

UNIVERSIDAD TÉCNICA FEDERICO SANTA
MARÍA

DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE



UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA



VISUALIZACIÓN VOLUMÉTRICA DE CUBOS DE DATOS ESPECTROSCÓPICOS

GERMÁN IGNACIO ORTEGA RODRÍGUEZ

Memoria de titulación para optar al título de Ingeniero Civil
en Informática.

Profesor Guía

:Mauricio Solar

Septiembre - 2016

UNIVERSIDAD TÉCNICA FEDERICO SANTA
MARÍA

DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE



UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA



VISUALIZACIÓN VOLUMÉTRICA DE CUBOS DE DATOS ESPECTROSCÓPICOS

GERMÁN IGNACIO ORTEGA RODRÍGUEZ

Memoria de titulación para optar al título de Ingeniero Civil
en Informática.

Profesor Guía

:Mauricio Solar

Profesor Correferente

:Mauricio Araya

Septiembre - 2016

Agradecimientos

Agradezco a los profesores Mauricio Solar y Mauricio Araya por el apoyo, tiempo y dedicación brindados para el desarrollo de la memoria.

Agradezco a la Doctora Amelia Bayo por su apoyo en la evaluación de la aplicación generada para la memoria.

Agradezco al proyecto de FONDEF IT15I10041: "Implementacion de Herramientas y Procesos del Observatorio Virtual Chileno" por su apoyo en este trabajo.

Dedicatoria

Dedico esta memoria a mis padres Germán de la Rosa Ortega Rodríguez y Margarita Angélica Rodríguez Rodríguez, quienes me han apoyado desde siempre en alcanzar mis metas y se han asegurado de que nada me falte.

Resumen

Los cubos de datos espectroscópicos son generados al captar ondas electromagnéticas y registrar sus detalles en archivos. El poder visualizar el contenido de estos archivos es una problemática de la computación actual. Se consideró particularmente el campo de la astronomía, para lo cual se analizó distintas bibliotecas existentes para la visualización considerando requisitos tales como el rendimiento gráfico, la velocidad de creación de gráficos, la interfaz de usuario, entre otros.

Finalmente se generó una propuesta desarrollada personalmente en base a los resultados obtenidos. Esta propuesta utilizó las bibliotecas *Matplotlib* y *Mayavi* del lenguaje *Python* para los gráficos creados, y se logró obtener un rendimiento y velocidades de creación adecuadas para el futuro desarrollo de una aplicación de uso científico.

Abstract

Spectroscopic data cubes are generated by capturing electromagnetic waves and keeping record on files. Visualizing the content of these files is one big problem of modern computer science. Considering particularly the astronomy field, diverse visualization libraries were analyzed with objectives such as graphics performance, rendering speed, user interface, among others.

Then, a personal application was created based on the results got in the analysis. This application made use of the Mayavi and Matplotlib libraries (both belonging to the Python language) to create all the plots and visualizations. The graphics performance and the rendering speed obtained were good enough to consider a future development for scientific fields.

Índice general

1. Introducción	1
1.1. Problemática General	2
1.2. Problema Específico	3
1.3. Contribución de la Memoria	6
1.4. <i>Outline</i> de la Memoria	6
2. Trabajo Previo	8
2.1. Métodos en 2D	8
2.1.1. Stacking	9
2.1.2. Mapa de Velocidades	9
2.1.3. Animación	11
2.2. Aplicaciones 3D	11
2.2.1. Gaia 3D	11
2.2.2. CARTA	12
2.2.3. Realidad Virtual	14
3. Implementaciones de Visualización Volumétrica	16
3.1. Matplotlib	18
3.1.1. Prueba de Concepto	19
3.1.2. Conclusiones	20
3.2. PyQtGraph	22
3.2.1. Prueba de Concepto	23
3.2.2. Conclusiones	26
3.3. Mayavi	27
3.3.1. Prueba de Concepto	28
3.3.2. Conclusiones	30

4. Implementación para Astronomía	33
4.1. Aplicación Generada	34
4.2. Validación con Astrónomo	36
5. Conclusiones	41
5.1. Trabajo Futuro	42
A. Instalación de bibliotecas	45
A.1. Matplotlib	45
A.2. PyQtGraph	45
A.3. Mayavi	46

Capítulo 1

Introducción

La visualización de datos computacionales sigue siendo uno de los mayores desafíos del mundo moderno. Las personas no pueden entender con facilidad tablas y matrices de números, por lo que es necesario otorgar la información de una manera sencilla e intuitiva, que permita a los investigadores analizar un objeto como podrían hacerlo en la vida real y así poder reconocer patrones y formas.

Esto se vuelve un problema de incremental complejidad cuando se trata con archivos de gran tamaño como en el caso de imágenes astronómicas, y en particular los cubos de datos espectroscópicos. Estos consisten de ondas electromagnéticas pertenecientes al espectro milimétrico y submilimétrico, no visibles normalmente al ojo humano.

Expertos afirman que se produce un efecto de “cuello de botella” en la ciencia cuando de visualización se habla, ya que por muy rápida que sean las máquinas, el avance será marcado finalmente por que tan rápido pueden ser procesados los datos por los investigadores [1].

En la presente memoria se definirá problema en detalle y se introducirán las soluciones existentes, así como también el *software* más relevante y proyectos que han abordado el problema, para luego proponer la solución desarrollada.

Se pondrá especial énfasis en el proyecto *ALMA* (*Atacama Large Millimeter/submillimeter Array*), ubicado en el desierto de Atacama, el cual entrega como producto final archivos de cubos de datos espectroscópicos del espectro microondas. Se analizará los archivos en sí, para proponer maneras de tratarlos a fin de obtener una buena y sencilla visualización tridimensional.

La propuesta que se busca en esta memoria debe cumplir con ser:

-
- Sencilla: Sin ejecutables pesados, sin requerimientos técnicos muy altos y de fácil uso. La interfaz de usuario debe ser intuitiva y fácil de usar, y los requerimientos de la aplicación no deben ser excesivos, tanto técnicos como de *software*. El objetivo es que desde un computador común de escritorio sea posible visualizar algunos cubos.
 - Confiable: Los resultados deben estar validados por profesionales entendidos en la materia y al tanto del estado del arte en aplicaciones de visualización. También deben ser comparados con resultados obtenidos y verificados previamente por laboratorios profesionales.
 - Abierta a la comunidad y al desarrollo. Esto implica que el código utilizado no debe ser demasiado complejo y no se debe utilizar *software* privativo.
 - Especializada en los archivos utilizados por *ALMA*, particularmente los cubos de datos en formato *fits*.

La línea de trabajo seguida en la memoria es la de investigar distintas alternativas para la visualización y crear una propuesta de uso en la astronomía. El código realizado debe ser sencillo a fin de abrir la posibilidad a futuro para continuar con el desarrollo e incluso con la posibilidad de aportar a una biblioteca para abrir la posibilidad de aportes de la comunidad.

1.1. Problemática General

Uno de los problemas del uso de herramientas computacionales actuales es la visualización multidimensional con datos que representan un objeto. La problemática principal es el cómo llevar estos datos, números y matrices a una imagen sencilla para analizar desde el punto de vista humano. Es necesario que la representación sea fidedigna y fácil de interpretar, y también que sea rápido y sencillo observar un objeto desde más de una perspectiva, tal como se realiza en el mundo real.

En particular con los datos en tres dimensiones, aparecen cubos de datos que representan un volumen, correspondiente a un objeto. Es posible encontrar estas representaciones en muchos campos de ciencia y tecnología, tal como en medicina, con simulaciones de partes internas del cuerpo humano. Por ejemplo, en la Figura 1.1 se aprecia la simulación de una dentadura y quijada humana, generada por el *software OGRE3D* [2].

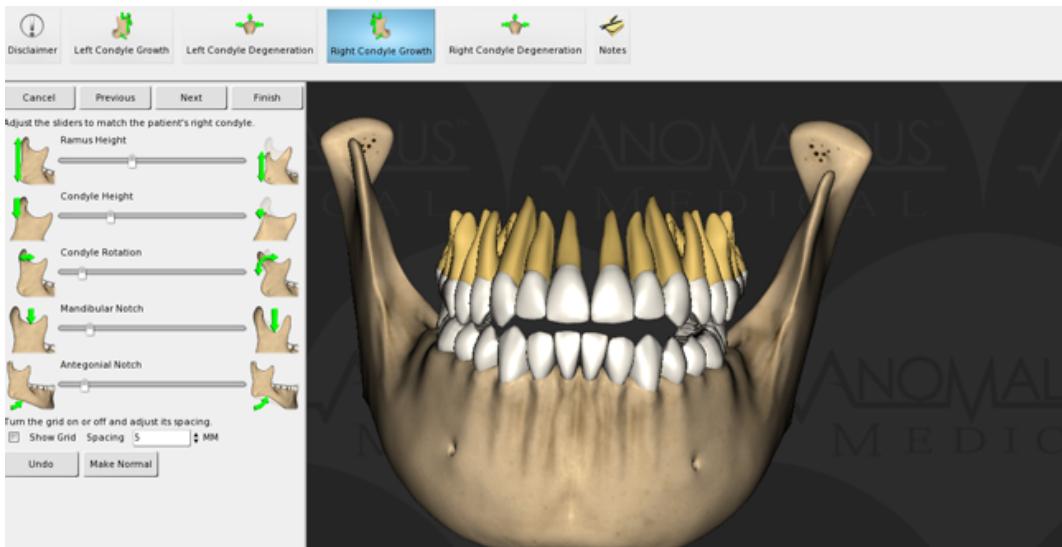


Figura 1.1: Modelo de dentadura hecho con *Ogre3D*. Fuente: <https://goo.gl/I7k8wa>

Otro uso importante se da en la minería y geología, a fin de modelar y analizar los contenidos de la tierra a distintas profundidades y determinar dónde construir pozos y hacer excavaciones. Por ejemplo en la Figura 1.2 se aprecia un diagrama de superficies utilizado por geólogos para encontrar pozos de petróleo [3].

Y así, la misma problemática se da en diversos campos tales como ingeniería, arquitectura, diseño de productos, astronomía, etc.

1.2. Problema Específico

En el campo de la astronomía el problema de la visualización se da, entre otros, con cubos espectroscópicos, guardados en archivos de gran tamaño. Estos cubos son generados por diversos proyectos astronómicos, entre los que puede mencionarse *ALMA* [4]. El proyecto *ALMA* consiste en antenas ubicadas en el desierto de Atacama, las cuales captan ondas electromagnéticas de longitudes de onda comprendidas dentro del espectro milimétrico y submilimétrico. Estas antenas se enfocan en un cuadrado en el cielo, el cual posee los ejes usados en la astronomía que son *right ascension (RA)* y *declination (DEC)*. La *right ascension* corresponde a la distancia angular con respecto

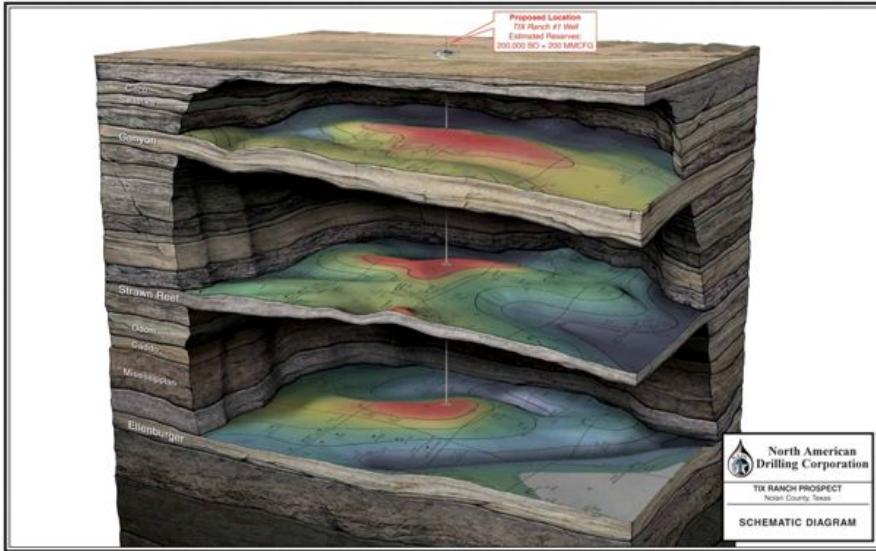


Figura 1.2: Esquema de pozos subterráneos de petróleo. Fuente: <https:// goo.gl/ltT08C>

al punto vernal (punto por el que pasa el sol durante el equinocio de otoño) a lo largo de la línea del Ecuador, mientras que la *declination* corresponde a la distancia angular de un punto con respecto al Ecuador terrestre, explicado gráficamente en la figura 1.3.

Las ondas que son captadas deben en primera instancia ser corregidas debido a la rotación de la tierra, y luego las distintas frecuencias captadas son separadas en canales, asignándoles distintas posiciones de profundidad en relación a la velocidad con la que se mueven respecto a la tierra. Así se añade a las 2 dimensiones existentes la de *velocity (VEL)*, generando finalmente un cubo de datos.

Estos cubos eran guardados en archivos de distintos formatos entre los distintos proyectos. En el año 1981 se creó el formato *fits* [5] para el almacenamiento y transporte de estos cubos, a fin de unificar el esfuerzo de la comunidad científica. Estos archivos vienen compuestos por un *header*, el cual contiene los metadatos (dimensiones del cubo de datos, historial de ajustes, entre otros) y de una *data*, la cual consiste en matrices (usualmente en 2D o 3D) que representan los datos captados. Este formato es el utilizado oficialmente por *ALMA* y será utilizado en la presente memoria.

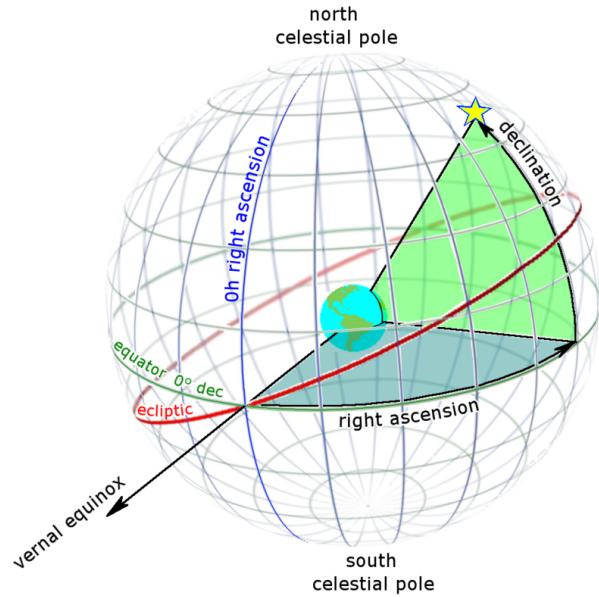


Figura 1.3: Coordenadas *right ascension* y *declination*. Fuente: <https://goo.gl/27UX70>

Estos archivos pueden alcanzar altísimas dimensiones de tamaño, debido a la naturaleza propia de la astronomía, por lo que es necesario una manera sencilla y confiable de interpretarlos y graficarlos.

Una manera alterna de visualizar los cubos es organizando la *data* en véxoles (equivalentes a píxeles en tres dimensiones), los cuales contienen el total de la *data* acumulada en una región cúbica. Estos cubos presentan complejidad al momento de analizarlos y es necesario el uso de un *threshold*, es decir, un valor bajo el cual un determinado véxel sea despreciable. Este valor es usado para eliminar ruido y errores en las mediciones, a fin de dejar solo los datos relevantes.

Dentro de los problemas específicos a resolver con respecto a lo mencionado anteriormente se encuentran:

- Los métodos de visualización existentes presentan limitaciones (mentionadas en el capítulo 2).
- El mejor *software* de visualización es demasiado específico y restrictivo a determinados proyectos.

-
- Es necesaria una manera sencilla y accesible de visualización, que no caiga en las limitaciones de los existentes.

Para esto será utilizada la biblioteca *astropy* [6] del lenguaje *Python*, la cual está especializada en el manejo de archivos de tipo *fits*, y posee además diversas herramientas astronómicas. Esta biblioteca es de uso libre y está orientada a la comunidad, por lo que cualquiera puede contribuir a su desarrollo.

1.3. Contribución de la Memoria

Con la presente memoria se buscó contribuir al campo de la visualización volumétrica, particularmente en el caso de cubos espectroscópicos de astronomía. Se mencionó distintos métodos existentes para la visualización, con sus ventajas y desventajas. Se mencionó también algunos proyectos existentes en el campo, con el objetivo de analizar sus puntos fuertes y débiles. Se analizó distintas bibliotecas de visualización volumétrica y se escogió una de ellas para llevarla más adelante. Se desarrolló una aplicación para solucionar el problema propuesto, utilizando para ello aportes de profesionales. Se analizó los alcances y limitaciones de esta propuesta, definiendo así trabajo para hacer a futuro.

1.4. *Outline* de la Memoria

En este capítulo se explica el tema a modo general del que trata la memoria, así como también el contexto actual relacionado al mismo. Luego, en la sección de Problemática General se entra en más detalles con respecto al problema a resolver, para definir claramente los detalles, formatos, conceptos claves y alcances en la sección de Problema Específico. Se explica luego en la sección de Contribución el impacto de la memoria.

En el capítulo de Trabajo Previo se analiza los métodos existentes para la visualización. Se analiza primero los métodos bidimensionales en la sección de Métodos en 2D, explicando conceptos tales como *Stacking*, Mapa de Velocidades y Animación, cada uno en su correspondiente sección con detalles y ejemplos gráficos. Luego se menciona algunas aplicaciones tridimensionales en la sección de Aplicaciones 3D, incluyendo *Gaia 3D*, *CARTA* y Realidad

Virtual, mencionando las características, estado actual y limitaciones en cada una de sus secciones.

En el capítulo de Implementaciones de Visualización Volumétrica se comienza ya el trabajo más práctico, explicando las plataformas, lenguaje y bibliotecas a analizar, así como también algunos elementos que son comunes a todos los análisis posteriores y requerimientos que se busca obtener con cada uno. Posteriormente se analiza 3 bibliotecas de visualización: Matplotlib, PyQtGraph y Mayavi, incluyendo elementos tales como descripción oficial, características técnicas, estado actual de desarrollo, entre otros. Para cada una de estas bibliotecas se realiza una Prueba de Concepto, en la que se detalla los procedimientos seguidos para la generación de visualización volumétrica y se muestra también el resultado visual obtenido, para posteriormente obtener Conclusiones de cada una en base a los requerimientos mencionados anteriormente.

Una de las bibliotecas analizadas es escogida en el capítulo de Implementación para Astronomía en base a comparaciones realizadas entre las 3 alternativas, para luego utilizarla y generar una aplicación con usos en astronomía. Esta será explicada con más detalles de funcionamiento, interfaz y resultados visuales en la sección de Aplicación Generada, seguida del *feedback* e ideas para trabajo a futuro otorgado por un profesional en la sección de Validación con Astrónomo.

Finalmente, en el capítulo de Conclusiones se resume los resultados obtenidos. Se menciona posteriormente el trabajo a hacer a futuro y también las posibilidades de desarrollo a partir de la memoria. Este capítulo es seguido por las Referencias citadas a lo largo de la memoria y finalmente el apéndice, que contiene las líneas de comando utilizadas para instalar las dependencias necesarias para cada biblioteca.

Capítulo 2

Trabajo Previo

El trabajo que se ha realizado en cuanto a visualización ha estado siempre limitado por la capacidad del *hardware* utilizado, principalmente procesadores y tarjetas gráficas. Antes era demasiado costosa la visualización gráfica y volumétrica, pero en 1986 se comenzó a poner énfasis en esta área, al considerar la comunidad científica que el *hardware* había alcanzado el nivel necesario para ello [7]. Desde entonces ha existido el campo de la computación gráfica dentro de la ingeniería en informática, con énfasis en la visualización tanto bidimensional como tridimensional, con usos en campos tales como la ciencia, ingeniería, videojuegos, publicidad, etc.

2.1. Métodos en 2D

La visualización volumétrica es inherentemente más costosa y lenta que la visualización bidimensional. No era posible obtener velocidades de interacción psicológicamente adecuadas, y es por eso que se ha desarrollado métodos en 2 dimensiones que buscan representar de la mejor manera posible los datos por medio de entregar una versión resumida de ellos. Este acercamiento de solución presenta desventajas frente a una visualización volumétrica pura, ya que se pierde información que el usuario pueda interpretar, además de que se pierde la noción general del objeto que se está observando.

En las secciones 2.1.1, 2.1.2 y 2.1.3 se menciona algunos de estos métodos, con ejemplos y características de cada uno.

2.1.1. Stacking

El método de *stacking* es un procesamiento de datos que genera una visualización en 2D de los cubos, que consiste en sumar (del inglés *stacking* “apilar”) los datos a lo largo de una dimensión, a fin de entregar una sola imagen bidimensional en la cual las distintas intensidades que había en la dimensión apilada aparecen agregadas. En los cubos de datos es posible generar un máximo de 3 distintas imágenes apiladas, una por cada dimensión que se suprime, y el resultado es similar al de la figura 2.1.

En esta imagen los distintos colores indican la suma total de las intensidades, siendo negro cercano a nulo, azul oscuro baja intensidad, celeste intensidad media, amarillo alta intensidad y rojo la mayor intensidad. Por lo tanto es posible apreciar que se trata de un objeto ovalado con materia concentrada principalmente en el centro (como una galaxia elíptica).

La principal desventaja de este método es la pérdida de visiones específicas, es decir, sólo se puede dar una vista general de lo que se tiene, perdiéndose así los detalles de cada zona en particular de la dimensión perdida. No es posible apreciar por ejemplo si las distintas emisiones captadas están alejando o acercando relativamente a la tierra, a diferencia de otras visualizaciones tridimensionales. Además, los colores presentados son relativos, y al momento de comprar una imagen por otra los mismos colores podrían diferir en significado.

2.1.2. Mapa de Velocidades

Un mapa de velocidad consiste en un tipo de visualización bidimensional similar a *stacking*, desde el punto de vista de que se pierde una dimensión, pero luego es coloreado en base a las velocidades de las distintas emisiones con respecto a la tierra, es decir, compensa uno de los problemas del método anterior. Para demostrar diferentes intensidades hace uso de líneas de relieve, siendo las zonas con mayor densidad de líneas las más intensas. Por ejemplo en la figura 2.2 se puede apreciar un ejemplo de este método.

En esta imagen puede apreciarse una escala de colores que indica de cierto modo la proximidad que cada sección tendría en el cubo original, dando una idea general de como es el cubo en sí.

Si bien soluciona uno de los problemas de *stacking*, sigue teniendo deficiencias en entregar vistas específicas, y le da prioridad a una vista general e interpretada (analizando automáticamente los datos), en lugar de otorgarle

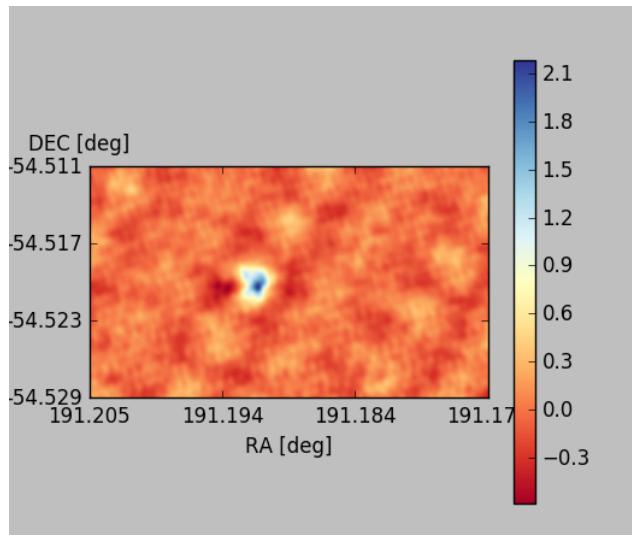


Figura 2.1: Imagen *fits* a la cual se le aplicó *stacking*. La coloración (indicada en la escala a la derecha) indica los valores de los datos acumulados. Fuente: Elaboración propia

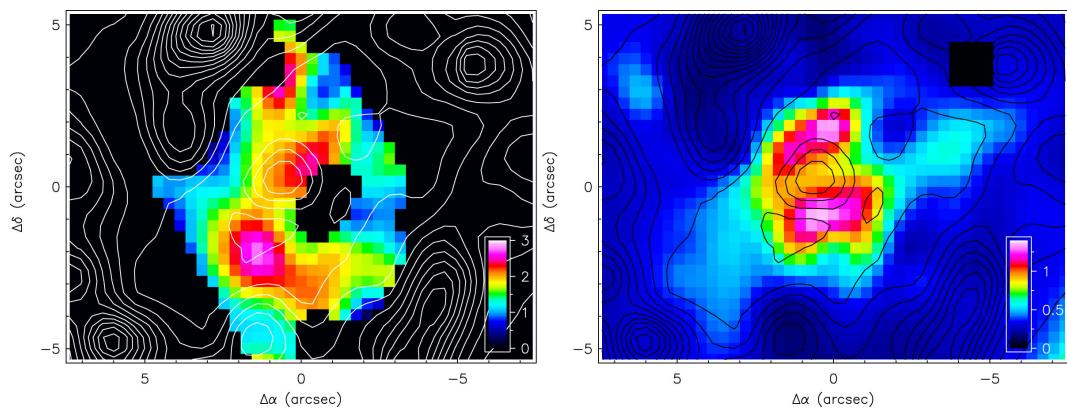


Figura 2.2: Mapa de velocidades de la galaxia M100. Fuente: *Discovery of a galactic wind in the central region of M100* [8].

mayor libertad al usuario al momento de analizar el objeto.

2.1.3. Animación

El método de animación consiste en mostrar cada una de las imágenes del archivo *fits* por separado, mostrando una por una y otorgándole el poder al usuario para detener y controlar la animación, a fin de poder enfocarse en alguna imagen en particular.

En la figura 2.3 se muestra un ejemplo de animación, en el cual se muestran 4 de las imágenes contenidas dentro de un mismo archivo.

Estas 4 imágenes, más las demás que las sucedan y precedan, generan un objeto al juntarlas. Este método tiene la ventaja de que le permite al usuario un mayor nivel de libertad al momento de moverse y ver específicamente una sección, e interpretar libremente, sin que el *software* lo haga de antemano.

La principal desventaja de este método es que al ser tan específico se pierde la noción de lo que se está mirando en general, y la única manera de hacerlo es analizando todas las imágenes existentes y generando una imagen mental. Además es engorroso moverse a una zona en específico, ya que es necesario revisar una por una las imágenes, o bien tenerlas todas abiertas, lo cual genera un exceso de información para el usuario.

2.2. Aplicaciones 3D

A medida que ha avanzado el *hardware* de visualización han aparecido múltiples aplicaciones (tanto comerciales como *open source*) dedicadas a la visualización volumétrica. Algunas de estas alternativas son específicas a un problema en particular, mientras que otras buscan crear herramientas generales para desarrollar distintos casos de visualización volumétrica.

En las secciones 2.2.1, 2.2.2 y 2.2.3 se analiza algunas de estas herramientas, dejando espacio también a posibles trabajos futuros.

2.2.1. Gaia 3D

Gaia 3D es un *software* de visualización en 3D, orientado principalmente a la enseñanza, el cual promete revolucionar la manera en que los alumnos aprenden sobre ciertas materias, permitiéndoles visualizar de manera realista los distintos objetos de las materias que estudian, como biología, química,

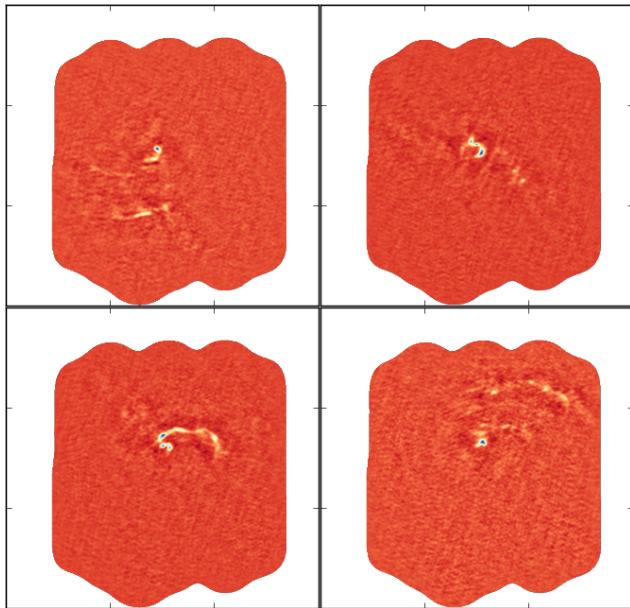


Figura 2.3: 4 imágenes de un archivo *fits* correspondientes a un mismo objeto pero a distintas profundidades. Fuente: Elaboración propia.

física, entre otros [9]. Por ejemplo se muestra en la figura 2.4 la visualización en 3D de un corazón humano (generada por el *software*).

Las principales desventajas que este *software* presenta parte por el hecho de que está optimizado para la educación, perdiendo potencial en otras áreas.

Por otro lado, el *software* no está orientado únicamente a la astronomía, sino que a muchas más disciplinas, lo cual presenta por un lado muchas herramientas que serían innecesarias, y por otro lado una carencia de especialización en la materia significa que ciertas herramientas más específicas no se podrán encontrar.

Finalmente, es un *software* privativo, que requiere de una licencia para ser utilizado, ofreciendo a lo más 30 días de prueba gratuita. Debido a esto no se solucionaría el problema de crear herramientas más generales orientadas a toda la comunidad científica.

2.2.2. CARTA

El proyecto *CARTA: The Cube Analysis and Rendering Tool for Astronomy* busca crear una nueva plataforma que reemplace a las existentes. A

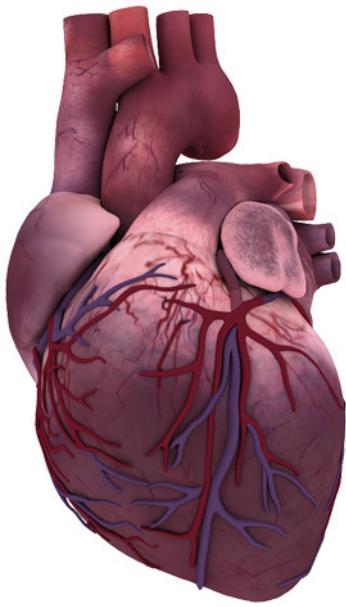


Figura 2.4: Corazón humano hecho en *Gaia 3D*. Fuente: <https://goo.gl/IumH47>

diferencia del caso anterior, está hecho pensando en los proyectos *ALMA* (mencionado en la sección 1.2), *VLA* (*Very Large Array*, proyecto de 27 antenas de radio ubicado en Nuevo México, Estados Unidos) y el *SKA* (*Square Kilometer Array*, proyecto en construcción de antenas de radio, con ubicación posible en Australia o Sudáfrica). El proyecto *CARTA* se enfoca en imágenes de gran tamaño (50 GB a 1 TB), con los objetivos de ser extensible y mantenable, otorgar posibilidad a la comunidad para desarrollar, entre otros [10].

En resumen, busca tomar lo positivo del *software* utilizado actualmente y añadirle la posibilidad de mantenimiento por parte de la comunidad. Este proyecto aún está en desarrollo. La figura 2.5 presenta una *screenshot* de como luce el *software* actualmente.

Actualmente sigue en desarrollo, buscando cubrir específicamente proyectos astronómicos de grandes laboratorios, siendo un referente a considerar durante el desarrollo.

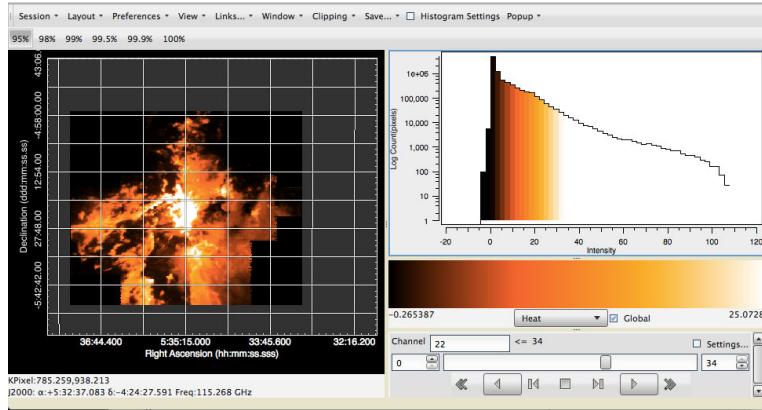


Figura 2.5: Estado actual del proyecto *CARTA* al 31 de Marzo, 2016. Fuente: <https://goo.gl/RuIWUB>

2.2.3. Realidad Virtual

En la Universidad Tecnológica de California (*Caltech*) se lleva a cabo un proyecto que consiste en aprovechar dispositivos de realidad virtual (como *Oculus Rift*) para llevar al espectador a una visualización realmente en 3D, en la cual se sumerja y sea capaz de moverse alrededor del objeto visto, siendo lo más cercano que pudiera llegar a observar el objeto en la vida real [1]. En la figura 2.6 se muestra como un avatar representa al usuario, siendo capaz de moverlo para así cambiar su perspectiva.

Este proyecto se encuentra aún en desarrollo, y tiene como objetivo llegar a masificarse, ya que ahora mismo los dispositivos necesarios para utilizarlo no se encuentran al alcance de todos, y además se enfoca en la visualización en general más que en la astronomía, que es lo que se busca con esta memoria. Cabe mencionar también que una virtualización requiere de una gran cantidad de procesamiento, reservando su uso solo para los mejores computadores cuando se trata de archivos de gran tamaño.

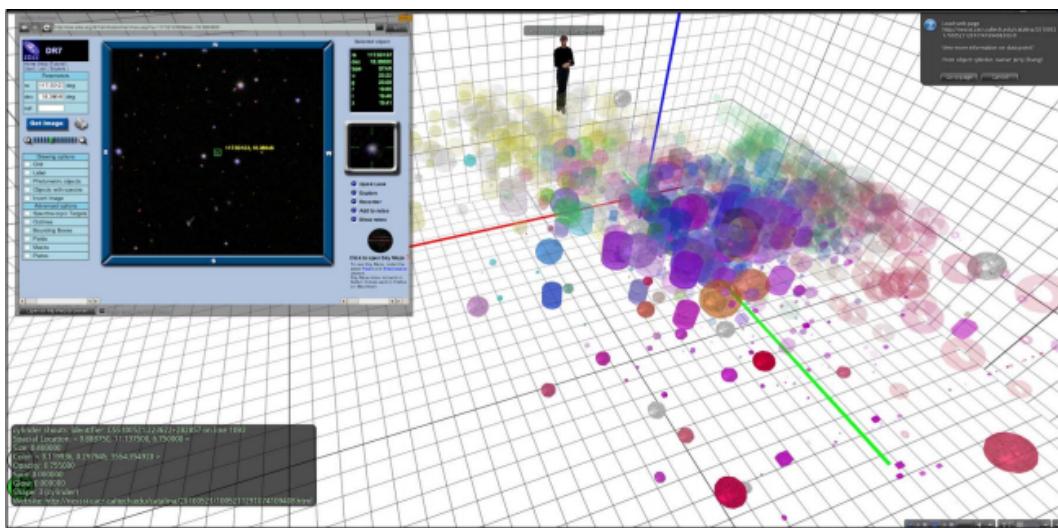


Figura 2.6: Avatar humano en simulación tridimensional. Fuente: <https://goo.gl/kLaJJ>

Capítulo 3

Implementaciones de Visualización Volumétrica

Existen múltiples alternativas para la visualización volumétrica en diversos lenguajes, pero solo las alternativas para el lenguaje *Python* serán analizadas, debido a que la biblioteca *astropy* es necesaria para el desarrollo propuesto. Entre estas alternativas puede mencionarse:

- *Matplotlib*: Biblioteca para la generación de gráficos.
- *PyQtGraph*: Biblioteca para la generación de gráficos orientada a ciencia e ingeniería.
- *Mayavi*: Biblioteca para visualizar datos científicos.
- *VisPy*: Biblioteca para visualización científica interactiva.

Las primeras 3 serán examinadas con el objetivo de determinar si serán útiles en este caso. En el caso de *VisPy* se decidió que sería dejada de lado, ya que aún se encuentra en pleno desarrollo y posee poca documentación al respecto.

Para este análisis se creará una visualización volumétrica con las herramientas que otorga cada biblioteca, utilizando para ello diversos archivos *fits* obtenidos desde la *Science Verification Data* de *ALMA* [11], la cual consiste en datos astronómicos bien conocidos utilizados para comprobar la correcta funcionalidad del observatorio. Esta visualización constará de un gráfico de volumen e idealmente opciones de *GUI* para el usuario, que permitan manejar la visualización.

Los requerimientos que se busca cumplir al analizar las alternativas son:

- **Rendimiento:** La visualización debe funcionar a velocidad de interacción, es decir, las operaciones básicas de gráficos (*zoom*, desplazamiento, rotación) no deben hacer esperar al usuario. Para este ítem se considerará rendimiento "Bajo" tiempos de espera de más de 5 segundos, "Medio" tiempo de entre 1 y 5 segundos y "Alto" tiempos menores a un segundo.
- **Velocidad de renderizado:** La velocidad a la cual se producen los gráficos inicialmente no debe ser excesiva. En este caso, a diferencia del anterior, es posible tolerar tiempos de espera no instantáneos, dentro de cierto margen, por lo tanto se considerará velocidad "Lenta" tiempos de espera de más de 20 segundos, "Media" tiempo de entre 5 y 20 segundos y "Rápida" tiempos menores a 5 segundos.
- **Opciones de personalización:** El poseer poder sobre el tamaño, el nombre de los ejes, los colores, entre otros para personalizar la visualización permite llevar el desarrollo más allá y ajustarlo a los requerimientos propios de cada institución astronómica. Para medir este ítem se creará en primera instancia un objetivo a alcanzar de visualización, y en base a este objetivo se considerará las opciones como "Bajas" al no ser capaz de personalizar todo lo planeado, "Medias" el tener opciones suficientes para cumplir con el mínimo y "Altas" en caso de que exista aun más opciones no necesarias para el objetivo planeado pero que permita mayor expansión a futuro.
- **Opciones de *GUI*:** Es necesario que el usuario tenga control sobre los datos a mostrarse y también que posea información relevante acerca de la visualización actual. El poder agregar *sliders*, botones y cajas de texto es esencial para este propósito. Al igual que en el ítem anterior, se planeará un objetivo mínimo de elementos de *GUI* necesarios para la visualización, calificando las opciones como "Bajas" el no poder cumplir el objetivo, "Medias" el tener el mínimo necesario y "Altas" el tener aún más de lo necesario, permitiendo ampliar y planear trabajo a futuro.

En todas las alternativas los datos del archivo serán almacenados como un arreglo tridimensional, el cual será copiado a una nueva variable para ser alterado según corresponda a fin de poder modificar la visualización sin

perder los datos originales. Este arreglo está hecho con *NumPy*, una biblioteca de *Python* especializada entre otras cosas en el manejo de arreglos de muchas dimensiones, la cual cuenta con poderosas herramientas para procesar los datos de manera eficiente [12].

Uno de los requerimientos básicos para las visualizaciones es la creación de un *threshold*, correspondiente a un valor bajo el cual se ocultarán los datos. Se considerará para todos los casos un valor base, correspondiente al *Root Mean Square (RMS)*, calculado según la ecuación 3.1:

$$RMS = \sqrt{\frac{1}{N} \sum_{n=1}^N X_n^2}. \quad (3.1)$$

Con N siendo la cantidad de datos en el archivo y X_n el dato en la posición n . Este valor inicial es necesario para eliminar el ruido propio de las observaciones astronómicas, reduciendo la cantidad de datos a mostrarse y generando una renderización más rápida como resultado. Luego, este valor será multiplicado por un factor escogido por el usuario a fin de poder ocultar los datos que sean menores a cierto valor, teniendo como tope máximo el máximo valor posible dentro del archivo.

Para todas las alternativas a analizar se presentará la visualización volumétrica del archivo *M100line.image.fits*, correspondiente a la galaxia espiral M100, también conocida como *Messier 100*. Al final de cada sección se analizará si la alternativa cumple con los requerimientos mencionados anteriormente.

3.1. Matplotlib

Matplotlib es una biblioteca para gráficos 2D en *Python* con énfasis en generar gráficos de calidad para presentaciones e informes [13]. *Matplotlib* busca generar gráficos de manera sencilla para el usuario, para que con unas pocas líneas de código sea posible generar gráficos personalizables. La biblioteca es de código abierto y está abierta a aportes de la comunidad.

A pesar del enfoque 2D de la biblioteca, posee también herramientas para gráficos tridimensionales, como líneas, *scatter plots*, *wireframes*, superficies, contornos, etc.

La versión actual estable de *Matplotlib* es la 1.5.1 y funciona en *Windows*, *Linux* y *OSX*. Los prerequisitos de sistema para el correcto funcionamiento

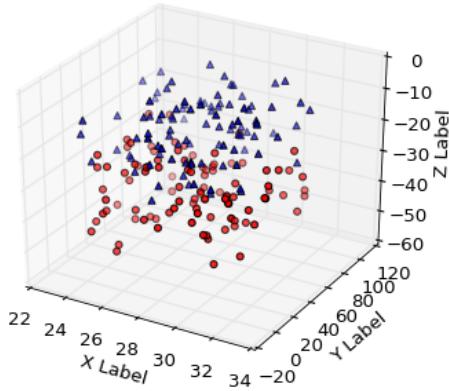


Figura 3.1: *Scatter plot* realizado con *Matplotlib*. Fuente: <http://goo.gl/crKrEK>

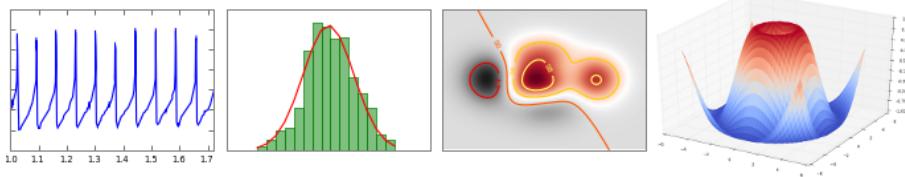


Figura 3.2: Ejemplos de gráficos realizados con *Matplotlib*. Fuente: <http://goo.gl/1EqBnC>

de la biblioteca incluyen *NumPy* y *SciPy* (procesamiento numérico). También, para este caso en particular es necesaria la biblioteca *astropy* para el manejo de archivos *fits*. Los comandos para la instalación de esto pueden encontrarse en el apéndice A.1.

3.1.1. Prueba de Concepto

La implementación del código para generar la visualización se hizo utilizando el *scatter plot* antes mencionado. Los pasos a seguir para esta fueron:

1. Se cargó el archivo *fits* a utilizarse.
2. Se extrajeron los datos, guardando la *data* en una matriz tridimensional de *NumPy*.

-
3. Se calculó el *threshold* de los datos.
 4. Se obtuvieron las coordenadas de aquellos datos que fueran mayor al *threshold*.
 5. Se llamó a la función *scatter* de *Matplotlib* con las coordenadas obtenidas como parámetro.
 6. Se mostró el resultado por pantalla.

La interfaz de usuario consta de los botones básicos otorgados por la biblioteca, los que permiten navegar entre las distintas *views* (es decir, la vista generada al poner la cámara en un determinado punto) y también guardar una imagen de la vista actual.

En cuanto al resultado visual, en la figura 3.3(a) se puede apreciar la visualización con el mínimo *threshold*. En la figura 3.3(b) se puede apreciar la visualización con un *threshold* equivalente a 5 veces el original. Por último, en la figura 3.3(c) se puede apreciar la visualización con el mismo *threshold* anterior desde una perspectiva diferente que permite visualizar más claramente la forma de la galaxia. Todas las figuras son de elaboración propia.

3.1.2. Conclusiones

En cuanto a los requerimientos que se busca y el desempeño mostrado por *Matplotlib* puede mencionarse:

- Rendimiento: La visualización posee un bajo rendimiento, generándose tiempos de espera cercanos a 15 segundos al rotar el gráfico. Esta medición fue realizada con el *threshold* al mínimo y por ende hay más puntos. Además no posee la opción de realizar *zoom* a los datos.
- Velocidad de renderizado: La velocidad inicial llega a ser considerablemente lenta, demorando aproximadamente 30 segundos en mostrar el gráfico por primera vez.
- Opciones de personalización: La biblioteca posee una cantidad alta de parámetros bien documentados e intuitivos para modificar los gráficos. Entre ellos se encuentran opciones tales como el poder definir el color de todos o cada uno de los puntos, y la opacidad de los mismos, entre otros.

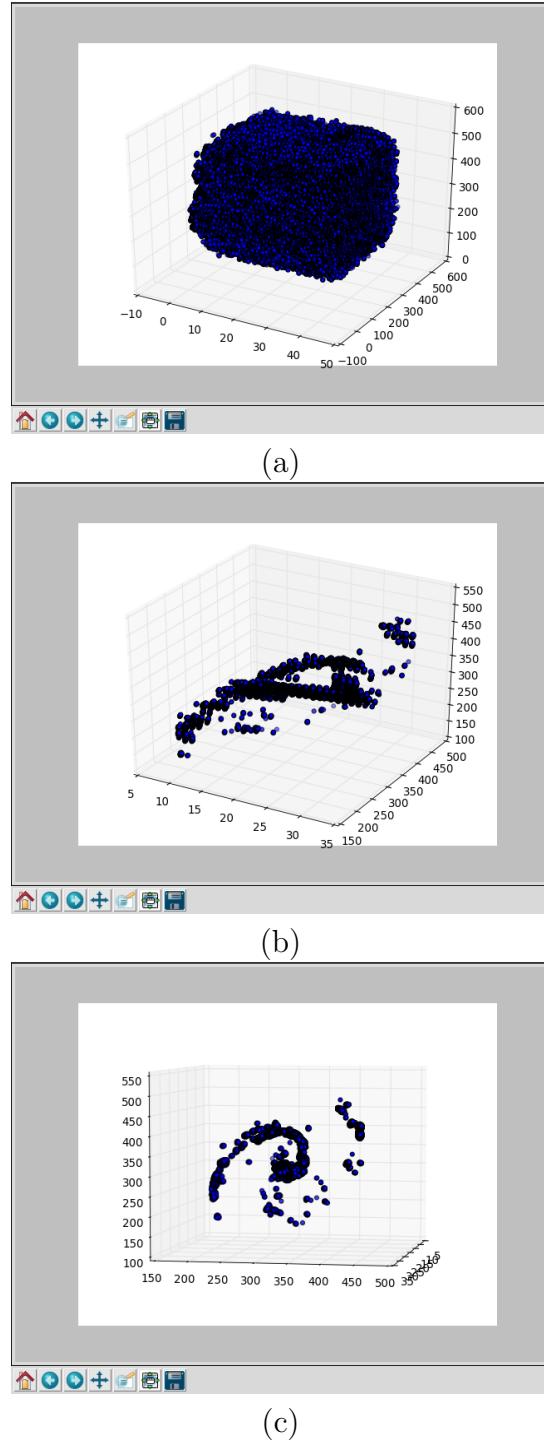


Figura 3.3: Resultado visual para la biblioteca *Matplotlib*.

-
- Opciones de *GUI*: Existen ciertas opciones de *GUI* en la documentación, pero debido al bajo rendimiento gráfico observado se decidió no continuar con la alternativa.

En resumen, puede concluirse que *Matplotlib* no está realmente optimizado para visualización volumétrica, debido a que no utiliza un motor gráfico con énfasis en el rendimiento como *OpenGL*, a diferencia de otras opciones analizadas.

Matplotlib está enfocado en la simpleza de los gráficos y el código para generarlos, siendo una herramienta potente para visualizar información tal como estadísticas y tablas, pero no para visualización volumétrica.

3.2. PyQtGraph

PyQtGraph es una biblioteca de gráficas e interfaces para *Python*, con enfoque en los campos de ciencia e ingeniería [14]. Utiliza para ello el *framework* de visualización gráfica de *Qt*, debido a su alto rendimiento. Para el procesamiento numérico utiliza la biblioteca *NumPy*.

Entre las funcionalidades principales se puede mencionar:

- Visualización básica de imágenes, líneas y gráficos de dispersión.
- Alto rendimiento gráfico, el cual permite poder cambiar la perspectiva o el *zoom* sin tiempos de espera excesivos.
- Exportación de gráficas como archivos de imagen PNG o SVG.
- *Widgets* para seleccionar regiones de interés en un gráfico.
- *Framework* para realizar los *widgets* mencionados anteriormente de manera personalizada.
- *Widgets* de interfaz gráfica basados en los que otorga *Qt*.

La versión actual de *PyQtGraph* es la 0.9.10 y funciona en *Windows*, *Linux* y *OSX*. Corre tanto en las versiones 2.7 como 3+ de *Python*. Los prerequisitos de sistema para el correcto funcionamiento de la biblioteca incluyen *NumPy*, *SciPy* (procesamiento numérico), *Qt* (interfaz) y *OpenGL* (rendimiento gráfico). También, para este caso en particular es necesaria la biblioteca *astropy* para el manejo de archivos *fits*. Los comandos para la instalación de esto pueden encontrarse en el apéndice A.2.

3.2.1. Prueba de Concepto

La implementación del código se basa en la lectura de datos desde el archivo *fits* para luego crear un gráfico de dispersión (*scatter plot*) para la representación de los puntos más importantes. El detalle de cada paso seguido se encuentra a continuación:

1. Se cargó el archivo *fits* a utilizarse.
2. Se extrajeron los datos del *header* y *data* del mismo. Los primeros son usados para inicializar variables que guarden las dimensiones de los datos, mientras que los segundos son los datos en sí, dentro de una matriz de *NumPy*.
3. Se calculó un *threshold* de los datos.
4. Se inicializaron los elementos propios de la *GUI*.
5. Se almacenaron las coordenadas de todos los puntos dentro de la *data* que cumplieran con la condición de tener un valor mayor al *threshold* multiplicado por un factor llamado *RMS*.
6. A cada uno de estos puntos se le asignó una coloración dependiendo de su frecuencia, siendo los de menor frecuencia más cercanos al rojo y los de mayor frecuencia más cercanos al violeta. A esto se le suma además un valor de transparencia basado en su frecuencia (mientras menor sea su frecuencia tendrá mayor valor de transparencia).
7. El resultado final se muestra dentro de la zona del *display*, quedando a la espera del control del usuario.
8. En caso de que se alteren los valores de *RMS* y transparencia, se repiten los pasos 5, 6 y 7.

La interfaz de usuario es otorgada por la biblioteca *Qt*, enfocada principalmente en darle el control al usuario para discriminar datos de menor importancia. Se basa en 2 elementos:

- *RMS control*: Un *slider* en la zona superior de la ventana permite al usuario aumentar o disminuir el valor del *RMS*, en un intervalo que va desde 1 hasta un máximo determinado en base al valor máximo del

arreglo, de manera de mostrar solo los mayores valores posibles en el último nivel. El incremento entre cada nivel es de 1, siendo un factor entero. Se incluye además en la zona inferior izquierda una ventana que sirve tanto como *display* del valor actual como también para modificarlo manualmente por teclado. Cada vez que este valor es cambiado, producirá un recálculo de los puntos a mostrar por pantalla, para cumplir que se muestren solo los que cumplan con la condición de ser mayores a $RMS * threshold$.

- *Alpha control*: De manera similar al elemento anterior, consiste en un *slider* en la zona izquierda de la pantalla, la cual permite manejar el nivel de transparencia entre los valores 0 y 1, siendo 1 el valor por defecto (todo opaco, sin transparencia) y teniendo un incremento mínimo de 0.01. Posee también una zona de *display* que es posible modificar manualmente por teclado. El nivel de transparencia no es aplicado igualmente a todos los puntos, sino que afecta más a los de menor frecuencia, de manera tal que alterar este parámetro permite hacer desaparecer paulatinamente los valores de menor frecuencia, enfocándose solo en los más significativos.
- **Controles**: Los controles de la visualización son los más básicos, otorgados por defecto por *PyQtGraph*, que consisten en rotar la pantalla al arrastrarla con el botón izquierdo del *mouse*, acercar y alejar la vista con la rueda del *mouse* y trasladar la pantalla al arrastrarla con el botón izquierdo del *mouse* manteniendo la tecla *shift* apretada.

En cuanto al resultado visual, en la figura 3.4(a) se puede apreciar la visualización con el mínimo *threshold* y con uno mayor en la figura 3.4(b). En la figura 3.4(c) se puede ver el detalle del centro del espiral de la galaxia. Luego, al incrementar aún más el *threshold* se puede apreciar que el centro es la zona de mayor concentración de frecuencias en la figura figura 3.4(d).

Por otro lado, en la figura 3.4(e) se puede apreciar la visualización volumétrica con el mínimo *threshold* y con transparencia aumentada, la cual oculta parcialmente valores inferiores sin hacerlos desaparecer por completo. Luego, en la figura 3.4(f) la transparencia está al máximo, la cual oculta casi todos los valores menos los más significativos. Finalmente en la figura 3.4(g) se ve el detalle del centro de la galaxia con las mismas características anteriores. Todas las figuras son de elaboración propia.

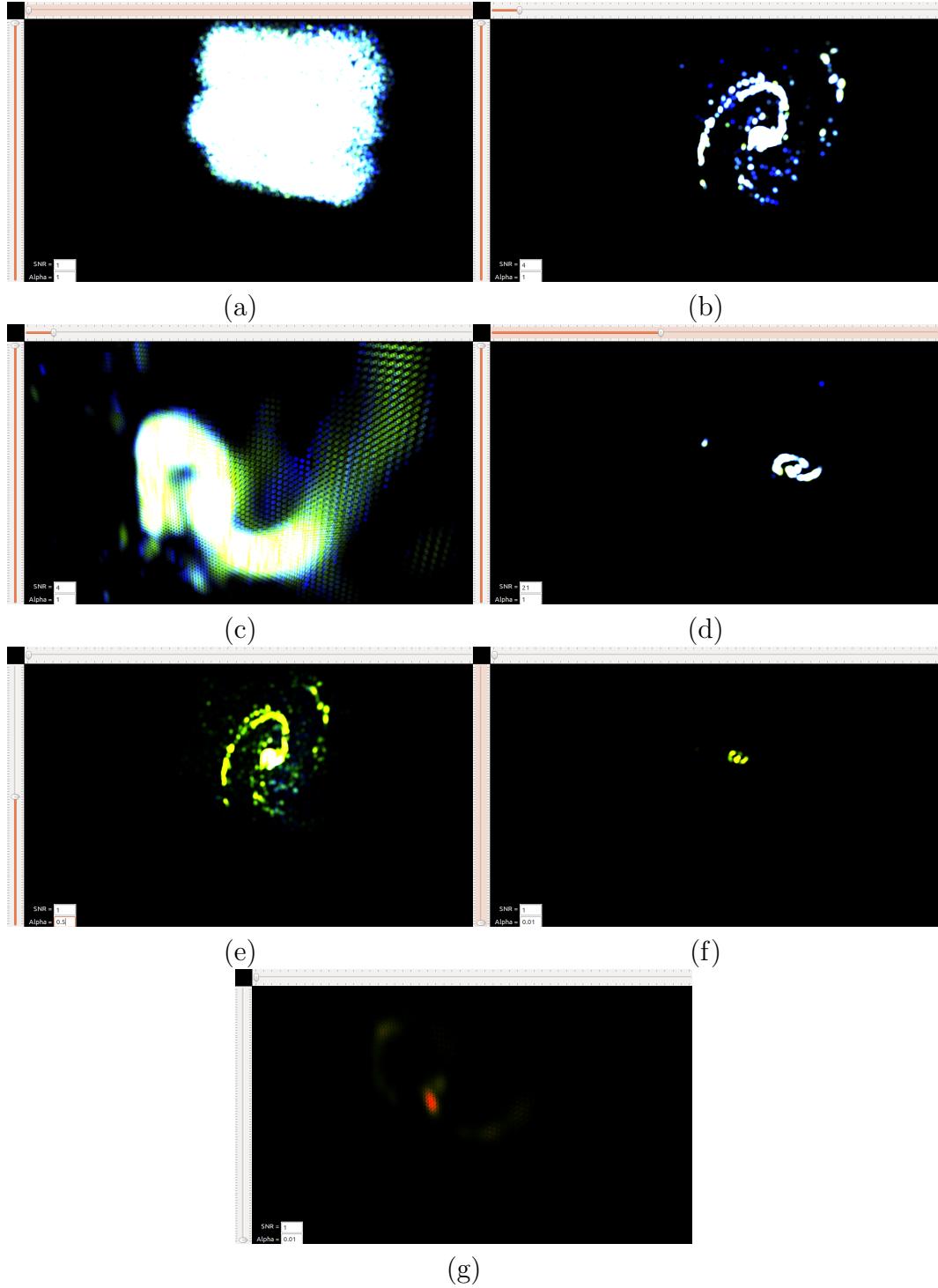


Figura 3.4: Resultado visual para la biblioteca *PyQtGraph*.

3.2.2. Conclusiones

En cuanto a los requerimientos que se busca y el desempeño mostrado por *PyQtGraph* puede mencionarse:

- Rendimiento: Posee un rendimiento gráfico alto, siendo posible manipular el volumen rápidamente incluso con una gran cantidad de puntos. Esto se debe a que la biblioteca utiliza *OpenGL* para la parte gráfica, lo cual asegura un desempeño adecuado. El tiempo medido de espera es casi nulo.
- Velocidad de renderizado: La velocidad del primer volumen también fue bastante rápida, con tiempos de espera no superiores a los 5 segundos, aún en los casos con mayor cantidad de puntos.
- Opciones de personalización: Aquí la biblioteca presentó la mayor cantidad de carencias con una baja cantidad de opciones para personalizar la visualización. Fue posible cambiar el color y transparencia de los puntos, así como su tamaño, pero no existían más opciones que permitieran por ejemplo cambiar los controles o crear ejes para un mejor entendimiento del gráfico, provocando que el desarrollo quedara estancado al no poder abarcar todos los puntos requeridos.
- Opciones de *GUI*: Utilizando elementos propios de *Qt* es posible tener una alta cantidad de elementos de *GUI* altamente personalizables, siendo uno de los puntos más fuertes de la biblioteca.

En resumen, *PyQtGraph* es una buena biblioteca para la representación volumétrica. Posee un alto rendimiento y elementos de *GUI* que permiten crear distintos casos de uso para el usuario.

Por otro lado, la documentación existente para la biblioteca es muy escasa, y la comunidad de usuarios no es tan grande como las de las otras alternativas, dificultando el proceso de desarrollo al presentarse problemas y limitando también el trabajo a futuro.

El resultado visual del *scatter plot* no cumplió con los requerimientos iniciales al no ser realmente un volumen sólido sino que un conjunto de puntos. Para conseguir un volumen sólido, se debe definir estructuras vectoriales, típicamente mediante vértices de los polígonos que componen al sólido. Debido a la escasa documentación y a la complejidad de implementar estos cálculos de forma manual en *PyQtGraph*, se decidió explorar otro tipo de representación en la siguiente alternativa.

3.3. Mayavi

Mayavi es una biblioteca que busca una visualización de datos 3D de manera sencilla e interactiva [15], utilizando para ello una interfaz de usuario intuitiva, una interfaz para desarrolladores simple de usar (utilizando *scripting* de *python*) y utilizando soporte de la biblioteca *VTK*. *VTK*, siglas de *Visualization Toolkit* es un *software* para la generación de imágenes 3D de código abierto, que incluye técnicas avanzadas de modelamiento, como reducción de polígonos, suavizado de superficies, contorneo, entre otros [16]. Al utilizar *Mayavi* no es necesario aprender a usar *vtk*, ya que lo hace automáticamente. *VTK* a su vez utiliza *OpenGL* para todas las representaciones, asegurando contar con la eficiencia gráfica característica de *OpenGL*.

Los detalles técnicos de *Mayavi* son:

- Uso de las matrices otorgadas por *NumPy*.
- Un claro diseño *MVC*, a diferencia de su predecesor.
- Componentes y módulos que facilitan el realizar extensiones.
- Una *GUI* basada en *Qt4* o bien en *wxPython*. En este caso se usará la primera, al compartir elementos en común con la anterior biblioteca utilizada.
- Un diseño no intrusivo y reusable.

Mayavi utiliza una arquitectura del tipo *pipeline*, creándose una jerarquía por la cual pasan los datos. En primera instancia los datos son cargados en *Mayavi*, ya sea desde un archivo o a través de un *script*. Estos datos son pasados a través de filtros opcionales, los cuales pueden ser parte de la interfaz o bien del *script*. Finalmente, son cargados dentro de la escena, la cual crea la visualización a partir de los datos. La razón para crear esta arquitectura separada es poder visualizar los mismos datos de diferentes maneras, por ejemplo, en la figura 3.5 se aprecia 3 representaciones distintas a partir del mismo conjunto de datos.

La biblioteca *Mayavi* corre en *Windows*, *Linux* y *OSX*. Las dependencias previas para su funcionamiento son *NumPy*, *SciPy* (procesamiento numérico), *Qt* (interfaz), *OpenGL* (rendimiento gráfico) y *VTK* (herramientas de visualización). También, para este caso en particular es necesaria la biblioteca *astropy* para el manejo de archivos *fits*. Los comandos para la instalación de esto pueden encontrarse en el apéndice A.3.

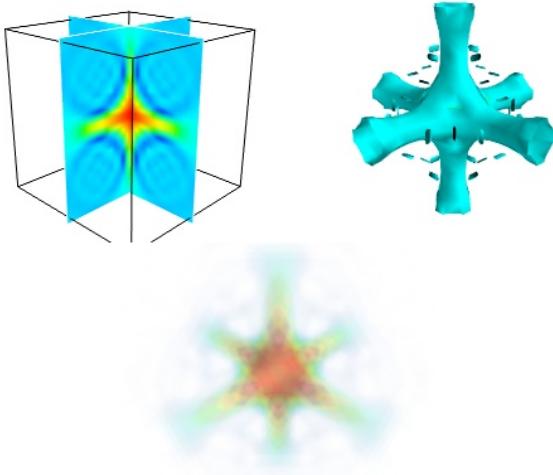


Figura 3.5: Modelos del mismo conjunto de datos al aplicarle módulos distintos. Fuente: <https://goo.gl/2kfqsW>

3.3.1. Prueba de Concepto

Mayavi ofrece múltiples opciones de visualización volumétrica, entre ellas se encuentra la visualización volumétrica simple (para analizar las formas generadas por los datos) y otra visualización más compleja compuesta por contornos (para analizar los distintos niveles entre los datos). La visualización volumétrica simple es más rápida de generar que la de contorno.

En primera instancia se creó la vista volumétrica, para la cual es necesario utilizar el *pipeline scalar_field* de *Mayavi*. Para generar este se debe primero crear un *mesh grid*¹ con las dimensiones del cubo, utilizando la función *mgrid* de *NumPy*. Luego se genera un nuevo cubo de datos idéntico al original, el cual posee al igual que antes un *threshold* para seleccionar datos a analizar. Finalmente el *mesh* creado y el cubo actualizado son pasados como parámetros al *pipeline*, generando la vista deseada.

Luego se creó la vista con contornos. En este caso la función *contour3d* debe ser utilizada, pasando por parámetros el *mesh* creado, el cubo actualizado, la cantidad de contornos que se desea y la opacidad de los mismos, lo

¹Un *mesh grid* consiste en una malla que define la forma de un objeto tridimensional sólido. Estas mallas suelen estar conformadas de triángulos pero pueden ser creadas también con otras figuras. Si se utiliza una figura pequeña el *mesh* será más detallado y más pesado de crear.

cual determinará qué tan transparentes serán.

En este caso no se creó ninguna opción personalizada de *GUI*, ya que *Mayavi* otorga una barra de herramientas por defecto, la cual puede apreciarse en la figura 3.6.

Esta contiene los siguientes elementos (de izquierda a derecha):

- *View the Mayavi pipeline*: Otorga acceso a las opciones de *pipeline* de *Mayavi*, las cuales pueden ser modificadas por el usuario. Se puede cambiar por ejemplo el color del fondo, el color del volumen generado, la opacidad del mismo, las dimensiones del gráfico, etc.
- *View along some axis*: Los 6 botones siguientes sirven para posicionar la cámara en un determinado eje orientada hacia la dirección positiva o negativa de cada eje.
- *Obtain an isometric view*: Posiciona la cámara en la diagonal generada por los 3 ejes, orientada hacia la dirección negativa de ellos.
- *Toggle parallel projection*: Elimina la perspectiva a lo largo del eje de profundidad, generando una vista plana de los datos. Puede activarse y desactivarse.
- *Toggle axes indicator*: Muestra en la esquina inferior izquierda un indicador de hacia dónde están orientados los 3 ejes, la cual rota junto con el objeto. Puede activarse y desactivarse.
- *Full screen*: Lleva la ventana de la visualización a modo de pantalla completa.
- *Save a snapshot of this scene*: Permite guardar como archivo de imagen la vista generada actualmente.
- *Configure the scene*: Permite configurar distintos aspectos de la visualización, como la calidad de la imagen, de la iluminación, etc.

En cuanto al resultado visual, se puede apreciar la visualización con el mínimo *threshold* a lo largo del eje Z en la figura 3.7(a) y a lo largo del eje X en la figura 3.7(b). En la figura 3.7(c) se puede ver el detalle del volumen luego de hacerle *zoom*, y de igual manera en la figura 3.7(d) pero ahora con un volumen de contornos, en el cual las zonas de colores amarillos y rojos son las de mayor intensidad. Finalmente en la figura 3.7(e) puede apreciarse



Figura 3.6: Barra de herramientas otorgada por *Mayavi*. Fuente: Elaboración propia.

la visualización volumétrica con un mayor *threshold*. Todas las figuras son de elaboración propia.

3.3.2. Conclusiones

En cuanto a los requerimientos que se busca y el desempeño mostrado por *Mayavi* puede mencionarse:

- Rendimiento: El rendimiento de la biblioteca es alto, ya que igual que la biblioteca anterior utiliza *OpenGL* para la parte gráfica, con tiempos de espera casi nulos para las operaciones de rotación y *zoom*.
- Velocidad de renderizado: La velocidad de renderizado es rápida. Eso si fue posible observar tiempos de espera aumentados en el caso de la visualización de contornos, lo cual se debe a la mayor complejidad del gráfico en comparación a los hechos anteriormente. Aún así, los tiempos de espera no excedieron los 5 segundos.
- Opciones de personalización: A través del *pipeline* de *Mayavi* es posible cambiar en tiempo de ejecución varios aspectos tales como el nombre de los ejes, los colores, las dimensiones, entre otros. Estos cambios también pueden hacerse en el código, siendo esta biblioteca la que tiene la cantidad más alta de opciones de personalización de las vistas hasta ahora.
- Opciones de *GUI*: *Mayavi* propone 2 alternativas de *GUI*: elementos integrados dentro del gráfico y la posibilidad de incluir la visualización dentro de una aplicación de *Qt*, junto con otros elementos. Esta segunda opción se decidió que sería considerada para el capítulo 4 de Implementación para Astronomía.

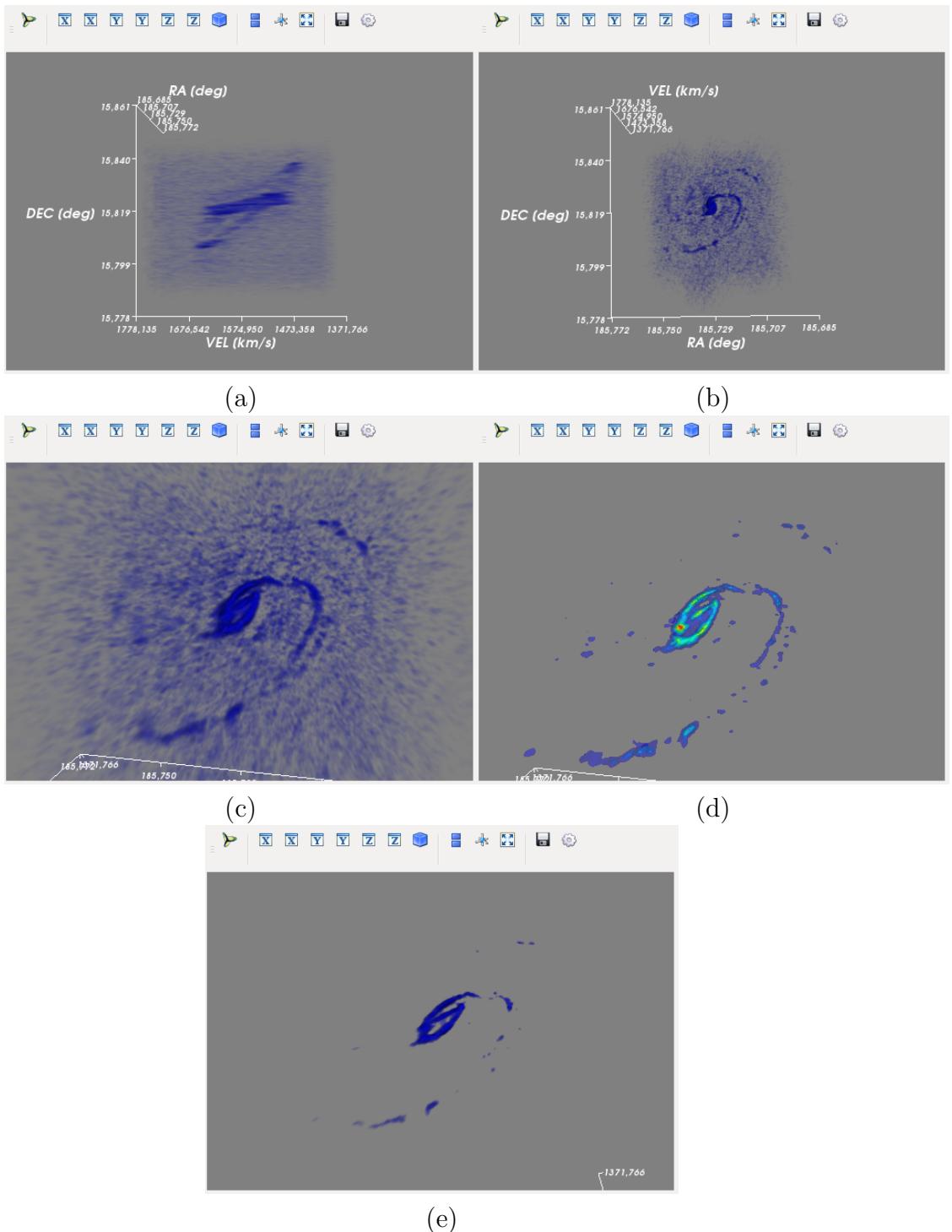


Figura 3.7: Resultado visual para la biblioteca *Mayavi*.

En resumen, *Mayavi* es una biblioteca con un gran potencial, una gran documentación y una comunidad considerablemente amplia. Su rendimiento es alto y los resultados de las visualizaciones fueron más cercanos a lo requerido.

Capítulo 4

Implementación para Astronomía

Para llevar a cabo la comparación final se utilizó el cuadro 4.1, con una comparación cualitativa entre distintas características y cómo califica cada una de las bibliotecas analizadas.

Finalmente, se decidió que *Mayavi* cuenta con las características apropiadas para la visualización volumétrica que se busca. El siguiente paso es desarrollar una aplicación con usos en la astronomía, con *feedback* de astrónomos profesionales para su desarrollo. También la biblioteca *Matplotlib* será utilizada pero en este caso para crear gráficos bidimensionales.

	<i>Matplotlib</i>	<i>PyQtGraph</i>	<i>Mayavi</i>
Rendimiento	Bajo	Alto	Alto
Velocidad de renderizado	Lenta	Rápida	Rápida
Opciones de personalización	Altas	Bajas	Altas
Opciones de <i>GUI</i>	Medias	Altas	Altas
Documentación	Alta	Baja	Medianamente alta
Comunidad	Numerosa	Baja	Moderada
Visualización 2D	Buena	Media	Media
Visualización 3D	Limitada	Media	Alta

Cuadro 4.1: Comparación entre bibliotecas. Fuente: Elaboración propia.

4.1. Aplicación Generada

La aplicación generada en esta instancia de desarrollo no es únicamente una visualización volumétrica de los datos, sino que tiene distintas secciones para entregar información al usuario, entre las que se encuentran:

- Visualización de los metadatos incluidos en el *header* del archivo, con información tal como las dimensiones del cubo, el historial de cambios al archivo, entre otros.
- La visualización volumétrica generada utilizando *Mayavi*, la cual tiene 2 opciones: volumétrica simple para visualizar la forma general de los datos, y de contorno, la cual genera distintas visualizaciones dentro de una, estando los datos de similares dimensiones dentro de la misma superficie, permitiendo así visualizar los distintos niveles. Esta última es más pesada y lenta de generar que la primera.
- Una vista *stacked* del cubo, en la cual la dimensión de *velocity* es eliminada para generar un gráfico bidimensional con los datos de ese eje acumulados. Este gráfico es creado utilizando *Matplotlib*.
- Un espectro de frecuencias acumuladas a lo largo del eje de *velocity*, indicando a qué profundidad del cubo se presenta la mayor acumulación de datos. Este gráfico es creado utilizando *Matplotlib*.

Para generar esto, en primera instancia el archivo *fits* es cargado en la aplicación, generando una variable constante dentro de la aplicación, la cual será copiada para crear nuevas instancias de los datos sin perder lo original.

Luego se genera la visualización volumétrica, la que tendrá una posición centralizada en la aplicación. El proceso para hacer ésta es el mismo descrito en la sección 3.3.1.

Posteriormente, se genera el gráfico de la vista *stacked* del cubo, para lo cual se genera un nuevo cubo idéntico al original pero limitado en sus dimensiones según las preferencias del usuario. A este cubo entonces se le elimina la dimensión de *velocity*, acumulando los valores y creando una matriz de 2 dimensiones (*right ascension* y *declination*). La función *imshow* de *Matplotlib* es utilizada para generar una imagen que representa la matriz generada anteriormente, con los valores de los ejes ajustados a los valores reales de las dimensiones mostradas.

Luego, se genera el espectro, el cual corresponde al mismo cubo original reducido en base a las preferencias del usuario. A este cubo se le eliminan las dimensiones de *right ascension* y *declination*, dejando solamente la de *velocity*, a fin de generar un gráfico que muestre las frecuencias acumuladas a lo largo de este eje. Al igual que antes, el eje correspondiente tiene las unidades reales de la dimensión correspondiente.

La interfaz de usuario consiste en una aplicación *Qt*, la cual contiene incrustada los siguientes elementos:

- Metadatos: Representados por una caja de texto amplia, la cual muestra los datos del archivo incluidos en el *header*.
- Visualización volumétrica: Otorgada por *Mayavi*, ubicada en el centro de la aplicación. Bajo esta visualización se encuentran los botones que permiten cambiar entre volumen simple y contorno, y también las cajas de texto de *threshold* (tanto mínimo como máximo). Esta visualización posee además un *wireframe*, consistente de un cubo que encierra el volumen, y que se actualiza cuando los *sliders* de rangos de los ejes son alterados por el usuario, a fin de encerrar la zona de interés seleccionada.
- Cajas de texto de *threshold*: Dos cajas de texto que permiten tanto visualizar el valor actual de cada *threshold* como también cambiarlo directamente, a fin de discriminar datos por debajo y por sobre valores independientes.
- Caja de texto de contornos: Permite alterar la cantidad de contornos generada por el gráfico volumétrico de contornos, siendo 10 el número por defecto. A mayor número se podrá apreciar mayor cantidad de detalles pero el gráfico será más lento de generar.
- Visualización *stacked*: Otorgada por *Matplotlib*. Los ejes de esta imagen corresponden a las unidades de los ejes de *right ascension* y *declination*. El tamaño de esta imagen se ajusta automáticamente al realizar cambios en los *sliders* de rangos de los ejes, según las preferencias del usuario.
- Espectro: Correspondiente a un gráfico creado con *Matplotlib*. Al igual que el gráfico anterior, este se actualiza al modificar los *sliders* de rangos de los ejes.

-
- *Sliders* de rangos de los ejes: Existen 3 *sliders* de rangos, uno para cada eje del cubo, los cuales permiten seleccionar un área de interés dentro del volumen, con cuyos datos se generarán los 2 gráficos mencionados anteriormente. Cada vez que uno de los *sliders* o bien la caja de texto de alguno de ellos son modificados se genera una actualización tanto en los 2 gráficos anteriores como en el *wireframe* creado en la vista volumétrica, a fin de encerrar la zona de interés que se está mostrando actualmente.

Todos los *sliders* de doble índice creados para la aplicación fueron otorgados por la biblioteca *QRangeSlider* [17].

En cuanto al resultado visual, fue utilizado para las *screenshots* el archivo *Orion.methanol.cbc.consub.image.fits*, correspondiente a la nebulosa *Kleinmann-Low*. En la figura 4.1(a) se aprecia la visualización inicial, en la que todos los *sliders* se encuentran en su máxima posición, por lo que se muestra la mayor cantidad posible de datos. De similar manera, en la figura 4.1(b) se puede ver la visualización inicial pero ahora con un volumen de contornos. Al aumentar el *threshold* mínimo en la figura 4.2(a) se reduce la cantidad de datos que se muestra en el volumen, pero no se modifican los otros 2 gráficos. Finalmente en la figura 4.2(b) se han modificado los *sliders* de los ejes, alterando los gráficos bidimensionales y el *wireframe* alrededor del volumen, indicando la zona de interés que se está analizando. Todas las figuras son de elaboración propia.

4.2. Validación con Astrónomo

La validación con un astrónomo fue llevada a cabo por la doctora Amelia Bayo, astrónoma que trabaja en la Universidad de Valparaíso (información de contacto¹), quien tuvo la oportunidad de utilizar la aplicación para luego hacer sus comentarios a modo de *feedback*. Estos comentarios fueron críticas hacia la aplicación, indicando lo que estaba bien hecho y lo que le faltaba. Entre los elementos faltantes se encuentran:

- Redimensionar los cubos de gran tamaño para evitar que la aplicación se demore demasiado en mostrarlos por primera vez, y también la posi-

¹Dra. Amelia Bayo, PhD. en Física. Teléfono: (+56 32) 250 5555. Correo electrónico: amelia.bayo@uv.cl. Página web: <https://goo.gl/zh8Dnm>

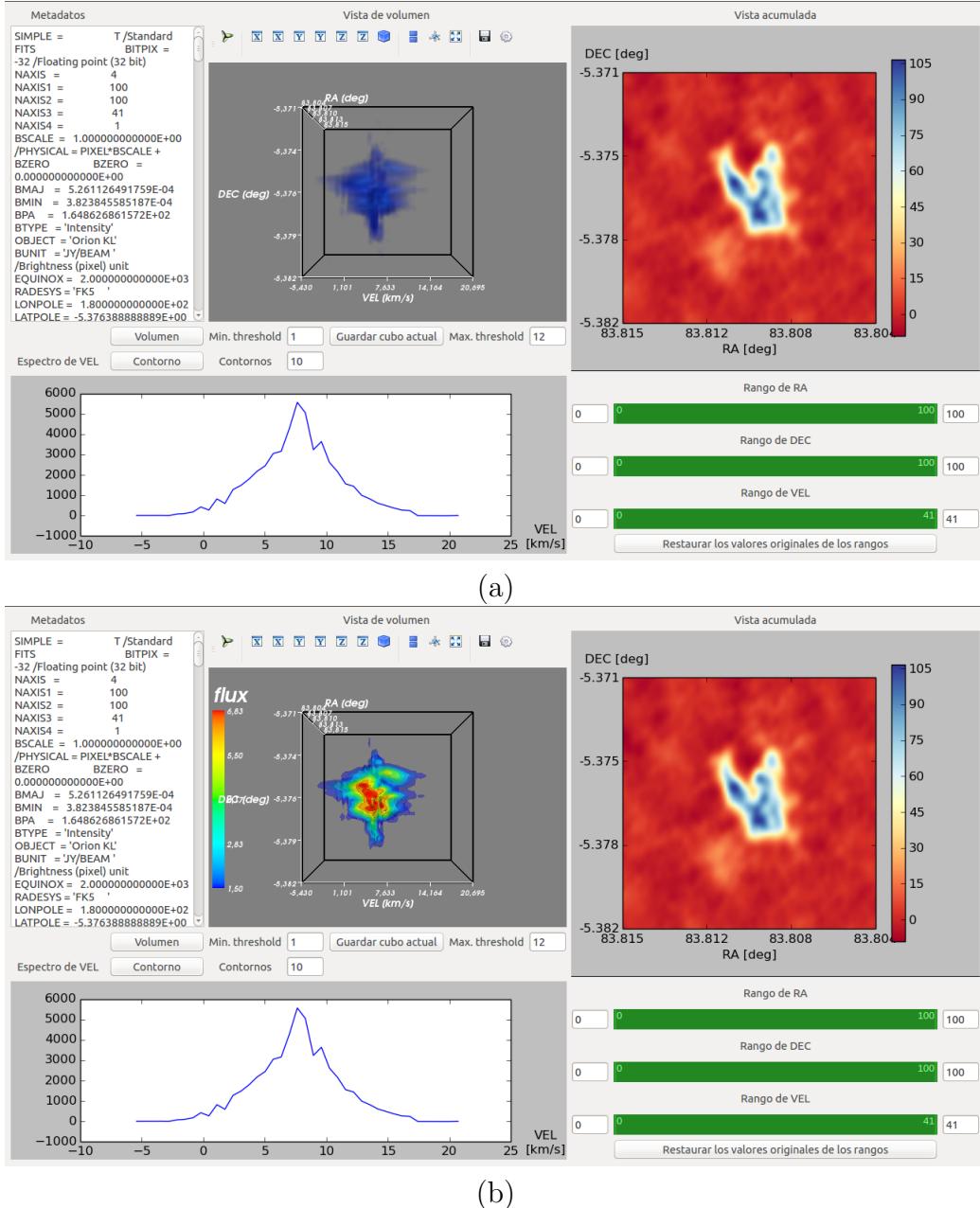


Figura 4.1: Aplicación generada (primera parte)

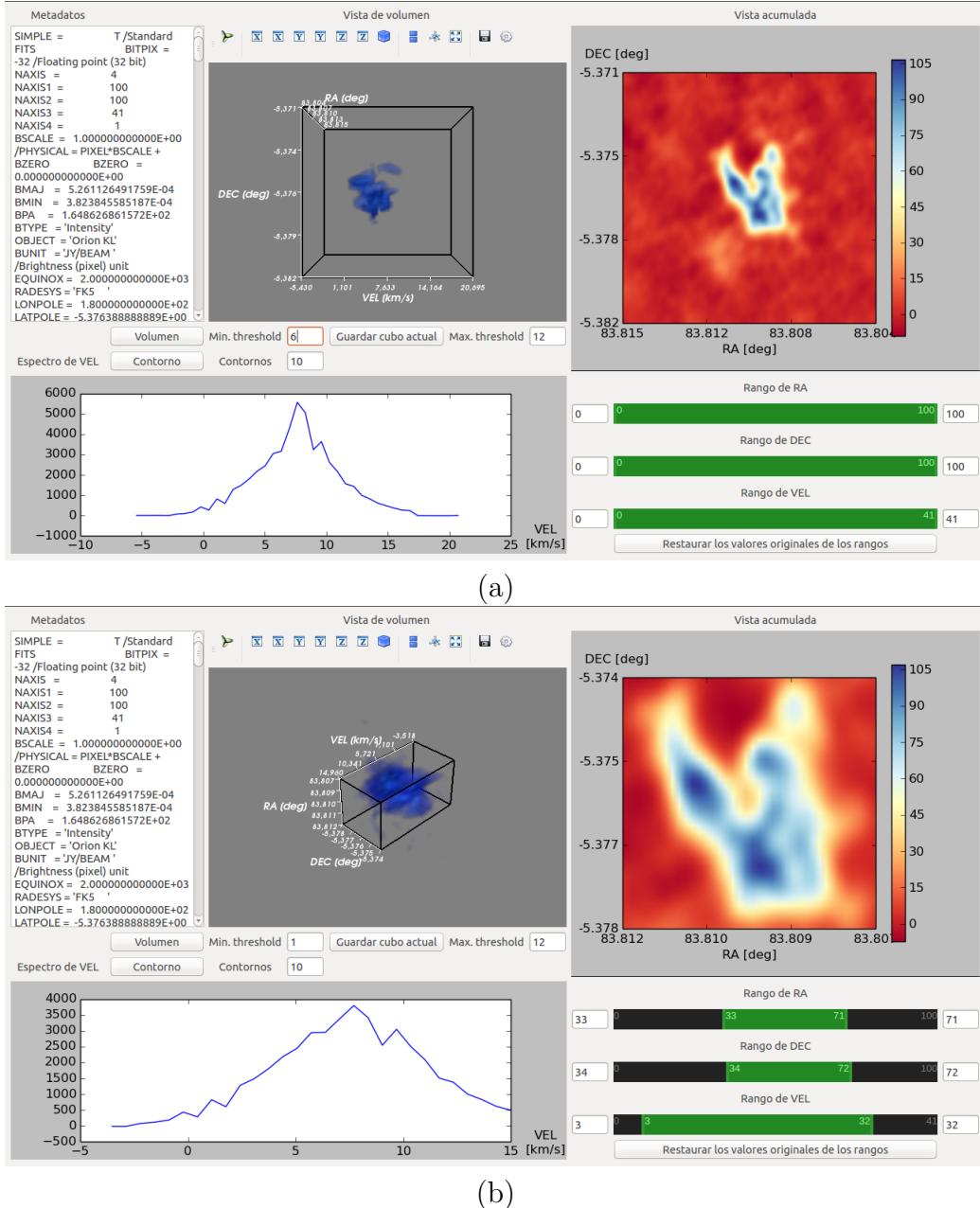


Figura 4.2: Aplicación generada (segunda parte)

bilidad de volver a las dimensiones originales al momento de enfocarse en zonas de interés para evitar perder detalles.

- El poder controlar la transparencia de ciertas partes del volumen para así poder enfocarse en partes específicas del mismo sin perder la visión general de su totalidad.
- Controlar el nivel de transparencia de aquellas secciones del volumen que se encuentran fuera del *wireframe*, a fin de que sea sencillo ver qué secciones están siendo mostradas en los gráficos bidimensionales.
- La posibilidad de cambiar el eje de la vista *stacked*, es decir, poder acumular los datos del cubo en sus 3 ejes en vez de solo uno.
- Nombres faltantes de los ejes en los gráficos mostrados, lo cual fue arreglado en la versión final de la aplicación.

Estos puntos serán considerados como trabajo a futuro para la aplicación, y de similar manera también hizo comentarios a modo de ideas que serían útiles para su trabajo, entre las que puede mencionarse:

- La opción de buscar algún dato específico en la vista de metadatos.
- El poder guardar como archivo *fits* un cubo actualizado equivalente al cubo actual pero limitado por el *wireframe*, reduciendo así sus dimensiones según las preferencias del usuario. Esta opción fue analizada determinando que es posible y fue implementada parcialmente en la versión final de la aplicación.
- El poder guardar como una imagen formato *.png* la vista actual del volumen.
- Poder cambiar la escala en la que se muestran los gráficos.
- Ser capaz de calcular distancias en el gráfico de *stacked* al seleccionar 2 puntos dentro del mismo, lo cual requeriría conocimiento acerca de las dimensiones originales del cubo y un gráfico interactivo.

Además, hizo mención de la biblioteca *Glue* para *Python*, la cual permite la visualización de datos de manera similar a la aplicación creada [18]. *Glue*

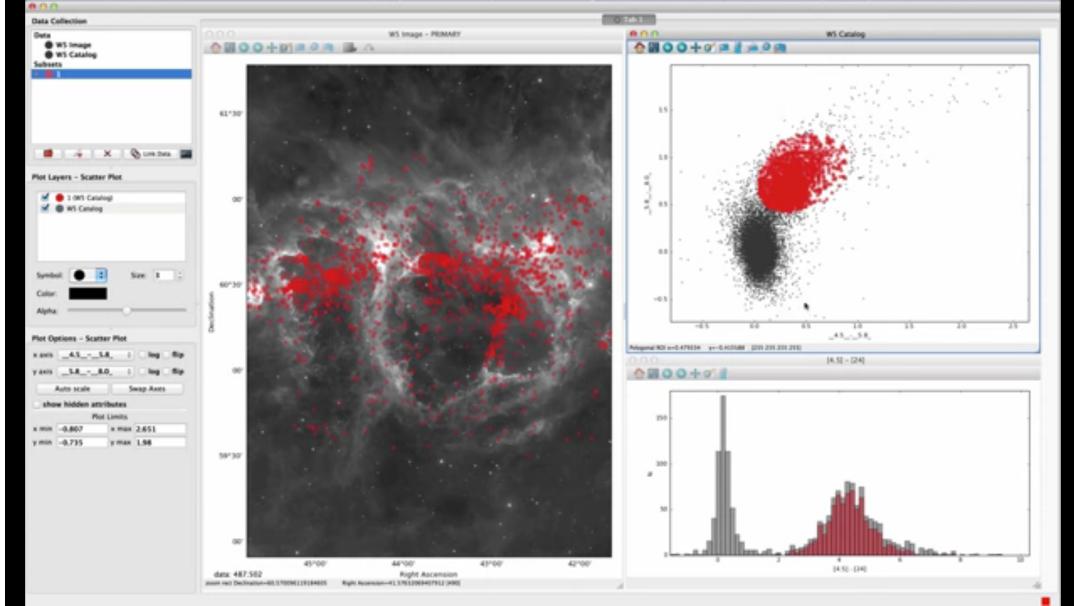


Figura 4.3: Captura de pantalla de la aplicación *Glue*. Las áreas marcadas en rojo de los 3 gráficos corresponden a los mismos datos pero representados de distinta manera. Fuente: <https://goo.gl/rjQHdF>

permite la visualizaciones de distintos conjuntos de datos (no solo astronómicos) con gráficos relacionados entre sí, es decir, las secciones de los distintos gráficos están relacionadas de manera tal que al seleccionar una zona de interés en uno de ellos se hará una selección a la representación equivalente de esos datos en los demás gráficos.

Los objetivos de *Glue* son otorgar al usuario la posibilidad de crear gráficos de dispersión, histogramas e imágenes (de 2 o 3 dimensiones) con sus datos. Esto pueden hacerlo utilizando la *GUI* por defecto que otorga la biblioteca o bien pueden añadir su propio código personalizado, ya que *Glue* es de código abierto y utiliza herramientas típicas de *Python*, como *NumPy*, *Matplotlib* y *Scipy*. Los gráficos generados son responsivos y reconocen como *input* selecciones de área utilizando el *mouse* sobre ellos.

Varios aspectos de esta aplicación, como los gráficos responsivos y la posibilidad de personalizar la aplicación deben ser considerados a modo de trabajo futuro para la aplicación de esta memoria.

Capítulo 5

Conclusiones

Al analizar los resultados obtenidos es posible concluir que existen múltiples opciones para la visualización, siendo *Mayavi* una fuerte alternativa para el trabajo de visualización volumétrica, y *Matplotlib* una buena alternativa para la visualización bidimensional.

Mayavi permite la creación y personalización de volúmenes con un alto rendimiento gráfico. Posee opciones que permiten la personalización de los gráficos por parte del desarrollador en el código y también por parte del usuario en la aplicación.

Matplotlib posee gran cantidad de herramientas para la creación de imágenes y gráficos en 2 dimensiones, tales como histogramas, espectros, gráficos de dispersión, mapas de calor, etc. Estos gráficos son de alto rendimiento y altamente personalizables por parte del desarrollador.

La diferencia de rendimiento observada entre las distintas bibliotecas deja en claro que es necesaria la utilización de *OpenGL* como motor gráfico para asegurar velocidades de interacción aceptables.

La aplicación generada tiene usos para la astronomía con cubos de datos de tamaño moderado, pero es necesario ampliar el trabajo para cubos de mayor tamaño.

Es necesario el uso de código libre para la aplicación y el crear una documentación para que la comunidad pueda continuar con el desarrollo. Desde el punto de vista funcional, es necesario el trabajo en gráficos responsivos y dinámicos para ser manejados de manera intuitiva por el usuario.

En cuanto al problema específico enunciado, se concluyó que es necesario el continuo trabajo tanto de *hardware* como de *software* para poder generar visualizaciones volumétricas de alto nivel para conjuntos grandes de datos,

con énfasis en la claridad de los resultados para poder ser interpretados por los expertos.

5.1. Trabajo Futuro

En la sección 4.2 se mencionó desde el punto de vista de la aplicación generada los puntos pendientes y trabajo futuro en base a los requerimientos mencionados. Aparte de eso, el trabajo a futuro de la memoria puede ampliarse de distintas maneras.

Por un lado, dentro del campo de la astronomía el uso de bibliotecas de visualización puede ser continuado a partir del código creado, ampliando las funcionalidades existentes o bien utilizando las funciones de visualización creadas en una nueva aplicación. Debido a la modularidad del código, es fácil separar las secciones de visualización y de creación de *GUI*, las cuales eran más específicas para la aplicación. El trabajo en esta área puede también ampliarse de manera *online*, con la posibilidad de utilizar esta aplicación en una página *web* para evitar el tener que instalarla en cada plataforma en la que se vaya a usar, y además poder mantenerla actualizada fácilmente.

Por otro lado, las funciones de visualización creadas y las capacidades propias de las bibliotecas permiten la creación de cualquier volumen siempre que esté definido como una matriz tridimensional. De esta manera, el trabajo e investigación realizados en la memoria pueden apoyar el trabajo en visualización volumétrica en otras áreas. Desde aquí puede ampliarse el trabajo de manera específica según las necesidades, teniendo como base la creación de visualizaciones volumétricas de alto rendimiento.

Bibliografía

- [1] Ciro Donalek, S. G. Djorgovski, Alex Cioc, Anwell Wang, Jerry Zhang, Elizabeth Lawler, Stacy Yeh, Ashish Mahabal, Matthew Graham y Andrew Drake, "Immersive and Collaborative Data Visualization using Virtual Reality Platform", 27 de Octubre, 2014.
<https://arxiv.org/ftp/arxiv/papers/1410/1410.7670.pdf>
- [2] Ogre3D, open source 3D graphics engine.
www.ogre3d.org
- [3] NADC, North American Drilling Corporation.
www.northamericandrillingcorp.com
- [4] Observing with ALMA: A Primer for Early Science (Cycle 3), Doc 3.1, ver. 4, Marzo, 2015.
<https://almascience.nao.ac.jp/documents-and-tools/cycle3/alma-early-science-primer>
- [5] W. D. Pence, L. Chiappetti, C. G. Page, R. A. Shaw and E. Stobie, "Definition of the Flexible Image Transport System (FITS)", version 3.0, 22 de Noviembre, 2010.
<http://www.aanda.org/articles/aa/pdf/2010/16/aa15362-10.pdf>
- [6] Astropy: A community Python package for astronomy, The Astropy Collaboration, 30 de Septiembre, 2013
<http://www.aanda.org/articles/aa/pdf/2013/10/aa22068-13.pdf>
- [7] McCormick, B.H., DeFanti, T.A., Brown, M.D., "Visualization in Scientific Computing", Computer Graphics, vol 21, no 6, ACM SIGGRAPH, 1 de Noviembre, 1987
<https://www.evl.uic.edu/documents/visc-1987.pdf>

-
- [8] Jiménez-Vicente et al., 2007, "Discovery of a galactic wind in the central region of M100", 382, L16.
http://articles.adsabs.harvard.edu/cgi-bin/nph-iarticle_query?letter=L&classic=YES&bibcode=2007MNRAS.382L..16J&page=&type=SCREEN_VIEW&data_type=PDF_HIGH&send=GET&filetype=.pdf
 - [9] Gaia 3D, 3D Visualization & Learning Technologies For Education.
www.gaia3d.co.uk
 - [10] CARTA: The Cube Analysis and Rendering Tool for Astronomy, Erik Rosolowsky, Universidad de Alberta.
www.adass2014.org/presentations/05-2.pdf
 - [11] Science Verification Data, ALMA project.
<https://almascience.eso.org/alma-data/science-verification>
 - [12] NumPy library for multi-dimensional arrays and matrices.
www.numpy.org/
 - [13] Hunter, J. D., "Matplotlib: A 2D graphics environment", Computing In Science & Engineering, Volume 9, Number 3, Pages 90–95, año 2007.
 - [14] Documentation for PyQtGraph.
<http://www.pyqtgraph.org/documentation/>
 - [15] Ramachandran, P. and Varoquaux, G., "Mayavi: 3D Visualization of Scientific Data", IEEE Computing in Science & Engineering, 13 (2), pp. 40-51 (2011).
 - [16] W. Schroeder et al.. "The Visualization Toolkit", 3rd Edition. Kitware, Inc., 2003"
 - [17] QRangeSlider v0.1 documentation.
<http://rsgalloway.github.io/QRangeSlider/>
 - [18] Glue multidimensional data exploration.
<http://www.glueviz.org/en/stable/index.html>

Apéndice A

Instalación de bibliotecas

A.1. Matplotlib

```
$ sudo apt-get install python-pip  
  
$ sudo apt-get install python-numpy python-scipy  
python-matplotlib ipython ipython-notebook  
python-pandas  
  
$ sudo pip install --no-deps astropy
```

A.2. PyQtGraph

```
$ sudo apt-get install python-pip  
  
$ sudo apt-get install python-numpy python-scipy  
ipython ipython-notebook python-pandas  
  
$ sudo pip install --no-deps astropy  
  
$ sudo apt-get install python-opengl python-qt4-gl
```

Luego de estos requerimientos es necesario descargar el archivo de instalación desde la página <http://www.pyqtgraph.org/>.

A.3. Mayavi

```
$ sudo apt-get install python-pip  
  
$ sudo apt-get install python-numpy python-scipy  
python-matplotlib ipython iipython-notebook  
python-pandas  
  
$ sudo pip install --no-deps astropy  
  
$ sudo apt-get install python-vtk python-setuptools  
python-configobj  
  
$ sudo pip install mayavi envisage  
  
$ sudo apt-get install python-opengl python-qt4-gl  
  
$ sudo pip install qrangeslider
```