosa no és gaire habitual a la realitat). Si es vol modelar l'efecte de la penombra es pot modelar la font de llum com un conjunt (o matriu) de fonts puntuals.

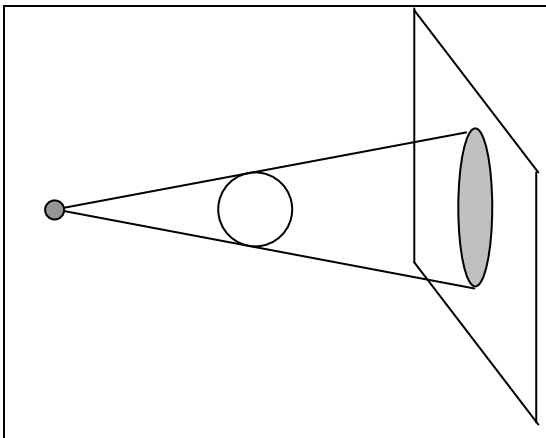


Figura 3.4. Font puntual: ombra.

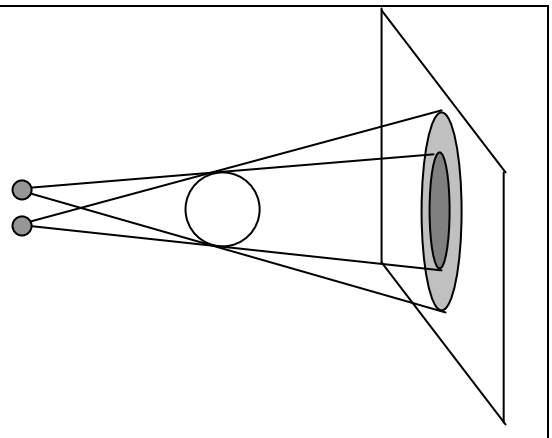


Figura 3.5. Font múltiple: ombra i penombra.

[1] De fet, la paraula perspectiva prové del llatí *perspicere* que vol dir "mirar al través".

Cal recordar que al *ray-casting* volumètric, que és el que ens interessa, no es fan servir fonts de llum.

3.1.4.2 Tipus de raigs

El raigs amb què es treballa poden ser dels següents tipus:

- **Raig de visió:** És el que s'envia des del punt de visió fins a l'escena passant per cada un dels pixels del rectangle de la finestra. Serveix per calcular el color del pixel corresponent a la imatge final. En el cas del ray-casting volumètric travessa una sèrie de voxels i compon els seus valors RGBA corresponents per calcular el color del pixel.
- **Raig d'il·luminació (només al *ray-casting* geomètric):** Una vegada el raig de visió intersecta un objecte de l'escena, s'envia un raig d'ombra des d'aquest punt d'intersecció a cada una de les fonts de llum definides a l'escena. Si el raig arriba a la font directament (sense travessar el propi objecte), llavors es considera la influència d'aquesta font per calcular la il·luminació final del punt.

- **Raig reflectit (només al *ray-casting* geomètric):** Amb la tècnica de ray-casting només hi ha reflexió difusa (i només en el cas del geomètric), que es produeix en totes direccions.

3.1.5 Limitacions

La visualització directa de volums també té algunes mancances, ja que si els nivells d'opacitat que ens retorna la funció de transferència són molt petits pot ser difícil apreciar els detalls de les zones d'interès. Contràriament, si l'opacitat és molt alta no podrem veure gaire més enllà de la superfície del volum, i l'interior quedarà ocult (veure Figura 3.6 per veure els 2 casos). Per solucionar aquests problemes afegirem informació addicional a la visualització que ajudarà a saber on estan situades les dades d'interès. Aquesta informació addicional serà en la forma de vistes alternatives del volum.

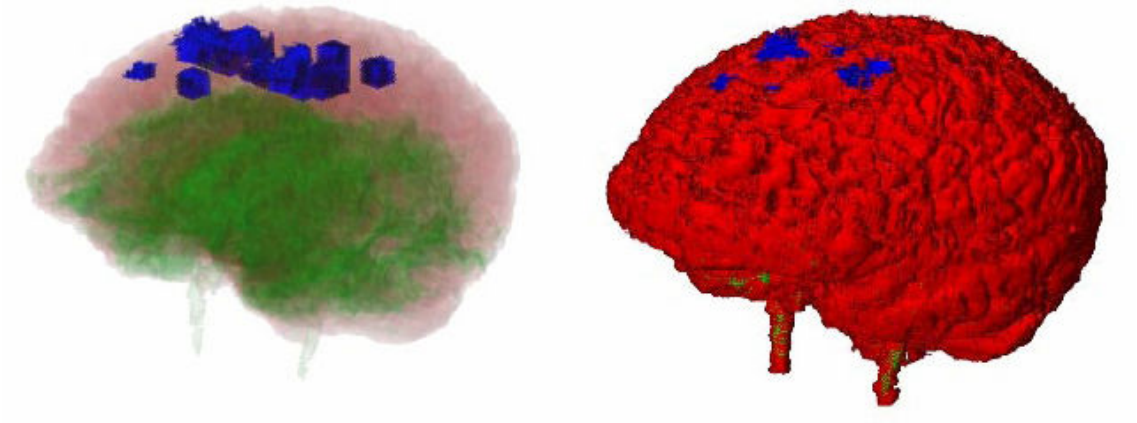


Figura 3.6. Esquerra: visualització del volum amb molt baixa opacitat; és difícil apreciar els detalls i fer-se una idea de l'estructura espacial de les dades. Dreta: visualització del volum amb molt alta opacitat; no es veu l'interior del volum.

Amb una sola imatge quieta és difícil apreciar correctament la profunditat, ja que la veiem en un dispositiu 2D com és una pantalla. Per aquest motiu K. Rehm i altres autors han proposat projectar les regions d'interès a les parets d'un entorn en forma de cub, per ajudar a veure el context i la situació d'aquestes regions. Amb aquestes vistes alternatives és molt més fàcil deduir la disposició espacial de les dades (veure Figura 3.7 per la comparació). Amb aquesta idea farem una extensió de la visualització directa de volums anomenada Magic Mirrors, que explicarem més endavant.

A la resta d'aquest apartat i el següent, referent als Magic Mirrors, la informació està treta principalment del text *Multiple Views and Magic Mirrors – fMRI Visualization of the Human Brain*, que es pot trobar a l'adreça següent:

http://www.cg.tuwien.ac.at/research/vis/vismed/MM/

També es pot trobar un estudi que explica perquè es fan servir plans al voltant del volum per les vistes addicionals i no un altre sistema –com per exemple més finestres– al text *Mental Registration of 2D and 3D Visualizations (An Empirical Study)* de Melanie Tory, disponible a la següent adreça:

http://www.cs.sfu.ca/~mktory/personal/publications/vis03.pdf

Continuant amb les múltiples vistes, les imatges dels plans són projeccions ortogonals del volum tal com es veuria des de darrera de cada pla (veure Figura 3.7, dreta). Aquestes és poden generar amb la mateixa tècnica de *ray-casting* i després enganxar-les com a textures als plans. Per aquesta aplicació la projecció ortogonal s'ha demostrat que és més útil que la perspectiva, ja que la distorsió de la perspectiva no ajuda l'usuari a interpretar la localització de les dades d'interès.

Aquests plans tenen característiques diferents a les que pot tenir un mirall per exemple. El que es veu en un mirall és diferent segons el punt de vista de l'usuari, en canvi, en aquests plans es veu sempre la mateixa informació independentment del punt de vista de l'usuari, que simplement veurà la informació de manera diferent. Els plans tenen aquestes propietats perquè d'aquesta manera resulten més útils.

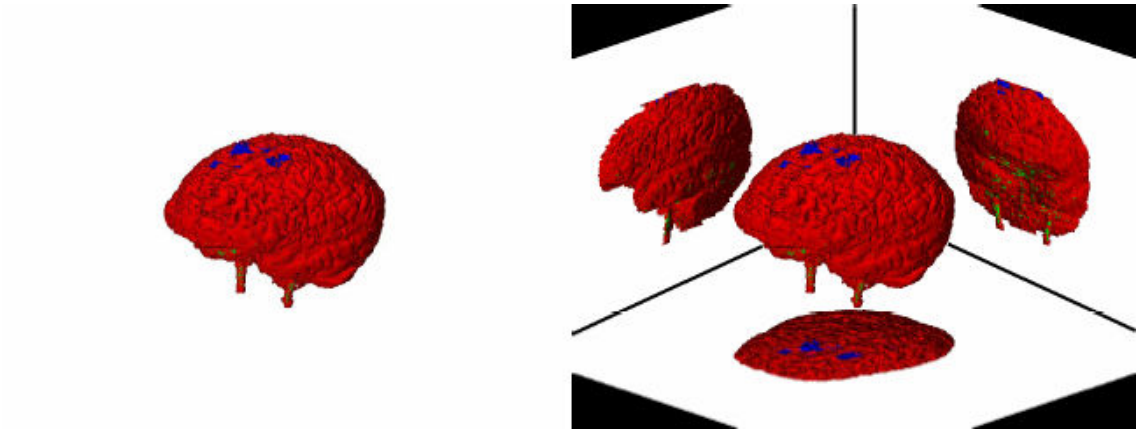


Figura 3.7. Esquerra: visualització directa d'un volum sense vistes múltiples. Dreta: visualització directa d'un volum amb vistes múltiples.

Els beneficis que aporten les vistes múltiples són els següents:

- La localització de les regions d'interès és òbvia fins i tot en imatges quietes, ja que el context tridimensional es pot reconèixer més fàcilment quan tenim

múltiples vistes a la mateixa escena. Això es nota especialment quan els valors de transparència són molt alts (veure Figura 3.8 esquerra i comparar amb Figura 3.6 esquerra).

- El problema de l'oclusió es resol bastant satisfactòriament. A la Figura 3.8 dreta per exemple s'ha tallat un bloc de dades del darrere, però gràcies a les vistes múltiples les localitzacions de les dades d'interès encara són visibles.

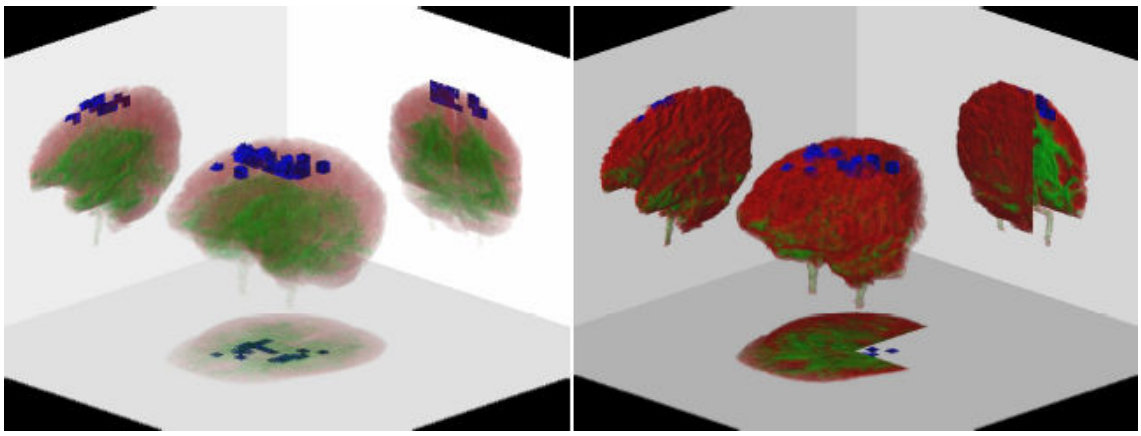


Figura 3.8. Esquerra: utilitzant múltiples vistes, es pot deduir la localització exacta de les activacions fins i tot amb nivells alts de transparència. Dreta: fins i tot les regions ocultes com el tall del darrera són òbvies amb l'ajuda de vistes múltiples.

3.1.6 Magic Mirrors

Les vistes múltiples aporten bons beneficis, però encara es poden millorar. Tenint en compte que els plans al voltant del volum no tenen unes característiques físiques reals (no són miralls), podem fer servir tècniques de visualització diferents per generar les textures dels plans que per dibuixar el volum. La versió estesa de les vistes múltiples s'anomena *Magic Mirrors* i aquestes són les seves propietats:

- *"Projecció"* vs. *"Reflexió"*: Les textures per les múltiples vistes són creades amb la tècnica del *ray-casting* fent servir un punt de vista des de darrera dels plans. Amb això obtenim un efecte semblant a la reflexió. Però podem posar fàcilment el punt de vista a l'altra banda del volum i d'aquesta manera obtenir les projeccions contràries (des de davant, des de dalt i des de la dreta).

- Es poden fer servir funcions de transferència diferents per cadascun dels Miralls Màgics (veure Figura 3.9). En un Mirall podem destacar unes certes propietats i en un altre Mirall unes altres propietats.

- Es poden afegir imatges obtingudes amb tècniques de visualització diferents de la visualització directa de volums per proporcionar informació complementària. Per exemple, en un Mirall podem posar una imatge del contorn del volum juntament amb les dades d'interès ressaltades (Figura 3.11).

Totes aquestes millores es poden combinar entre elles per oferir més versatilitat als Miralls Màgics.

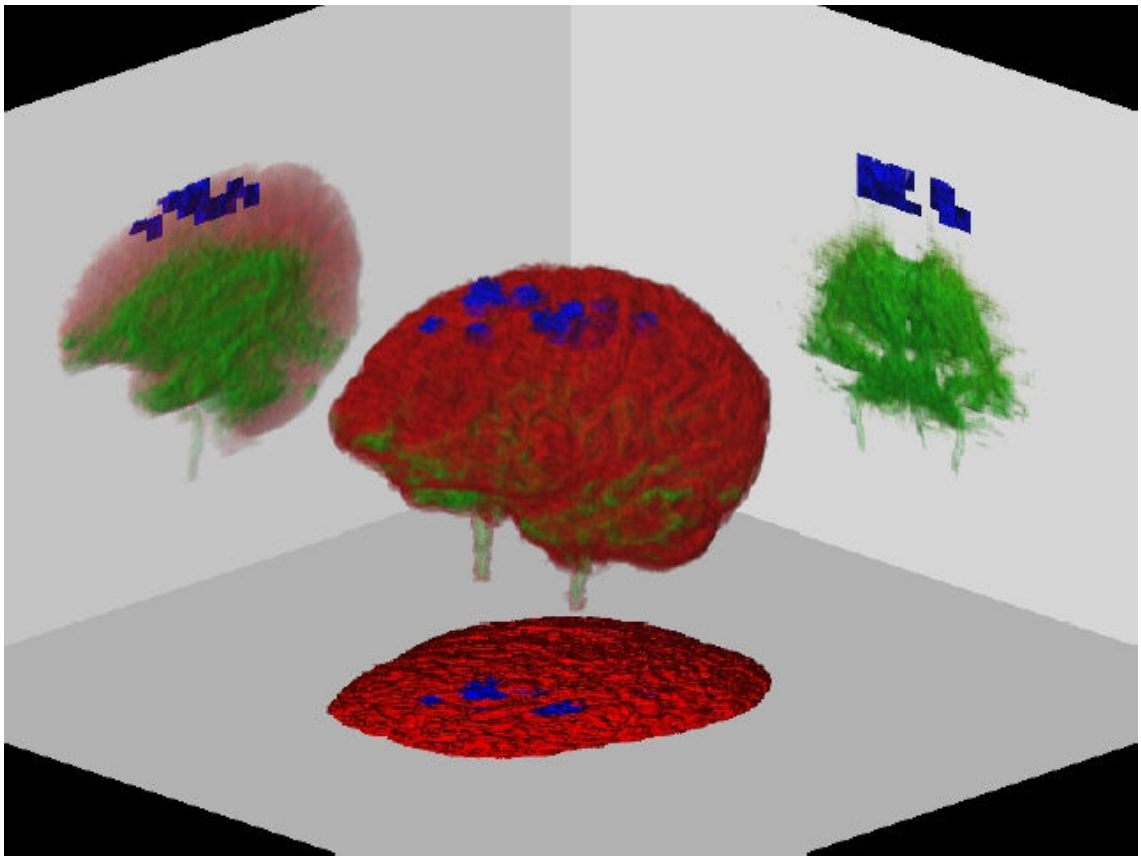


Figura 3.9. Magic Mirrors: es poden fer servir funcions de transferència diferents a cada lloc (el volum central i els Miralls). En aquesta imatge hi ha 4 funcions de transferència diferents, on el que varia és el grau d'opacitat de la zona vermella.

3.1.7 Visualització de Models Registrats

A l'hora de realitzar visualitzacions de models registrats ens trobem amb els mateixos problemes que a l'hora de visualitzar els volums no registrats. Hem de ser capaços de definir una funció de transferència que ens indiqui quines propietats visuals té una propietat determinada. Cal tenir en compte que al tractar-se de volums registrats no tindrem només una sola propietat sinó que en tindrem més d'una.

Voxel d'un model simple:
només hi ha una propietat

Voxel d'un model registrat:
hi ha dues o més propietats

Figura 3.10. Tipus de voxels.

Si considerem que el volum registrat representa la informació de dos volums v_1 i v_2, pels quals hem definit les funcions de transferència f_1 i f_2, les principals opcions de pintat que tenim són les que es presenten a continuació:

1. Pintar el volum registrat usant la funció de transferència f_1. En aquests cas la imatge que es genera és la mateixa que obtindrem si haguéssim pintat directament el volum v_1.
2. Pintar el volum registrat usant la funció de transferència f_2. En aquests cas la imatge que es genera és la mateixa que obtindrem si haguéssim pintat directament el volum v_2.

Aquestes dues primeres opcions no tenen massa sentit, ja que només estem considerant informació d'un únic volum.

3. La tercera opció es calcular un nou valor de propietat a partir de les mostres representades en el voxel (promig, màxim, mínim...). Pel nou valor calculat es defineix una nova funció de transferència. El volum registrat es pinta usant aquesta nova funció.
4. La darrera possibilitat és aplicar una combinació de valors d'un model i de l'altre.

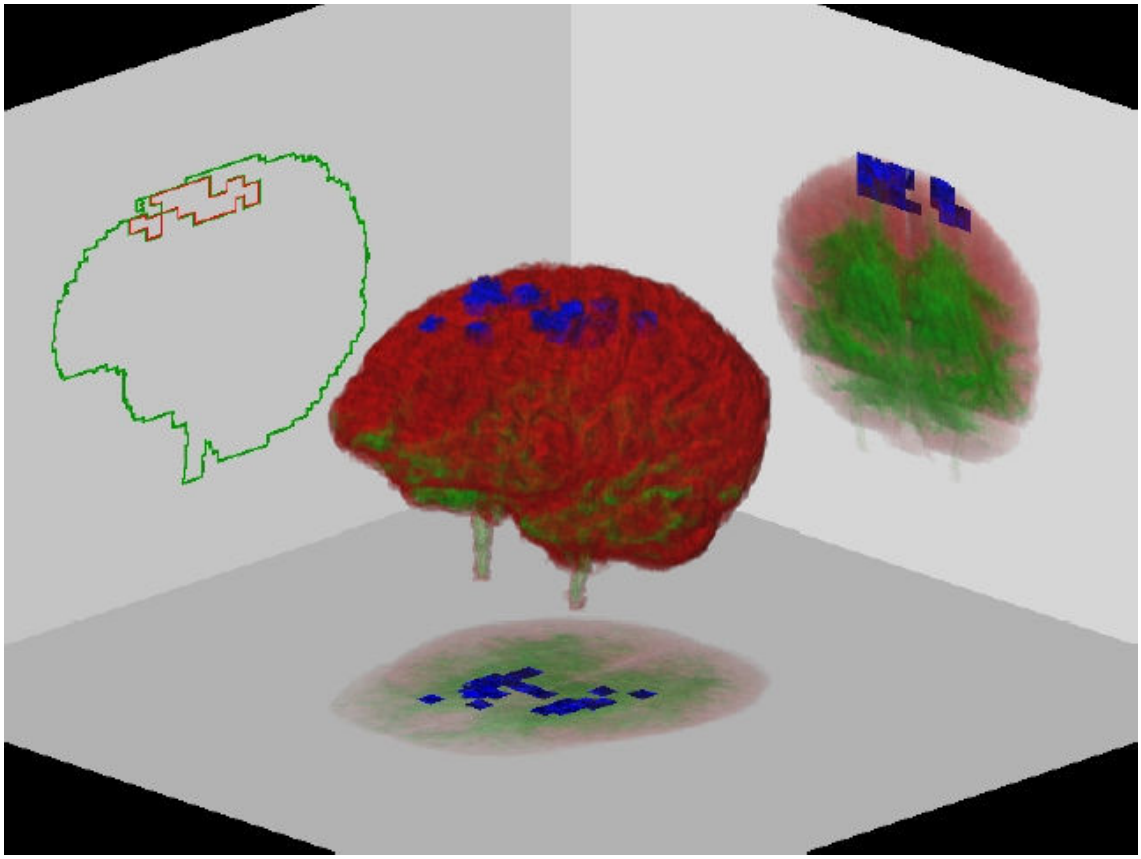


Figura 3.11. Una altra possibilitat dels Magic Mirrors: al Mirall de l'esquerra es visualitza el contorn del volum i de les propietats destacades.

3.1.8 Magic Mirrors per Models Registrats

Els Magic Mirrors són una bona solució a l'hora de visualitzar models registrats. Amb els Magic Mirrors podem visualitzar propietats diferents a cada lloc.

Per exemple, si tenim un model registrat amb tres propietats podríem veure les tres propietats juntes al volum central (usant la 3a o la 4a opció de l'apartat

anterior), una propietat al Mirall de l'esquerra, una altra al Mirall de la dreta i l'altra al de baix.

També podríem veure totes les propietats a tots els Miralls, però amb diferents combinacions de les funcions de transferència.

3.2 Fonaments Pràctics

3.2.1 Qt

La informació d'aquest apartat està treta del *Qt 3.3 Whitepaper*, disponible des de la pàgina web de Trolltech, juntament amb la documentació completa de Qt, tutorials i les llibreries per diferents plataformes:

http://www.trolltech.com

3.2.1.1 Introducció

Les Qt són unes llibreries per C++ que permeten desenvolupar aplicacions amb interfície gràfica d'usuari (GUI, *Graphical User Interface*). Són totalment orientades a objectes i ofereixen un conjunt de classes que faciliten la programació d'aplicacions gràfiques, multiplatarforma i multillengua.

Les llibreries Qt proporcionen un conjunt de *widgets* que permeten fer les GUIs. També tenen un sistema de comunicació entre objectes anomenat *"signals i slots"*. També hi ha un model d'events convencional que permet gestionar els clics del ratolí, les tecles premudes, etc. Les Qt permeten crear tots els elements de les GUIs d'avui en dia, com menús, menús contextuals i barres d'eines.

Juntament amb les Qt hi ha un programa, el Qt Designer, que permet dissenyar gràficament les interfícies d'usuari i veure com queden amb diferents estils. Un altre programa, el Qt Linguist permet traduir les aplicacions que fan servir Qt a altres idiomes.

Les Qt suporten gràfics en 2D i en 3D, permeten connexions a bases de dades independents de la plataforma, i poden emular l'aparença de varis sistemes operatius i entorns gràfics.

3.2.1.2 Widgets

Les Qt disposen d'una àmplia varietat de *widgets*, que són elements visuals que es combinen per crear interfícies d'usuari. Exemples de *widgets* són botons, menús, *scrollbars* i finestres. Tots els *widgets* de Qt poden servir com a controls i com a contenidors, i es poden crear nous *widgets* com a subclasses dels existents. A les figures Figura 3.12, Figura 3.13, Figura 3.14 i Figura 3.15 hi ha imatges de diferents *widgets*.

Figura 3.12. Un QLabel i un QPushButton organitzats fent servir un QHBox.

A les Qt hi ha la classse QWidget, que és el *widget* més bàsic, i la resta de *widgets* són subclasses d'aquesta.

Figura 3.13. Un QDateTimeEdit, un QLineEdit, un QTextEdit i un QComboBox organitzats dins d'un QGroupBox.

Un *widget* pot contenir qualsevol nombre de *widgets* fills, que són dibuixats a l'interior del *widget* pare. Els *widgets* fills s'organitzen dins del pare fent servir diferents tipus de *layouts* bàsics que es poden combinar per aconseguir distribucions més complexes. El *widget* de més alt nivell és el que no té pare, generalment la finestra principal. Quan es deshabilita, s'amaga o es destrueix un *widget* s'aplica la mateixa acció a tots els fills recursivament.

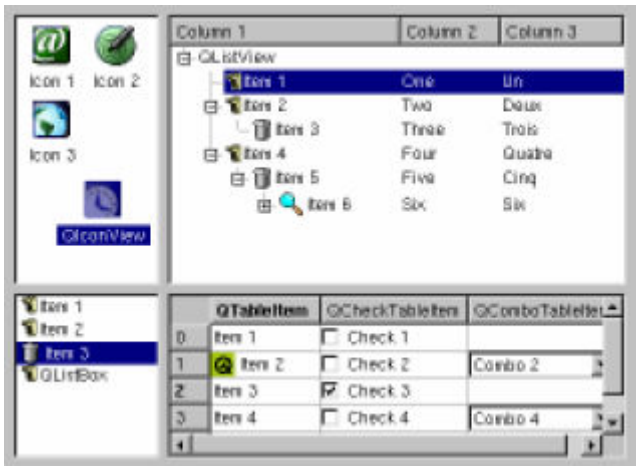


Figura 3.14. Un QIconView, un QListView, un QListBox i un QTable organitzats fent servir un QGrid.

Figura 3.15. *Widget* personalitzat (subclasse de QLCDNumber) que actua com un rellotge digital.

3.2.1.3 Signals i slots

Les aplicacions amb interfície gràfica d'usuari responen a les accions de l'usuari. Per exemple, quan un usuari clica un ítem del menú o un botó, l'aplicació executa un tros de codi. Més generalment és útil poder comunicar objectes de qualsevol tipus entre ells, relacionar un determinat event amb un codi concret.

El sistema que proporcionen les Qt per fer això s'anomena mecanisme de *"signals i slots"*. Els *signals i slots* són flexibles, orientats a objectes i implementats amb C++.

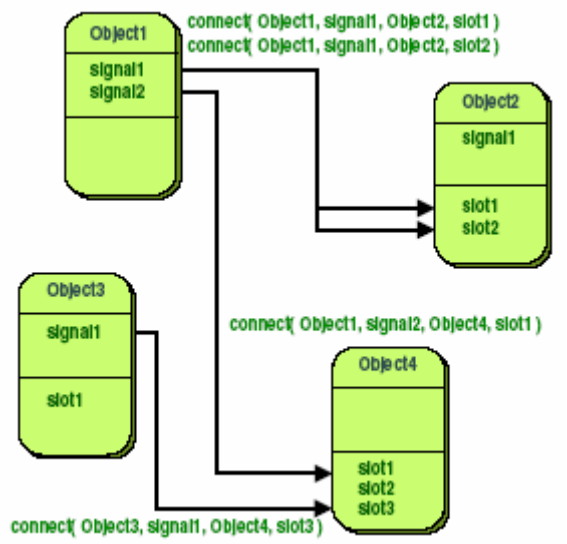


Figura 3.16. Vista abstracta d'algunes connexions de *signals* i *slots*.

Els *widgets* de Qt emeten *signals* quan es produeixen events. Per exemple, si un usuari clica un botó el botó emet un *signal* "clicked". Aquest *signal* es pot connectar a un *slot* (que és una funció) d'un altre objecte fent servir la funció connect(). Aquest mecanisme permet connectar objectes de classes totalment independents, sense coneixement l'una de l'altra, i d'aquesta manera ajuda a crear classes reutilitzables. A la Figura 3.16 es pot veure un esquema de connexions de *signals i slots*.

Com a exemple de funcionament, si volem fer que quan l'usuari cliqui un botó s'acabi l'aplicació, es faria amb el següent codi:

```
connect( button, SIGNAL(clicked()), qApp, SLOT(quit()) );
```

Es poden fer i desfer connexions entre *signals* i *slots* en qualsevol moment durant l'execució d'una aplicació de Qt.

Perquè una classe pugui fer servir el mecanisme de *signals i slots* ha d'heredar de QObject o una de les seves subclasses i incloure la macro `Q_OBJECT` a la definició de la classe. Els *signals* es declaren a la secció `signals` i els *slots* a les seccions `public slots`, `protected slots` o `private slots`. Exemple:

```
class BankAccount : public QObject
{
      Q_OBJECT

public:
      BankAccount() { curBalance = 0; }
      int balance() const { return curBalance; }

public slots:
      void setBalance( int newBalance );

signals:
      void balanceChanged( int newBalance );
```

```
private:
      int curBalance;
};
```

Quan s'emet un *signal* s'executen tots els *slots* que estan connectats amb ell. Els *slots* són com la resta dels mètodes de la classe, però amb la propietat que poden ser connectats a *signals*.

Implementació de setBalance():

```
void BankAccount::setBalance( int newBalance )
{
      if ( newBalance != curBalance ) {
            curBalance = newBalance;
            emit balanceChanged( curBalance );
      }
}
```

Els *signals* s'emeten fent servir la paraula `emit` seguida del nom del *signal* amb els paràmetres corresponents. A diferència dels *slots*, els *signals* no s'implementen.

Per connectar dos BankAccounts diferents es faria així:

```
BankAccount x, y;
connect( &x, SIGNAL(balanceChanged(int)), &y,
SLOT(setBalance(int)) );
```

Quan es fa la connexió només s'indiquen els tipus dels paràmetres, però no els noms. També és possible connectar un *signal* a un altre *signal* directament, i quan s'emet el primer s'emet el segon automàticament. Perquè la connexió funcioni el *signal* i el *slot* (o els dos *signals*) han de tenir la mateixa signatura pel que fa als tipus, encara que el receptor pot tenir menys paràmetres i d'aquesta manera ignoraria els altres. Si no és així, donarà un *warning* en temps d'execució perquè no podrà fer la connexió. Passaria el mateix si el *signal* o el *slot* no existissin.

Per compilar les classes que fan servir el mecanisme de *signals* i *slots* aquestes han de passar pel *Meta-Object Compiler* (`moc`), inclòs amb les Qt, que converteix el codi dels *signals i slots* en codi C++ estàndard. El `moc` s'invoca automàticament amb els makefiles que genera el `qmake` (una altra aplicació de les Qt).

3.2.1.4 Qt Designer

El Qt Designer (Figura 3.17) és una aplicació de les Qt que permet dissenyar ràpidament una interfície gràfica amb Qt. És senzill i intuïtiu. Només cal buscar el *widget* que volem entre els menús o les barres d'eines, i llavors afegir-lo al lloc que vulguem del *widget* pare. Després es poden canviar les propietats del *widget* amb l'editor de propietats. També es poden aplicar diferents tipus de *layouts* (organitzacions dels *widgets*).

El Qt Designer té plantilles per diferents tipus de GUIs: finestres principals, quadres de diàleg, etc. També es poden crear noves plantilles i afegir *widgets* personalitzats per fer-los servir en altres interfícies gràfiques.

Un cop dissenyada la interfície es pot veure com queda amb diferents estils sense necessitat de compilar. D'aquesta manera s'estalvia molt temps si s'han de corregir petits detalls.

Amb el Qt Designer també es poden fer les connexions entre *signals* i *slots* dels diferents *widgets* de l'interfície, i afegir *signals*, *slots*, atributs i mètodes a la pròpia interfície. També porta un editor de C++ per implementar el codi que necessiti la classe creada.

Les interfícies d'usuari creades amb el Qt Designer es guarden en fitxers amb extensió `.ui` que conté un codi XML que serveix per generar el codi font de la classe. Això es fa amb el programa `uic` (*User Interface Compiler*) de les Qt.

El Qt Designer permet agrupar les classes i interfícies gràfiques en projectes de Qt, que es guarden en fitxers `.pro`. El `qmake` genera automàticament els makefiles necessaris per compilar els projectes de Qt, cridant els diferents programes (`qmake`, `uic`, `moc`) quan és necessari.

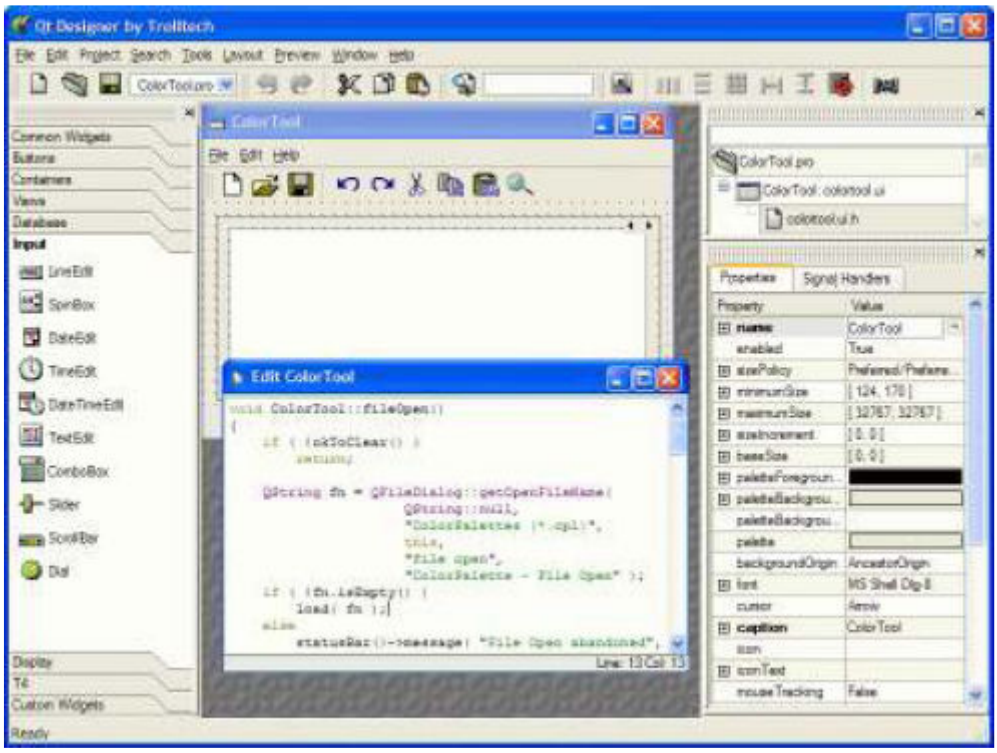


Figura 3.17. Imatge del Qt Designer.

3.2.1.5 Ús en el projecte

Les Qt han servit per dissenyar la part d'interfície gràfica del projecte. La interfície que té els controls per configurar els paràmetres de visualització ha estat dissenyada amb el Qt Designer. Les comunicacions internes entre els elements d'aquesta interfície i la comunicació amb la classe principal de control (MagicMirrors) s'han implementat aprofitant el mecanisme de *signals i slots*, que proporciona una certa independència entre les classes.

Les Qt també han servit per poder integrar aquest projecte a la plataforma de visualització d'imatges mèdiques.

3.2.2 ITK

La informació d'aquest apartat està treta de la pàgina web d'ITK, on també hi ha la documentació i les llibreries:

http://www.itk.org

3.2.2.1 Introducció

ITK (*Insight ToolKit*) són unes llibreries *open-source* per realitzar operacions de registre i segmentació d'imatges. La segmentació és el procés d'identificar i classificar les dades que es troben en una representació mostrejada digitalment. Normalment aquesta representació és una imatge obtinguda amb instruments mèdics com *scanners* de CT o MRI. Una operació de registre consisteix en alinear o desenvolupar correspondències entre dades. Per exemple, en l'entorn mèdic, es poden alinear les dades d'una CT amb les d'una MRI per combinar la informació de les dues.

Les ITK estan implementades amb C++ fent servir programació genèrica (*templates*) i són multiplataforma.

Algunes característiques:

- Proporcionen representació de dades i algoritmes per efectuar operacions de segmentació i registre. Estan enfocades cap a aplicacions mèdiques, encara que poden processar altres tipus de dades.
- Proporcionen representacions de dades de forma general per imatges de dimensions arbitràries i malles no estructurades.
- Suporten processament paral·lel amb *multi-threading* i memòria compartida.
- Estan organitzades al voltant d'una arquitectura de flux de dades. Les dades es representen amb objectes de dades que són processats per objectes de procés (filtres). Els objectes de dades i els objectes de procés es connecten en *pipelines*. Les *pipelines* són capaces de processar les dades a trossos segons el límit de memòria que especifiqui l'usuari.
- Els objectes s'instancien utilitzant factories d'objectes (patró de disseny *Factory Method*).
- Els events es processen fent servir el patró de disseny *Observer*.
- Les llibreries estan implementades seguint els principies de la programació genèrica (*templates* de C++).
- Són multiplataforma (Unix, Windows i MacOS X).
- El model de memòria depèn dels *"smart pointers"*, que tenen un comptador de referències a objectes.

3.2.2.2 Ús en el projecte

Les llibreries ITK només les he fet servir a la classe MagicMirrors, en el mètode per afegir una nova imatge. El mètode rep com a paràmetre una imatge d'ITK i llavors fa servir un filtre d'ITK per convertir-la en una imatge de VTK, i aquí s'acaba l'ús de les ITK.

3.2.3 VTK

La informació d'aquest apartat està treta de la pàgina web de VTK, on també hi ha la documentació i les llibreries:

http://www.vtk.org

3.2.3.1 Introducció

Les VTK (Visualization ToolKit) són unes llibreries *open-source* gratuïtes per a gràfics per computador, processament d'imatges i visualització. Estan implementades amb C++ (orientades a objectes) i són multiplataforma.

Els model de gràfics de VTK està a un nivell d'abstracció més alt que altres llibreries gràfiques com OpenGL o PEX, i per tant es poden crear aplicacions gràfiques més fàcil i ràpidament.

És un sistema de visualització complet amb molts algoritmes de visualització geomètrics i volumètrics, a més de nombrosos filtres per tractar les imatges abans de visualitzar-les. També és possible barrejar imatges 2D, 3D i dades.

Algunes característiques:

- Implementat amb C++ i orientat a objectes.
- Es pot integrar amb varis sistemes de finestres, entre ells Qt.
- Les finestres de VTK es poden fer interactives.
- Tracta els events seguint el patró *Observer*.
- Visualització de superfícies i volums amb diferents algoritmes, entre ells *Ray Casting*.
- Característiques de sistemes de visualització: llums, cameres, textures, etc.
- El codi per fer la visualització és independent del dispositiu de visualització.
- Gran quantitat de filtres per tractar les dades.
- Sistema de visualització amb *pipelines* (flux de dades) que pot tractar les dades per parts.
- Execució paral·lela de *pipelines* i filtres (*multi-threading*).
- Comptadors de referències pels objectes de VTK.
- Creació d'objectes amb *Factory Method*.

Les VTK estan formades per dos models d'objectes: el model de gràfics i el model de visualització.

3.2.3.2 El Model de Gràfics

L'objectiu del model de gràfics és transformar dades gràfiques en imatges.

El model de gràfics inclou els actors i volums (*props*), llums, cameres, propietats i *mappers* dels *props*, transformacions, LUTs (*Look-Up Tables*) i funcions de transferència, *renderers*, *render windows* i *render window interactors*.

Combinant aquests elements es crea una escena.

Els *props* (actors i volums) són els objectes que es veuen a l'escena. La seva aparença és controlada per una propietat (fent servir LUTs o funcions de transferència) i la seva geometria per un *mapper* (que proporciona una interfície

entre el model de visualització i el model de gràfics). Els *props* també guarden una transformació interna.

Les cameres i les llums funcionen de forma semblant a altres sistemes de visualització, però a VTK són objectes.

El *renderer* és l'objecte que dibuixa la imatge a la *render window*, que és la finestra que mostra l'escena. El *render window interactor* serveix perquè l'usuari pugui interactuar amb l'escena a través de la finestra.

3.2.3.3 El Model de Visualització

L'objectiu del model de visualització és transformar informació en dades gràfiques.

Aquest model inclou dos tipus d'objectes: els objectes de dades i els objectes de procés.

Els objectes de dades representen dades de diferents tipus, amb estructura o sense. Les imatges de VTK són un tipus de dades.

Els objectes de procés o filtres processen objectes de dades d'entrada per produir nous objectes de dades de sortida. Representen els algoritmes del sistema.

Els dos tipus d'objectes es connecten per formar *pipelines* de visualització, que s'executen només quan es demanen les dades.

3.2.3.4 Ús en el projecte

Les VTK han servit per implementar tota la part de visualització: *Ray Casting*, funcions de transferència, volums, Miralls, cameres, etc.

4 Anàlisi i Disseny de l'Aplicació

4.1 Anàlisi de Requeriments

Com s'ha dit a la introducció, l'aplicació desenvolupada s'havia d'integrar en una plataforma de visualització d'imatges mèdiques, que en el moment de l'anàlisi permetia obrir un model de voxels. Per tant en l'anàlisi hem suposat que ja tenim un model de voxels carregat a memòria, en una variable del programa inicial.

Tenint en compte això, necessitem un mòdul que agafi aquest volum i el tracti de la forma necessària per poder implementar les característiques dels Magic Mirrors, explicades anteriorment.

A més a més ha d'oferir la possibilitat de visualitzar un model registrat. Aquest també hem suposat que estava guardat en una variable del programa, si bé en l'etapa d'anàlisi encara no estava implementat, però sí més endavant, durant l'etapa de disseny.

D'aquesta manera, els requeriments de l'aplicació són:

- Manipular el volum lliurement de forma senzilla, per poder-lo veure des de diferents angles.
- Escollir la funció de transferència desitjada per visualitzar el volum.
- Hi ha d'haver els Miralls en els quals hi ha d'haver la projecció del volum.
- Escollir funcions de transferència diferents pel volum a cada Mirall.
- Per un model registrat:
    - Visualitzar el model registrat escollint quina propietat volem veure a cada Mirall i al centre.
    - Seleccionar funcions de transferència diferents per cada propietat i Mirall.
    - Decidir la prioritat de visualització en cas que les dues propietats hagin de ser visibles.

Donats aquests requeriments, també hi haurà d'haver una interfície gràfica d'usuari que s'integri a la plataforma de visualització d'imatges mèdiques i que permeti a l'usuari controlar totes aquestes opcions.

Perquè la interfície gràfica i la resta de l'aplicació es puguin integrar a la plataforma, hem de fer servir les mateixes llibreries:

- Per fer la interfície gràfica farem servir Qt, perquè és el mateix que fa servir la interfície gràfica de la plataforma de visualització.
- Per accedir al model carregat a memòria necessitarem ITK, ja que es guarda en una variable de tipus imatges d'ITK.
- Per manipular i visualitzar el model farem servir VTK, ja que implementa vàries tècniques de visualització volumètrica.

Els mòduls dels quals es compondrà l'aplicació seran:

- **Mòdul de Control**: Un mòdul que controli la manipulació dels volums en l'espai 3D i la visualització. Aquest mòdul rebrà una imatge d'ITK i la convertirà en una imatge de VTK per visualitzar-la.

- **Mòdul de Volums**: Un mòdul que agrupi els diferents paràmetres de visualització per cada volum, ja que seran diferents per cada Mirall.
- **GUI**: La interfície gràfica d'usuari que ha de permetre totes les opcions explicades.

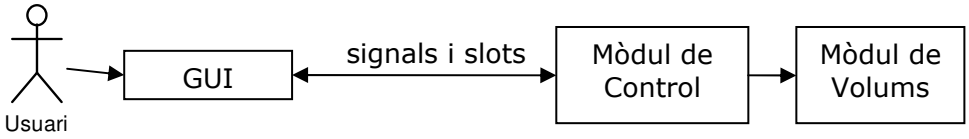


Figura 4.1. Comunicació entre els mòduls de l'aplicació.

4.2 Casos d'Ús

4.2.1 Diagrama de casos d'ús

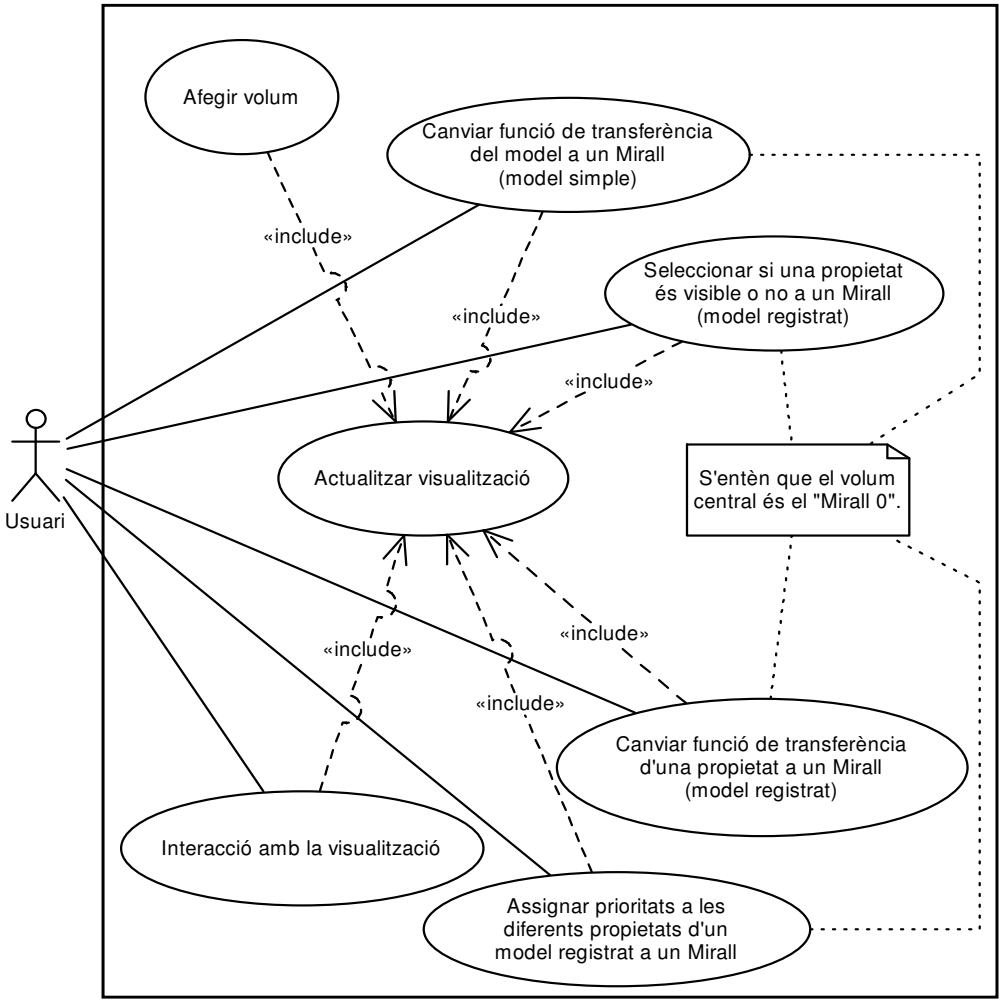


Figura 4.2. Diagrama de casos d'ús.

4.2.2 Fitxes de casos d'ús

CAS D'ÚS	Afegir volum
Descripció	S'afegeix un volum d'una propietat als Magic Mirrors.
Actors	Sistema
Precondició	
Flux principal	1. Afegir el volum a la llista de volums 2. Assignar funcions de transferència per defecte 3. Actualitzar visualització
Postcondició	El volum ha estat afegit i es visualitza el model
Comentaris	El sistema detecta els models oberts a la plataforma general i els afegeix aquí. Si s'afegeix un volum es considera un model simple. Si se n'afegeix més d'un es considera un model registrat. S'ha de cridar com a mínim una vegada perquè funcioni la resta de l'aplicació.

CAS D'ÚS	Canviar funció de transferència del model a un Mirall (model simple)
Descripció	L'usuari canvia la funció de transferència d'un model simple a un dels Miralls.
Actors	Usuari
Precondició	Hi ha un model simple carregat
Flux principal	1. L'usuari selecciona el Mirall 2. L'usuari canvia els paràmetres de la funció de transferència 3. Canviar la funció de transferència del volum corresponent al Mirall seleccionat 4. Actualitzar visualització
Postcondició	El model es visualitza amb la nova funció de transferència al Mirall seleccionat

CAS D'ÚS	Seleccionar si una propietat és visible o no a un Mirall (model registrat)
Descripció	L'usuari tria si una propietat del model registrat és visible o no a un dels Miralls.
Actors	Usuari
Precondició	Hi ha un model registrat carregat
Flux principal	1. L'usuari selecciona la propietat 2. L'usuari selecciona el Mirall 3. L'usuari selecciona si és visible o no 4. Aplicar la visibilitat seleccionada al volum corresponent a la propietat seleccionada al Mirall seleccionat 5. Actualitzar visualització
Postcondició	El model es visualitza amb els canvis aplicats

CAS D'ÚS	Canviar funció de transferència d'una propietat a un Mirall (model registrat)
Descripció	L'usuari canvia la funció de transferència per una propietat i Mirall.
Actors	Usuari
Precondició	Hi ha un model registrat carregat

Flux principal	1. L'usuari selecciona la propietat 2. L'usuari selecciona el Mirall 3. L'usuari canvia els paràmetres de la funció de transferència 4. Canviar la funció de transferència del volum corresponent a la propietat seleccionada al Mirall seleccionat 5. Actualitzar visualització
Postcondició	El model es visualitza amb la funció de transferència canviada

CAS D'ÚS	Assignar prioritats a les diferents propietats d'un model registrat a un Mirall
Descripció	L'usuari assigna una prioritat de visualització a cada propietat per un Mirall. D'aquesta manera, si les dues propietats han de ser visibles segons les funcions de transferència, la que es visualitza finalment és la que té més prioritat.
Actors	Usuari
Precondició	Hi ha un model registrat carregat
Flux principal	1. L'usuari selecciona el Mirall 2. L'usuari assigna la prioritat per cada propietat 3. Aplicar canvis 4. Actualitzar visualització
Postcondició	El model es visualitza amb les prioritats assignades

CAS D'ÚS	Interacció amb la visualització
Descripció	L'usuari interacciona amb la pantalla girant el volum, movent la camera, fent *zoom*, etc.
Actors	Usuari
Precondició	Visualització activa
Flux principal	1. L'usuari efectua la interacció 2. Actualitzar visualització
Postcondició	La visualització està actualitzada d'acord amb la interacció

CAS D'ÚS	Actualitzar visualització
Descripció	S'actualitza la visualització en pantalla dels Magic Mirrors.
Actors	Usuari
Precondició	Hi ha un o més volums afegits
Flux principal	1. Per cada Mirall: Fer visualització des del punt de vista del Mirall Convertir imatge visualitzada en textura Enganxar textura al pla del Mirall 2. Fer visualització del volum central i els Miralls
Postcondició	La visualització està actualitzada
Comentaris	El sistema detecta els models oberts a la plataforma general i els afegeix aquí. Si s'afegeix un volum es considera un model simple. Si se n'afegeix més d'un es considera un model registrat.

4.3 Disseny de l'Aplicació

En aquesta secció explicaré primer els dissenys inicials, sense entrar en tots els detalls, i després el disseny definitiu, aquest detalladament. Per les versions més importants del disseny hi haurà un diagrama de classes i una explicació de què fa i per a què serveix cada classe. Els noms de les classes dels diagrames són de diferents colors, depenent de la seva tipologia:

- Amb text normal, les classes que ja formaven part de la plataforma.
- En **negreta**, les classes que he fet jo.
- En gris, les classes de STL.
- En vermell, les classes de Qt.
- En verd, les classes d'ITK.
- En blau, les classes de VTK.

4.3.1 Primers dissenys

Al principi vaig fer alguns dissenys molt simples que no estaven integrats a la plataforma i es limitaven a visualitzar un volum simple sense poder canviar les funcions de transferència. Com que estaven molt lluny del disseny final no els explicaré, perquè no tenen gaire importància.

A la Figura 4.3 hi ha el diagrama de classes general d'un dels primers dissenys on ja hi havia una integració amb la plataforma, però encara no tenia en compte els models registrats.

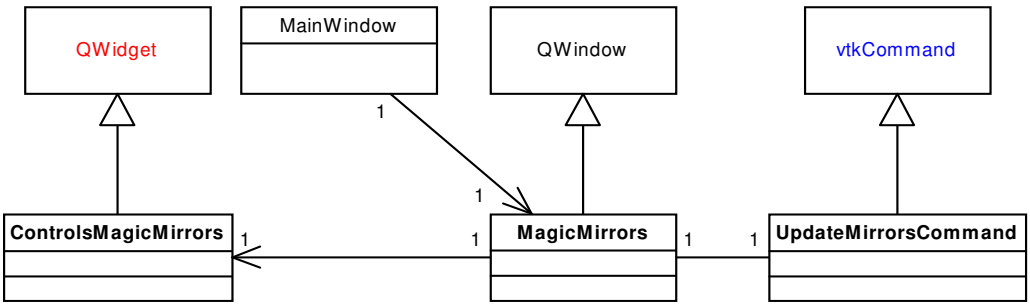


Figura 4.3. Diagrama de classes general (primers dissenys).

En aquest disseny era la classe MagicMirrors la que guardava una referència a la interfície, que era ControlsMagicMirrors (veure Figura 4.4). Aquesta era una interfície molt bàsica que servia només per rotar el volum, encara no es podien configurar les funcions de transferència. La interfície es veia a la part de dalt de la finestra principal de MagicMirrors.

Figura 4.4. ControlsMagicMirrors.

Lògicament aquest disseny s'havia de canviar molt perquè no permetia canviar funcions de transferència ni treballar amb models registrats. A més a més quedava millor si la interfície no formava part de la finestra principal, ja que així semblava més integrat a la plataforma de visualització d'imatges mèdiques, que té els controls per les diferents funcions en un *toolbox* a l'esquerra.

El disseny de la interfície gràfica va passar per una altra fase abans d'arribar al definitiu. En aquesta fase encara era una sola classe que es deia

ConfiguracioMagicMirrors, però ja estava separada de MagicMirrors (veure Figura 4.5). Tenia un aspecte semblant al de MirrorSetup (veure Figura 4.6) i permetia configurar les funcions de transferència per tots els Miralls a la vegada.

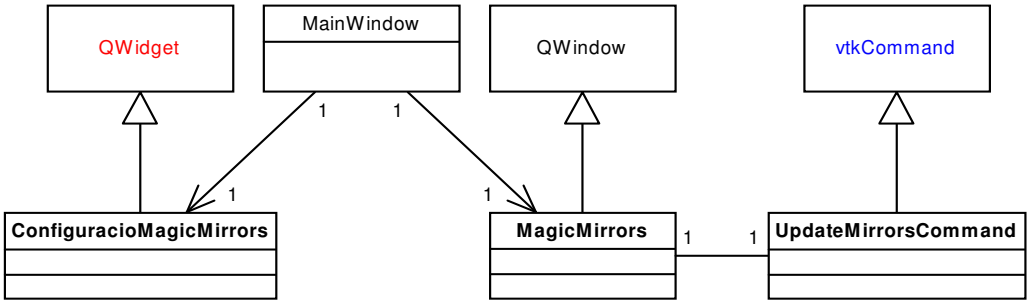


Figura 4.5. Diagrama de classes general (primers dissenys).

Aquesta interfície gràfica va haver de ser substituïda per la del disseny final perquè no permetia configurar funcions de transferència diferents per cada Mirall i volum, i la resta del disseny perquè encara no suportava models registrats.

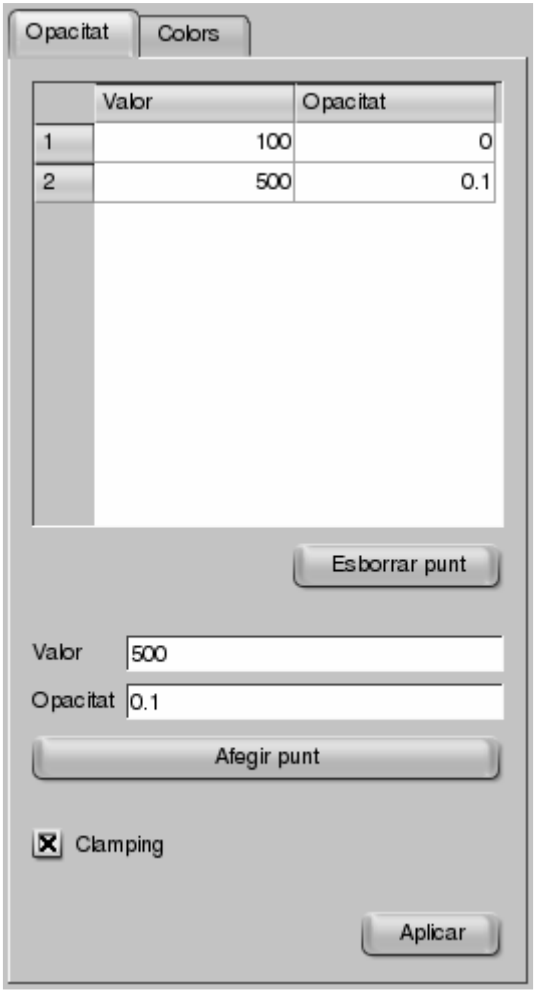


Figura 4.6. ConfiguracioMagicMirrors.

Després ja es va passar a un disseny més proper al final, amb el mòdul de volums incorporat, encara que aquest no era encara com el de la versió final. El diagrama

de classes general ja era com en el disseny final, però el del mòdul de volums no (veure Figura 4.7).

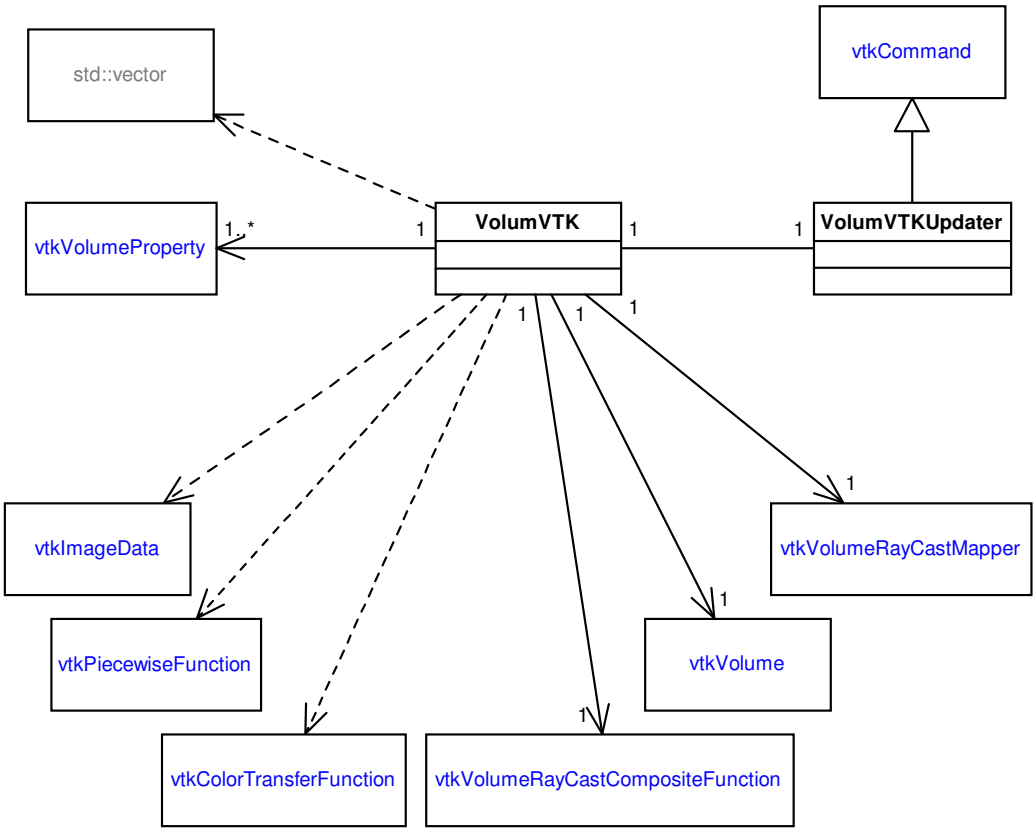


Figura 4.7. Diagrama de classes del mòdul de volums (primers dissenys).

La classe central d'aquest mòdul és VolumVTK, que s'ajuda de VolumVTKUpdater per actualitzar-se quan és necessari. En aquesta versió del disseny VolumVTK guardava un volum (vtkVolume), un *mapper* de *Ray Casting* (vtkVolumeRayCastMapper i vtkVolumeRayCastCompositeFunction) i un vector de propietats (vector de STL i vtkVolumeProperty). Cada propietat tenia associada una funció de transferència d'opacitat (vtkPiecewiseFunction) i una de color (vtkColorTransferFunction), de les quals en guardava una referència ella mateixa i per tant no calia guardar-la a la classe VolumVTK. Es guardava una propietat pel volum central a la posició 0 del vector i una per cada Mirall a les posicions següents. Per fer-ho ampliable el nombre de propietats que havia de guardar era un paràmetre del constructor.

Amb aquest disseny es pretenia estalviar memòria guardant només les diferents propietats (funcions de transferència) però només un volum, i abans de fer la visualització per cada Mirall i pel volum central seleccionar la propietat que li corresponia i assignar-la al volum, i llavors fer la visualització. És a dir:

1. Per cada Mirall:
   1.1. Seleccionar la propietat corresponent al Mirall
   1.2. Visualitzar el model des de la camera del Mirall[2]
2. Seleccionar la propietat corresponent al volum central
3. Visualitzar el volum central

[2] Després de la visualització caldria fer el procés d'agafar la imatge com a textura i enganxar-la al Mirall, però aquest és un esquema simplificat.

Aquest sistema no va funcionar perquè no es podia controlar el procés de visualitzar primer un Mirall, canviar la funció de transferència i visualitzar-ne un altre, i al final es veia el model amb la mateixa funció de transferència al mig i als Miralls.

Encara hi va haver un disseny de Magic Mirrors abans del disseny final. El diagrama de classes és el de la Figura 4.8.

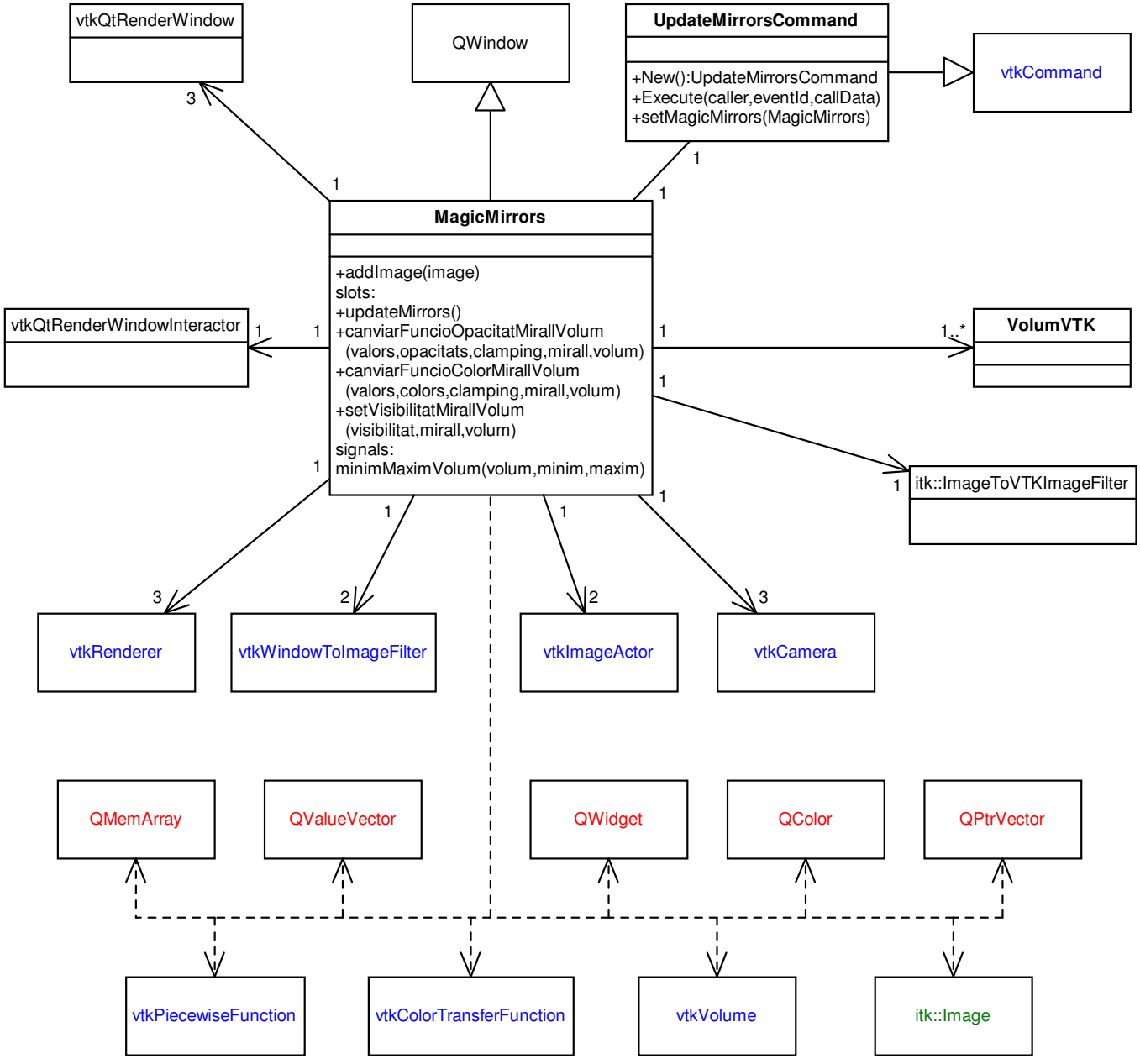


Figura 4.8. Diagrama de classes del mòdul de control (primers dissenys).

La classe MagicMirrors no tenia el mètode ajustarModelRegistrat() i el MagicMirrorsUpdater es deia UpdateMirrorsCommand. La diferència principal és que en aquest cas es guardava per cada Mirall una vtkQtRenderWindow i un vtkWindowToImageFilter associats a ell. Els filtres agafaven les imatges de les finestres i les enganxaven directament als Miralls. El problema era que no ho feia

bé: enganxava la imatge corresponent al segon Mirall al primer, i al segon hi havia una imatge indefinida. Fent proves vaig descobrir que enganxava la imatge de l'últim filtre executat al primer mirall dibuixat. Sembla ser, per tant, que és un problema de les VTK, i per això vaig haver de canviar aquest disseny pel definitiu.

4.3.2 Disseny final

El disseny final té les classes següents per cada mòdul:

- **Interfície gràfica:**
  - ColorTableItem
  - MirrorSetup
  - VolumeSetup
  - MagicMirrorsSetup
- **Mòdul de control:**
  - UpdateMirrorsCommand
  - MagicMirrors
- **Mòdul de volums:**
  - VolumVTKUpdater
  - VolumVTK

A continuació explicaré el disseny a nivell general i després cada mòdul.

4.3.2.1 General

La Figura 4.9 representa el diagrama de classes general de l'aplicació. Només es veuen les classes que he fet jo, les seves superclasses i les relacions entre elles. També hi apareix la classe MainWindow perquè és la finestra principal de la interfície de la plataforma i és la que inicia la part de l'aplicació corresponent a aquest projecte. Per cada mòdul hi ha un diagrama més detallat.

A la classe MainWindow hi ha el mètode startMagicMirrors(), que és un *slot* privat. Aquest mètode crea la interfície gràfica per MagicMirrors (objecte MagicMirrorsSetup) i l'afegeix al *toolbox* de la interfície de la plataforma. Després crea un objecte MagicMirrors i a continuació connecta els *signals* i *slots* entre MagicMirrorsSetup i MagicMirrors:

- De MagicMirrorsSetup a MagicMirrors s'enviaran els canvis a les funcions de transferència i la visibilitat, sempre especificant el Mirall i volum que s'han de canviar.
- De MagicMirrors a MagicMirrorsSetup s'enviarà informació sobre el valors mínim i màxim de cada volum afegit, per poder mostrar aquesta informació a la interfície gràfica.

Un cop fet això s'afegeix el model obert a MagicMirrors, i si s'ha registrat amb un altre model també s'afegeix el segon, amb la transformació ja aplicada[3].

Aquest mètode es crida des d'un menú de la finestra principal, però aquest menú està desactivat fins que s'obri un volum, perquè l'aplicació de Magic Mirrors necessita com a mínim un volum per funcionar. Un cop obert un model l'usuari pot

[3] En aquests moments la plataforma només permet registrar dos volums, i per tant és el màxim que es pot afegir des d'aquest mètode, però tots els mòduls de la part de Magic Mirrors estan preparats per treballar amb *n* volums.

iniciar directament els Magic Mirrors a través de l'opció del menú o fer primer un registre amb un altre model i després iniciar els Magic Mirrors.

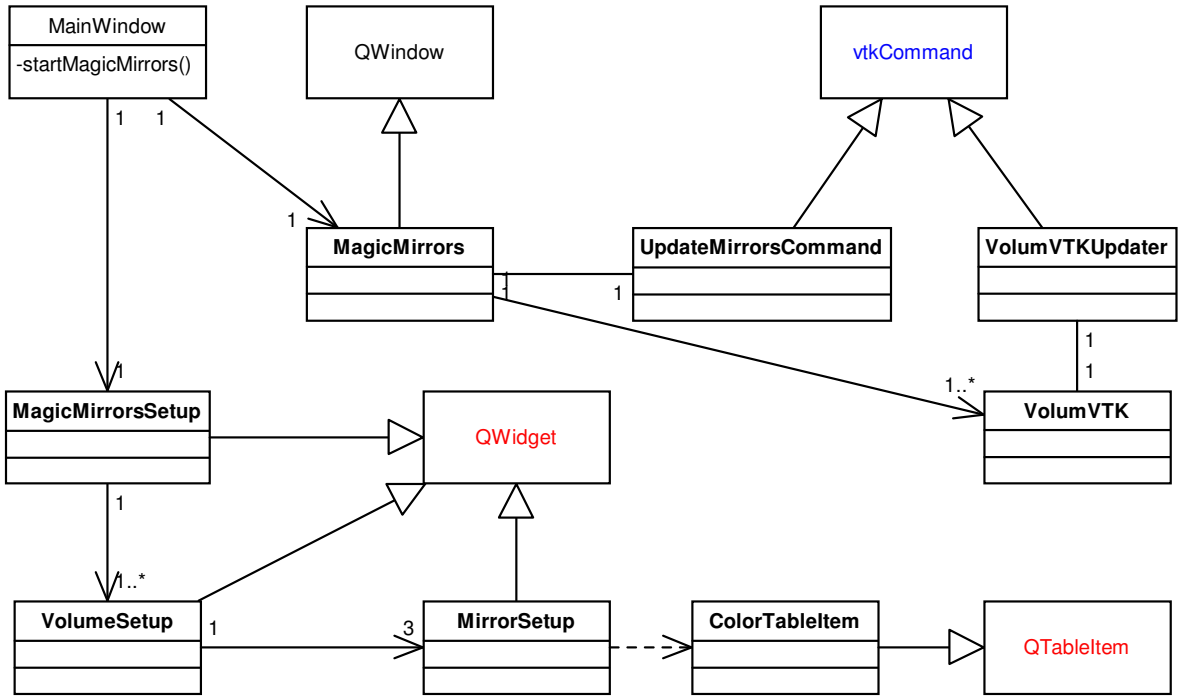


Figura 4.9. Diagrama de classes general (disseny final).

4.3.2.2 Interfície Gràfica d'Usuari

La interfície gràfica ha estat implementada amb Qt. Les classes d'aquest mòdul són subclasses de QWidget excepte ColorTableItem, que ho és de QTableItem.

El diagrama de classes per aquest mòdul es pot veure a la Figura 4.10. No hi surten tots els tipus de *widgets* que faig servir com a elements de la interfície perquè quedaria un diagrama molt embolicat i no aportaria informació útil, però si que hi surten les classes que faig servir com a paràmetres de mètodes, *slots* o *signals*.

L'estructura és la següent: la classe MagicMirrorsSetup té un VolumeSetup per cada volum, i VolumeSetup té un MirrorSetup pel volum central i un per cada Mirall. El ColorTableItem és per afegir-lo a una QTable de MirrorSetup.

A continuació, la descripció de les classes i mètodes.

4.3.2.2.1 ColorTableItem

Aquesta classe no és res més que un QTableItem que es pinta amb un determinat color, que és el que guarda com a atribut.

Mètodes:

- paint(QPainter, QColorGroup, QRect, selected: booleà) → Reimplementat de QTableItem. Pinta el ColorTableItem del color que té assignat.
- setColor(QColor) → Assigna aquest color al ColorTableItem.

- getColor(): QColor → Retorna el color que té assignat el ColorTableItem.

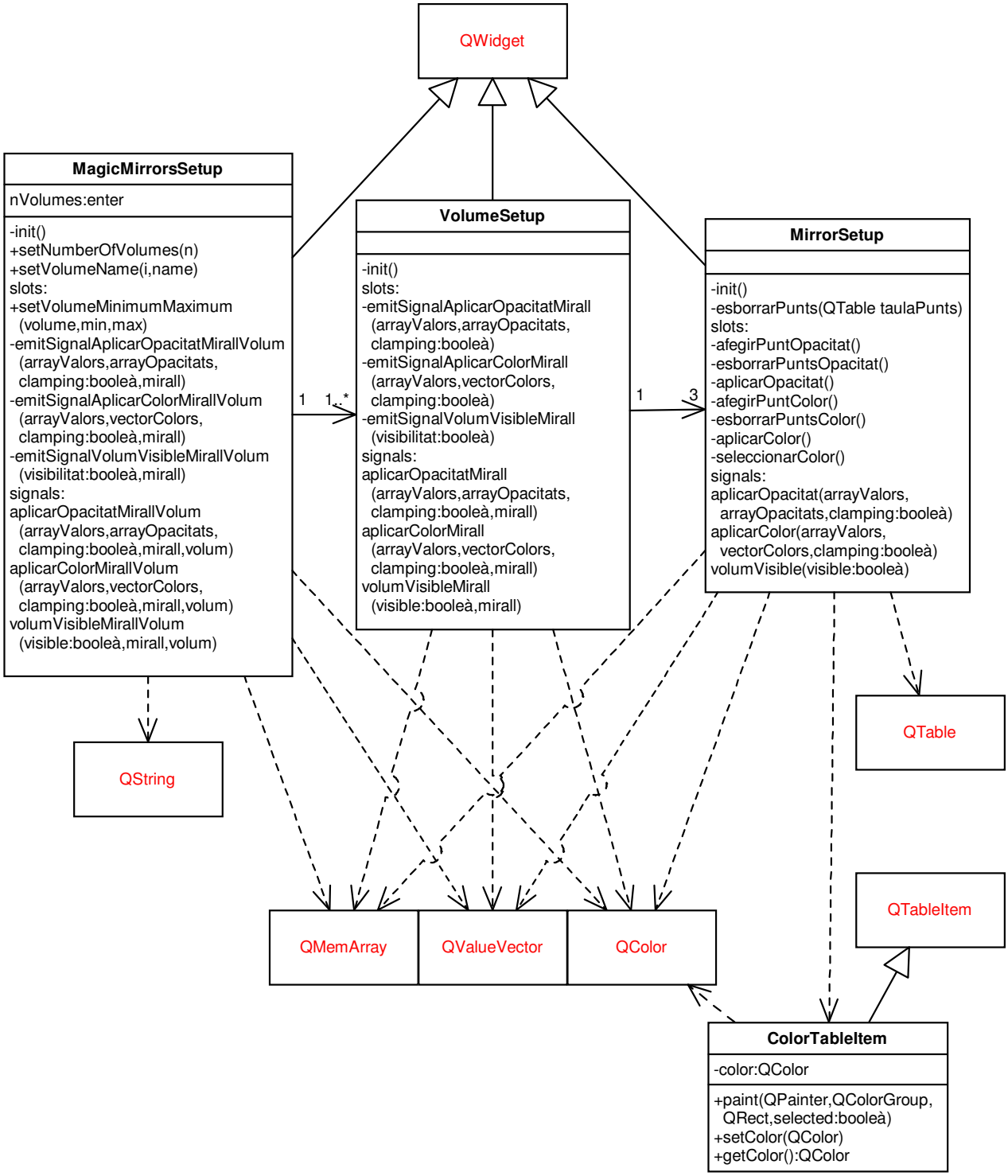


Figura 4.10. Diagrama de classes de la interfície gràfica d'usuari (disseny final).

4.3.2.2.2 MirrorSetup

Aquesta classe té la interfície gràfica necessària per configurar un Mirall o el volum central. Es pot triar si es vol configurar la funció de transferència d'opacitat o la de

color. Per cada una d'aquestes es veu una taula amb els valors actuals de la funció, i els controls per esborrar i afegir punts a la funció. Aquests punts són els que determinen la funció. L'opció *clamping* determina com s'extrapolen els punts; si està marcada s'extrapola amb els valors dels extrems i si no està marcada s'extrapola a 0.

Valor de propietat	Opacitat
812	0,844
1021	0,82
1272	0,208
3794	0,015

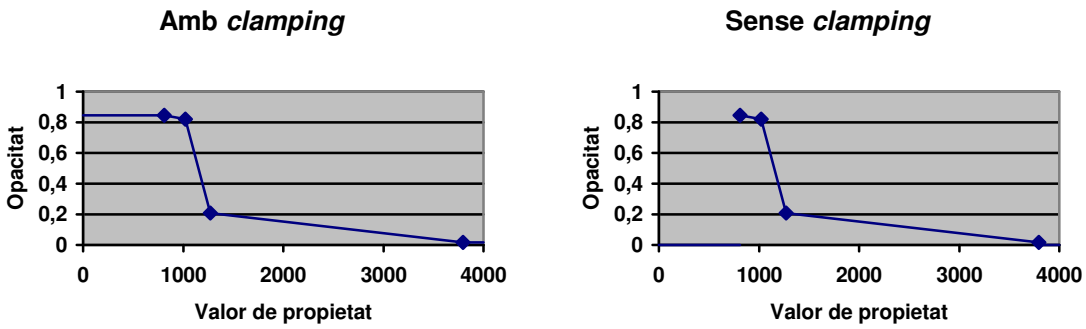


Figura 4.11. Exemple de funció de transferència.

En el cas de la funció de transferència d'opacitat els punts estan formats pel valor de propietat (enter positiu) i el nivell d'opacitat (real en el rang [0..1]). En el cas de la de color són el valor de propietat i el color (QColor).

Mètodes:

- init() → Aquest mètode el crida el constructor. Fa algunes inicialitzacions i connecta *signals* amb altres *signals* (el Qt Designer només permet connectar *signals* amb *slots*).
- esborrarPunts(QTable taulaPunts) → Aquest mètode esborra els punts (files) seleccionats de la taula que rep com a paràmetre.

Slots:

- afegirPuntOpacitat() → Afegeix una fila a la taula de punts d'opacitat que representa un punt de la funció de transferència d'opacitats.
- esborrarPuntsOpacitat() → Esborra les files seleccionades de la taula de punts d'opacitat cridant esborrarPunts().
- aplicarOpacitat() → Emet el *signal* aplicarOpacitat() amb els valors de la taula de punts d'opacitat i del *checkbox* de *clamping* d'opacitat.
- afegirPuntColor() → Afegeix una fila a la taula de punts de color que representa un punt de la funció de transferència de colors. El color a la taula es representa amb un ColorTableItem.
- esborrarPuntsColor() → Esborra les files seleccionades de la taula de punts de color cridant esborrarPunts().
- aplicarColor() → Emet el *signal* aplicarColor() amb els valors de la taula de punts de color i del *checkbox* de *clamping* de color.
- seleccionarColor() → Obre un quadre de diàleg per seleccionar un color i pinta el botó de seleccionar color del color triat.

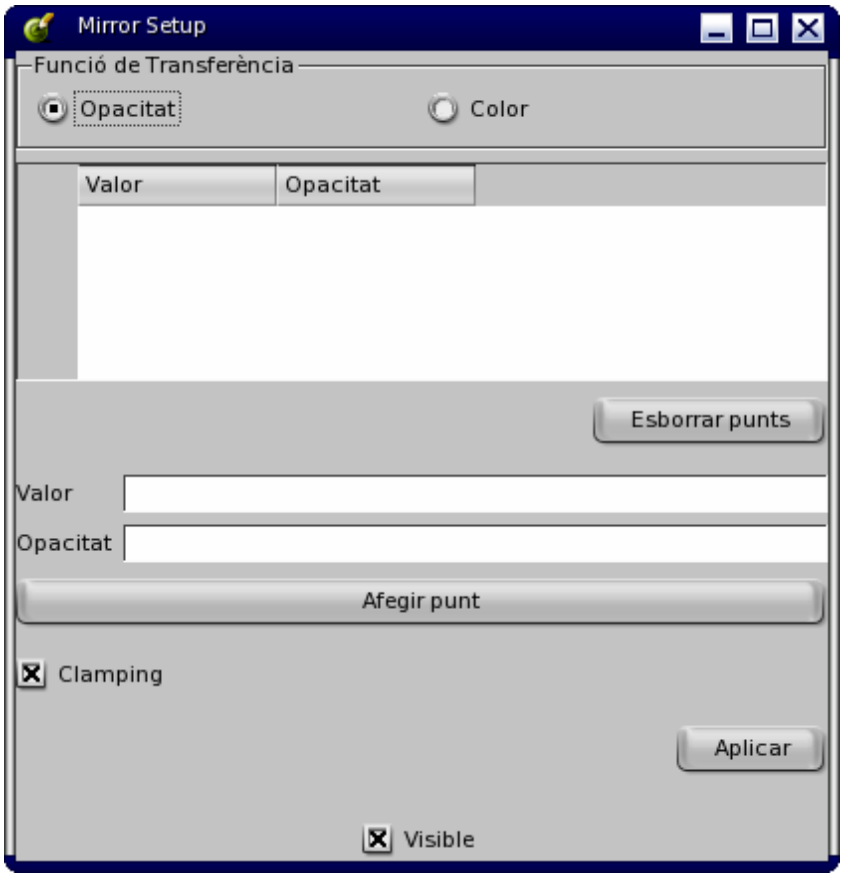


Figura 4.12. MirrorSetup.

Signals:

- aplicarOpacitat(arrayValors, arrayOpacitats, clamping: booleà) → El primer paràmetre és un QMemArray amb els valors de propietat, el segon un QMemArray amb els valors d'opacitat i el tercer indica si hi ha d'haver *clamping* o no.
- aplicarColor(arrayValors, vectorColors, clamping: booleà) → El primer paràmetre és un QMemArray amb els valors de propietat, el segon un QValueVector amb els colors i el tercer indica si hi ha d'haver *clamping* o no.
- volumVisible(visible: booleà) → S'emet quan es modifica el *checkbox* "Visible", que vol dir si el volum ha de ser visible o no al Mirall actual.

4.3.2.2.3 VolumeSetup

Aquesta classe té la interfície gràfica necessària per configurar un volum. Es pot triar el Mirall[4] que es vol configurar clicant a la pestanya corresponent. A cada pestanya hi ha un MirrorSetup per configurar les funcions de transferència.

Mètodes:

- init() → Aquest mètode el crida el constructor. Fa algunes connexions de *signals* i *slots*.

[4] El volum central es considera com un Mirall més, ja que té les mateixes propietats a nivell de funcions de transferència.

Figura 4.13. VolumeSetup.

Slots:

- emitSignalAplicarOpacitatMirall(arrayValors, arrayOpacitats, clamping: booleà) → Emet el *signal* aplicarOpacitatMirall() amb els valors dels paràmetres i el Mirall seleccionat.
- emitSignalAplicarColorMirall(arrayValors, vectorColors, clamping: booleà) → Emet el *signal* aplicarColorMirall() amb els valors dels paràmetres i el Mirall seleccionat.
- emitSignalVolumVisibleMirall(visibilitat: booleà) → Emet el *signal* volumVisibleMirall() amb el valor del paràmetre i el Mirall seleccionat.

Signals:

- aplicarOpacitatMirall(arrayValors, arrayOpacitats, clamping: booleà, mirall) → El primer paràmetre és un QMemArray amb els valors de propietat, el segon un QMemArray amb els valors d'opacitat, el tercer indica si hi ha d'haver *clamping* o no i el quart és el Mirall al qual s'ha d'aplicar l'opacitat.
- aplicarColorMirall(arrayValors, vectorColors, clamping: booleà, mirall) → El primer paràmetre és un QMemArray amb els valors de propietat, el segon un QValueVector amb els colors, el tercer indica si hi ha d'haver *clamping* o no i el quart és el Mirall al qual s'ha d'aplicar el color.
- volumVisibleMirall(visible: booleà, mirall) → El primer paràmetre vol dir si el volum ha de ser visible o no i el segon és el Mirall al qual afecta.

4.3.2.2.4 MagicMirrorsSetup

Aquesta classe té la interfície gràfica necessària per configurar l'aplicació de MagicMirrors. Es pot triar el volum que es vol configurar amb un *combo*. Per cada opció del *combo* hi ha una pàgina amb un VolumeSetup per configurar el volum.

Inicialment el MagicMirrorsSetup es crea amb una sola pàgina, per un volum, però es pot canviar el nombre de volums una vegada per configurar els volums que faci falta. També es poden assignar noms als volums (per exemple el nom del fitxer) que sortiran al *combo*. També pot oferir informació sobre el mínim i el màxim de cada volum, si rep aquesta informació a través d'un *slot*.

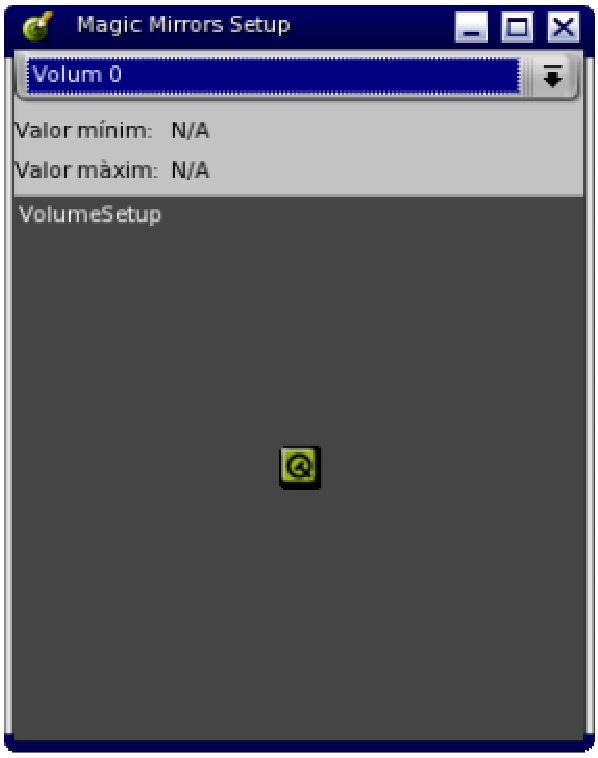


Figura 4.14. MagicMirrorsSetup.

Mètodes:

- init() → Aquest mètode el crida el constructor. Inicialitza el nombre de volums a 1 i fa algunes connexions de *signals* i *slots*.
- setNumberOfVolumes(n) → Posa el nombre de volums a *n* i crea les opcions del *combo*, pàgines i VolumeSetups addicionals necessaris per configurar-los. Només ho fa quan es crida per primera vegada.
- setVolumeName(i, name) → Posa el nom *name* al *combo* per al volum *i*.

Slots:

- setVolumeMinimumMaximum(volume, min, max) → Assigna els valors de propietat mínim i màxim del volum *volume*.
- emitSignalAplicarOpacitatMirallVolum(arrayValors, arrayOpacitats, clamping: booleà, mirall) → Emet el *signal* aplicarOpacitatMirallVolum() amb els valors dels paràmetres i el volum seleccionat.
- emitSignalAplicarColorMirallVolum(arrayValors, vectorColors, clamping: booleà, mirall) → Emet el *signal* aplicarColorMirallVolum() amb els valors dels paràmetres i el volum seleccionat.
- emitSignalVolumVisibleMirallVolum(visibilitat: booleà, mirall) → Emet el *signal* volumVisibleMirallVolum() amb els valors dels paràmetres i el volum seleccionat.

Signals:

- aplicarOpacitatMirallVolum(arrayValors, arrayOpacitats, clamping: booleà, mirall, volum) → El primer paràmetre és un QMemArray amb els valors de propietat, el segon un QMemArray amb els valors d'opacitat, el tercer indica si hi ha d'haver *clamping* o no, el quart és el Mirall al qual s'ha d'aplicar l'opacitat i el cinquè el volum implicat.

- aplicarColorMirallVolum(arrayValors, vectorColors, clamping: booleà, mirall, volum) → El primer paràmetre és un QMemArray amb els valors de propietat, el segon un QValueVector amb els colors, el tercer indica si hi ha d'haver *clamping* o no, el quart és el Mirall al qual s'ha d'aplicar el color i el cinquè el volum implicat.
- volumVisibleMirallVolum(visible: booleà, mirall, volum) → El primer paràmetre vol dir si el volum ha de ser visible o no, el segon és el Mirall al qual afecta i el tercer el volum implicat.

4.3.2.3 Mòdul de Control

El mòdul de control és el que controla l'execució de l'aplicació Magic Mirrors. Es comunica amb la interfície gràfica i utilitza el mòdul de volums per gestionar els volums.

El diagrama de classes per aquest mòdul es pot veure a la Figura 4.15. La classe principal és MagicMirrors, que conté tots els mètodes de visualització i modificació dels volums. La segona classe és MagicMirrorsUpdater, que serveix per avisar a MagicMirrors quan s'han d'actualitzar els miralls.

A continuació, la descripció de les classes i mètodes.

4.3.2.3.1 MagicMirrorsUpdater

Aquesta és una subclasse de vtkCommand, que és una classe que serveix per tractar events de VTK. Com que MagicMirrorsUpdater és una subclasse de vtkCommand podrà observar un altre objecte per veure quan produeix un determinat event, i llavors tractar-lo. En aquest cas serà associat events dels objectes que fa servir MagicMirrors (concretament quan s'acaba la interacció amb un volum i quan un dels volums d'un model registrat és modificat) i el tractament consistirà en actualitzar els Miralls o els altres volums del model registrat. Per poder fer la seva feina guarda una referència de l'objecte MagicMirrors que ha d'actualitzar.

Mètodes:

- New(): MagicMirrorsUpdater → Aquest mètode de classe retorna un objecte MagicMirrorsUpdater. És la manera de crear objectes a l'estil VTK.
- Execute(caller, eventId, callData) → Aquest és el mètode que executa la funció de la classe. Actualitza els Miralls del MagicMirrors associat cridant el seu mètode updateMirrors() i actualitza els volums d'un model registrat fent que coincideixin les seves posicions cridant el mètode ajustarModelRegistrat().
- setMagicMirrors(MagicMirrors) → Assigna el MagicMirrors del paràmetre a l'objecte MagicMirrorsUpdater. Aquest objecte MagicMirrors serà el que s'actualitzarà.

4.3.2.3.2 MagicMirrors

Aquesta és la classe principal d'aquest mòdul de control. És una subclasse de QWindow i amb això es pot integrar a la plataforma com una finestra i pot fer servir el mecanisme de *signals i slots* de Qt, que li servirà per comunicar-se amb la interfície gràfica.

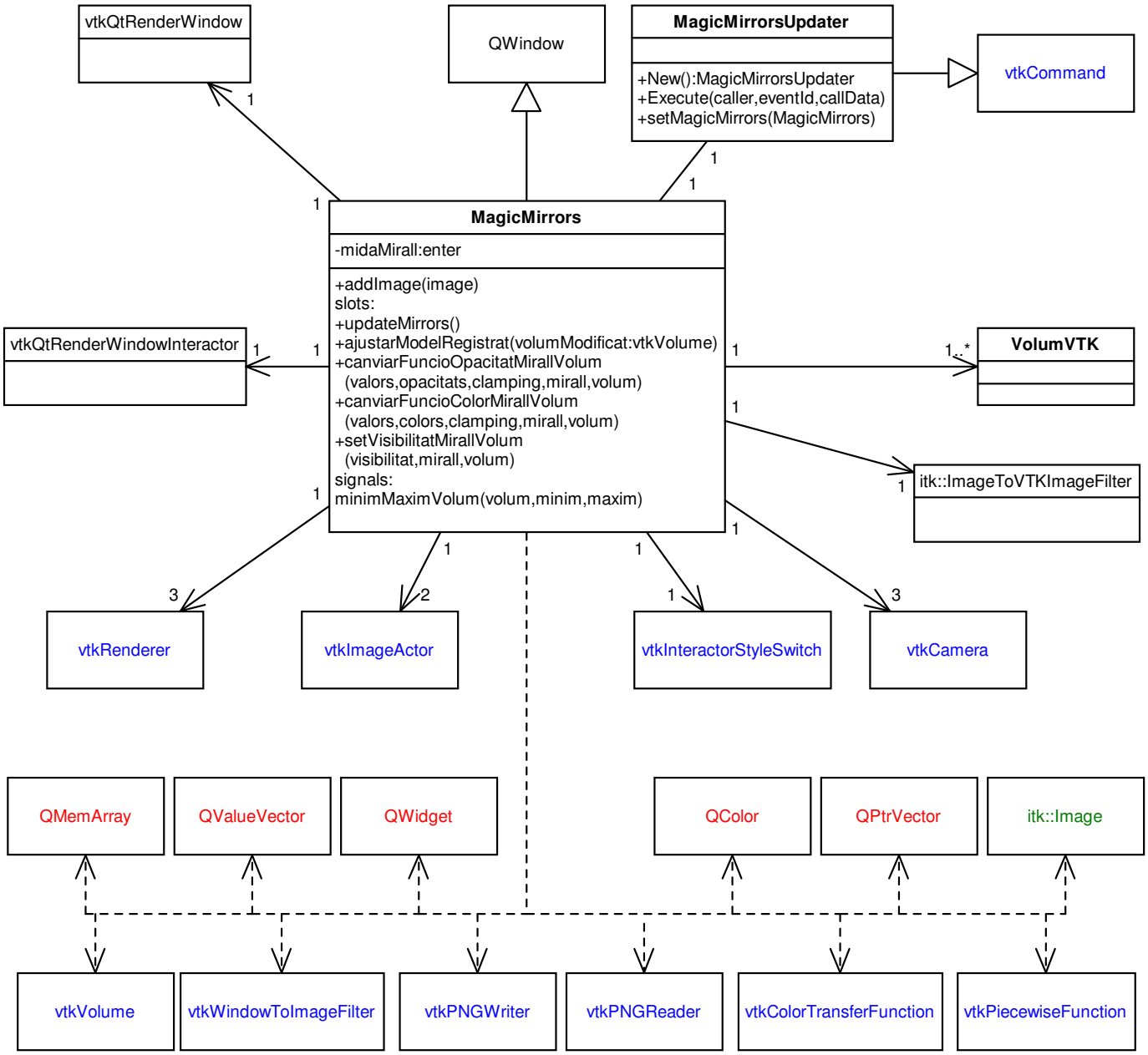


Figura 4.15. Diagrama de classes del mòdul de control (disseny final).

Com es pot veure al diagrama de classes, per fer la seva feina, MagicMirrors fa servir classes de Qt, d'ITK, de VTK i algunes que ja hi havia a la plataforma.

Aquest diagrama és per una implementació amb 2 Miralls, i d'aquí surten tots els dosos i tresos de les associacions: 2 quan es necessita un objecte per cada Mirall i 3 quan es necessita un objecte per cada Mirall i un pel volum central. Amb un Mirall més s'incrementarien en 1 tots els dosos i tresos.

Hi ha una referència a una vtkQtRenderWindow. És la finestra principal on es veuran els volums i els Miralls des de la camera principal. També es fan servir finestres temporals al mètode updateMirrors() per visualitzar el contingut dels Miralls. La classe vtkQtRenderWindow té les mateixes característiques que vtkRenderWindow però a més es pot integrar en una interfície de Qt.

Hi ha una referència a un vtkQtRenderWindowInteractor que serveix per interactuar amb la finestra principal. Mitjançant el ratolí i el teclat l'usuari pot girar el volum, moure la camera, etc. vtkQtRenderWindowInteractor és una subclasse de vtkRenderWindowInteractor que treballa amb una vtkQtRenderWindow. També hi ha una referència a un vtkInteractorStyleSwitch que defineix l'estil d'interacció.

Hi ha 3 referències a vtkRenderers, un per cada Mirall. Són els que s'encarreguen de dibuixar la imatge a les finestres.

També hi ha 3 vtkCameras, una per la finestra principal i una per cada Mirall.

Es fan servir vtkWindowToImageFilters per agafar les imatges de les finestres temporals dels Miralls i convertir-les en textures. Aquestes textures les donen a uns vtkPNGWriters que les guarden en fitxers temporals al disc, per ser llegides després.

Els Miralls són 2 vtkImageActors, que són uns plans que visualitzen una imatge 2D en un entorn 3D. Són els Miralls que es veuen a la finestra principal, al voltant del volum central. En aquest cas visualitzen la textura que els és assignada per vtkPNGReaders, que llegeixen els fitxers temporals de les textures dels Miralls.

MagicMirrors també guarda una referència d'un itk::ImageToVTKImageFilter, que és un filtre que converteix una imatge d'ITK en una imatge de VTK. Es crea al principi i després serveix per tots els volums afegits.

Finalment, per guardar els volums utilitza un QPtrVector (un vector de Qt que guarda punters als seus elements) de VolumVTKs. Aquests VolumVTKs encapsulen el tractament dels volums, i poden retornar el vtkVolume desitjat amb el mètode getVolume().

A més a més també hi ha un MagicMirrorsUpdater, que s'associa a l'event de final d'interacció del vtkInteractorStyleSwitch i a l'event de modificació de cada volum afegit. D'aquesta manera, cada vegada que s'acaba de girar el model s'actualitzen els Miralls i quan es gira un volum d'un model registrat també es giren els altres.

Mètodes:

- addImage(image) → Aquest mètode s'encarrega d'afegir un volum a MagicMirrors. *image* és una imatge d'ITK, per tant, abans de fer res més, s'ha de passar la imatge pel filtre conversor, i després es crea un VolumVTK amb la imatge de VTK resultant. Després afegeix els vtkVolumes que encapsula el VolumVTK als *renderers* (un volum a cada *renderer*). Actualitza els Miralls i emet un *signal* amb els valors de propietat mínim i màxim del volum. Aquest mètode també posiciona les cameres quan s'afegeix el primer volum, a una distància perquè el volum es vegi bé.

Slots:

- updateMirrors() → Com el seu nom indica, s'encarrega d'actualitzar els Miralls. Aquest mètode el crida MagicMirrorsUpdater. Per actualitzar els Miralls guarda uns fitxers temporals amb les textures d'aquests i després llegeix els fitxers per enganxar les textures als plans dels Miralls. Al final de la primera execució posiciona els Miralls.
- ajustarModelRegistrat(volumModificat:vtkVolume) → Aquest mètode s'encarrega que quan un dels volums d'un model registrat és modificat

(girat, mogut, etc.) s'apliqui la mateixa transformació als altres volums del mateix model registrat. Aquest mètode és cridat pel MagicMirrorsUpdater.

- canviarFuncioOpacitatMirallVolum(valors, opacitats, clamping, mirall, volum) → Crea una funció de transferència d'opacitat a partir dels valors i opacitats i el paràmetre *clamping* i l'assigna al volum indicat per al Mirall indicat.
- canviarFuncioColorMirallVolum(valors, colors, clamping, mirall, volum) → Crea una funció de transferència de colors a partir dels valors i colors i el paràmetre *clamping* i l'assigna al volum indicat per al Mirall indicat.
- setVisibilitatMirallVolum(visibilitat, mirall, volum) → Fa que el volum indicat sigui visible o no al Mirall indicat segons el que digui el paràmetre *visibilitat*.

Signals:

- minimMaximVolum(volum, minim, maxim) → Dóna informació sobre els valors de propietat mínim i màxim del volum *volum*.

4.3.2.4 Mòdul de Volums

El mòdul de volums encapsula en una sola classe tots els vtkVolumes que corresponen a un sol volum conceptualment. Es necessita un vtkVolume pel volum central i un per cada Mirall, perquè és l'única manera de tenir funcions de transferència diferents a cada lloc. Els vtkVolumes que corresponen a un sol volum conceptualment són els que representen les mateixes dades i per tant han estat creats a partir de la mateixa imatge de VTK.

El diagrama de classes per aquest mòdul es pot veure a la Figura 4.16. La classe principal és VolumVTK, que conté tots els mètodes per tractar amb els vtkVolumes relacionats de forma senzilla. La segona classe és VolumVTKUpdater, que serveix per avisar a VolumVTK quan s'han d'actualitzar els volums. Entre totes dues i amb l'ajuda d'altres formen aquest mòdul de volums.

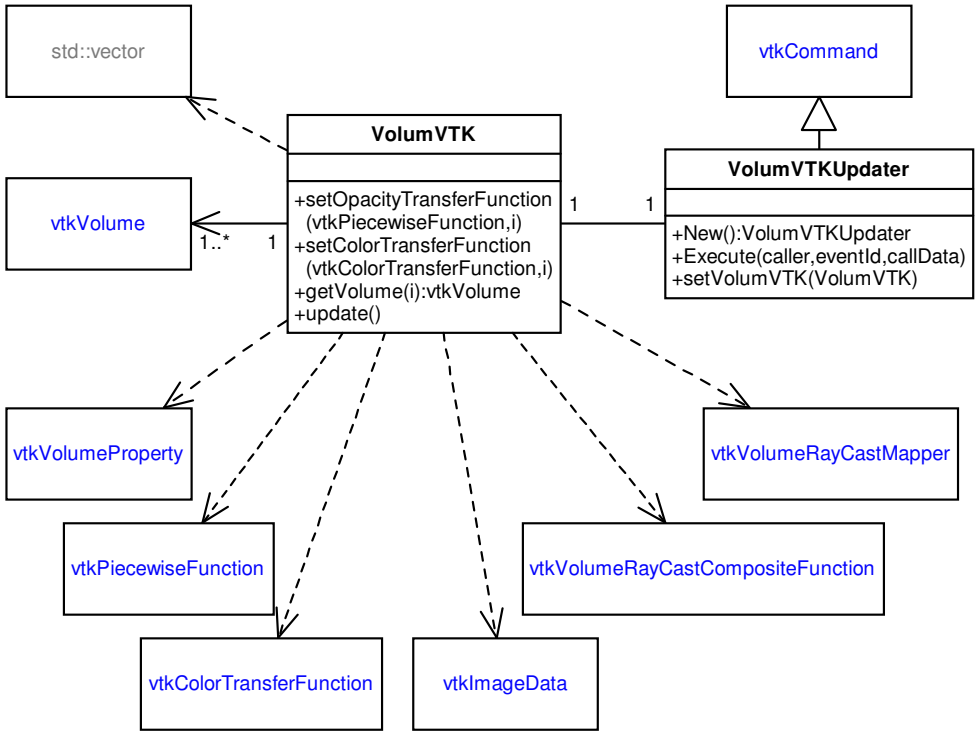


Figura 4.16. Diagrama de classes del mòdul de volums (disseny final).

A continuació, la descripció de les classes i mètodes.

4.3.2.4.1 VolumVTKUpdater

Aquesta també és una subclasse de vtkCommand. En aquest cas un VolumVTKUpdater serà associat a un event d'un dels vtkVolumes de VolumVTK (concretament quan es modifica un volum) i el tractament de l'event consistirà en actualitzar els altres vtkVolumes. Això servirà perquè quan l'usuari giri el volum central també giraran els dels Miralls. S'han de fer girar manualment perquè són objectes diferents internament, però des del punt de vista de l'usuari són el mateix. Per poder fer la seva feina, VolumVTKUpdater guarda una referència de l'objecte VolumVTK que ha d'actualitzar.

Mètodes:

- New(): VolumVTKUpdater → Aquest mètode de classe retorna un objecte VolumVTKUpdater. És la manera de crear objectes a l'estil VTK.
- Execute(caller, eventId, callData) → Aquest és el mètode que executa la funció de la classe. Actualitza els vtkVolumes del VolumVTK associat cridant el seu mètode update().
- setVolumVTK(VolumVTK) → Assigna el VolumVTK del paràmetre a l'objecte VolumVTKUpdater. Aquest objecte VolumVTK serà el que s'actualitzarà.

4.3.2.4.2 VolumVTK

Aquesta és la classe principal d'aquest mòdul de volums. Fa servir vàries classes de VTK i el vector de STL per fer la seva feina.

El vector serveix per guardar els vtkVolumes i ells mateixos guarden referències dels seus *mappers* i propietats.

VolumVTK també guarda una referència a un VolumVTKUpdater, que s'associa a l'event de modificació del volum central. D'aquesta manera, cada vegada que es modifica el volum central canviant la seva transformació (per exemple fent una rotació), la mateixa transformació s'aplica als altres volums.

El constructor d'aquesta classe rep com a paràmetre una imatge de VTK (vtkImage) i el nombre de volums que ha de crear. Llavors crea el vector amb la mida justa i crea cada volum amb un *mapper* i una propietat associats. Els *mappers* són de la classe vtkVolumeRayCastMapper, per fer la visualització amb *Ray Casting*. Les propietats són de la classe vtkVolumeProperty, i el constructor crea unes funcions de transferència per defecte per assignar a aquestes propietats. També crea el VolumVTKUpdater i l'associa a l'objecte i event mencionats anteriorment.

Mètodes:

- setOpacityTransferFunction(vtkPiecewiseFunction, i) → Assigna la funció de tipus vtkPiecewiseFunction com a funció de transferència d'opacitat per la vtkProperty del vtkVolume *i*.
- setColorTransferFunction(vtkColorTransferFunction, i) → Assigna la funció de tipus vtkColorTransferFunction com a funció de transferència de color per la vtkProperty del vtkVolume *i*.
- getVolume(i): vtkVolume → Retorna el vtkVolume de la posició *i* del vector.

- update() → Actualitza els vtkVolumes a partir de la posició 1 del vector, corresponents als Miralls. Per fer-ho els assigna la mateixa matriu de transformació que té el de la posició 0, corresponent al volum central.

5 Implementació

En aquest capítol explicarem el funcionament de l'aplicació a través de les diferents interfícies, què vol dir cada paràmetre o opció que es pugui triar i l'efecte que té sobre la visualització.

5.1 Iniciant l'Aplicació

Un cop engegada l'aplicació apareix la interfície gràfica d'usuari de la plataforma de visualització d'imatges mèdiques, la qual es pot veure a la Figura 5.1.

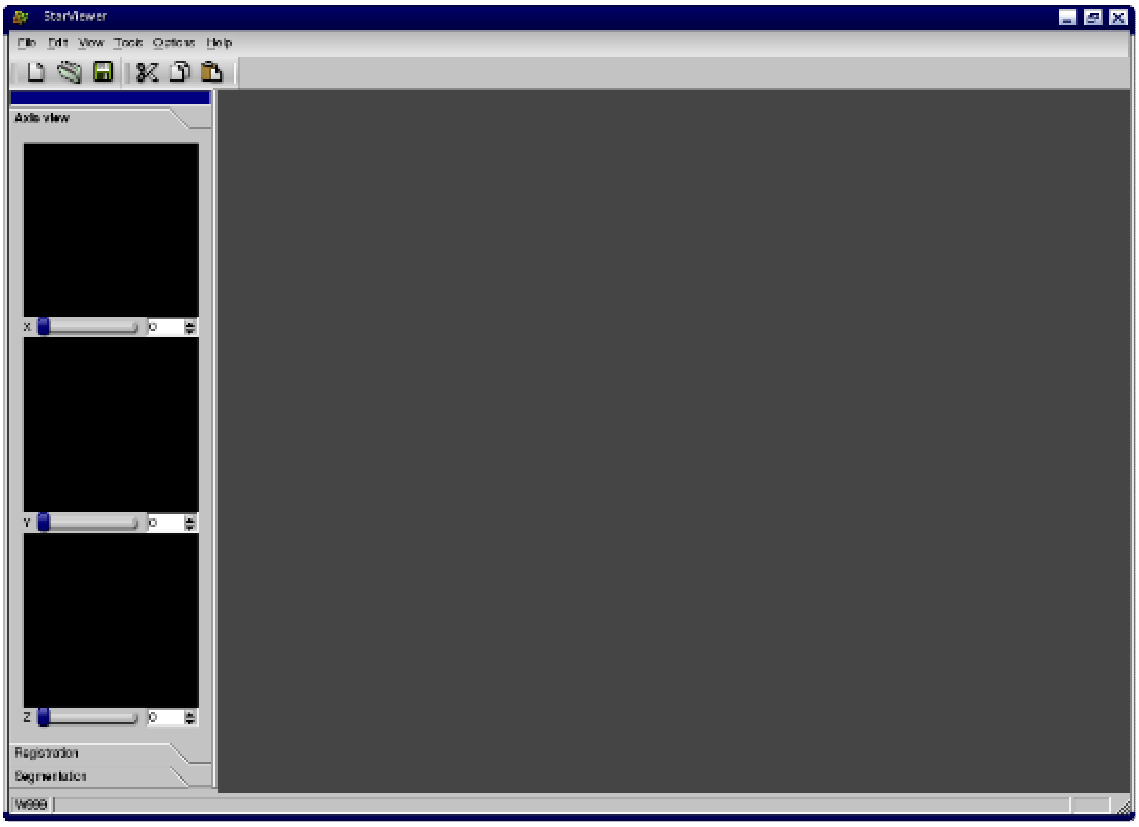


Figura 5.1. Plataforma de visualització d'imatges mèdiques.

En aquesta interfície hi ha diversos menús i botons, però el que interessa per ara és obrir un model. Això es pot fer clicant al botó d'obrir o a través del menú "Fitxer" ▸ "Obrir...". Al quadre de diàleg que apareix només cal buscar el model i obrir-lo.

Per exemple, obrim el model "headsq.mhd". Llavors apareixeran les imatges de les llesques del model als quadrats de l'esquerra de la plataforma, com es veu a la Figura 5.3.

Llavors es poden iniciar els Magic Mirrors anant al menú "Eines" ▸ "Iniciar Magic Mirrors" (Figura 5.4). Aquest menú estarà desactivat mentre no hi hagi cap model obert.

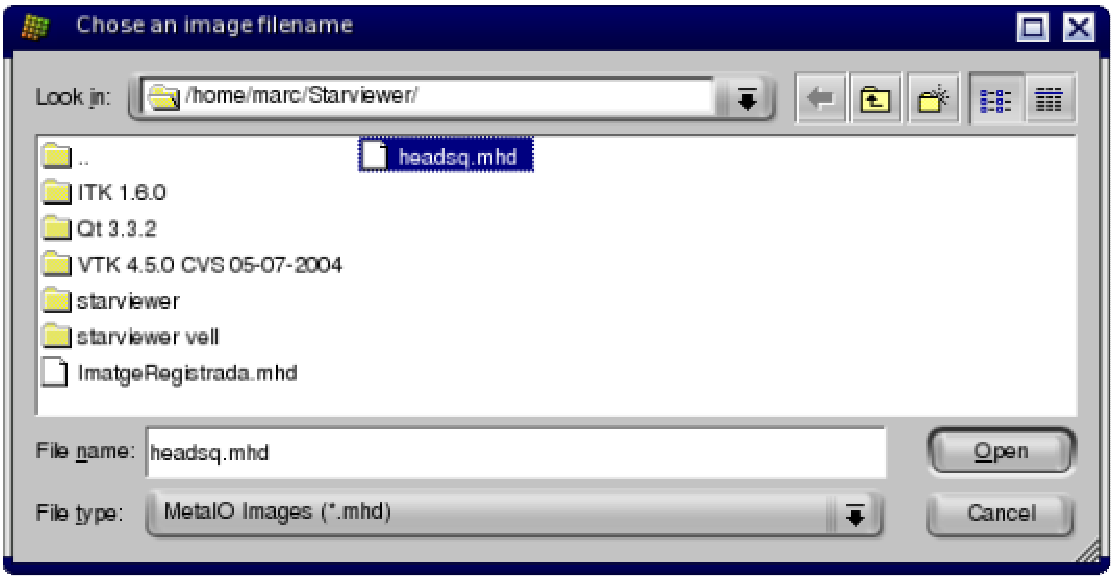


Figura 5.2. Quadre de diàleg per obrir un fitxer.

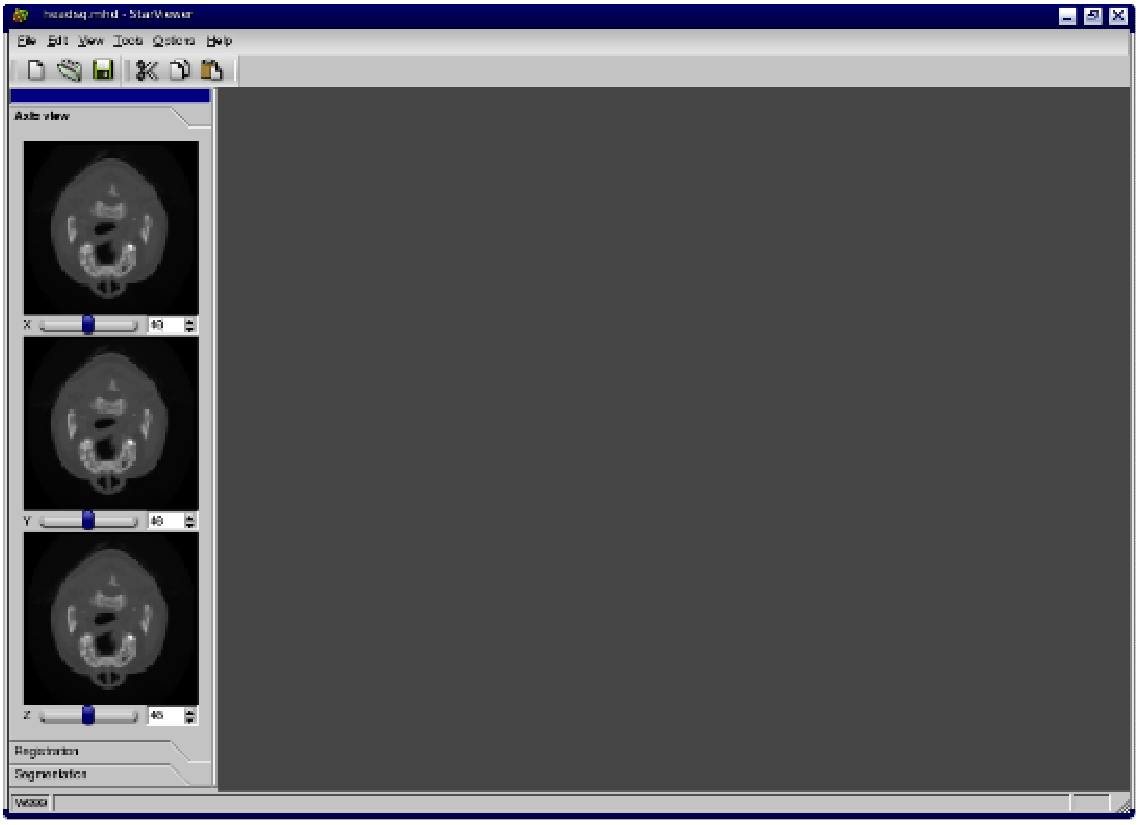


Figura 5.3. Plataforma després d'obrir el fitxer.

Figura 5.4. Iniciar Magic Mirrors.

Quan s'hagin iniciat els Magic Mirrors es veurà una imatge com la següent:

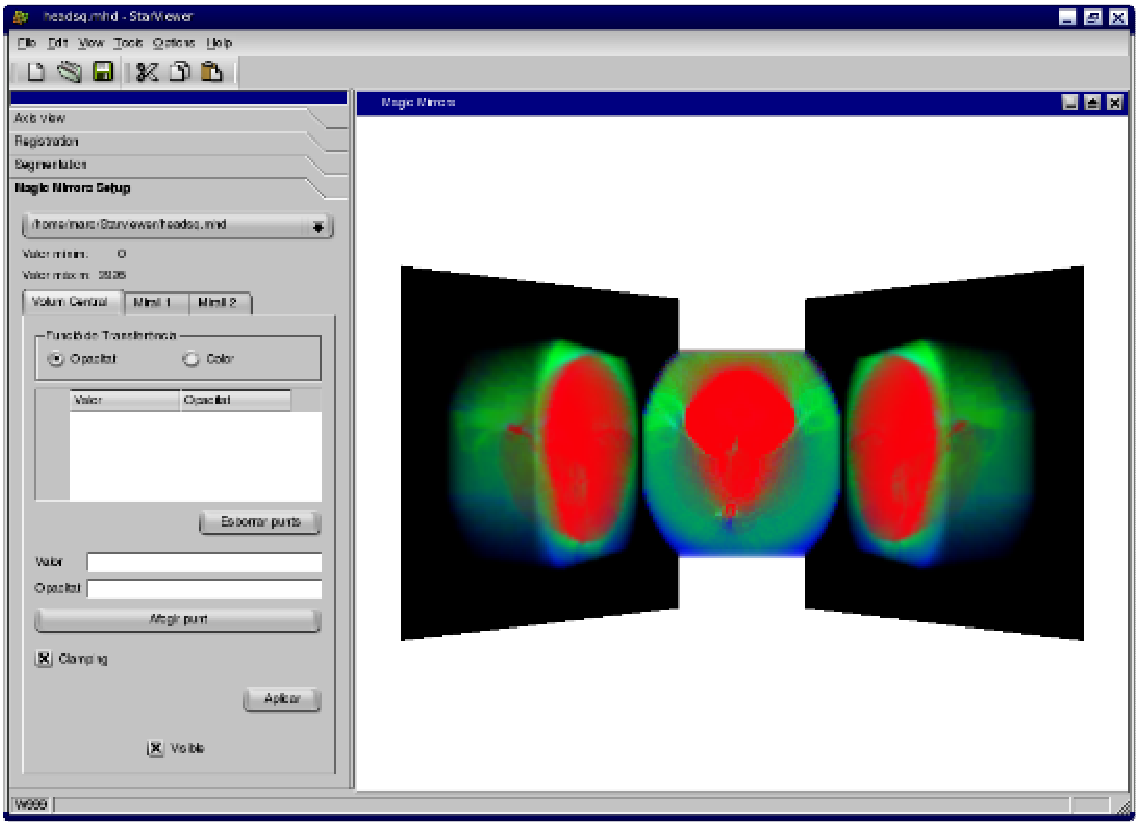


Figura 5.5. Magic Mirrors.

5.2 Treballant amb un Model Simple

5.2.1 Girant i movent el model

Un cop apareix la visualització es pot interactuar amb ella amb l'ajuda del ratolí i el teclat. Amb la configuració inicial el que es mou és la camera però prement la tecla 'A' es pot moure el model. Es pot tornar a la camera amb la tecla 'C'. Es pot girar el model o la camera lliurement amb el ratolí. Per fer-ho s'ha de clicar amb el botó esquerre –per moure el model s'ha de clicar al model– i arrossegar el ratolí cap a un costat; mentre es mantingui el botó premut la camera o el model anirà girant. Amb el botó del mig o la roda l'efecte és que es trasllada la camera o el model, i amb el botó dret es fa *zoom* o s'escala el model. També es pot triar l'estil d'interacció amb les tecles 'J' i 'T' (*joystick* i *trackball*); el primer és el que hi ha per defecte.

Mentre l'usuari està interactuant amb el model la imatge dels Miralls es manté estàtica (Figura 5.6), però quan para s'actualitzen els 2 Miralls (Figura 5.9). Per fer-ho es grava un fitxer temporal al disc per cada Mirall amb la textura del Mirall (Figura 5.7). Aquests fitxers s'esborren automàticament un cop ha finalitzat l'execució de l'aplicació.

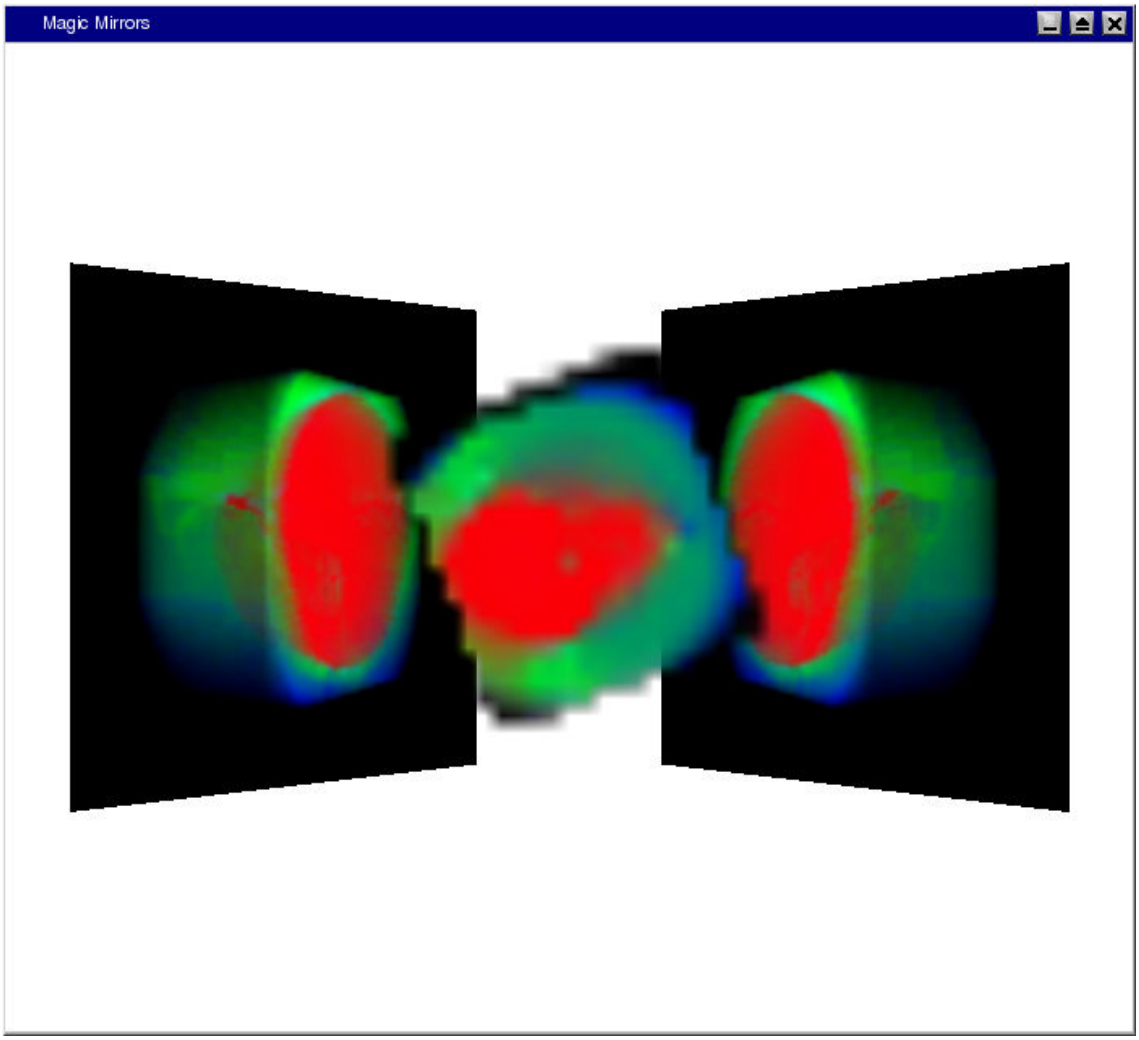


Figura 5.6. Girant el model.

Figura 5.7. Fitxers temporals.

5.2.2 Explicació de la interfície de control

Quan s'inicien els Magic Mirrors apareix una nova secció al *toolbox* de l'esquerra de la interfície de la plataforma. Aquesta secció conté la interfície de control dels Magic Mirrors. Aquesta interfície està formada per diferents elements:

- **Un *combo* amb el nom del model obert.** Aquest *combo* serveix per seleccionar el model que volem configurar. Només té utilitat pels models registrats, ja que obrint un model simple només té una opció.

Figura 5.8. *Combo* de models.

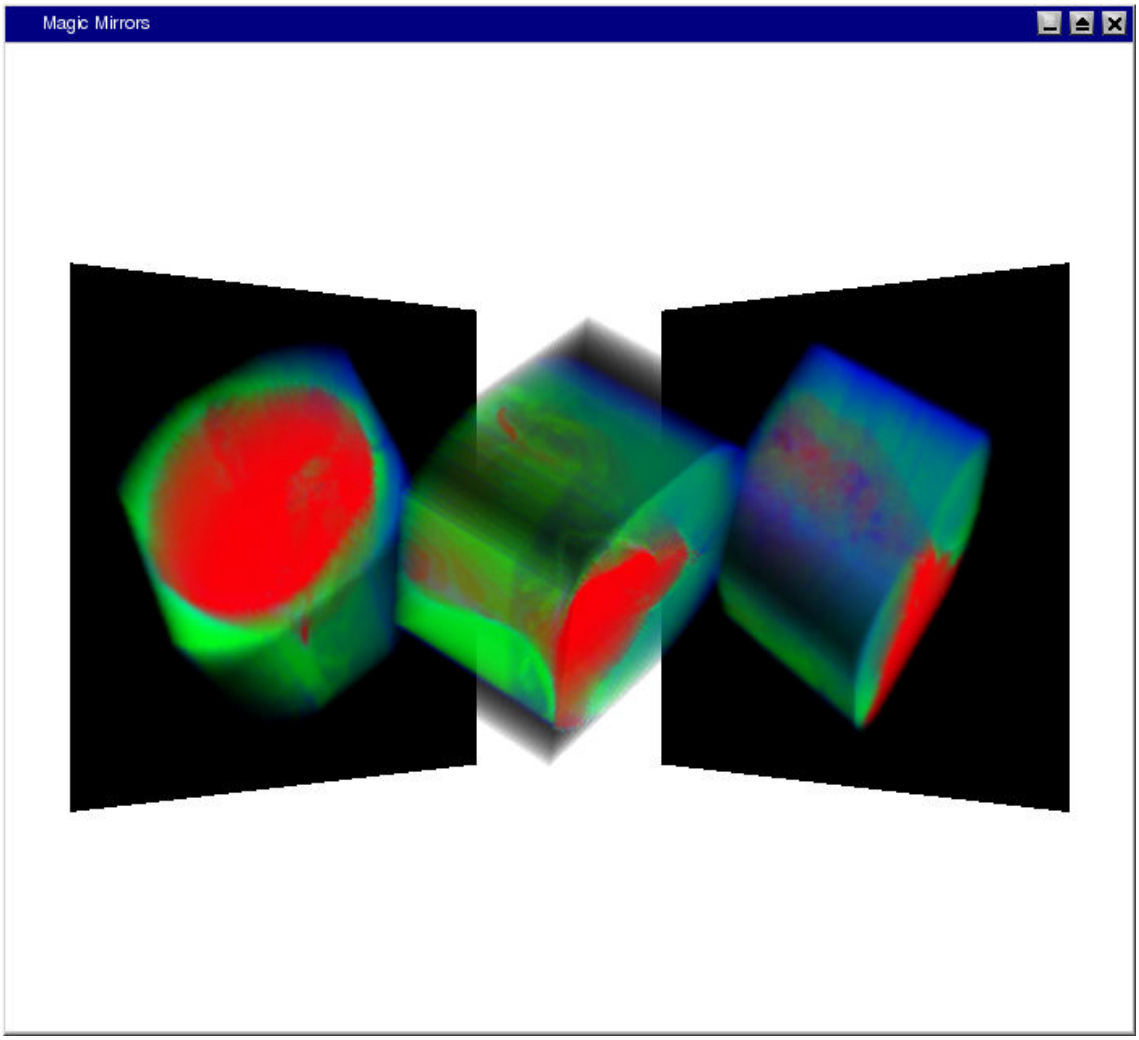


Figura 5.9. Model girat.

- **Valors mínim i màxim.** Aquests són els valors de propietat mínim i màxim del model seleccionat. Són purament informatius, per ajudar l'usuari a configurar les funcions de transferència, que d'aquesta manera sap el rang de valors que ha de tenir en compte.

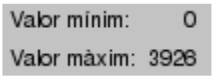


Figura 5.10. Valors mínim i màxim del model.

- **Pestanyes de selecció de Mirall.** Amb aquestes pestanyes l'usuari pot seleccionar a quin Mirall vol canviar la funció de transferència. També es pot seleccionar el volum central, que també és com un "Mirall" pel que fa a configuració.

Figura 5.11. Pestanyes de selecció de Mirall.

- ***Radio buttons* "Funció de Transferència" ("Opacitat" i "Color").** Amb aquests *radio buttons* l'usuari tria el tipus de funció de transferència que vol configurar: la d'opacitat o la de color. Segons quina triï apareixeran a sota els controls de configuració corresponents.

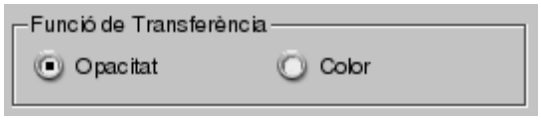


Figura 5.12. *Radio buttons* "Funció de Transferència".

- **Taula de punts de la funció de transferència.** En aquesta taula apareixeran els punts de la funció de transferència que vagi afegint l'usuari. Per la funció de transferència d'opacitat els punts estan formats per un valor de propietat i un grau d'opacitat de 0 a 1. Per la de color estan formats per un valor de propietat i un color. Per saber què volen dir aquests punts veure secció 4.3.2.2.2. Inicialment està buida, no mostra els valors per defecte.

Figura 5.13. Taula de punts de la funció de transferència d'opacitat.

- **Botó "Esborrar punts".** Aquest botó serveix per esborrar els punts seleccionats de la taula.

Figura 5.14. Botó "Esborrar punts".

- **Camps de text "Valor" i "Opacitat" (opacitat).** Aquí l'usuari introdueix el valor de propietat i el grau d'opacitat respectivament del punt que vol afegir.

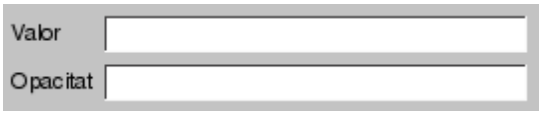


Figura 5.15. Camps de text "Valor" i "Opacitat".

- **Camp de text "Valor" i botó de seleccionar color (color).** Aquí l'usuari introdueix el valor de propietat i selecciona el color –amb un quadre de diàleg– respectivament del punt que vol afegir.

Figura 5.16. Camp de text "Valor" i botó de selecció de color.

- **Botó "Afegir punt".** Aquest botó serveix per afegir el punt d'opacitat o de color configurat a la taula de punts corresponent.

Figura 5.17. Botó "Afegir punt".

- ***Checkbox* "Clamping".** És una opció de la funció de transferència per decidir com tracta els valors que queden fora de l'interval configurat. Per a més informació veure secció 4.3.2.2.2.

Figura 5.18. *Checkbox* "Clamping".

- **Botó "Aplicar".** Aplica la nova funció de transferència al Mirall corresponent.

Figura 5.19. Botó "Aplicar".

- ***Checkbox* "Visible".** Determina si el volum és visible o no al Mirall seleccionat. Si es desmarca el volum desapareix del Mirall actual, independentment de la funció de transferència. Quan es marca es torna a fer visible. És útil per aplicar una opacitat nul·la a tot el volum ràpidament.

Figura 5.20. *Checkbox* "Visible".

5.2.3 Configurant funcions de transferència

5.2.3.1 Funció de transferència d'opacitat

5.2.3.1.1 Afegir punts

El primer que s'ha de fer és seleccionar el Mirall pel qual es vol configurar la funció de transferència. Per l'exemple configurarem el volum central, clicant a la pestanya corresponent si no està ja activada. Després s'ha de seleccionar el *radio button* "Opacitat" si no ho està. (Figura 5.21)

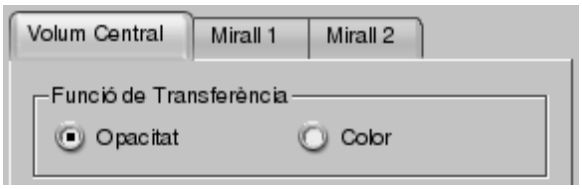


Figura 5.21. Seleccionem el volum central i la funció de transferència d'opacitat.

El següent pas és introduir els valors dels punts.

El camp de text del valor de propietat accepta valors de 0 a 65535, però els valors que tenen sentit són els que estan dins del rang de valors del model. Per aquest model els valors van de 0 a 3926 (Figura 5.22). Per tant introduirem valors dins d'aquest rang.

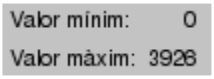


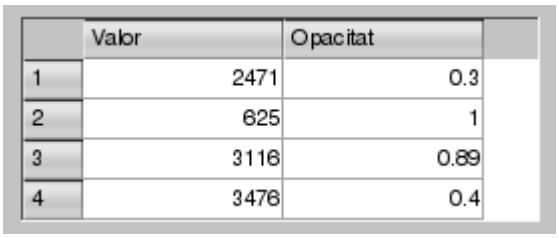
Figura 5.22. Rang de valors del model.

El camp de text del grau d'opacitat accepta valors reals de 0 a 1 amb 3 decimals.

Un cop introduïts els valors del punt s'ha de clicar al botó "Afegir punt" (Figura 5.23). És possible assignar dos o més graus d'opacitat diferents pel mateix valor de propietat, però només es tindrà en compte l'últim.

Figura 5.23. Afegint un punt.

Per l'exemple introduïm els següents valors:



	Valor	Opacitat
1	2471	0.3
2	625	1
3	3116	0.89
4	3476	0.4

Figura 5.24. Taula de punts d'opacitat.

També hem de triar si volem *clamping* o no. Per defecte està activat. En aquest cas deixem el *checkbox* marcat.

Clicant el botó "Aplicar" s'aplica la nova funció de transferència al volum central i s'actualitza la visualització amb la imatge de la Figura 5.25.

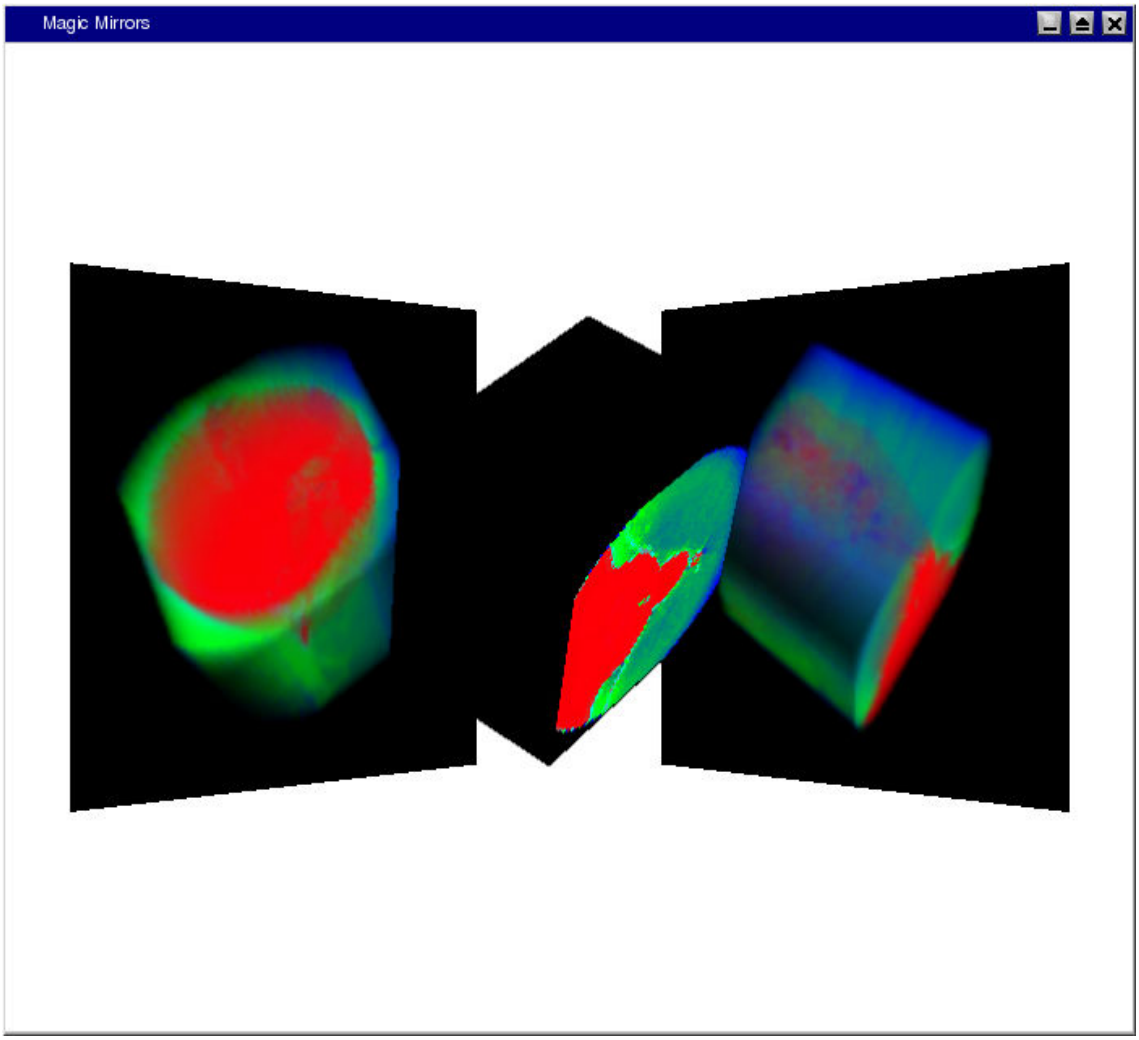


Figura 5.25. Després d'aplicar la funció de transferència d'opacitat.

5.2.3.1.2 Esborrar punts

Esborrar punts de la funció de transferència és tan fàcil com seleccionar els punts que es volen esborrar de la taula i clicar el botó "Esborrar punts" (Figura 5.26).

	Valor	Opacitat
1	2471	0.3
2	625	1
3	3116	0.89
4	3476	0.4

Esborrar punts

Figura 5.26. Punts d'opacitat seleccionats per esborrar.

Després d'esborrar els punts la taula queda com a la Figura 5.27.

	Valor	Opacitat
1	2471	0.3
2	3476	0.4

Figura 5.27. Punts d'opacitat esborrats.

A continuació es poden afegir o esborrar més punts, canviar l'opció de *clamping* o aplicar la funció de transferència. A la Figura 5.28 i a la Figura 5.29 hi ha exemples amb una nova funció de transferència d'opacitat, en els quals es pot distingir la forma del crani al volum central. Ajustant una mica els valors segurament es podria veure del tot bé.

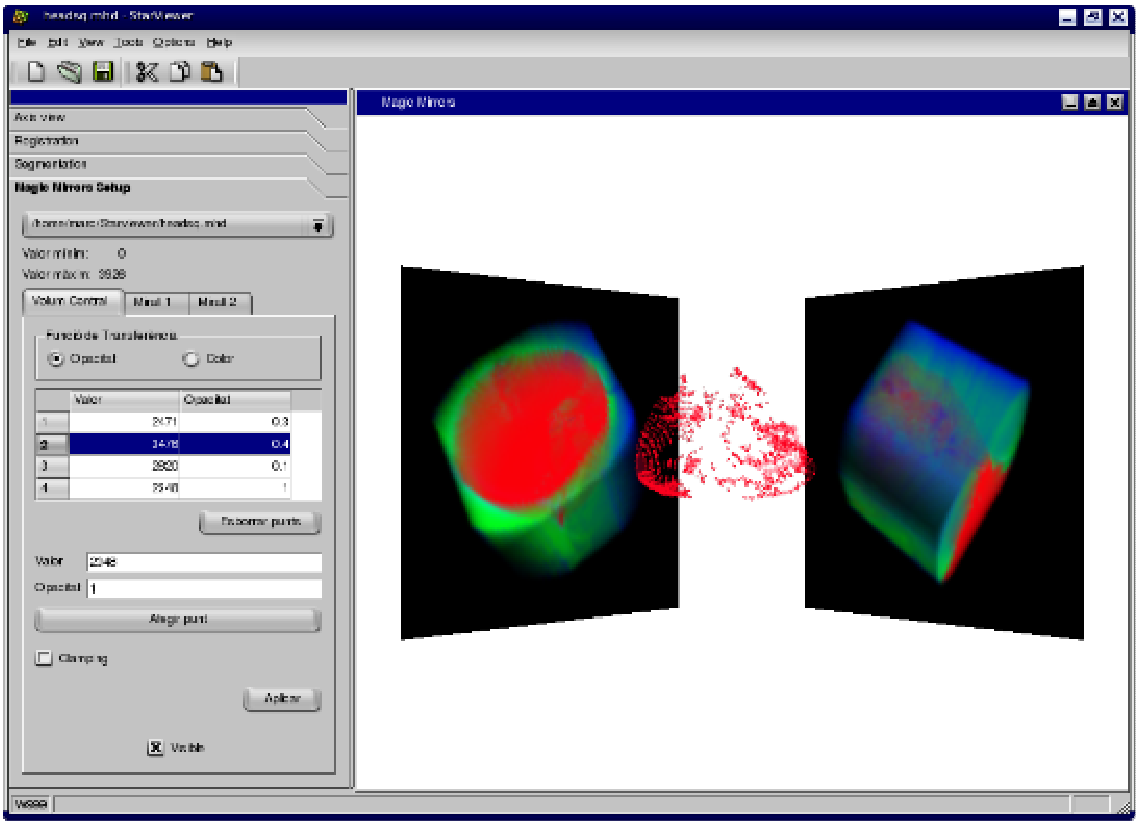


Figura 5.28. Amb una nova funció de transferència d'opacitat.

5.2.3.2 Funció de transferència de color

5.2.3.2.1 Afegir punts

El procés és molt semblant al de la funció de transferència d'opacitat. Un cop seleccionat el Mirall, per exemple el Mirall 1, seleccionem el *radio button* "Color" (Figura 5.30).

Després ja es poden afegir els punts de la funció de transferència.

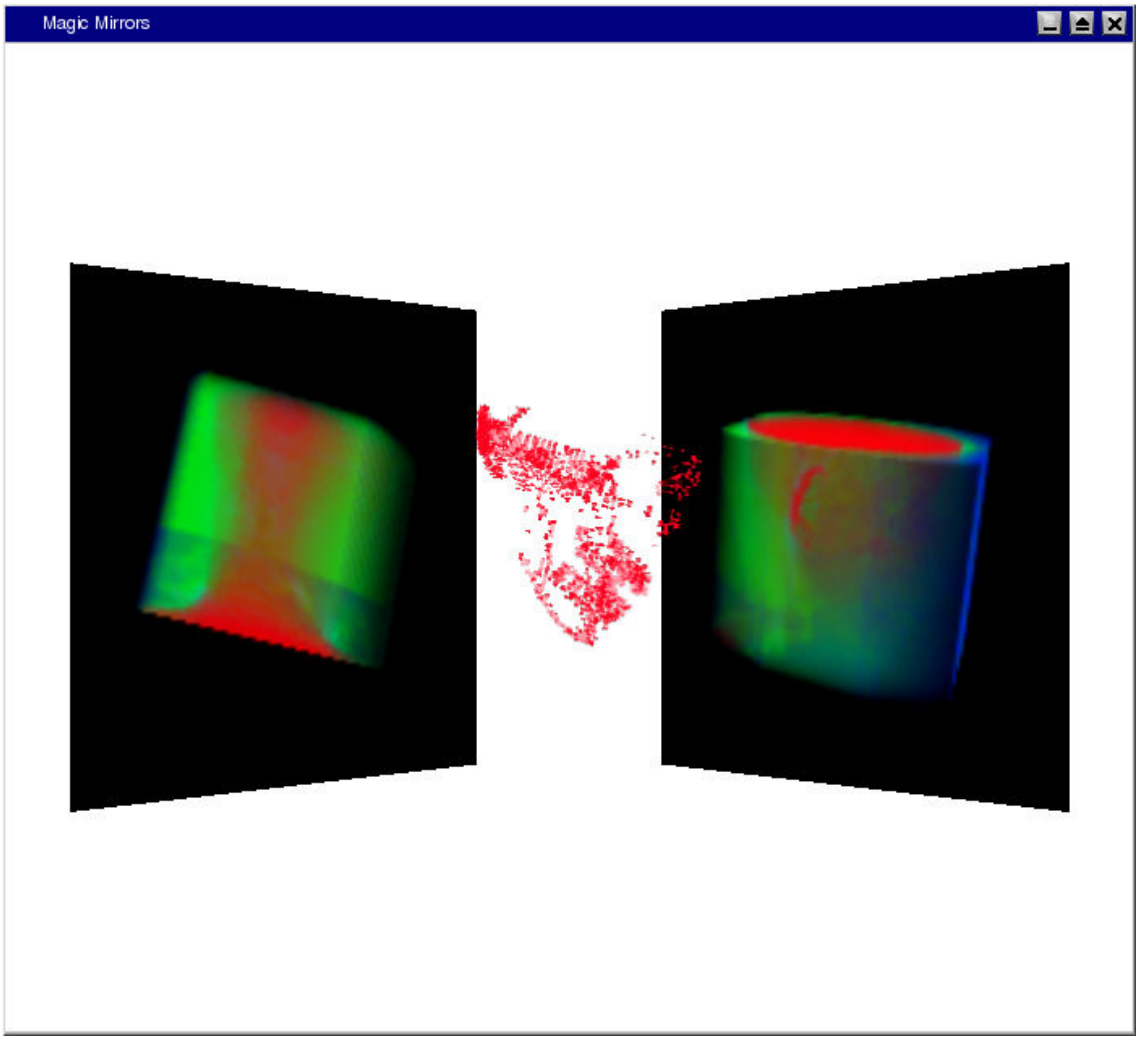


Figura 5.29. Rotant una mica el model es pot arribar a distingir la forma del crani.

Figura 5.30. Seleccionem el Mirall 1 i la funció de transferència de color.

El valor de propietat s'introdueix de la mateixa manera.

El color es selecciona clicant un botó i triant el color al quadre de diàleg que apareix (Figura 5.32). Llavors el botó quedarà pintat del color seleccionat (Figura 5.31).

Figura 5.31. El botó de selecció de color queda pintat amb el color seleccionat.

Un cop introduïts els valors del punt s'ha de clicar al botó "Afegir punt". És possible assignar dos o més colors diferents pel mateix valor de propietat, però només es tindrà en compte l'últim.

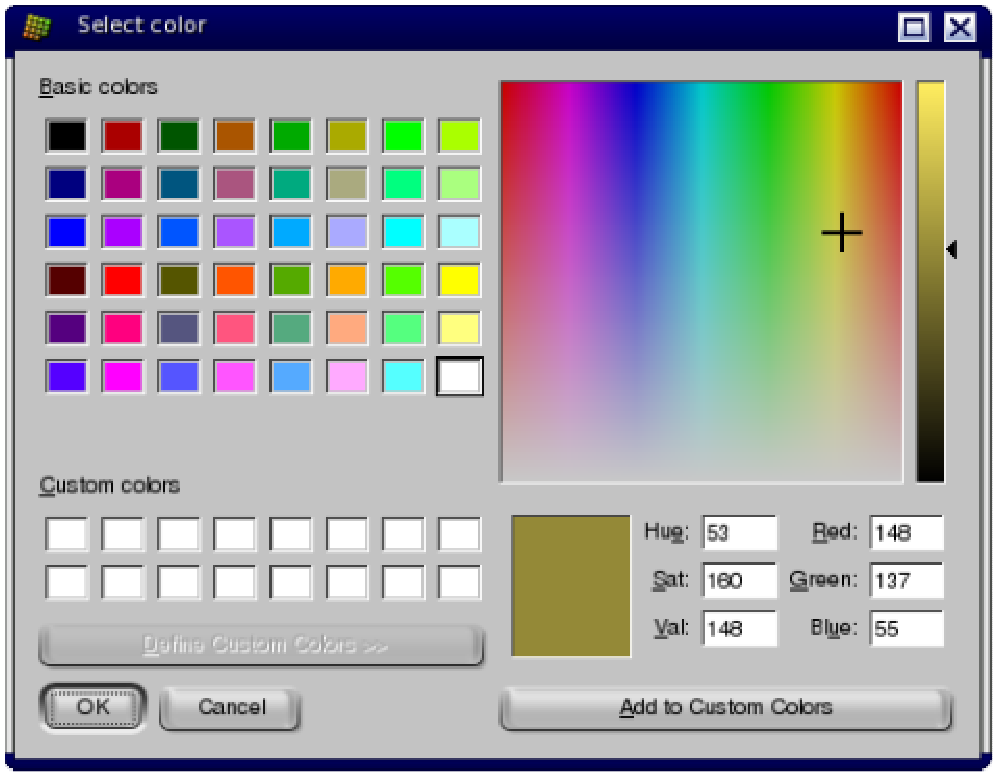


Figura 5.32. Quadre de diàleg de selecció de color.

Per l'exemple introduïm els següents valors:

	Valor	Color
1	261	
2	3836	
3	2821	
4	3102	

Figura 5.33. Taula de punts de color.

Com abans, també podem triar si volem *clamping* o no. En aquest cas deixem el *checkbox* desmarcat.

Clicant el botó "Aplicar" s'aplica la nova funció de transferència al Mirall 1 i s'actualitza la visualització amb la imatge de la Figura 5.34.

5.2.3.2.2 Esborrar punts

Es fa de la mateixa manera que per la funció de transferència d'opacitat. Com es veu a la Figura 5.35, el color es veu igual fins i tot quan està la fila seleccionada. D'aquesta manera no hi ha confusions.

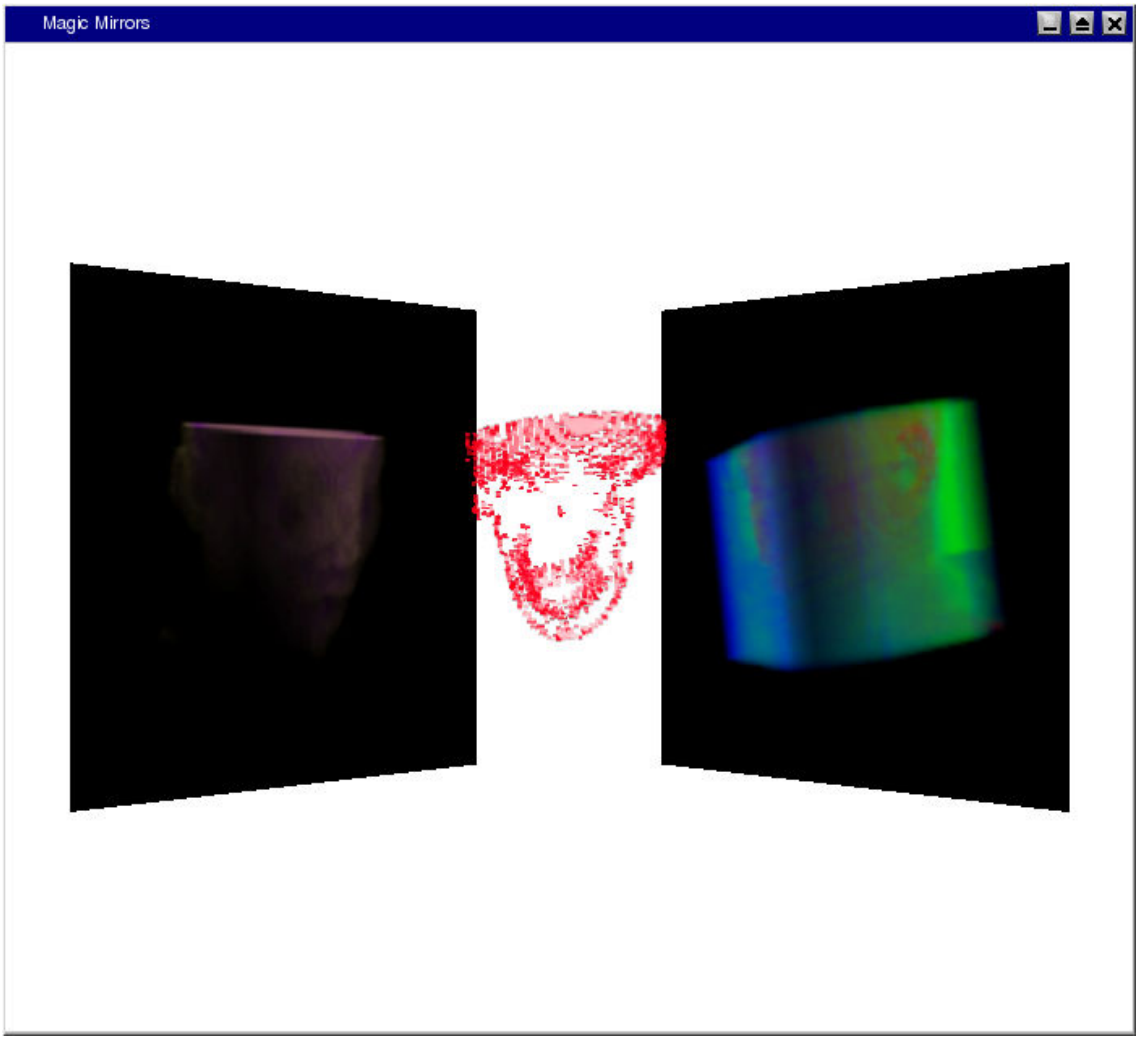


Figura 5.34. Després d'aplicar la funció de transferència de color.

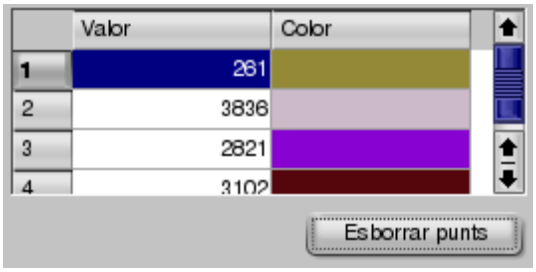


Figura 5.35. Punt de color seleccionat per esborrar.

A la Figura 5.36 es pot veure el resultat d'aplicar una funció de transferència de color diferent. Al Mirall de l'esquerra es pot arribar a veure entre tota la "boira" la forma del crani. A la Figura 5.37 es veu aquest Mirall després d'enfosquir alguns colors amb un programa d'edició gràfica. Aquí es veu clarament la forma del crani.

5.2.4 Visibilitat del model

Es pot fer que el model sigui visible o invisible en un determinat Mirall. Per defecte és visible a tots. Això té utilitat pels models registrats, ja que pels models simples

el més normal és que siguin visibles a tots els Miralls, però igualment es pot fer servir l'opció, per si algun usuari li troba utilitat.

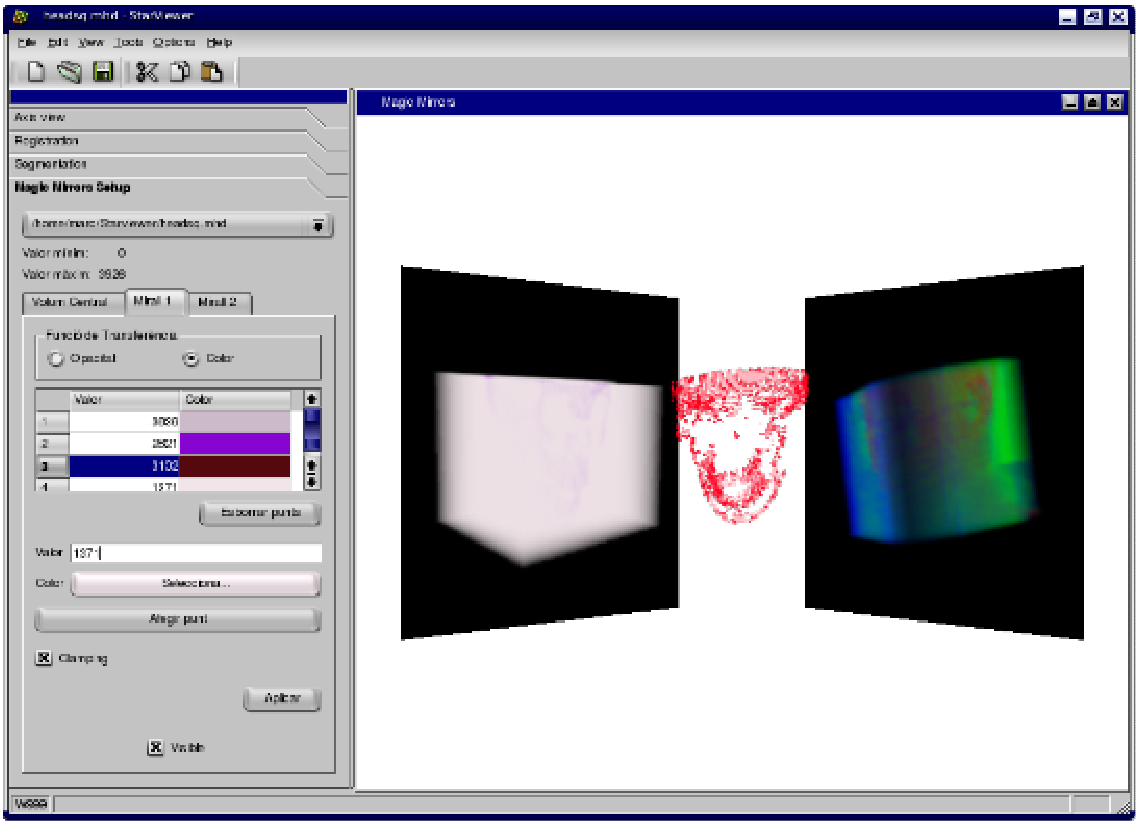


Figura 5.36. Amb una nova funció de transferència de color.

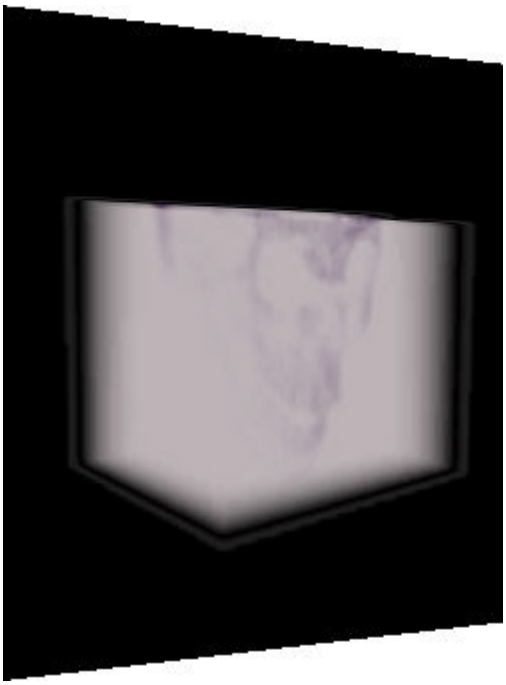


Figura 5.37. El Mirall de l'esquerra després d'enfosquir els colors neutres amb un editor gràfic: també es pot distingir la forma del crani.

Per exemple podem seleccionar el volum central i fer que el model no sigui visible allà desmarcant el *checkbox* "Visible". El resultat, a la Figura 5.38.

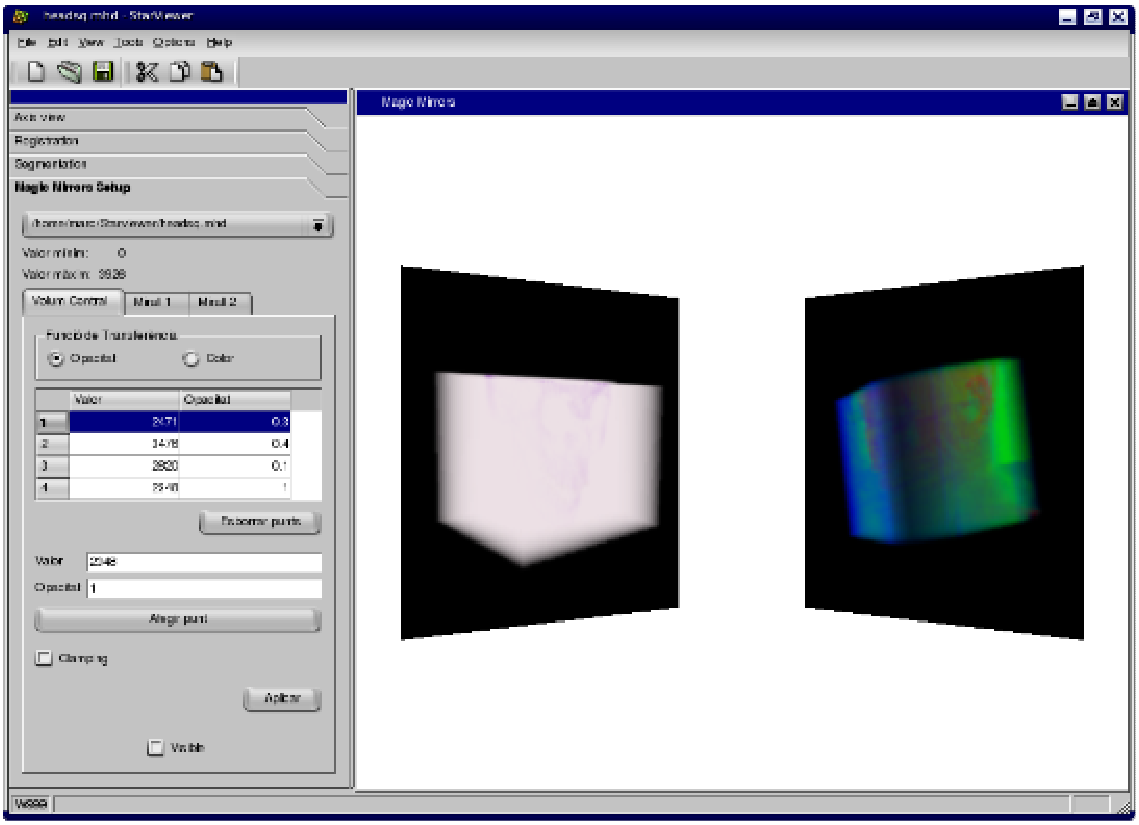


Figura 5.38. El volum central és invisible.

5.2.5 Altres exemples de configuracions

En aquest secció hi ha vàries imatges de proves amb diferents funcions de transferència.

A la Figura 5.39 s'ha aplicat una nova funció de transferència de color al Mirall 1 i es veu la forma del cap dins d'una "boira" lila.

A la Figura 5.40 hi ha una petita variació de la funció de transferència anterior. Concretament s'ha eliminat el punt corresponent al valor de propietat 91 i el color lila i s'ha desactivat el *clamping*. Com a resultat es veu netament la forma del cap.

A la Figura 5.41 s'ha aplicat una nova funció de transferència de color al Mirall 2 i s'ha girat una mica el model perquè es vegi la cara al Mirall 2.

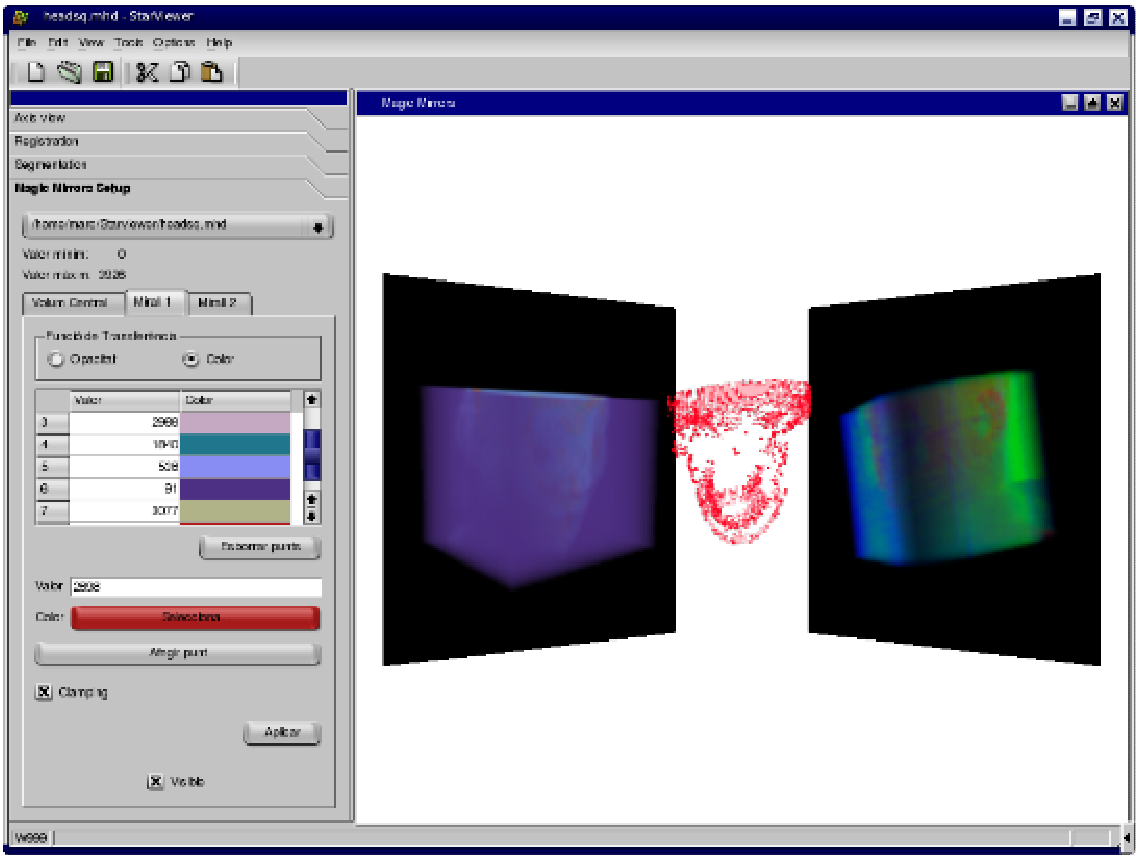


Figura 5.39. Nova funció de transferència de color al Mirall 1.

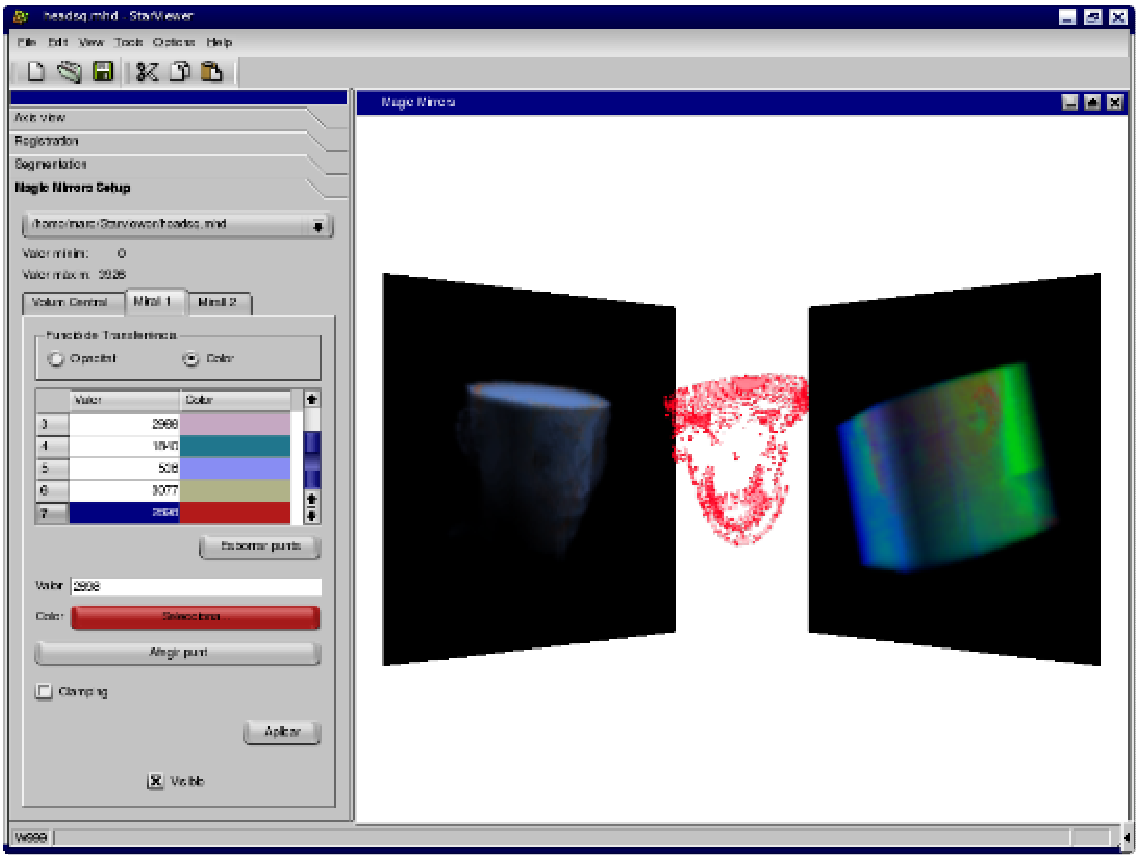


Figura 5.40. Una petita variació de l'anterior.

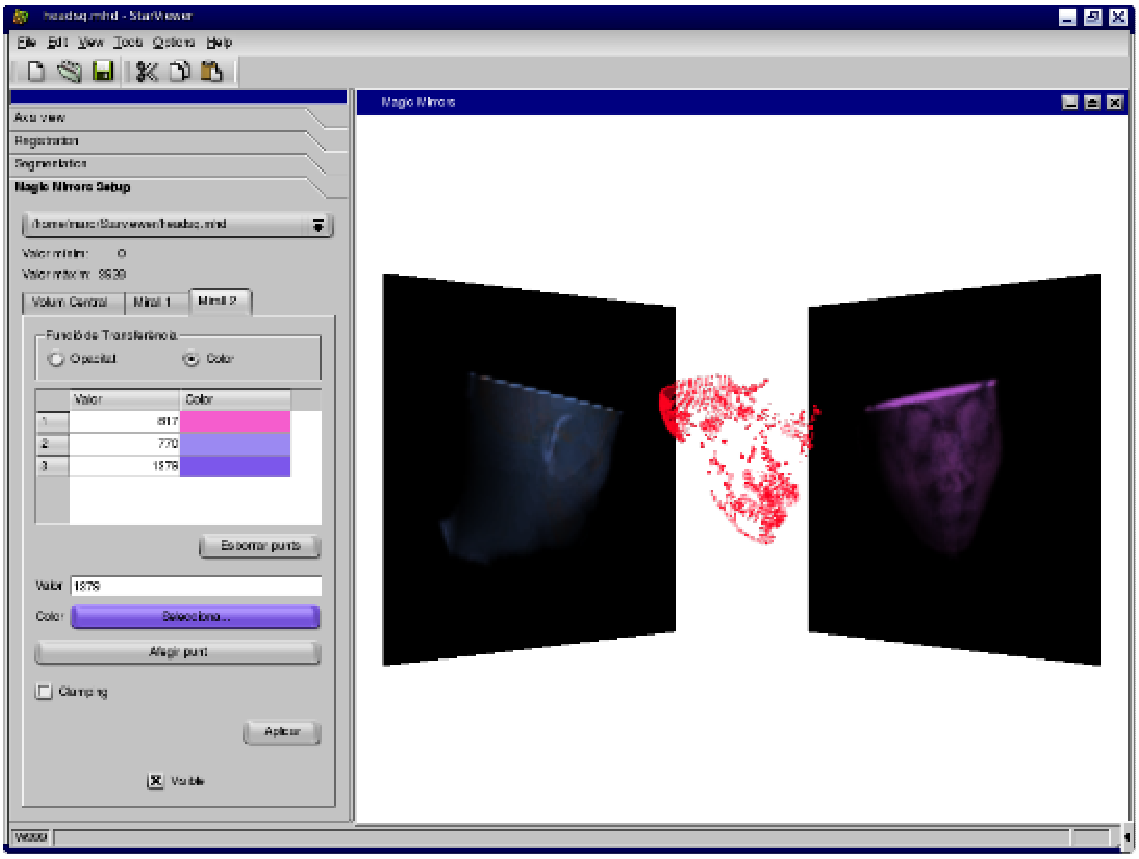


Figura 5.41. Nova funció de transferència de color al Mirall 2.

5.3 Treballant amb un Model Registrat

Treballar amb un model registrat és pràcticament igual que amb un model simple: hi ha els mateixos controls a la interfície gràfica i es pot interactuar de la mateixa manera amb la visualització. El que canvia més és el que s'ha de fer abans d'iniciar els Magic Mirrors.

5.3.1 Registre de dos models

El primer que s'ha de fer, com en el cas del model simple, és obrir un fitxer. En aquest exemple obrim el fitxer "ct.mhd".

Després d'obrir el model cliquem al *toolbox* allà on diu "Registration". Llavors apareixerà la interfície de registre (Figura 5.42). Aquesta interfície no l'explicaré perquè no forma part d'aquest projecte, però hi ha moltes opcions que afecten al procés de registre.

El següent pas és clicar el botó "Open" per obrir el segon model, que serà el "mr_PD.mhd".

Un cop obert apareixeran un conjunt de finestres com a la Figura 5.43.

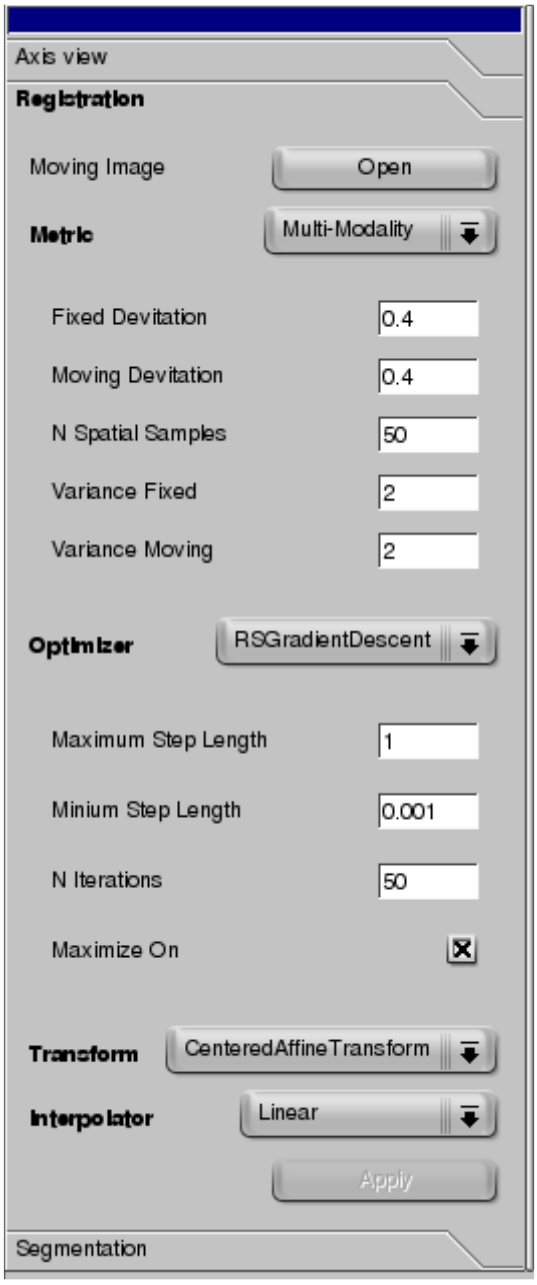


Figura 5.42. Interfície de registre.

Després d'això s'han de configurar els paràmetres per fer el registre. En aquest cas només cal seleccionar "Mattes Mutual" al *combo* de "Metric" i deixar la resta tal com està per defecte (Figura 5.45). Després prémer el botó "Apply" i esperar que es completi el registre. Si no surt bé a la primera s'ha d'anar intentant reiniciant cada cop l'aplicació fins que surti bé. Es pot saber que ha sortit malament quan surt un missatge d'error a la consola. Quan surt bé fa 16 iteracions.

Quan s'ha completat correctament el procés de registre apareixen noves finestres i la visualització queda com a la Figura 5.46.

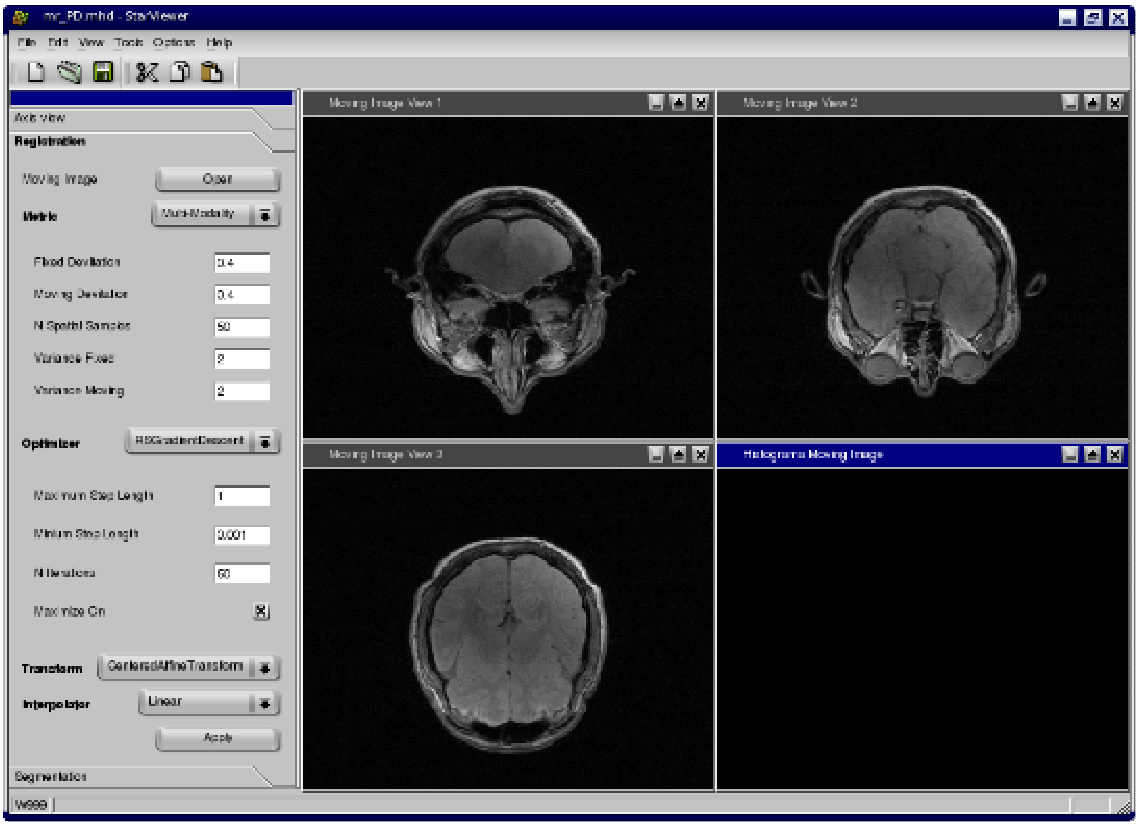


Figura 5.43. Després d'obrir el segon model.

Un cop fet això ja podem tancar totes les finestres obertes (imprescindible en el meu cas per un *bug* de les targetes gràfiques ATI Radeon, que no poden tenir obertes les finestres de registre i la de Magic Mirrors a la vegada, per unes característiques d'aquestes finestres) i iniciar els Magic Mirrors amb l'opció del menú.

5.3.2 Magic Mirrors amb models registrats

Quan s'inicien els Magic Mirrors apareix una visualització igual que en el cas de models simples. La interacció amb la visualització es fa de la mateixa manera: també es pot moure la camera i el model i triar entre 2 estils d'interacció diferents. Les tecles i botons són els mateixos.

A la Figura 5.47 hi ha una imatge agafada després de moure la camera perquè es vegi una mica des de dalt.

El procés de treball és bàsicament el mateix que amb un model simple. La diferència és que ara abans de cada canvi de funció de transferència s'ha de triar el volum pel qual es vol canviar. Això es fa amb el *combo*, que conté els noms dels diferents models que han estat oberts i registrats (Figura 5.44).

Figura 5.44. *Combo* per un model registrat.

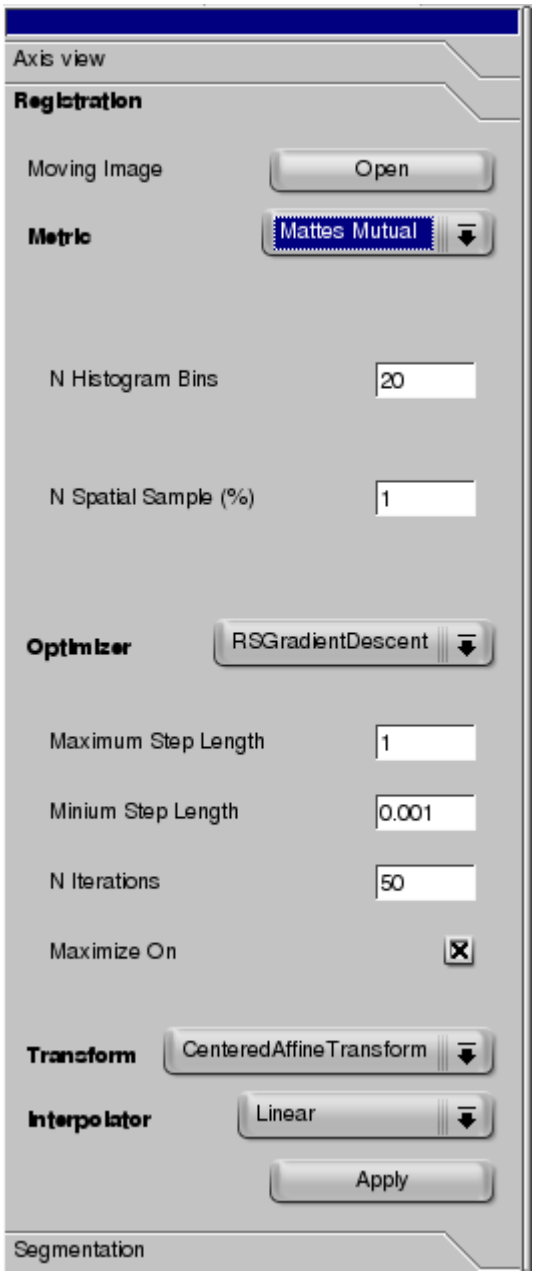


Figura 5.45. Paràmetres del registre.

El primer que pot interessar, però, és decidir quines propietats (o volums) es veuran a cada Mirall. Per exemple, podem fer que al mig es vegin els 2 volums, al Mirall de l'esquerra el volum 1 i al de la dreta el volum 2. Com que són tots visibles a tot arreu per defecte, el que hem de fer és fer-los invisibles on no els vulguem. Pel cas de l'exemple, hauríem de seleccionar el primer volum i el Mirall 2 i desactivar el *checkbox* "Visible", i després fer el mateix amb el segon volum i el Mirall 1. D'aquesta manera la visualització quedaria com a la Figura 5.48.

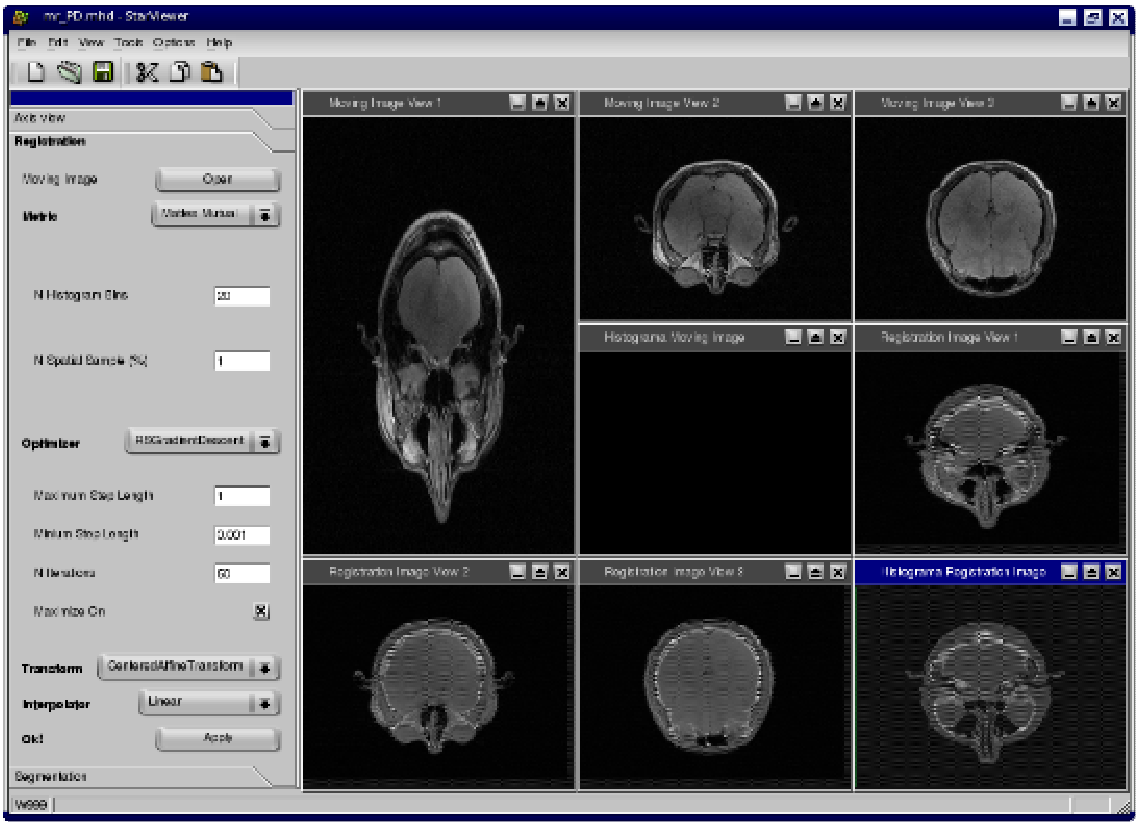


Figura 5.46. Resultat del registre.

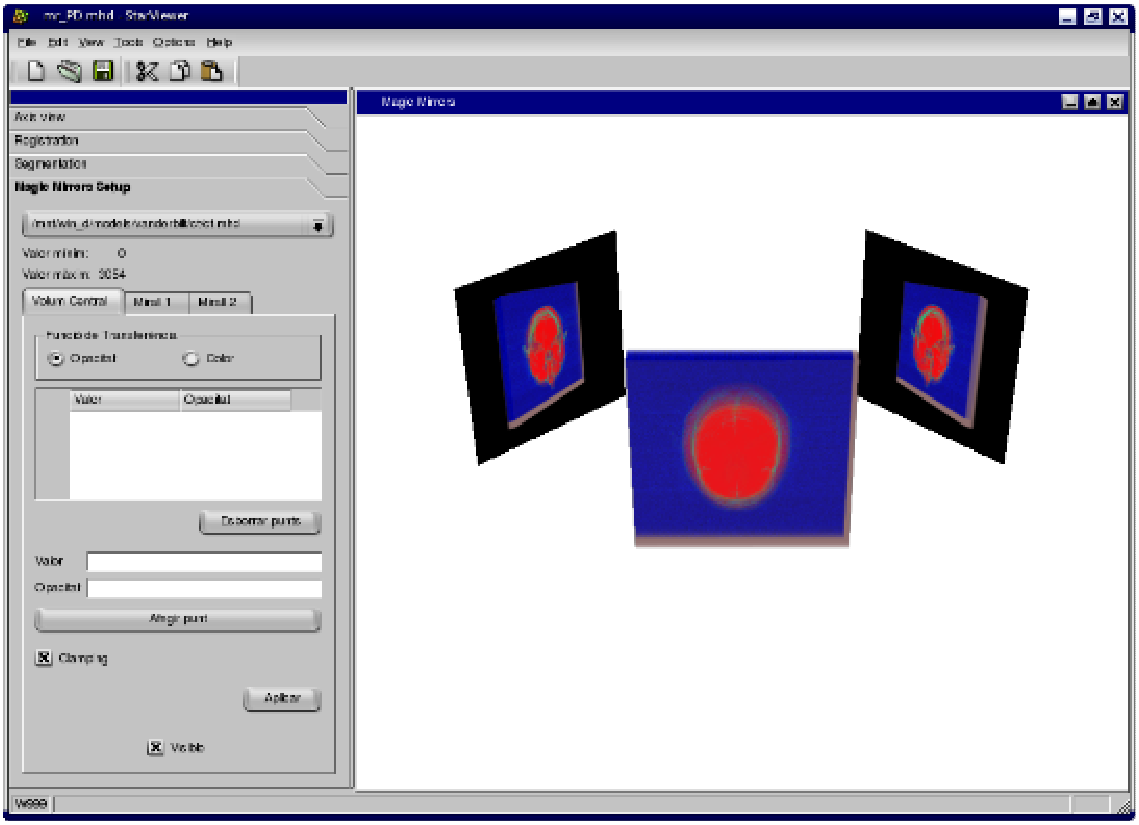


Figura 5.47. Magic Mirrors amb el model registrat després de moure la camera.

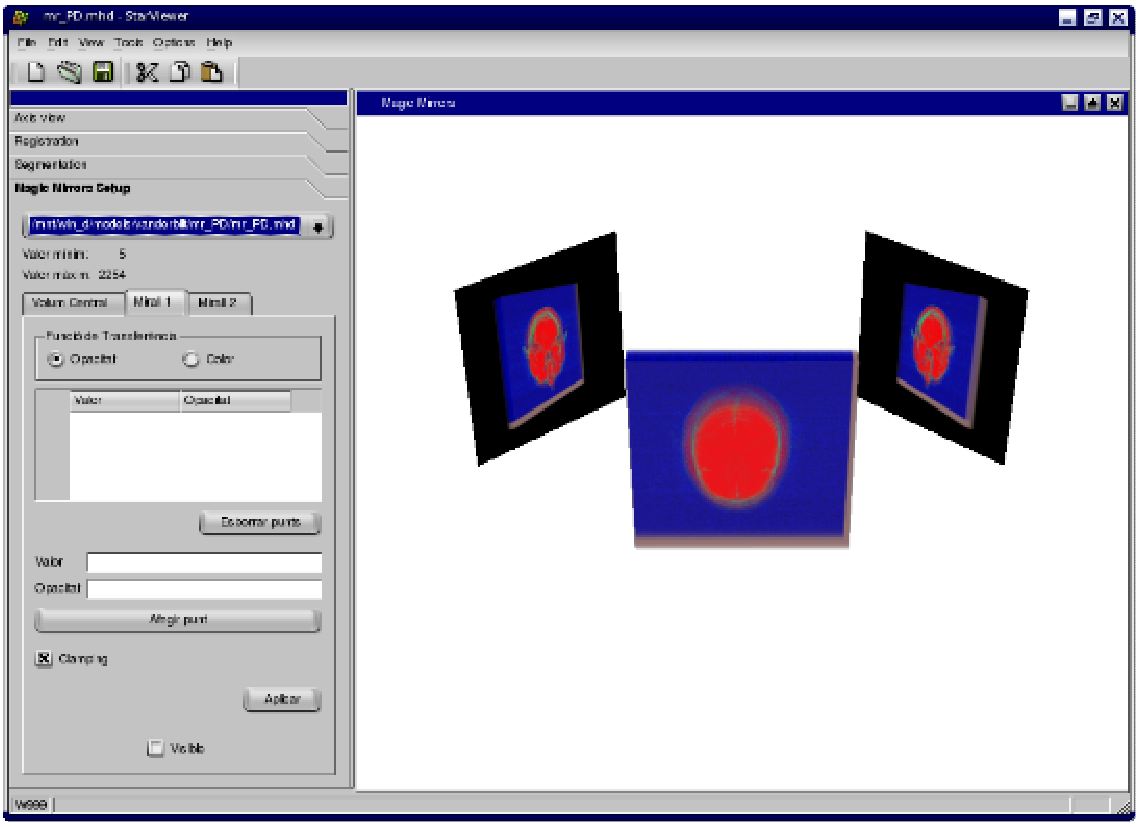


Figura 5.48. Magic Mirrors: Al volum central són visibles els dos volums, al Mirall de l'esquerra només el primer i al de la dreta només el segon.

La diferència és pràcticament imperceptible en aquest cas perquè els dos models són molt semblants (es pot comprovar executant els Magic Mirrors amb cadascun per separat), però amb models més diferents s'hauria de notar la diferència.

Un aspecte a destacar és sobre la prioritat de visualització. Amb la implementació actual no hi ha manera d'establir una prioritat amb la interfície gràfica (s'hauria d'afegir algun control per establir-la i el mètode corresponent a la classe MagicMirrors), però és possible assignar-la indirectament. Donat que el vtkRenderer dibuixa els objectes que té en l'ordre en què li han estat afegits es pot aprofitar això perquè dibuixi primer el que hauria de tenir menys prioritat i després el de més prioritat. Això es pot fer des de la interfície gràfica jugant amb les visibilitats dels objectes. Per fer un volum invisible el que faig és treure'l del *renderer*, i per fer-lo visible l'hi torno a posar. D'aquesta manera, si un volum es fa invisible en un Mirall, quan es torni a fer visible passarà a tenir prioritat sobre l'altre.

Aprofitant aquesta característica no és impossible assignar prioritats, però queda pendent afegir a la interfície gràfica els controls necessaris per fer-ho de forma senzilla.

Després d'aquest parèntesi hi ha alguns exemples de diferents configuracions de les funcions de transferència. El procediment és bàsicament el mateix: triar el volum, triar el Mirall, triar la funció de transferència (opacitat o color), afegir i esborrar punts i decidir *clamping*, i aplicar.

A la Figura 5.49 s'ha definit una nova funció de transferència de color pel model de CT al Mirall 1. Amb aquesta es veuen millor els detalls del model.

A la Figura 5.50 hi ha una vista des de darrera del Mirall 1, per veure'l de més a prop.

A la Figura 5.51 hi ha una nova funció de transferència de color pel model de MR al Mirall 2. Aquesta no es veu tan bé perquè els colors són molt semblants, però serveix com a exemple.

A la Figura 5.52 hi ha una imatge de la visualització després de girar el model aproximadament uns 180º al voltant de l'eix X. Així es veu el que abans es veia al darrera, però de cap per avall respecte a com es veia abans.

Finalment, a la Figura 5.53 hem fet visible el model de MR al Mirall 1 i li hem assignat una nova funció de transferència d'opacitat. Com que ha estat l'últim en fer-se visible, té prioritat de visualització respecte a l'altre volum.

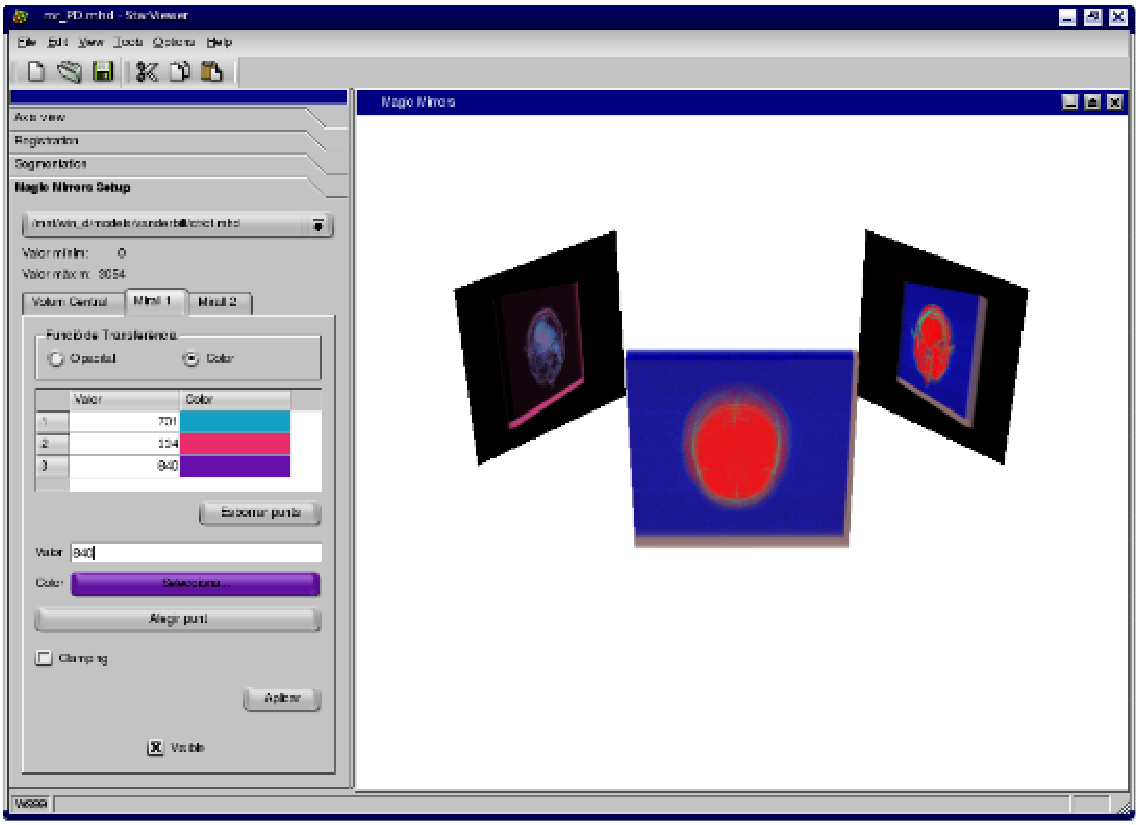


Figura 5.49. Nova funció de transferència de color pel CT al Mirall 1.

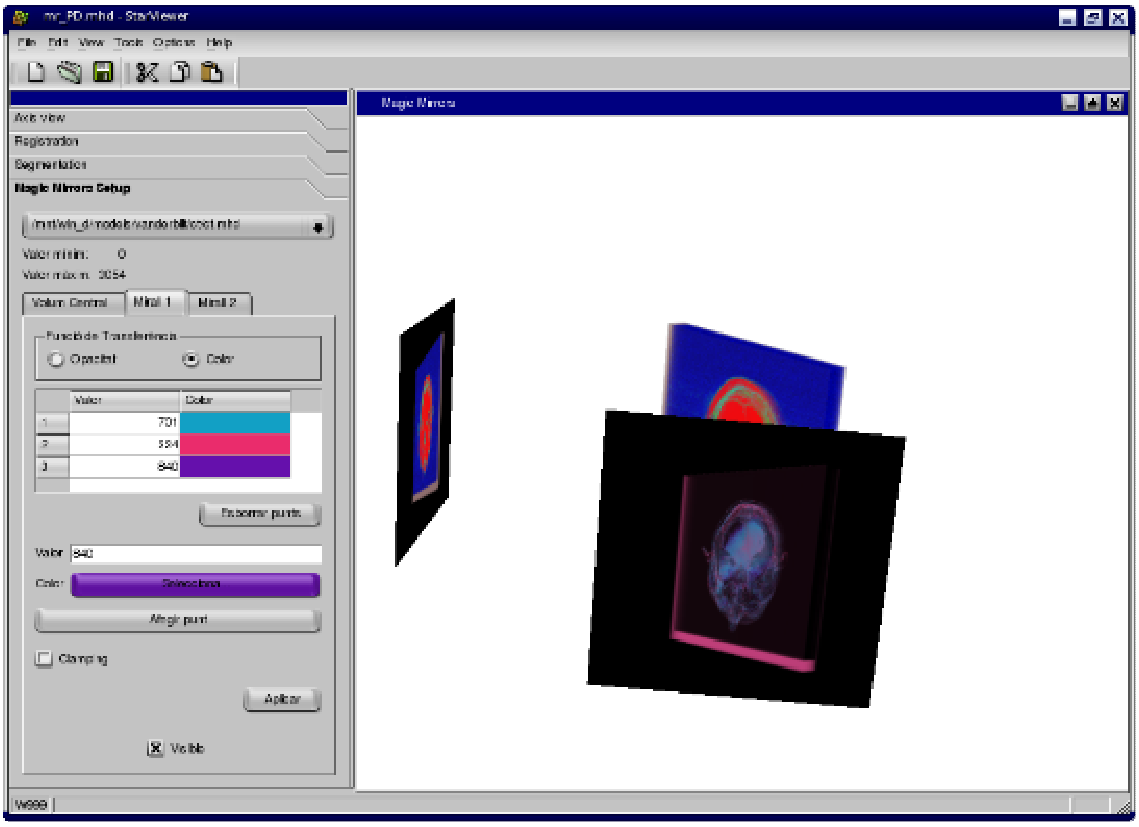


Figura 5.50. El mateix d'abans però amb la camera darrera del Mirall 1.

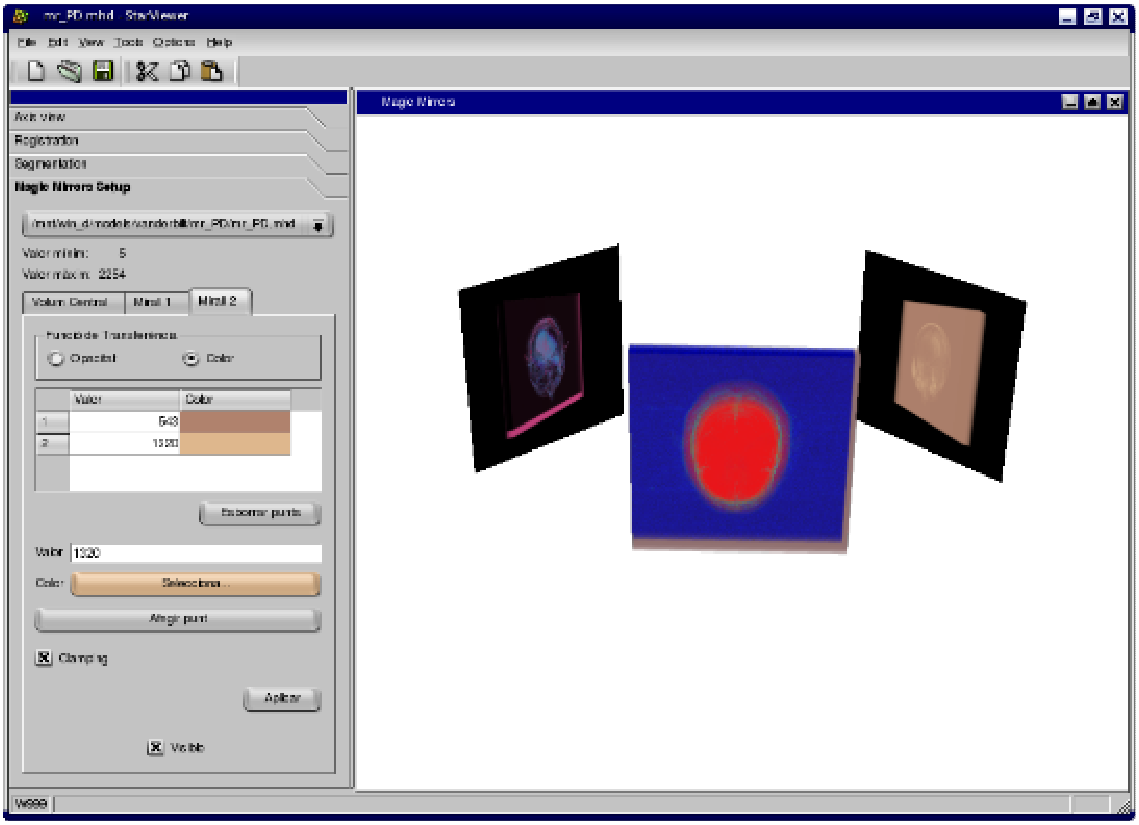


Figura 5.51. Nova funció de transferència de color pel MR al Mirall 2.

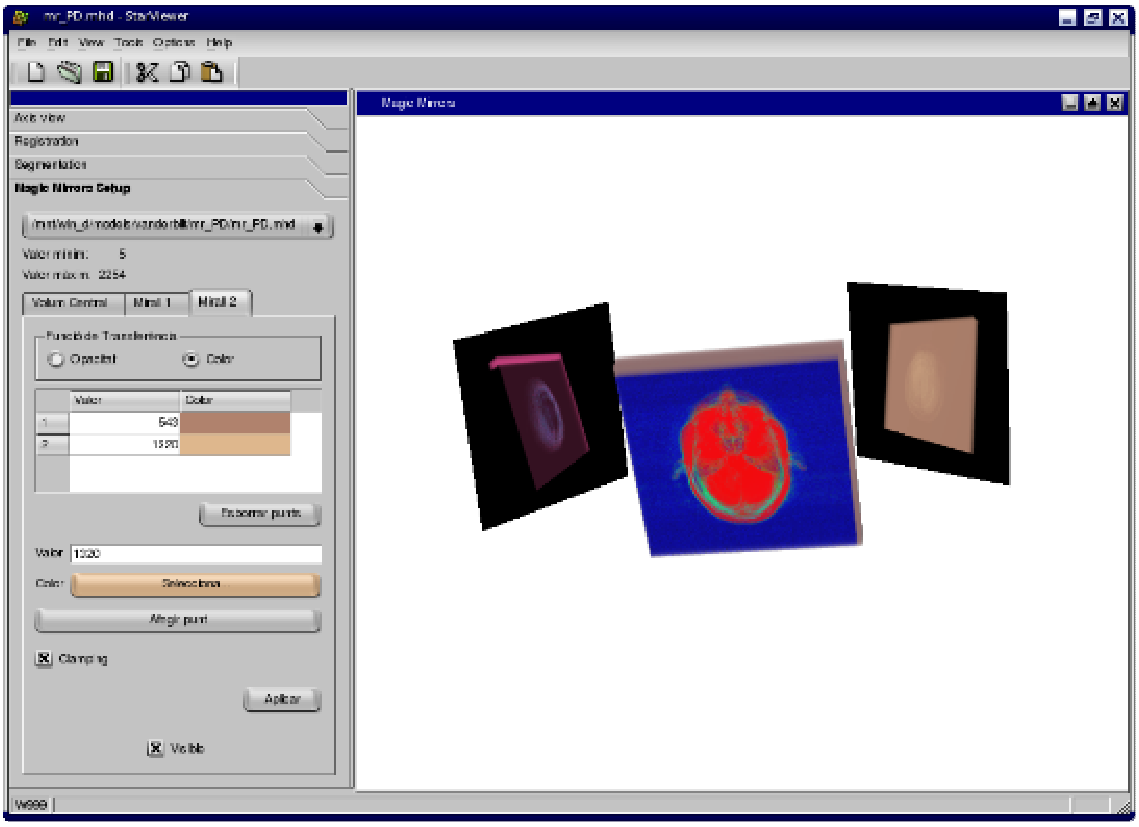


Figura 5.52. Havent girat el model.

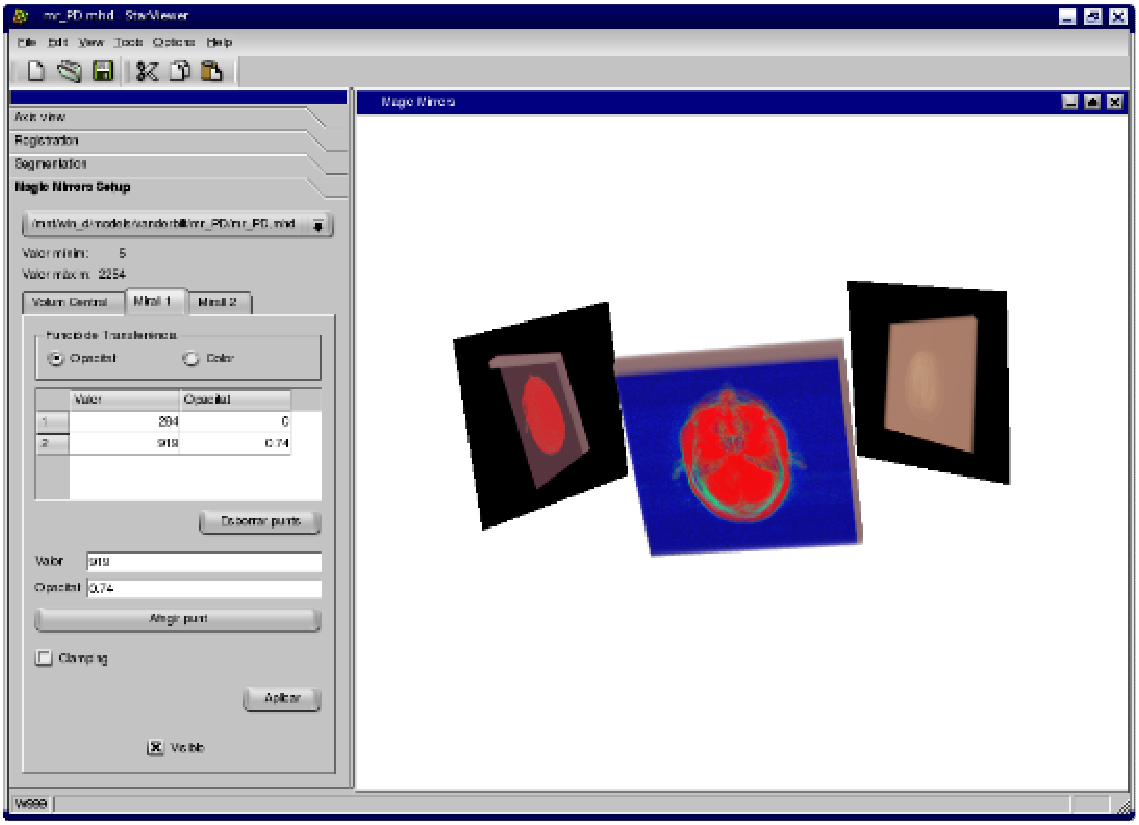


Figura 5.53. Nova funció de d'opacitat pel MR al Mirall 1, on ara és visible.

6 Avaluació dels Resultats

6.1 Ordinador de Proves

Les proves les he fet amb l'ordinador personal de casa, amb les característiques següents:

Processador Intel Pentium 4 2.8 GHz
512 MB RAM
Targeta gràfica ATI Radeon 9000 Series 128 MB RAM
Sistema Operatiu Mandrake Linux 10.0 Official
Kernel 2.6.3
XFree86 4.3
GCC 3.3.2
KDE 3.2
Qt 3.2.3

També cal dir que els temps d'execució que sortiran al següent apartat són per la versió de publicació de l'aplicació (no la de depuració), i això fa que siguin més petits.

6.2 Proves i Resultats

En aquest apartat mostraré alguns resultats obtinguts provant diferents combinacions de funcions de transferència amb diferents models. Inclouré el temps que tarden a actualitzar-se els Miralls (calculats amb un cronòmetre, o sigui que hi pot haver un cert error), les dades de les funcions de transferència per cada Mirall i una imatge del resultat.

6.2.1 Model Simple "headsq.mhd"

Temps d'actualització dels Miralls: 0.71 s.

Dades de les funcions de transferència:

Volum central		**Mirall 1**		**Mirall 2**	
Opacitat	Color	Opacitat	Color	Opacitat	Color
(1454, 0.83)	(946,)	(1109, 0)	(1111,)	(2155, 0.512)	(256,)
(1653, 1)	(1627,)	(1157, 0)		(2901, 0.19)	(1676,)
(3917, 0.364)	(2445,)	(1423, 0.952)		(3727, 0.8)	
		(1923, 0)			
		(2201, 0.4)			
		(2324, 0.951)			
		(2791, 1)			
Clamping off	*Clamping* off	*Clamping* on	*Clamping* on	*Clamping* off	*Clamping* on

Resultat:

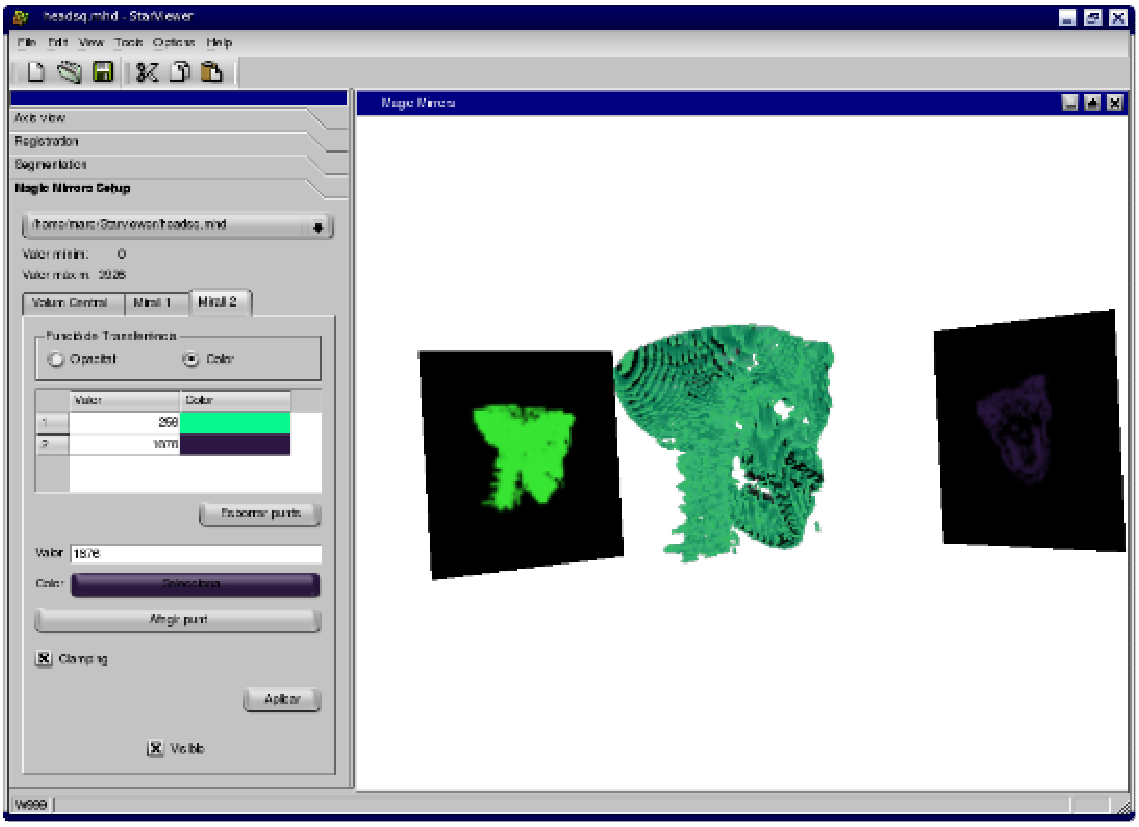


Figura 6.1. headsq.mhd

6.2.2 Model Registrat "ct.mhd" + "mr_PD.mhd"

Temps d'actualització dels Miralls: 1.52 s.

Al volum central són visibles tots 2 amb prioritat del MR, al Mirall 1 tots dos amb prioritat del CT, i al Mirall 2 només el CT.

Dades de les funcions de transferència:

ct.mhd:

Volum central		**Mirall 1**		**Mirall 2**	
Opacitat	Color	Opacitat	Color	Opacitat	Color
(1263, 0.722)	(2599, ■)	(459, 0.5)	(1037, ■)	(821, 0.846)	(690, ■)
(1465, 0.29)		(3004, 1.0)	(1230, ■)	(1980, 0.1)	(1625, ■)
(2092, 0.8)			(1968, ■)	(2947, 0.1)	(3017, ■)
Clamping off	*Clamping* on	*Clamping* off	*Clamping* on	*Clamping* off	*Clamping* on

mr_PD.mhd:

Volum central		**Mirall 1**	
Opacitat	Color	Opacitat	Color
(1312, 0.408)	Per defecte	(768, 0.04)	(1163, ■)
(1928, 0.533)		(1320, 0.4)	
(2203, 0)			
Clamping off		*Clamping* off	*Clamping* on

Resultat:

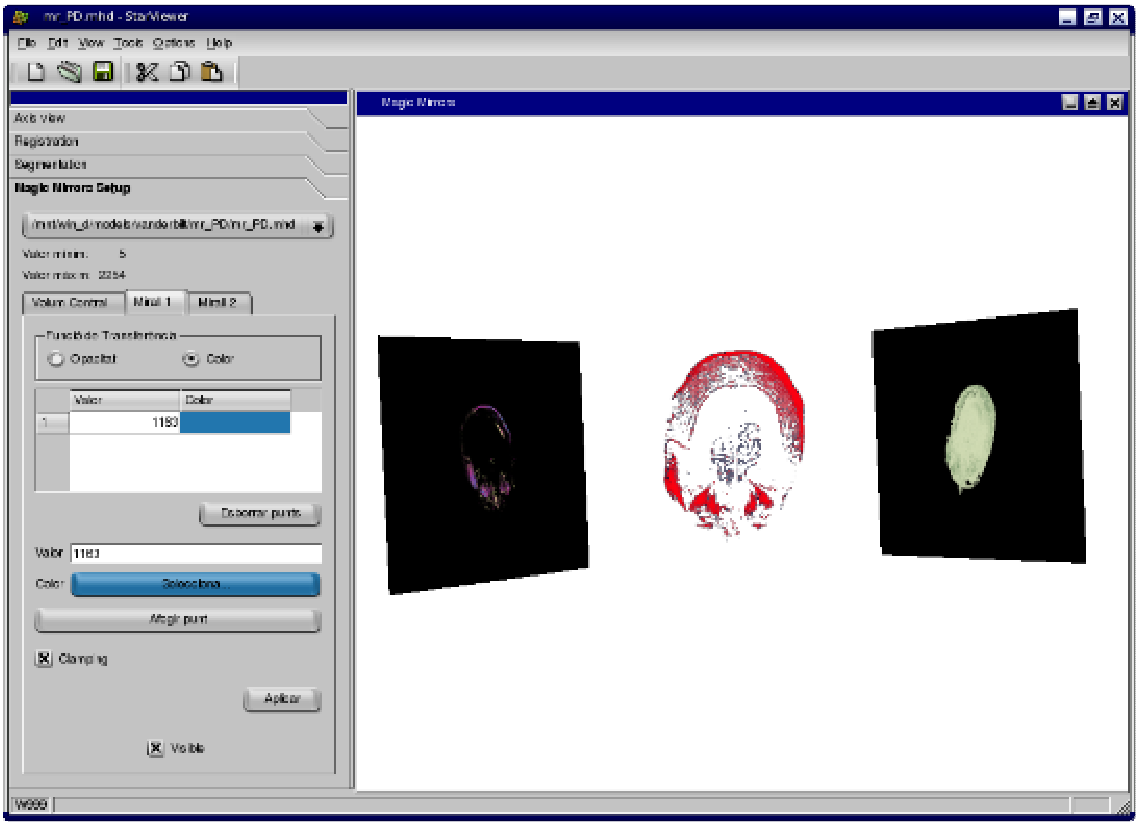


Figura 6.2. ct.mhd + mr_PD.mhd

6.2.3 Model Simple "brain1.mhd"

Temps d'actualització dels Miralls: 1.16 s.

Dades de les funcions de transferència:

Volum central		**Mirall 1**		**Mirall 2**	
Opacitat	Color	Opacitat	Color	Opacitat	Color
(89, 0.206)	Per defecte	(28, 0.25)	(15,)	(161, 1)	(162,)
(202, 0)		(70, 0.6)	(36,)	(219, 0.301)	(212,)
			(77,)		(250,)
Clamping off		*Clamping* off	*Clamping* on	*Clamping* off	*Clamping* on

Resultat:

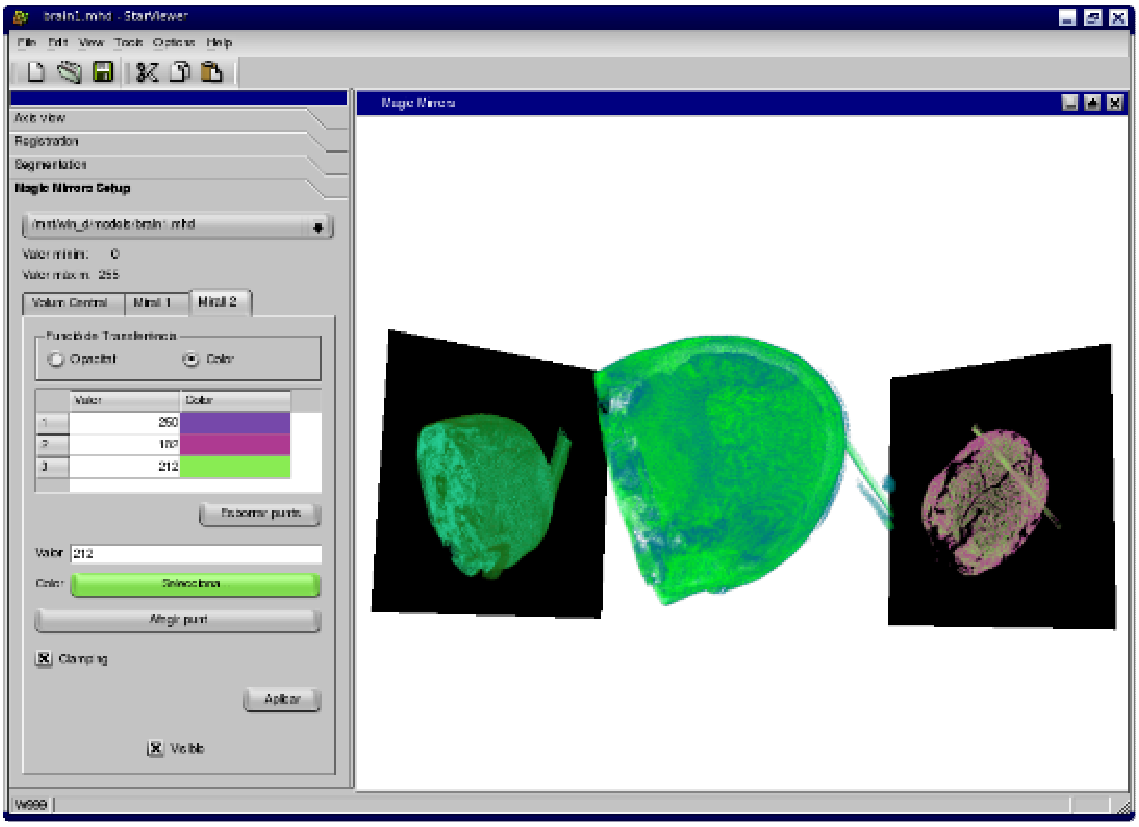


Figura 6.3. brain1.mhd

6.3 Avaluació

Si mirem els temps d'actualització dels Miralls veiem que no són gaire alts. Amb el primer model és gairebé instantani (menys d'un segon), i és perquè és un model simple i de poca resolució (la mida dels Miralls surt de 64 x 64). En canvi amb els altres dos el temps és superior a un segon, i això és perquè són imatges de més resolució (la mida dels Miralls surt de 256 x 256) i en el cas del model registrat perquè són 2 volums, i per això és el que més triga. Tot i això tampoc són temps excessivament elevats, i amb això podem concloure que amb un ordinador una mica menys potent que el de proves el temps d'actualitzar els Miralls encara seria acceptable. Per tant podem dir que l'eficiència és relativament bona, però s'haurien de fer proves en altres màquines per determinar els requisits mínims.

A banda d'aquests temps d'actualització dels Miralls també s'ha de tenir en compte el temps de resposta quan l'usuari vol rotar el model. Aquests no els he cronometrat, però amb el primer model era pràcticament instantani, amb el model registrat trigava prop d'un segon però amb el model del cervell trigava alguns segons a respondre. L'explicació segurament deu ser que tarda més perquè el model és més voluminós, més que el primer i més que els dos volums del model registrat junts. Això faria augmentar els requisits mínims, perquè s'hauria d'intentar que el temps de resposta no passés mai d'un segon. En canvi, el temps de resposta per moure la camera és molt petit pels 3 casos.

Pel que fa a les funcions de transferència que he aplicat en cada cas, el que he fet és anar afegint punts i provant com es veia. Si es veia bé la deixava com estava o provava d'afegir un altre punt, i si es veia malament provava de canviar algun punt o afegir-ne de nous. He seguit aquest procés fins que he aconseguit una imatge

satisfactòria. Els resultats de les imatges són els millors que he pogut aconseguir segons el meu punt de vista. El que he procurat és que no es veiés la "caixa" que sempre hi ha al voltant de les dades importants, ja que en principi no aporta cap informació útil. De tota manera, els que hauran de fer proves i avaluar els resultats obtinguts són els usuaris finals, els membres del grup de neuro-radiologia de l'Institut de Diagnòstic per la Imatge de l'Hospital Universitari Dr. Josep Trueta de Girona. Ells tenen informació sobre el tipus de material que correspon a cada valor de propietat de les diferents proves, i amb això poden dissenyar funcions de transferència més ràpidament, ja que ells saben el que han de veure i com ho volen veure.

7 Millores i Treball Futur

Tot i que les tècniques i mètodes que s'han implementat i explicat al llarg de la memòria ens han permès assolir els objectius que nosaltres ens havíem marcat, som conscients de que moltes d'aquestes tècniques poden ser millorades. En aquest capítol recollim de forma molt sintetitzada algunes d'aquestes possibles millores.

- **Implementar un sistema de prioritats de visualització pels models registrats.**

  Es podria fer fàcilment aprofitant la característica del vtkRenderer que dibuixa els objectes en l'ordre en què li han estat afegits, tal com he explicat al capítol d'implementació. Quan es defineixi una prioritat (caldria dissenyar la interfície per fer-ho) només caldria treure els objectes del *renderer* i afegir-los en l'ordre adient, de menys prioritat a més prioritat.

- **Fer que es puguin moure els Miralls.**

  Aquesta seria una mica més difícil de fer, perquè s'hauria de pensar com es pot fer de forma senzilla. Una possible solució seria que hi hagués un botó per passar al mode de moure Miralls i llavors canviés a la camera corresponent al Mirall. Llavors l'usuari mouria la camera fins on volgués i d'alguna manera diria que ja està posicionada. Llavors es mouria el Mirall fins a la mateixa posició i se li aplicaria la rotació adient. Amb això el Mirall ja estaria posicionat i l'usuari tornaria a veure l'escena des de la camera principal.

- **Que es puguin afegir Miralls.**

  S'haurien de guardar els Miralls i cameres en un vector, i també pensar com es defineix la rotació i posició dels Miralls (potser com en el cas anterior).

- **Que no calgui guardar fitxers temporals.**

  S'hauria de mirar si funciona l'assignació directa de la textura del vtkWindowToImageFilter al Mirall, amb el disseny actual. O sinó estudiar l'opció per escriure a memòria del vtkPNGWriter i si n'hi ha alguna per llegir de memòria al vtkPNGReader.

- **Actualització en temps real dels Miralls.**

  Seria difícil de fer amb el disseny actual. S'hauria d'implementar un mètode d'actualització dels Miralls molt menys costós.

- **Posar-ho tot en un sol idioma i fer traduccions.**

  Per exemple posar-ho tot en català (ara està barrejat català i anglès) i posar fitxers d'altres idiomes com a *plugins*. En teoria és fàcil amb el Qt Linguist. S'hauria de poder canviar l'idioma des d'un menú.

- **Posar *tooltips* a la interfície.**

  Només cal agafar el Qt Designer i entretenir-s'hi una estona.

- **Que es puguin guardar funcions de transferència.**

  Quan l'usuari troba una funció de transferència bona per un model l'hauria de poder guardar en un fitxer per després carregar-la fàcilment. Es podria guardar en un fitxer de text estructurat. També es podria fer que la funció de transferència es pogués associar al model, i quan l'usuari carrega el model que es carregui la funció de transferència automàticament. Es podria fer posant a la funció de transferència el mateix nom que el model (per exemple, "nom_model.mhd" i "nom_model.tf").

- **Que es puguin definir les funcions de transferència per defecte per cada Mirall.**

  Es podria fer amb un fitxer d'inicialització.

- **Millorar la interfície, potser.**

  Si els usuaris finals no estan satisfets amb la interfície, sempre es pot millorar o canviar completament, sempre que mantingui els mateixos *signals* i *slots*, aprofitant la independència de classes que proporciona aquest mecanisme de Qt. La nova interfície s'hauria de fer tenint en compte l'opinió dels usuaris finals.

- **Suggeriments dels usuaris finals.**

  Sempre es poden fer millores que suggereixin els usuaris finals.

8 Conclusions

L'objectiu del projecte era *implementar un mètode basat en els Miralls Màgics per facilitar la visualització i interpretació de models de voxels simples i registrats dins d'una plataforma de visualització d'imatges mèdiques*.

Recordem que els sistema de Miralls Màgics consisteix en visualitzar el model de voxels al centre en 3D i al seu voltant posar uns plans que semblen miralls on hi ha la visualització del model de voxels des de la posició del mirall.

A més a més s'havia de fer una interfície gràfica per permetre a l'usuari controlar els paràmetres de visualització: colors, opacitats, propietats...

Tot això s'havia d'integrar a la plataforma de visualització d'imatges mèdiques que ens havien proporcionat, i això implicava un disseny modular.

Doncs bé, arribats aquí podem dir que hem complert els objectius.

S'ha implementat una aplicació de visualització de models de voxels utilitzant la tècnica dels Miralls Màgics. Gràcies a les llibreries VTK el resultat final ha sigut bastant eficient pel que fa a velocitat, si bé encara es podria millorar una mica. La implementació s'ha fet amb 2 Miralls però és fàcilment ampliable a 3 o més sempre i quan sigui un nombre fix. Per fer un nombre variable de Miralls s'hauria de modificar més el codi.

L'usuari pot moure, rotar i escalar el model i moure, girar i fer *zoom* amb la camera. Això ho pot fer de forma senzilla amb el ratolí i el teclat, gràcies a les Qt i les VTK.

L'aplicació permet visualitzar models simples i registrats oferint les mateixes possibilitats de configuració de la visualització per a tots dos tipus. L'usuari pot canviar les funcions de transferència d'opacitat i de color per cada Mirall i volum, i també decidir si un determinat volum és visible en un determinat Mirall.

Els models poden ser de vàries mides. La mida i la posició dels Miralls s'adapten a la mida dels models.

Tots els paràmetres de visualització són configurables amb una interfície gràfica d'usuari dissenyada amb l'ajuda de les llibreries Qt i el Qt Designer.

S'ha intentat que la interfície fos intuïtiva i fàcil de fer servir. S'ha aconseguit bastant bé, però falta que la provin els usuaris finals.

L'aplicació dels Magic Mirrors està totalment integrada a la plataforma de visualització d'imatges mèdiques gràcies a un disseny modular. El disseny final consta de 3 mòduls: interfície gràfica, mòdul de control i mòdul de volums. Els mòduls són bastant independents entre ells.

Aquest disseny modular també fa que l'aplicació sigui fàcilment modificable i ampliable.

L'execució, almenys a l'ordinador de proves, és bastant ràpida i permet la interactivitat de l'usuari.

En principi no hi hauria d'haver problemes per compilar el codi en altres plataformes diferents de Linux, tot i que per compilar per Windows es necessita una llicència de pagament de Qt per Windows.

S'han fet servir eines de domini públic per desenvolupar l'aplicació. Totes són gratuïtes en la seva versió per Linux.

Per acabar, tot i que els objectius han estat assolits sempre és possible fer millores a l'aplicació perquè ofereixi més possibilitats i sigui més eficient i útil.

9 Bibliografia

- A. KÖNIG, H. DOLEISCH i E. GRÖLLER. *Multiple Views and Magic Mirrors – fMRI Visualization of the Human Brain*. Projecte VisMed (http://www.vismed.at), 1998.
  http://www.cg.tuwien.ac.at/research/vis/vismed/MM/

- MELANIE TORY. *Mental Registration of 2D and 3D Visualizations (An Empirical Study)*. Graphics, Usability, and Visualization (GRUVI) lab, School of Computing Science, Simon Fraser University.
  http://www.cs.sfu.ca/~mktory/personal/publications/vis03.pdf

- *The VTK User's Guide*. Kitware, Inc. (http://www.kitware.com), 2003.

- *Qt 3.3 Whitepaper*. Trolltech (http://www.trolltech.com).
  http://www.trolltech.com/pdf/whitepapers/qt33-whitepaper-a4.pdf

- *Qt Documentation*. Trolltech (http://www.trolltech.com).
  http://doc.trolltech.com

- *Insight Segmentation and Registration Toolkit*. Kitware, Inc. (http://www.kitware.com).
  http://www.itk.org

- *ITK 1.6 Documentation*. Kitware, Inc. (http://www.kitware.com).
  http://www.itk.org/Doxygen16/html/index.html

- *VTK Home Page*. Kitware, Inc. (http://www.kitware.com).
  http://www.vtk.org

- *VTK 4.5.0 Documentation*. Kitware, Inc. (http://www.kitware.com).
  http://www.vtk.org/doc/nightly/html

- RAMON MAS SANSÓ. *Informàtica Gràfica 2, Tema IV*. Departament de Ciències Matemàtiques i Informàtica, Universitat de les Illes Balears (http://dmi.uib.es).
  http://dmi.uib.es/~ramon/Docencia/ig2/IGII_tema4.doc

10 Manual d'Usuari

10.1 Iniciant Magic Mirrors

Primer de tot s'ha d'engegar l'aplicació de la plataforma de visualització d'imatges mèdiques. Un cop iniciada s'ha d'obrir un model prement el botó "Obrir" o anant al menú "Fitxer" ▸ "Obrir...". Apareixerà un quadre de diàleg per escollir el model a obrir.

Quan s'hagi obert el model ja es poden iniciar els Magic Mirrors anant al menú "Eines" ▸ "Iniciar Magic Mirrors", que estava desactivat fins ara.

També es pot optar per fer un procés de registre abans d'iniciar els Magic Mirrors, per treballar amb el model registrat. Consultar les instruccions pròpies del mòdul de registre per saber com es fa.

10.2 Finestra Principal de Magic Mirrors

Un cop s'inicia el mòdul de Magic Mirrors l'aplicació passarà a tenir un aspecte semblant al de la Figura 10.1.

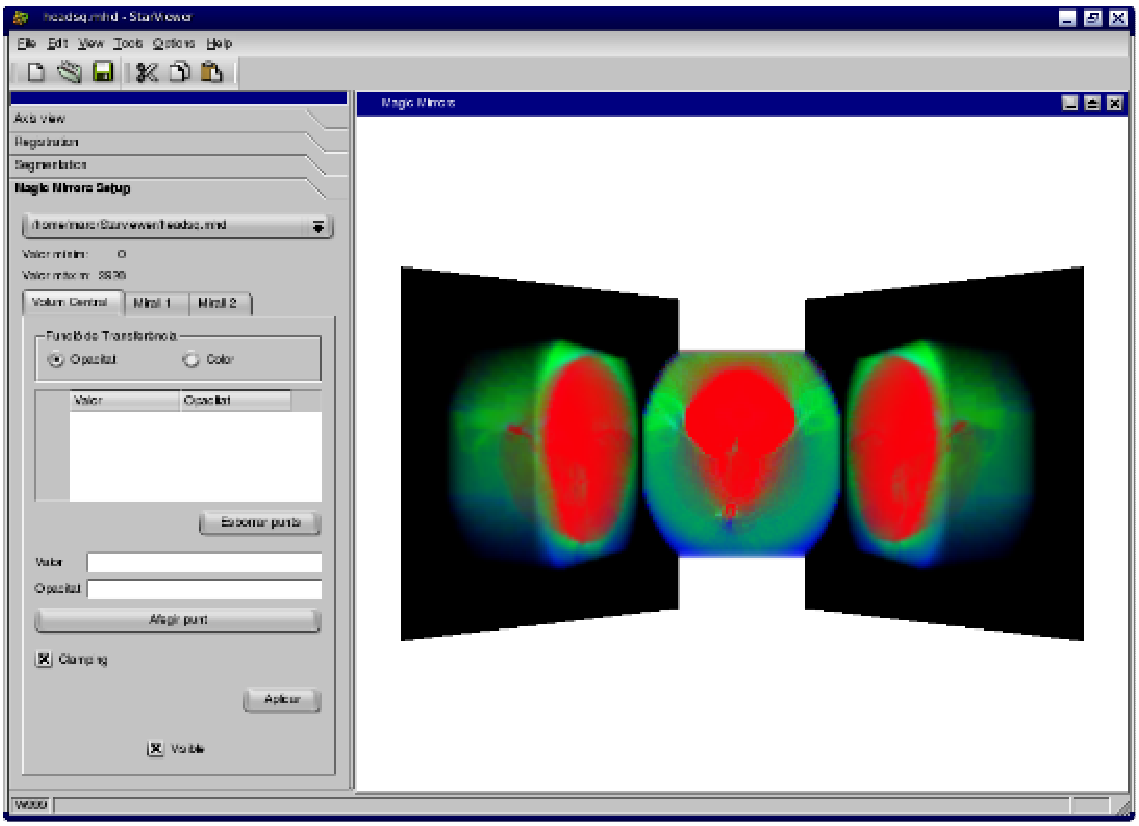


Figura 10.1. Magic Mirrors.

La finestra principal de Magic Mirrors serveix per mostrar la visualització del model de voxels i els Miralls Màgics i també permet interactuar amb la camera i el model

de voxels. Tota la interacció que es fa en aquesta finestra es fa amb el ratolí i el teclat.

La interacció es caracteritza per dues coses:

- L'element que rep es modifica a causa de la interacció. Pot ser un "actor", en aquest cas el model de voxels, o pot ser la camera.
- Com els moviments del ratolí afecten a l'element modificat per la interacció. En aquest aspecte hi ha 2 estils d'interacció: "joystick" i "trackball". Quan és "joystick" és la posició del ratolí durant la interacció la que afecta als moviments de l'objecte. Quan és "trackball" és el moviment del ratolí el que afecta als moviments de l'objecte.

Els dos parells de característiques es poden combinar de totes les maneres i formen els 4 possibles estils d'interacció. L'estil d'interacció es pot canviar en qualsevol moment amb les següents tecles:

'A' La interacció passa a un estil de tipus "actor". La interacció modificarà el model de voxels i s'ha de clicar a sobre seu per fer-ho.

'C' La interacció passa a un estil de tipus "camera". La interacció modificarà la camera.

'J' La interacció passa a un estil de tipus "joystick".

'T' La interacció passa a un estil de tipus "trackball".

Aquests són els controls per cada tipus d'interacció:

- **Joystick Actor.** Mantenint clicat el botó esquerre gira el model de voxels si està a sota del punter. La velocitat de gir depèn de la distància del punter al centre del model de voxels. Amb el botó dret s'escala el model, si el punter està cap a dalt el model es fa més gros, i si està cap a baix el model es fa més petit. Amb el botó del mig o la roda es desplaça el model de voxels. El model tendeix a anar cap a sota del punter.

- **Joystick Camera (per defecte).** Mantenint clicat el botó esquerre es gira la camera al voltant del punt focal, que inicialment és el centre del model de voxels. La velocitat de gir depèn de la distància del punter al centre de la finestra. Amb el botó dret es controla el *zoom*, cap a dalt s'augmenta i cap a baix es redueix. Amb el botó del mig o la roda es desplaça la camera i també el punt focal. La velocitat del desplaçament depèn de la distància del punter al centre de la finestra. Aquest és l'estil per defecte.

- **Trackball Actor.** Amb el botó esquerre es gira el model de voxels si està a sota del punter. El model gira mentre es mogui el ratolí. Amb el botó dret s'escala mentre es mogui el ratolí, cap a dalt es fa gros i cap a baix petit. Amb el botó del mig o la roda es desplaça el model de voxels, que es manté a sota el punter.

- **Trackball Camera.** Amb el botó esquerre es gira la camera mentre es mogui el ratolí. Amb el botó dret es controla el *zoom*, movent cap a dalt s'augmenta i movent cap a baix es redueix. Amb el botó del mig o la roda es desplaça la camera mentre es mogui el ratolí.

La millor manera d'aprendre a fer servir tots aquests controls de la finestra principal és anar provant els diferents estils i moviments.

Quan s'acaba una interacció de tipus actor (quan l'usuari deixa anar el botó del ratolí) s'actualitzen els Miralls. Apareixeran momentàniament 2 petites finestres que fan la visualització dels Miralls i després es tancaran i s'actualitzarà la finestra principal.

A vegades pot passar que la imatge dels Miralls no surti bé. La solució és tornar a interactuar amb el model de voxels per forçar una nova actualització.

10.3 Interfície de Control dels Magic Mirrors

A més a més de la finestra principal dels Magic Mirrors hi ha la interfície de control. Aquests són els elements d'aquesta interfície:

- **Un *combo* amb el nom del model obert.** Aquest *combo* serveix per seleccionar el model que es vol configurar. Només té utilitat pels models registrats, ja que obrint un model simple només té una opció.

Figura 10.2. *Combo* de models.

- **Valors mínim i màxim.** Aquests són els valors de propietat mínim i màxim del model seleccionat. Serveixen per saber el rang de valors de propietat a tenir en compte.

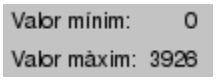


Figura 10.3. Valors mínim i màxim del model.

- **Pestanyes de selecció de Mirall.** Amb aquestes pestanyes es pot seleccionar a on es vol canviar la funció de transferència: al volum central, al Mirall 1 o al Mirall 2.

Figura 10.4. Pestanyes de selecció de Mirall.

- ***Radio buttons* "Funció de Transferència" ("Opacitat" i "Color").** Aquests *radio buttons* serveixen per triar el tipus de funció de transferència que es vol configurar: la d'opacitat o la de color. Segons l'opció triada apareixeran uns controls o uns altres a sota.

Figura 10.5. *Radio buttons* "Funció de Transferència".

- **Taula de punts de la funció de transferència.** En aquesta taula apareixeran els punts de la funció de transferència que es vagin afegint. Per la funció de transferència d'opacitat els punts estan formats per un valor de propietat i un grau d'opacitat de 0 a 1. Per la de color estan formats per un valor de propietat i un color. Aquests punts determinen la funció de transferència. Inicialment està buida, no mostra els valors per defecte.

Figura 10.6. Taula de punts de la funció de transferència d'opacitat.

- **Botó "Esborrar punts".** Aquest botó serveix per esborrar els punts seleccionats de la taula.

Figura 10.7. Botó "Esborrar punts".

- **Camps de text "Valor" i "Opacitat" (opacitat).** Aquí s'introdueixen el valor de propietat i el grau d'opacitat respectivament del punt que es vol afegir a la funció de transferència d'opacitat.

Figura 10.8. Camps de text "Valor" i "Opacitat".

- **Camp de text "Valor" i botó de seleccionar color (color).** Aquí s'introdueix el valor de propietat i es selecciona el color –amb un quadre de diàleg– respectivament del punt que es vol afegir a la funció de transferència de color.

Figura 10.9. Camp de text "Valor" i botó de selecció de color.

- **Botó "Afegir punt".** Aquest botó serveix per afegir el punt d'opacitat o de color configurat a la taula de punts corresponent.

Figura 10.10. Botó "Afegir punt".

- ***Checkbox* "Clamping".** És una opció de la funció de transferència per decidir com tracta els valors que queden fora de l'interval configurat. Està explicada més endavant.

Figura 10.11. *Checkbox* "Clamping".

- **Botó "Aplicar".** Aplica la nova funció de transferència al Mirall corresponent.

Figura 10.12. Botó "Aplicar".

- ***Checkbox* "Visible".** Determina si el volum és visible o no al Mirall seleccionat. Si es desmarca el volum desapareix del Mirall actual, independentment de la funció de transferència. Quan es marca es torna a fer visible. És útil per aplicar una opacitat nul·la a tot el volum ràpidament. Serveix per decidir les propietats que s'han de veure a cada Mirall quan es treballa amb un model registrat.

Figura 10.13. *Checkbox* "Visible".

Les funcions de transferència determinen el color i el grau d'opacitat del model de voxels. Aquestes funcions de transferència estan determinades per punts, de la manera que es veu a l'exemple de la Figura 10.14. L'opció *clamping* determina com s'extrapolen els punts per valors que queden fora del rang especificicat; si està marcada s'extrapola amb els valors dels extrems i si no està marcada s'extrapola a 0.

Valor de propietat	Opacitat
812	0,844
1021	0,82
1272	0,208
3794	0,015

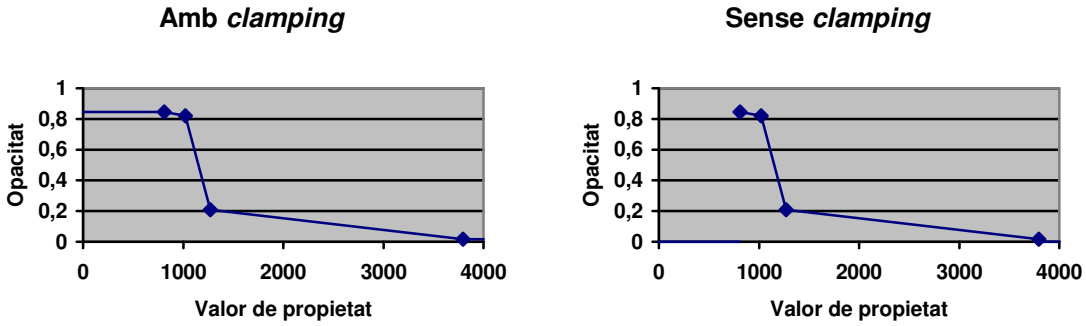


Figura 10.14. Exemple de funció de transferència.

Vist això, els passos per canviar una funció de transferència són els següents:

- Si s'està treballant amb un model registrat, seleccionar el volum al *combo*.

- Seleccionar la pestanya corresponent al volum central o al Mirall pel qual es vol canviar la funció de transferència.
- Seleccionar la funció de transferència que es vol canviar, la de d'opacitat o la de color, clicant al *radio button* corresponent.
- Afegir i treure punts de la taula fins que quedi la funció de transferència definida com es vol. El valor de propietat accepta valors de 0 a 65535 i el grau d'opacitat de 0 a 1 amb 3 decimals. Si alguna cosa no està bé surt una finestra amb un missatge d'error. El color es selecciona amb un quadre de diàleg. Per esborrar punts s'han de seleccionar a la taula i clicar el botó d'esborrar.
- Activar o desactivar l'opció de *clamping*, segons es vulgui.
- Aplicar-la prement el botó "Aplicar".

Quan s'aplica la nova funció de transferència s'actualitzen els Miralls i la visualització immediatament.

A més a més de la funció de transferència, també es pot decidir la visibilitat d'un volum al centre o a un Mirall. Això és útil principalment en el cas de visualitzar models registrats.

Aquests són els passos per canviar la visibilitat d'un volum:

- En el cas de models registrats, seleccionar el volum al *combo*.
- Seleccionar la pestanya corresponent al volum central o al Mirall pel qual es vol canviar la visibilitat.
- Marcar o desmarcar el *checkbox* "Visible", segons es vulgui.

Quan es canvia la visibilitat s'actualitzen els Miralls i la finestra principal immediatament, no cal tocar cap botó per aplicar-la.