Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Departamento de Computación

Interacción con Objetos Deformables

Tesis que presenta

Miriam Mecate Zambrano

para obtener el Grado de

Maestra en Ciencias

en la Especialidad de

Computación

Director de la Tesis

Dr. Luis Gerardo de la Fraga

México, D.F. **Noviembre 2008**

Resumen

En este trabajo se probaron tres dispositivos manipuladores con seis grados de libertad (que entregan tres coordenadas espaciales y tres ángulos de orientación); estos dispositivos se usaron para deformar una malla de cuadriláteros en un espacio tridimensional. Se incorporó también visión estereoscópica activa para facilitar la visualización y manipulación de la malla deformable.

Los tres sensores que se probaron fueron un Phantom Omni de la compañía Sensable, el Patriot de la compañía Pholemus, y el Head Tracker de la compañía Logitech. La deformación de la malla esta basada en un sistema masa-resorte-amortiguador en cada una de sus aristas, que se resuelve utilizando el método de diferencias finitas.

Se construyó una interfaz gráfica con las bibliotecas de Qt y OpenGL, en el ambiente de GNU/Linux, para mostrar la integración de todos los dispositivos. A través de esta interfaz se despliega el estado del objeto deformable, se selecciona el dispositivo a interactuar y se definen las propiedades de la malla deformable y de la estereoscopía del escenario. Así mismo, través de la interfaz se puede digitalizar cualquier punto en el espacio que defina la estructura de un objeto; se puede guardar un conjunto de coordenadas (x, y, z) que se reciben a través de cualquiera de los dispositivos.

Finalmente, se realizó una comparación del desempeño y la adaptabilidad de cada uno de los dispositivos integrados en un ambiente virtual. Los resultados obtenidos de la comparación permiten observar las características de cada uno, sus ventajas y desventajas, y sugieren la mejor forma de integrarlos a una aplicación.

Abstract

In the present work, three manipulator devices were proved. Each one with six degrees of freedom (they give three spatial coordinates and three orientation angles); these devices were used to deform a rectangular mesh in a three-dimensional space. The stereoscopic vision was also incorporated in order to facilitate the visualization and handling of the deformable mesh.

The three sensors that were proved are: Phantom Omni from Sensable Tecnologies Inc., Patriot from Pholemus Inc. and Head Tracker from Logitech Inc.. Deformation of the mesh is based on a mass-spring-damper system in every mesh edges, and this system is solved using the finite differences method.

A graphical interface with Qt and OpenGL libraries was developed in the GNU/Linux environment, with the purpose to show the integration of each of the three devices. With this interface, the user can select the interaction device, it is possible to deform the mesh, also to select the properties of the deformable mesh and the stereoscopy of the scene. Using the interface it is also possible to digitalize an object by collecting a set of points defining that object, this is, the set of (x, y, z) coordinates received form any device can be stored in a file.

Finally, it was made a comparison about the performance and the adaptability of each device integrated to a virtual environment. The results obtained from the comparison allows us to observe the characteristics of each device, their advantages an disadvantages, and that suggest the best way to integrate them to a virtual application.

Agradecimientos

Al Centro de Investigación y Estudios Avanzados del I.P.N.
por brindarme la formación y las herramientas suficientes para el desarrollo del presente trabajo de tesis

Al Consejo Nacional de Ciencia y Tecnología
por proporcionarme la beca que me permitió llegar hasta este punto

Al proyecto 80965
del Consejo Nacional de Ciencia y Tecnología

Al Dr. Luis Gerardo de la Fraga
por su instrucción como asesor y profesor

A mis padres
que me brindaron su comprensión, apoyo y cariño

A mis hermanos y amigos
que confiaron en mí siempre

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

La visualización de modelos a través de ambientes virtuales, se ha convertido en una tarea importante en diversos campos de la tecnología, la ciencia, la educación y el entretenimiento.

Un sistema de realidad virtual, es un sistema de visualización que se encarga de representar algún fenómeno del mundo real valiéndose de varios dispositivos de despliegue que se encargan de dicha visualización, sensores que detectan las acciones del usuario y computadoras que procesan estos eventos generando una salida.

Para simular y generar experiencias virtuales, se trabaja principalmente con ambientes virtuales, también conocidos como mundos virtuales. Dentro de éstos, se encuentran los objetos virtuales cuyas características se presentan al usuario a través de varios sistemas sensoriales.

La interacción con los objetos es un componente importante en un ambiente virtual ya que le permite al espectador no sólo ver o sentir los objetos en el espacio, sino que también le permiten modificarlos o afectarlos de alguna manera.

Para crear una interacción más natural del usuario en un ambiente virtual, se debe tener en cuenta el espacio tridimensional, la movilidad de todo el cuerpo y los órganos sensoriales del mismo. Todo esto para que durante la interacción se tenga una sensación más real. Una sensación negativa debida a un mal diseño de la interfaz, puede causar incomodidad y rechazo a dicha experiencia.

Por otra parte, existen diversas limitantes que se deben superar tanto en los recursos de hardware como con los recursos de software, que deben permitir generar un sistema funcional y ergonómico, características que a su vez causan conflicto con la naturalidad, la usabilidad y la eficiencia de las tareas [1].

En este trabajo de tesis se expone la construcción de una interfaz virtual para la interacción con un cuerpo deformable; se incluye visión estereoscópica y tres sensores

de posición, todos, con seis grados de libertad (trabajan en los tres ejes coordenados con sus tres ángulos de orientación). Estos dispositivos interactúan con el objeto deformable a través de un puntero dentro del espacio virtual.

Se utilizó como objeto deformable una malla de cuadriláteros. En cada arista de la malla se acopló un sistema mecánico simple consistente de una masa, un resorte y un amortiguador, que le confiare al objeto su característica deformable.

Finalmente, se realizó la programación en GNU/Linux para usar los tres dispositivos de interacción. También se hizo una comparación entre las características de los dispositivos, mostrando su adaptabilidad en un ambiente virtual.

1.1. Los modelos deformables

En el área de visualización y gráficos por computadora, se ha trabajado con modelos deformables desde hace ya varios años. Se tienen diversas investigaciones para la visualización y la animación física para modelos de objetos deformables y fluidos.

Se considera que un sólido que es elásticamente deformable, se puede representar con distintos modelos físicos [2]:

- Métodos basados en mallas de Lagrange.
  - Métodos basados en mecánica continua. Consisten de una estructura formada por un conjunto de puntos continuos conectados en tres dimensiones, donde cada punto se le conoce como *coordenada del material* y su deformación está definida por vectores de desplazamiento.
  - Sistemas de masa-resorte. Son los métodos más utilizados, consisten en resolver ecuaciones diferenciales parciales en mallas irregulares. En este caso, el objeto se considera como un volumen continuo conectado que se discretiza utilizando una malla irregular.
- Métodos libres de mallas de Lagrange
  - Sistemas de partículas débilmente acopladas. Son utilizadas cuando el objeto a simular no tiene una superficie bien definida, como el fuego, las nubes o el agua. Constan de un sistema de partículas con propiedades como posición, velocidad, temperatura, forma, tiempo de vida, etc; definiendo un comportamiento dinámico.
  - Hidrodinámica suavizada con partículas (SPH) [3]. Es una técnica de Lagrange basada en la teoría de interpolación que permite a cualquier función y sus derivadas, expresarse en términos del valor de la función en la posición de las partículas. Éste método es muy útil para aplicaciones interactivas dado que reduce la complejidad de computacional.

    - Métodos libres de mallas enfocados a la solución de Ecuaciones Diferenciales Parciales (PDEs). Son técnicas para el modelado de cuerpos deformables que constan de un campo de partículas sobre las cuáles se calcula la fuerza elástica aplicada a una unidad de volumen y varían según su densidad de energía.

- Modelos de deformación reducida y análisis modal. Son modelos reducidos que aproximan la deformación en un punto localizado, utilizando la superposición linear de campos de desplazamiento.

- Métodos basados en Euler y en parte en Lagrange. Consideran la forma en que se discretizan los objetos para trabajar con ellos numéricamente y la forma en que se definen los límites del mismo objeto. Desde el punto de vista de Lagrange, se describe el objeto como un conjunto de puntos en movimiento con propiedades especificas; mientras que desde el punto de vista de Euler, se observa ese conjunto de puntos estáticamente para calcular el cambio de las propiedades en cada uno puntos sin importarle si los límites del objeto están definidos.

    - Gases y fluidos
    - Objetos que se derriten

Los métodos más comunes son los basados en mallas, a través de los cuales se considera que el volumen de los objetos es continuo y conectado. Dentro de estos métodos, se encuentran los sistemas de masa-resorte que son uno de los modelos más sencillos e intuitivos entre los modelos deformables.

Por otra parte, para realizar una representación de un objeto deformable, no sólo es necesario definir un modelo físico para representar su comportamiento, sino que también es indispensable desplegarlo gráficamente para lo que se requiere definir su geometría. Existen diversos modelos para representar geométricamente a un objeto [4]:

- Modelos alámbricos
- Modelos de superficies
    - Modelos de superficies poligonales
    - Modelos de superficies curvas (splines)
- Modelos de sólidos
- Fractales
- Otros

En un escenario del mundo real se encuentran distintos tipos de objetos deformables donde aquellos que pueden representarse en un mundo virtual, a través de un modelo físico basado en mallas (i.e. a través de un sistema masa-resorte) y además puedan modelarse utilizando mallas de polígonos, son los objetos utilizados para el presente caso de estudio.

1.2. Malla deformable

Una malla deformable es una extensión dinámica de puntos en los cuales está definida la superficie de un objeto en un espacio tridimensional. Los puntos en dicha extensión permiten el movimiento y el desenvolvimiento de la representación de un objeto, en respuesta a un conjunto de fuerzas externas y otras restricciones [5].

Las mallas deformables pueden utilizarse para resolver diversos problemas en el área de visión y diseño de gráficos por computadora. Utilizando el modelo físico de masa-resorte y el modelo de mallas de cuadriláteros, la representación de modelos de objetos con contorno dinámico o elásticamente deformable quedan envueltos bajo las restricciones de masa y fuerza apropiadas.

1.2.1. Mallas de polígonos

Un polígono es una figura bidimensional que encierra un área delimitada al menos por tres segmentos de recta, i.e. aristas, que se conectan entre sí en sus extremos, i.e. vértices. Los polígonos (Figura 1.1) se pueden clasificar de acuerdo a su número de aristas (e.g. triángulos, cuadriláteros, etc.) y por la forma de su contorno (e.g. cóncavos, convexos, regulares, etc.).

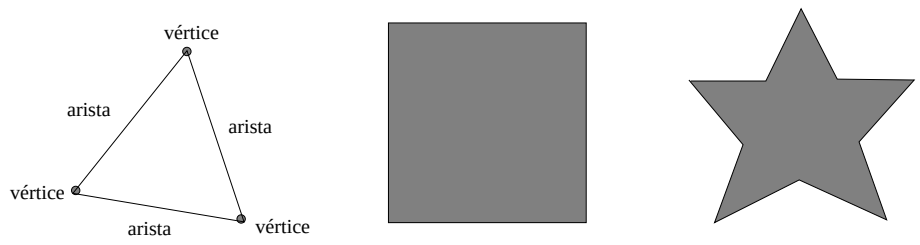

Figura 1.1: Ejemplo de polígonos

Una malla de polígonos (Figura 1.2) es un conjunto de polígonos unidos por sus aristas de tal forma que cada una es compartida a lo más por dos de ellos. Los polígo-

nos que conforman la malla deben ser coplanares y convexos.

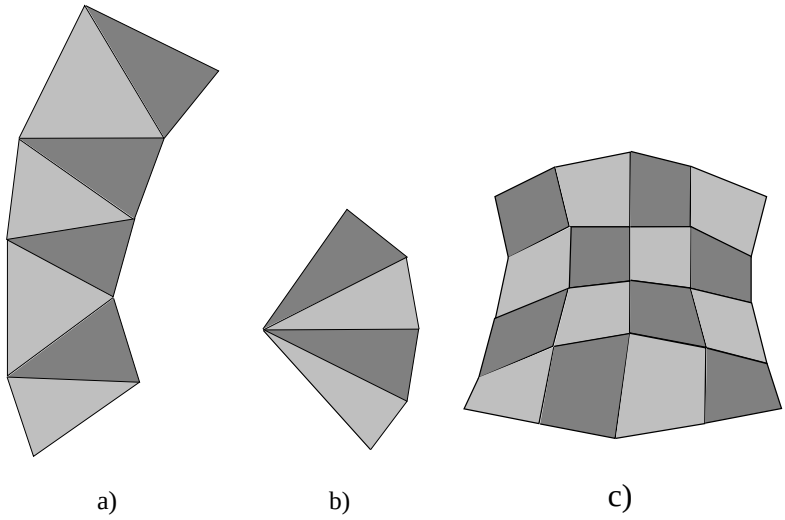

Figura 1.2: Mallas conocidas. a) Cinta de triángulos. Contiene $(n-2)$ triángulos conectados para n vértices. b) Abanico de triángulos. Contiene $(n-2)$ triángulos conectados para n vértices. c) Malla de cuadriláteros. Contiene $(n-1)*(m-1)$ cuadriláteros para $n*m$ vértices.

Las mallas de polígonos, usualmente son utilizadas para el modelado tridimensional de gráficos así como para otros propósitos computacionales. Las mallas más comunes son las de triángulos, sin embargo, para cierto tipo de aplicaciones se prefieren las mallas de cuadriláteros.

1.2.2. Mallas de cuadriláteros

Las mallas de cuadriláteros son utilizadas en el área de visualización y gráficos por computadora para el diseño de superficies y animaciones. Debido a su naturaleza vectorial, los cuadriláteros son preferidos en los esquemas de subdivisión de superficies así como en los dominios de parametrización para las representaciones de superficies curvas (Figura 1.3) [6].

1.3. Planteamiento del problema

La restricción de que dos objetos materiales no pueden ocupar el mismo punto en el espacio al mismo tiempo, es una característica importante del mundo real. Ésta condición física de impenetrabilidad permite la manipulación e interacción de los objetos [7], de tal forma que se puedan moverlos y tocarlos, así como deformar su estructura

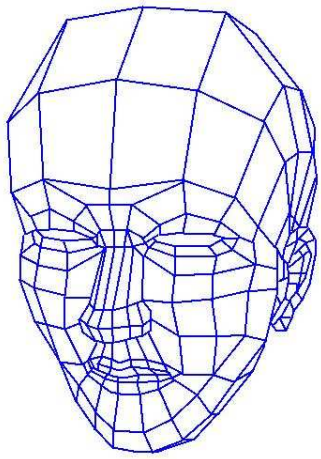

Figura 1.3: Malla tridimensional de cuadriláteros

para el caso de objetos deformables. En un ambiente simulado, esta propiedad se vuelve indispensable si se busca que el objeto de un mundo virtual se comporte tan aproximadamente como sea posible a como lo hiciera en el mundo real, para lo que se requiere de brindarle una estructura física con las características apropiadas que permitan la retroalimentación entre las fuerzas de contacto y la respuesta del objeto.

Con el desarrollo de este trabajo se buscó integrar los tres sensores de posición: Phantom Omni, Patriot y Head Tracker, a un espacio virtual en el que pudieran interactuar con un objeto de estructura deformable y de esta manera probar su eficiencia y adaptabilidad. Así mismo se incluyó la digitalización de puntos para la descripción de objetos y se acopló la visión estereoscópica. De esta manera se exploraron los temas de investigación sobre *Objetos Deformables y* Visión estereoscópica dentro de el área de Visualización y gráficos por computadora y se integraron en un prototipo de sistema funcional.

El ambiente virtual se programó en el ambiente de GNU/Linux, utilizando los lenguajes C y C++, y las herramientas Qt y OpenGL. Una parte importante de esta tesis fue la programación de los tres dispositivos, que sirven para interactuar con el mundo virtual, en GNU/Linux.

1.4. Estado del arte

En el área de visualización y gráficos por computadora, se ha trabajado con modelos deformables desde hace ya varios años. Se tienen numerosas investigaciones para la visualización y la animación física para modelos de objetos deformables y fluidos; las áreas en las que se han hecho estas contribuciones son diversas como: modelado

de objetos, fracturas, plasticidad, animación de ropa, simulación de fluidos estables, adaptabilidad en espacio-tiempo, modelado de resoluciones múltiples, así como simulaciones en tiempo real.

Un sólido que es plásticamente o elásticamente deformable, se puede representar con distintos modelos. Para este trabajo se propone utilizar un sistema de masa-resorte, que consiste en una masa acoplada a un resorte. La fuerza sobre cada punto de masa se calcula en relación a sus resortes vecinos y a las fuerzas externas que actúan sobre él. Los modelos relacionados al comportamiento físico del sólido se mencionan en la sección 1.1.

Elegir una técnica para modelar un objeto deformable, depende en gran medida de los objetivos que se deseen lograr. Teschner et al. [8], hace una investigación de las técnicas enfocadas principalmente a la jerarquía de límites volumétricos, así como una recopilación de trabajos para la detección de colisiones en dicha materia; así mismo, expone temas como métodos inexactos, campos de distancia, subdivisiones espaciales y de imagen-espacio. Menciona de manera general avances relevantes que han sido obtenidos hasta estos días. Esta compilación de temas y trabajos también muestra la gran variedad de aplicaciones que se pueden encontrar dentro de ésta área.

Las simulaciones enfocadas a modelos de ropa y personajes virtuales vestidos, es una de las áreas más estudiadas. Una propuesta para la resolución de este problema, es la de Zhang y Yuen [9], en la que presentan un modelo basado en coherencia para la detección de ambos tipos de colisiones: entre el modelo mismo y colisiones entre el personaje virtual y el modelo de ropa. La eficiencia en la detección de colisiones de un modelo geométrico depende en parte del número de intersecciones que se desean probar. Ha habido algunos métodos que utilizan la propiedad de coherencia, para detectar colisiones por medio de poliedros y superficies curvas. Sin embargo, asumen que el objeto es la unión de poliedros convexos y superficies convexas, lo cual se torna difícil de aplicar en modelos humanos, que sin embargo sí conservan la propiedad de coherencia. El trabajo que hacen está basado en mallas de triángulos, trabajan con límites volumétricos simplificando la detección de colisiones y utilizan regiones de Voronoi y la técnica de voxel para encontrar regiones potenciales de colisión en los vértices de la ropa. Una región de Voronoi alrededor de un punto dado, tiene la propiedad de encerrar la zona de todos los puntos más cercanos él, según alguna medida de distancia [10] (Figura 1.4). En general, presentan un algoritmo que permite la simulación de personajes con vestimenta y hacen una comparación de sus resultados con los de otros métodos existentes.

Otra de las principales áreas a las que se enfocan estos trabajos es a la medicina, en el ámbito de simulaciones quirúrgicas como es el caso del trabajo de Huynh [11]. Las mallas de tetraedros son importantes para el corte de volúmenes, mientras que las mallas de superficies tridimensionales lo son para el corte sobre el límite o término

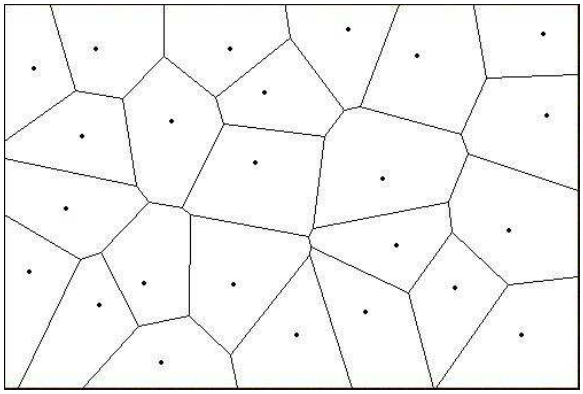

Figura 1.4: Diagrama de Voronoi cubriendo el plano; cada polígono es una región de Voronoi alrededor de cada punto.

de un cuerpo. Las técnicas de corte pueden ser divididas en dos clases: una conformada por aquellas que remueven las mallas intersecadas, reduciendo los elementos de la malla; la otra clase se conforma por las técnicas que regeneran las mallas intersecadas, recreando el camino de corte. En su trabajo, proponen una nueva estrategia para el corte sobre mallas de superficie y hacen la simulación visual con una pantalla de despliegue X3D y utilizan el dispositivo *Phantom* (que consiste de una pluma que ejerce una resistencia sobre un punto, brindando la sensación de tocar el objeto) para la manipulación en 3D.

La mayor parte de los trabajos encontrados, se basan en el uso de mallas de triángulos, que es una estructura conformada, como su nombre lo dice, de triángulos conectados por sus ejes; en el trabajo de Kavan[12], se muestra un nuevo método para la determinar colisiones a través de modelos deformables con esqueleto. Un modelo de éste tipo, consiste de una malla de triángulos como piel digital, un esqueleto y vértices. Trabaja con árboles n-arios de esferas asegurándose de que cada triángulo de la malla quede dentro de un límite esférico, y su principal aportación es que la reinstalación de esferas de acuerdo a un algoritmo que optimiza el proceso. Su algoritmo requiere de un pre-procesamiento inicial para generar el árbol de esferas, y todo este trabajo en su conjunto, agiliza el tiempo de procesamiento para la detección de colisiones, utilizando para esto un algoritmo estándar.

Simular un objeto volumétrico deformable es un problema complejo. En un trabajo reciente, Mezger y Straber [13] proponen un modelo interactivo utilizando elementos finitos. El cerebro humano está entrenado para estimar el comportamiento de cuerpos suaves y determinar sus propiedades físicas al tocarlos. Una animación de un objeto de este estilo requiere basarse en métodos físicos para acercar la simulación lo mejor posible a la realidad. En su trabajo modelan al objeto como un sólido continuo, con base en las leyes de la física y con parámetros del mundo real. Las fuerzas elásticas de un material visco-elástico las representan con una cubierta lineal. También emplean

elementos tetraedricos cuyas funciones de interpolación son utilizadas de igual forma, sobre la geometría del objeto. Muestran un modelo para determinar la estructura del objeto, calcular las fuerzas elásticas e integrarlo en un sistema eficiente. Sin embargo, no trabajan la detección de colisiones.

Por otro lado, se encuentra el trabajo de Schoner[14], el cual muestra una forma interesante de digitalizar un objeto deformable: a través de la observación del objeto. El límite para realismo de una simulación de este tipo, está ligado al modelo en cuestión. Emplean mallas triangulares donde cada vértice tiene su propio factor de desplazamiento o límite de movimiento y utilizando la matriz para la función discreta de Green muestran una combinación con los sistemas de partículas, obtienen un modelo que alcanza a igualarse con el comportamiento observado. Usan radios ajustables que permiten tener ciertos grados de libertad, de esta forma controlan no sólo la magnitud, sino también, la dirección del movimiento. La forma en la que se hace la adquisición del comportamiento de un objeto, consta de aplicar varios desplazamientos en la superficie de dicho objeto y observar su comportamiento tomando estéreo-imágenes antes y después del desplazamiento; también se hace mecánicamente utilizando un brazo que mide la fuerza y el desplazamiento al contacto. Cabe mencionar que todo el algoritmo fue programado en Python, C y C++.

La construcción de caracteres que se deformen naturalmente y cuyo comportamiento sea controlado interactivamente, es un problema que ha sido tratado en pocas ocasiones. Sin embargo, en el trabajo de Turner y Gobbetti [15] se realizó la simulación de superficies deformables alrededor de figuras cinemáticas articuladas. Uno de los principales problemas que atacan, es visualizar con la mayor naturalidad posible, la deformación de piel de un personaje, al mismo tiempo que permiten el control al animador mediante un ambiente virtual, generado por dispositivos de entrada en 3D. Hacen un análisis de la anatomía de un personaje y presentan un modelo elástico por capas que mejora el realismo de dicha animación. Así mismo, en su trabajo exponen un modelo diferencial, basado en un modelo físico que permite calcular las fuerzas elásticas y la restricción de impenetrabilidad. La interfaz del usuario, es una estación de trabajo de alto rendimiento en 3D de Silicon Graphics; está compuesta por un seguidor de movimiento de seis grados de libertad LEMAN y lentes de visión estereoscópica CrystalEyes. Los dispositivos le permiten al diseñador construir un esqueleto del personaje, agregar músculos, conectar la piel y esculpir el modelo con una capa de grasa representada por un espacio geométrico entre el músculo y la piel. Finalmente, la superficie se representa por una malla de cuadriláteros la cual contiene los valores de la animación: posición, velocidad, elasticidad y constantes de resorte. En general, este trabajo habla de la construcción y animación de un personaje, lo cual no permite interactuar con él al final, es decir, sólo se puede manipular su construcción.

Un trabajo más reciente que se enfoca también a la deformación de modelos, es el

de Lanquentin [16], en el cual, se muestra un método que permite deformar objetos de acuerdo a las limitantes dadas por un usuario. Utiliza el modelo de Deformaciones de Formas Libres que consiste en un espacio de deformación en el cual se actúa únicamente sobre los puntos de acción, sin tomar en cuenta la infracción sobre la topología, la geometría o del vecindario. Trabaja con una malla de triángulos la cual sirve para la subdivisión de la superficie del objeto. Dicha malla tiene la propiedad de poder ser subdividida en nuevos triángulos de acuerdo a las necesidades del usuario, permitiendo con esto, modificar el nivel de deformación sobre el objeto.

Uno de los retos importantes para la detección de colisiones, es el rendimiento, Alexander Greb et al [17] hace un trabajo basado en dispositivos GPU (Graphics Processing Unit), que es básicamente un procesador dedicado específicamente al procesamiento de gráficos. De igual forma Greb trabaja con jerarquías de límites volumétricos y utiliza NURBS (Non Uniform Rational B-Splines) un modelo matemático eficiente para la representación de superficies; muestra algunas técnicas para la detección de colisiones, pruebas para la superposición de volúmenes y colisiones entre el mismo objeto. Trabaja con cadenas de almacenamiento sobre las cuales está basado todo su algoritmo.

Continuando con este tema, Lovisach [18] desarrolla un trabajo para procesador GPU en el que su objetivo es obtener un movimiento realista en modelos de ropa, trabaja con mallas de triángulos y logra un modelo que permite procesar eficientemente la deformación de dicho objeto: compresiones, dobleces, desplazamientos, iluminación y textura, todo en tiempo real.

Finalmente, el trabajo de tesis de Julio Moctezuma [19] es fundamental para la presente propuesta de desarrollo. En su trabajo, se presenta un sistema para la manipulación en tiempo real de objetos deformables virtuales sin retroalimentación de fuerzas. Los objetos son construidos a base de mallas de simplejos y en cada arista de la malla se acopla un sistema mecánico simple, compuesto por una masa, un resorte y un amortiguador, el cual le da a la malla su capacidad deformable. El modelo deformable se encuentra bajo la acción de un guante sensorizado en el que se acoplan esferas rígidas virtuales; de esta forma cada esfera representa cada uno de las llamas de los dedos y son las que interactúan y deforman al modelo deformable.

En el área de detección de colisiones, se encuentran diversas propuestas y variadas soluciones a los problemas. Sin embargo, para modelos deformables, el panorama es distinto. La investigación realizada es muy reducida en comparación con la realizada para modelos sólidos. Es importante mencionar que la detección de colisiones, la simulación de retroalimentación de fuerzas de contacto o fricción, y la digitalización de objetos deformables, son actualmente una de las áreas de investigación más activas y con varios retos por cubrir. Sin embargo, en su conjunto no han sido trabajadas y el problema que se pretende solucionar, no ha sido cubierto actualmente.

1.5. Organización de la tesis

Dentro del capítulo 2 se desarrolla el tema de visión estereoscópica describiendo sus características y los avances desarrollados para generar una representación sintética de la profundidad. Específicamente se presenta la solución utilizada dentro del prototipo de sistema desarrollado: la visión activa.

En el capítulo 3 se muestra la estructura de la interfaz gráfica del prototipo de sistema propuesto y específicamente la caracterización del objeto deformable utilizado. Se da una introducción a los sistemas deformables más conocidos dentro de la literatura especializada y se plantean las restricciones definidas para el caso desarrollado.

En el capítulo 4 se da una descripción detallada de cada uno de los tres sensores de posición integrados al prototipo de sistema desarrollado, se explica su funcionalidad y adaptabilidad y finalmente se realiza una comparación entre estos. Cada dispositivo trabaja con una tecnología muy deferente, de esta manera se pueden resaltar las ventajas y desventajas de cada uno respecto a los otros.

Finalmente, dentro del capítulo 5 se presentan las conclusiones a las que se llegaron después del desarrollo del trabajo de tesis y se describe brevemente el trabajo a futuro.

Capítulo 2

Visión estereoscópica

La percepción de profundidad, también conocida como visión estereoscópica, permite generar una imagen en tres dimensiones; esto lo hace automáticamente el cerebro al recibir las imágenes observadas por cada ojo y combinarlas, dando como resultado, la apreciación distancias y volúmenes de los objetos del entorno.

Se sabe que Euclides y Leonardo da Vinci fueron los primeros en observar y estudiar el fenómeno de la visión estereoscópica. También Kepler dejó unos estudios que comentaban los principios de la misma. Pero hasta 1838, el físico Charles Wheatstone construyó el primer aparato, un visor estereoscópico, que permitía percibir la tridimensionalidad partiendo de dos imágenes.

En 1849, David Brewster diseñó y construyó la primera cámara estereoscópica, y años más tarde, Oliver Wendell Holmes construyó lo que seria el estereoscopio de mano más popular del siglo XIX.

En los años 30 resurgió la estereofotografía con la aparición de las cámaras en 3D, y a mediados de siglo hubo diferentes intentos de impulsar las películas en 3D sin demasiado éxito, puesto que las técnicas utilizadas provocaban problemas de visión. Pero no fue hasta los 80s que, con la creación de películas de alta resolución, que se consiguieron mejores resultados como con el formato IMAX 3D.

Actualmente los avances de la tecnología permiten generar y presentar imágenes tridimensionales en los monitores de la computadora y utilizarlas para aplicaciones científicas, industriales o de entretenimiento. Recientemente la NASA ha utilizado la estereoscopía como una herramienta para ver en 3D y analizar las imágenes de Marte enviadas por la sonda Pathfinder [20].

2.1. Parámetros visuales

La mayoría de las personas tienen la capacidad de la visión estereoscópica (Figura 2.1), la posición de los ojos en la cabeza y su separación, permiten tener dos imágenes dispares, es decir, la misma vista sobre un objeto o escenario pero con un pequeño ángulo de diferencia para cada ojo, de ésta manera, las imágenes serán muy similares, pero con una pequeña porción de información distinta que permitirá formar una imagen tridimensional.

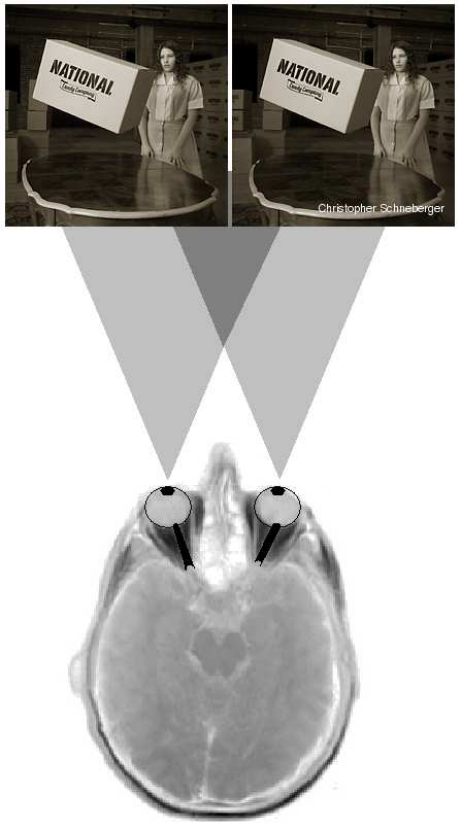

Figura 2.1: Visión estereoscópica, la información capturada por cada ojo es interpretada por el cerebro.

Una característica fisiológica que también es utilizada por el cerebro, es el movimiento de los músculos oculares, cuando se observa un objeto cercano, los ojos giran hasta que los ejes ópticos convergen sobre el objeto, la separación común entre los ojos es de 65 mm, sin embargo, varía entre los 45 mm y los 75 mm. Al observar objetos muy lejanos los ejes ópticos de los ojos quedan paralelos y a ciertas distancias, entre los 60m y los 100m, ya no se tiene la capacidad de distinguir la profundidad.

Otro parámetro visual de importancia es el campo de visión (FOV - Field Of View Figura 2.2), que se refiere al ángulo correspondiente a la superficie visible desde la

posición del observador. Las personas tienen un campo de visión entre los 120 y los 180 grados respecto al eje ocular, pero debido a que se puede girar el cuello entonces se puede alcanzar un campo de visión de 360 grados.

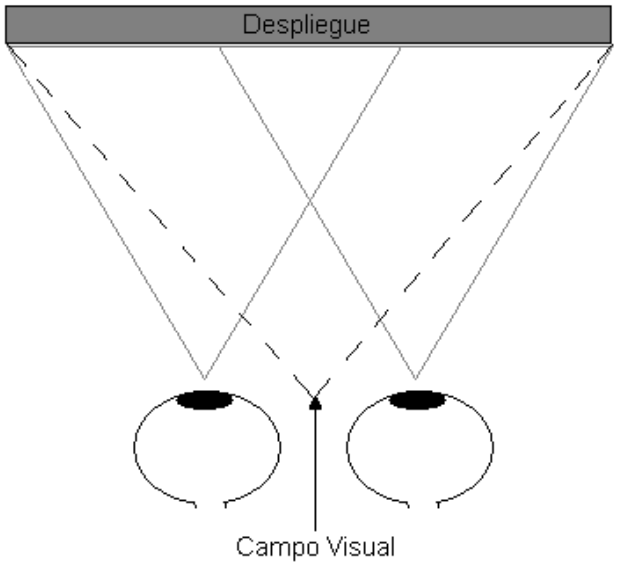

Figura 2.2: Campo visual, área cubierta por la visión con ambos ojos.

En realidad, el proceso que realiza el cerebro para percibir la tridimensionalidad es más amplio, ya que en él influyen otros mecanismos como la perspectiva, el tamaño conocido de los objetos, los detalles, la superposición o solapamiento, la iluminación y las sombras, la acomodación, la convergencia y la disparidad binocular (Figura 2.3) [21].

Utilizando la perspectiva, se puede definir la lejanía ó cercanía de un objeto en el escenario; conforme un objeto esté más lejano su tamaño será más reducido, mientras que dos líneas paralelas convergerán con la distancia. El tamaño de los objetos debe ser proporcional al área que se esté cubriendo del escenario y las líneas paralelas convergen en el llamado punto de fuga.

Asociado al mecanismo de perspectiva se encuentra que por experiencia, las personas adquieren el conocimiento sobre el tamaño que deben tener los objetos en el mundo real, de ésta manera si se observa una taza de té del mismo tamaño que un elefante, se considerará que el elefante se encuentra muy distante y la taza demasiado cerca. De la misma forma, se sabe que el acercamiento hacia un objeto, permite observar más detalles que si se observara a una distancia mayor.

La superposición permite conocer qué objetos se encuentran en el primer plano de nuestro escenario, es decir, un objeto A que obstruye a otro objeto B, es señal de

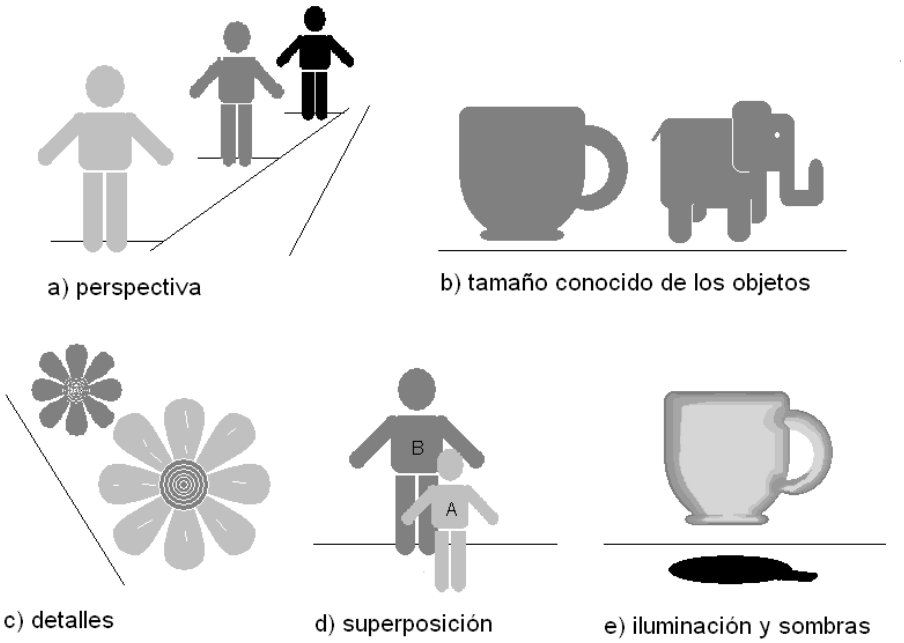

Figura 2.3: Mecanismos en 2D que permiten interpretar elementos en 3D.

que el primero, A, se encuentra delante de el segundo, B.

El grado de iluminación y la proyección de sombras, es un mecanismo que trata de la interpretación de diversas señales dadas por la cantidad de luz en un escenario y en los objetos que lo conforman; el reflejo de la luz en un objeto permite observar su grado de curvatura, al igual que las sombras pueden indicar obstrucciones.

Los mecanismos anteriores: la perspectiva, el tamaño conocido de los objetos, los detalles, la superposición o solapamiento y la iluminación y las sombras, son sencillos de representar en escenarios o imágenes en dos dimensiones, y son características que se pueden encontrar en fotografías, pinturas y dibujos, entre otros, pero solamente brindan una idea del escenario tridimensional, mas no permiten observarlo en tres dimensiones. La acomodación, la convergencia y la disparidad binocular, son los mecanismos que permiten apreciar el escenario tridimensional, por esta razón, se puede decir que son mecanismos indispensables para la visión estereoscópica [1].

2.1.1. Acomodación y convergencia

Cuando una persona observa el objeto de un escenario, el ojo automáticamente rota los ejes oculares, determina la forma de la lente ocular y enfoca, entonces la luz reflejada por el objeto forma su imagen en la retina.

Se le conoce como acomodación (Figura 2.4)a la acción del ojo para acomodarse para que la visión no se perturbe cuando varía la distancia o la luz del objeto que se mira. Este enfoque es más fino para objetos lejanos, mientras que para objetos cercanos la lente se engruesa. Es una característica propia de cada ojo que proporciona el reconocimiento de distancias.

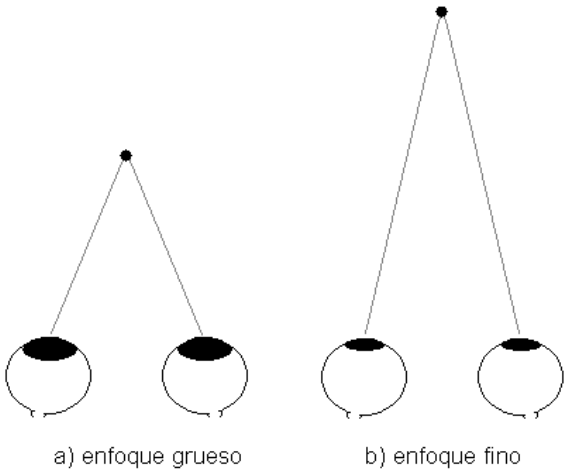

Figura 2.4: Acomodación.

Se le conoce como convergencia (Figura 2.5) a la interacción de ambos ojos para enfocar un objeto, es decir, el grado de movimiento de los dos ejes oculares. Entre más lejano se encuentre un objeto, menor será la convergencia.

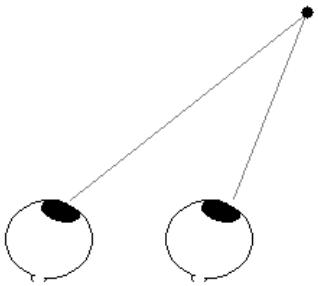

Figura 2.5: Convergencia.

La acomodación y la convergencia son mecanismos estrechamente relacionados, de los cuales el cerebro interpreta la información generada para conocer la profundidad de un objeto o de una escena.

2.1.2. Disparidad binocular

Se le llama disparidad a la diferencia de información generada por cada ojo. La disparidad binocular trata la diferencia entre las imágenes percibidas por cada ojo debido a la proyección de los objetos en sus distintas profundidades. Si se analiza la Figura 2.6, se puede observar la relación entre disparidad y profundidad relativa: la profundidad de un objeto enfocado en un punto A, se convierte en el punto de disparidad cero, éste servirá de referencia para que otros puntos del escenario, como el punto B, sean localizados según su proyección, y dada su lejanía o su cercanía. La disparidad total, es la suma de las disparidades de ambos ojos [22].

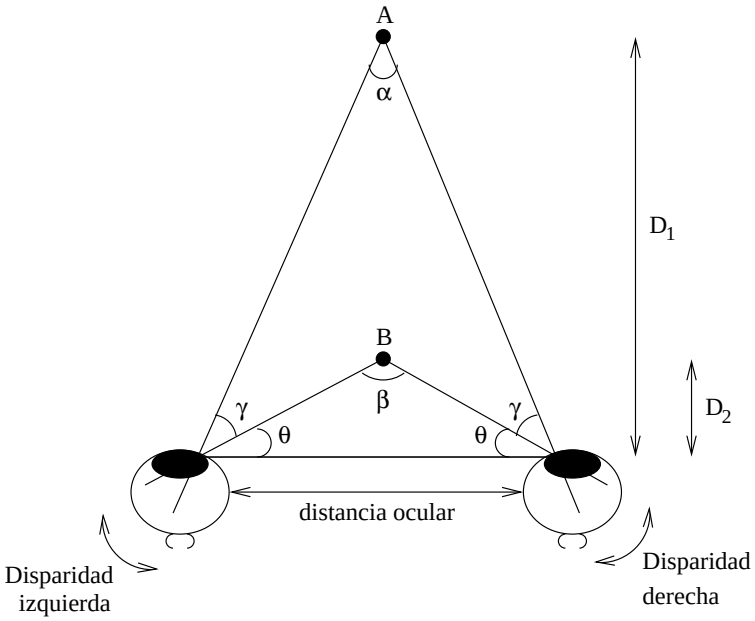

Figura 2.6: Elementos que interactúan en la disparidad binocular.

La disparidad está relacionada con la diferencia en los ángulos de convergencia de los puntos A y B; es directamente proporcional a la diferencia entre los ángulos de convergencia de ambos puntos. De la Figura 2.6, también se puede deducir la profundidad relativa, que es la substracción entre las distancias de los puntos:

$$\text{profundidad relativa} = D_1 - D_2 = \frac{i}{2\tan\left(\tan\frac{\alpha}{2}\right)} - D_2$$

También, se pueden calcular otros parámetros como la diferencia de los ángulos de convergencia y la disparidad de las retinas:

$$\text{diferencia de los ángulos de convergencia} = \alpha - \beta = -2\gamma$$

La suma de los ángulos internos de cualquier triángulo es 180 grados, por lo tanto,

de la misma Figura 2.6 tenemos:

$$\begin{aligned} \beta + 2\theta &= 180 \\ \alpha + 2(\gamma + \theta) &= 180 \\ \alpha - \beta &= 180 - 2(\gamma + \theta) - (180 - 2\theta) \\ &= -2\gamma \end{aligned}$$

La paralaje, es el fenómeno óptico que se da cuando el espectador, en un escenario, se desplaza de una posición a otra lo suficientemente alejada de tal manera que se puede observar un desplazamiento aparente de los objetos cercanos respecto a los objetos lejanos. El ángulo de paralaje permite calcular distancias y se obtiene mediante una triangulación:

$$\text{ángulo de paralaje} = \alpha = 2 \arctan\left(\frac{i}{2D_2}\right)$$

2.2. Generado de pares estereoscópicos

Brindar información sobre la profundidad de los objetos es muy importante para una apreciación tridimensional más natural en un ambiente virtual. Si se desea implementar la visión estereoscópica, lo primero que se necesita es mostrar al mismo tiempo una imagen diferente para cada ojo, de tal manera que únicamente el ojo derecho observe la escena que le corresponde, igual que el ojo izquierdo, para que el cerebro pueda interpretarlas correctamente.

El principal problema es que no se conoce adecuadamente el área que cubre cada ojo, y el despliegue puede causar imágenes erróneas. Normalmente se asume que los ejes oculares observan al infinito y a partir de ésta condición, se obtienen los demás valores relacionados. Pero aunque se conozca el punto de convergencia sigue existiendo el problema de la inconsistencia, la superficie de despliegue estará fija con respecto al movimiento de los ojos y la cabeza.

En la visión natural, la acomodación y la convergencia son mecanismos automáticos que enfocan adecuadamente hacia un punto. Sin embargo, si se quiere presentar una imagen virtual en una pantalla, se debe considerar que el observante tendrá una distancia de separación entre la pantalla y él, por lo que sus ojos estarán enfocando con relación a esa distancia.

Se le llamará paralaje horizontal a la distancia horizontal entre las proyecciones para el ojo izquierdo y el derecho. Dado que el objeto virtual se despliega en la superficie del monitor, se asume que los puntos tridimensionales yacen en el plano de despliegue: paralaje cero , y por lo tanto, éstos serán los puntos de disparidad. Los puntos que se encuentran más allá del plano, generarán una paralaje positiva, mientras que los puntos que se encuentran más cerca que el plano de despliegue generarán

una paralaje negativa ó cruzada (Figura 2.7). Entre más se acerque un objeto al espectador, la paralaje negativa se incrementará hacia el infinito, en consecuencia, se dice que la paralaje es directamente proporcional a la disparidad binocular.

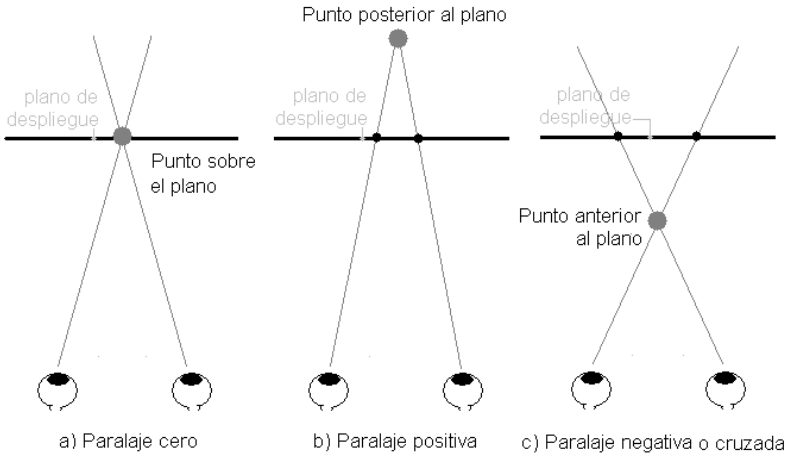

Figura 2.7: Paralaje horizontal.

Cuando una persona enfoca su vista sobre un objeto sobre área de despliegue, éste tendrá una distancia fija, pero el ángulo de convergencia cambia, cuando se mira desde una diferente posición hacia el mismo objeto del escenario.

De forma análoga, se llamará paralaje vertical a la distancia vertical entre las proyecciones para el ojo izquierdo y el derecho.

En los sistemas virtuales, lo más común es utilizar pares estereoscópicos para mostrar escenas en tres dimensiones. Existen dos métodos fundamentales: "toe-in" y "off-axis" (Figura 2.8), el primero es incorrecto, sin embargo se sigue utilizando porque sus costos de filmado son menores [23].

Para el método de "toe-in", se considera que las cámaras están fijas y tienen una separación simétrica y apuntan al mismo punto de enfoque. En éste caso, se generan problemas con la paralaje vertical, que causa incomodidad al mostrar las imágenes de una escena. La paralaje vertical se incrementa conforme la apertura de las cámaras aumenta.

El método "off-axis" es la forma correcta de generar pares estereoscópicos, se puede comparar fácilmente con la configuración común de una visión estereoscópica normal, además de que no genera paralaje vertical.

El efecto que se tendrá al presentar un objeto o un escenario en tres dimensiones, depende tanto de la distancia entre las cámaras y el plano a la que se haya capturado,

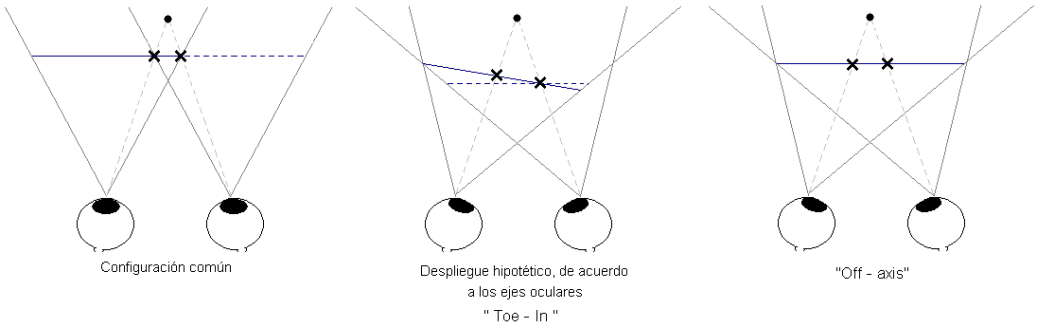

Figura 2.8: Paralaje vertical.

así como de la separación entre ambas cámaras. Se recomienda tener una separación de las cámaras de 1/20 de la distancia al plano de proyección, de igual manera se recomienda que el ángulo de paralaje no exceda los 15 grados.

2.3. Visión Activa

Se pueden utilizar diversas técnicas para desplegar escenas en tres dimensiones, cada técnica requiere de un análisis profundo y de alta tecnología. Una clasificación de los sistemas para la observación estereoscópica más comunes en sistemas de realidad virtual [1], es la siguiente:

- Sistemas auto-estereoscópicos. No requieren usar más dispositivos que ellos mismos.
  - Barreras de Paralaje
  - Hojas lenticulares
- Sistemas no-auto-estereoscópicos. Existen diversos sistemas de éste tipo, pero todos se basan en el mismo principio: generar dos imágenes, una para cada ojo.
  - Visión pasiva. Son sistemas que se basan en el filtrado cromático (por colores) y de otras propiedades de la luz.
    - Anáglifos
    - Luces polarizadas
  - Visión activa
    - Multiplexado en tiempo
  - Visión a través de un dispositivo montado en la cabeza

Dentro de esta clasificación se encuentra la visión activa, también conocida como método de multiplexado en tiempo, éste consta de generar las dos imágenes necesarias para cada ojo, y filtrarlas bloqueando y desbloqueando la visibilidad de cada ojo. El

intercambio de imágenes debe ser rápido y sincronizado, de tal manera que cada ojo vea la imagen apropiada, apreciando el escenario correctamente.

Para realizar éste proceso, se utilizan lentes obturadores (shutter glasses), éstos se sincronizan en conjunto con las imágenes desplegadas en el monitor conforme a la señal de un emisor infrarrojo. Cuando se despliega la imagen para el ojo izquierdo, el lente derecho bloquea la visión del ojo derecho, por lo que el ojo izquierdo adquiere la imagen proporcionada, y de la misma forma para el despliegue y apreciación del ojo derecho (Figura 2.9) [24].

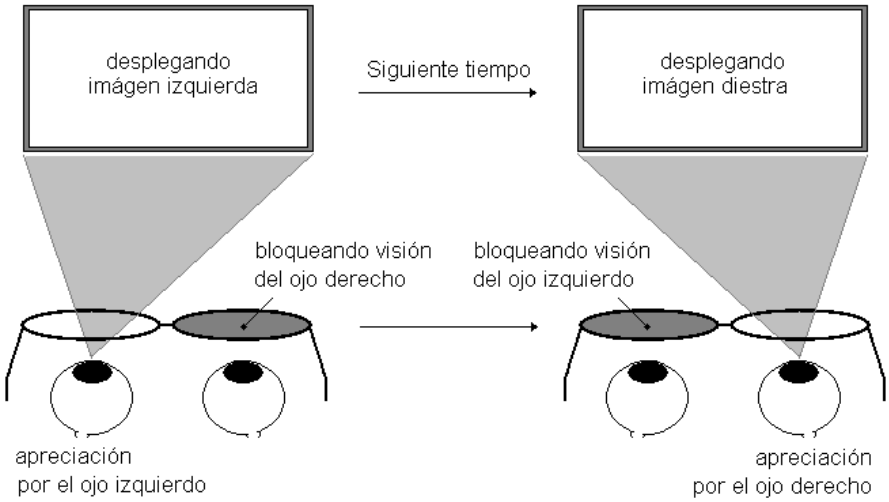

Figura 2.9: Lentes obturadores LCD (shutter glasses).

2.3.1. Hardware utilizado

Los lentes utilizados para este proceso son de cristal líquido (LCD) y de visión estereoscópica sin cables (CrystalEyes 3 de StereoGraphics Co.). El emisor infrarrojo (E-2 Emitter de StereoGraphics Co.) que se encarga de enviar la señal de sincronización a los lentes, se conecta a una tarjeta gráfica (NVIDIA QuadroFx 1500) que permite la visión en estéreo; los lentes deben estar alineados y colocados dentro del radio de alcance del emisor. El cristal líquido de los lentes es sincronizado, obscureciéndolo, para bloquear alternativamente la imagen derecha o izquierda desplegada en el monitor (G225f de ViewSonic), de forma que cada ojo tenga una vista ligeramente compensada. Los obturadores de cristal líquido se abren y cierran entre 80 y 160 veces por segundo.

Dado que la obstrucción es alternada tan rápido, el usuario no puede apreciar las dos imágenes desplegadas, sino una sola en tres dimensiones.

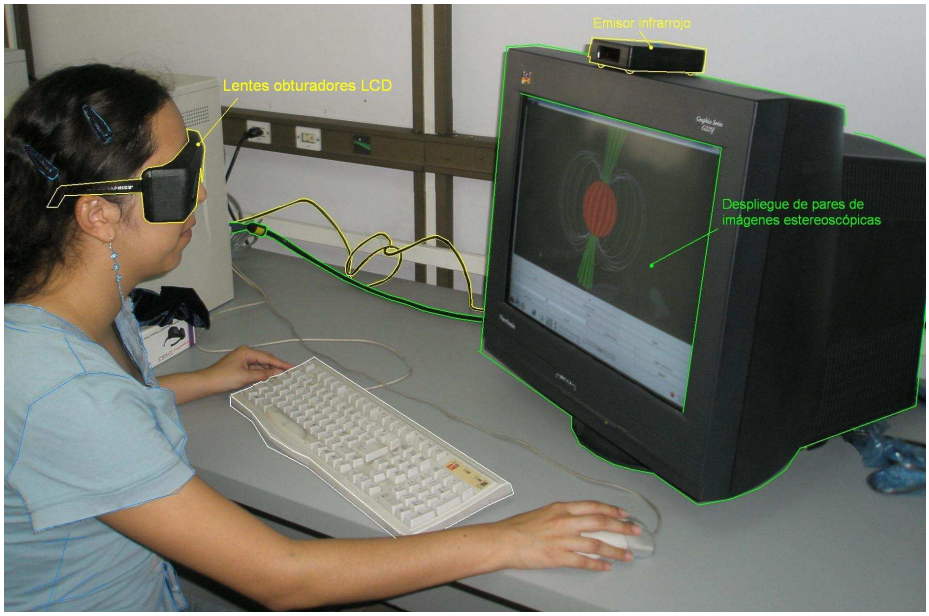

Figura 2.10: Hardware utilizado para la visión activa.

Trabajando en un ambiente Linux (Red Hat Enterprise 3), se debe instalar la tarjeta de vídeo, el monitor, y los dispositivos de hardware para la visión activa. La configuración del emisor se hace por defecto en el momento de configurar la tarjeta de vídeo, especificando habilitar el modo de trabajo en estéreo. Para configurar la tarjeta de vídeo y permitir el modo estéreo, deben estar sincronizados los tiempos que se tienen en la configuración, para esto se debe especificar uno de los generadores disponibles que se utilizarán.

La instrucción gtf ayuda a calcular los modos GTF (Generalized Timing Format), los primeros dos parámetros se refieren a la resolución horizontal y vertical deseadas, mientras que el tercer parámetro se refiere a la tasa de refresco (en Hz). En línea de comandos:

```
$ gtf 1280 1024 118
```

En éste caso, se debe asentar la configuración, agregando la siguiente línea en el archivo de configuración X, /etc/X11/XF86.conf, en la sección del monitor:

```
ModeLine  "1280x1024_118.00" 229.7 1280 1384 1528 1776 1024 1025 1028 1096
-hsync +vsync
```

Y la siguiente sub-sección en el mismo archivo de configuración, en la sección de pantalla, se agrega el modo:

```
SubSection     "Display"
     Depth       24
     Modes      "1280x1024_118.00" "1024x768_120.00" "1280x1024" "1280x960"
                "1152x864" "1024x768"
EndSubSection
```

Mediante el paquete de controladores para las tarjetas de vídeo NVIDIA, se pueden agregar modos de trabajo por medio de la siguiente instrucción:

```
$ nvidia-xconfig -mode 1600x1200
```

Se recomienda revisar los manuales de instalación de cada dispositivo, para lograr el funcionamiento correcto de cada sistema.

2.4. Solución propuesta: Visión activa con Qt y OpenGL

En esta sección se muestra cómo generar la visualización estereoscópica de un objeto virtual en C++, utilizando OpenGL y Qt [25]. Se considera que se tienen los dispositivos de hardware necesarios para trabajar la visión activa; se tienen instalados y configurados. El ejemplo que se describe, es un programa mediante el cual el objeto virtual que se desea mostrar, se enfoca desde dos puntos separados, y las vistas adquiridas se proyectan en el monitor.

Se puede crear un par de estructuras que pueden servir como variables auxiliares dentro del programa, con esto se puede facilitar el entendimiento y manejo del programa.

Se declara una estructura para el manejo de coordenadas:

```
typedef struct {
    double x,y,z;
} XYZ;
```

Se declara una estructura para las propiedades de la cámara (Figura 2.11):

```
typedef struct {
    XYZ vp;              /* Posición del lente           */
    XYZ vd;              /* Vector dirección del lente   */
    XYZ vu;              /* Dirección superior del lente */

    double focallength;  /* Longitud del enfoque por vd  */
    double aperture;     /* Apertura de la cámara        */
    double eyesep;       /* Separación ocular            */

    int screenwidth, screenheight;
} CAMARA;
```

2.4.1. Estereoscopía

Trabajar la visión estereoscópica de un objeto virtual, utilizando las herramientas que brinda OpenGL y Qt, no es muy complicado. Lo primero que se necesita es definir la clase en la que se va a desarrollar la aplicación, pero a ésta se le debe indicar que se trabajará en modo estéreo, pasando el formato como parámetro en el constructor:

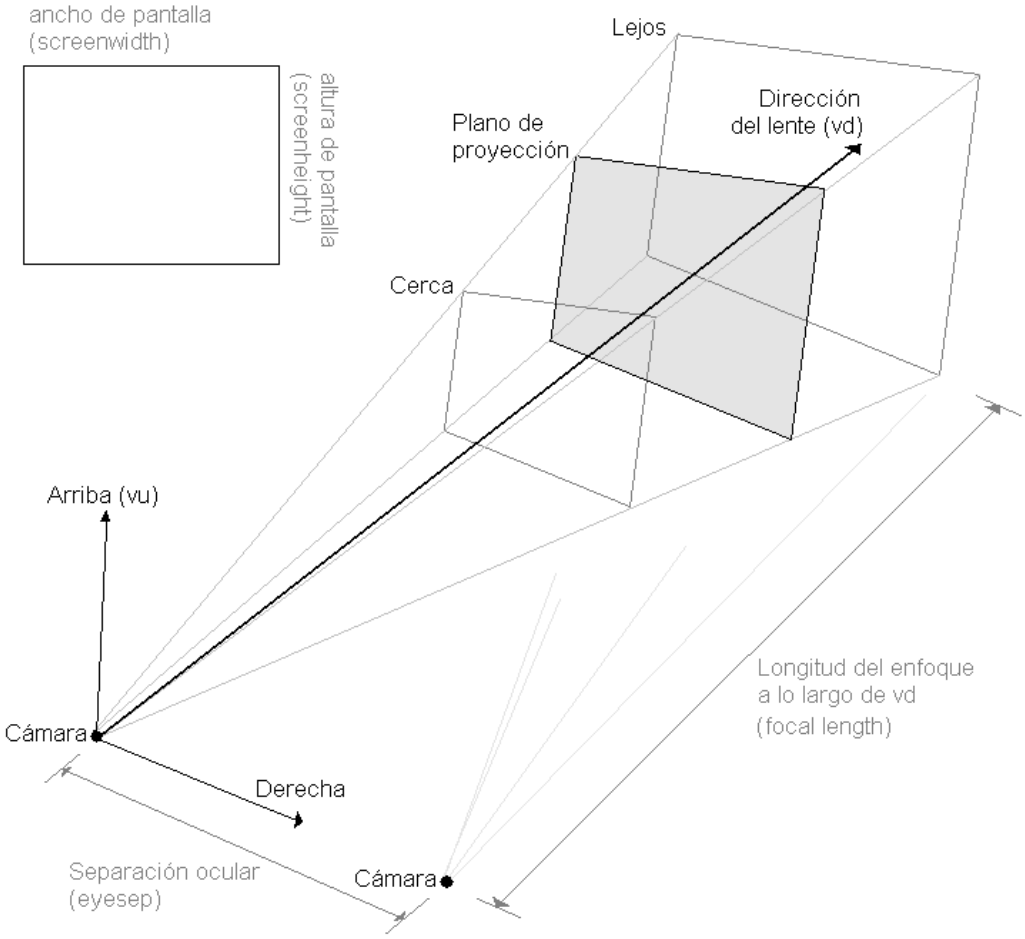

Figura 2.11: Características de una cámara, para la asignación de propiedades en la estructura del programa.

```
QGLFormat fmt;
fmt.setStereo( TRUE );

Estereo *stereoscopic = new Estereo( fmt, this, "Stereo Model - Neihtah" );
```

Mientras que el constructor recibe el parámetro:

```
Estereo::Estereo( QGLFormat fmt, QWidget *parent, const char *name )
          : QGLWidget( fmt, parent, name )
{
    ...
}
```

Para realizar el despliegue con OpenGL, se deben definir las características generales que se van a utilizar durante dicho proceso:

```
void Estereo::initializeGL( )
{
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_POLYGON_SMOOTH);
    glShadeModel(GL_SMOOTH);

    glDisable(GL_LINE_SMOOTH);
    glDisable(GL_POINT_SMOOTH);
    glDisable(GL_DITHER);
    glDisable(GL_CULL_FACE);

    /* Colores del objeto */
    glClearColor(0.0,0.0,0.0,0.0);
    glEnable(GL_COLOR_MATERIAL);
}
```

Por otra parte, se debe inicializar el objeto y su posición de origen, así como los atributos de la cámara.

```
Estereo::Estereo( QWidget *parent, const char *name )
                : QGLWidget( parent, name )
{
    camara.screenwidth  = 400;
    camara.screenheight = 300;
    camara.aperture  = 50;
    camara.focallength  = 70;
    camara.eyesep  = camara.focallength / 20;

    camara.vp.x = 39;
    camara.vp.y = 53;
    camara.vp.z = 22;
    camara.vd.x = -camara.vp.x;
    camara.vd.y = -camara.vp.y;
    camara.vd.z = -camara.vp.z;

    camara.vu.x = 0;
```

```
        camara.vu.y = 1;
        camara.vu.z = 0;
    }
```

Es importante señalar que para utilizar el modo en estéreo en OpenGL, se debe hacer uso de dos buffers: uno para el despliegue de la imagen del ojo izquierdo, `GL_BACK_IZQUIERDA` y otro para el derecho, `GL_BACK_DERECHA`. Para seleccionar el buffer en el que se desea cargar la vista, se hace uso de la función glDrawBuffer().

También se puede limpiar cada buffer seleccionándolo y utilizando la instrucción: `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)`.

2.4.2. Proyecciones usando método "off-axis"

Como ya se mencionó, existen dos métodos para desplegar los pares estereoscópicos de un objeto virtual: "toe-in" y "off-axis"; cuyas proyecciones dependen de la posición de las cámaras y su alineación. Entonces se debe seleccionar cada buffer y almacenar la escena apropiada para poderla dibujar en pantalla adecuadamente.

En el método "toe-in", las cámaras del ojo izquierdo y del derecho, intersecan su enfoque sobre un sólo punto. El método correcto es "off-axis", también conocido en inglés como *"parallel axis asymetric frustrum perspective projection"*, en éste los vectores para cada cámara se colocan paralelos y la instrucción de OpenGL, `glFrustrum()`, se utiliza para definir la perspectiva de la proyección.

```
/* Misc stuff */
radio  = camara.screenwidth / (double)camara.screenheight;
radianes = DTOR * camara.aperture / 2;
wd2     = near * tan(radianes);
ndfl    = near / camara.focallength;

if (estereo) {
    CROSSPROD(camara.vd,camara.vu,r);
    Normalise(&r);
    r.x *= camara.eyesep / 2.0;
    r.y *= camara.eyesep / 2.0;
    r.z *= camara.eyesep / 2.0;

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    izquierda = - radio * wd2 - 0.5 * camara.eyesep * ndfl;
    derecha   =   radio * wd2 - 0.5 * camara.eyesep * ndfl;
    arriba    =   wd2;
    abajo     = - wd2;
    // Se usa para describir la proyección en perspectiva
    glFrustum(izquierda,derecha,abajo,arriba,near,far);
```

```
        glMatrixMode(GL_MODELVIEW);

        //selección del buffer
        glDrawBuffer(GL_BACK_DERECHA);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        gluLookAt(camara.vp.x + r.x,camara.vp.y + r.y,camara.vp.z + r.z,
                  camara.vp.x + r.x + camara.vd.x,
                  camara.vp.y + r.y + camara.vd.y,
                  camara.vp.z + r.z + camara.vd.z,
                  camara.vu.x,camara.vu.y,camara.vu.z);

        MakeLighting();
        MakeGeometry();

        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        izquierda  = - radio * wd2 + 0.5 * camara.eyesep * ndfl;
        derecha =   radio * wd2 + 0.5 * camara.eyesep * ndfl;
        arriba    =   wd2;
        abajo = - wd2;
        glFrustum(izquierda,derecha,abajo,arriba,near,far);

        glMatrixMode(GL_MODELVIEW);
        glDrawBuffer(GL_BACK_IZQUIERDA);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        gluLookAt(camara.vp.x - r.x,camara.vp.y - r.y,camara.vp.z - r.z,
                  camara.vp.x - r.x + camara.vd.x,
                  camara.vp.y - r.y + camara.vd.y,
                  camara.vp.z - r.z + camara.vd.z,
                  camara.vu.x,camara.vu.y,camara.vu.z);
        MakeLighting();
        MakeGeometry();
    } else {
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        izquierda  = - radio * wd2;
        derecha =   radio * wd2;
        arriba    =   wd2;
        abajo = - wd2;
        glFrustum(izquierda,derecha,abajo,arriba,near,far);

        glMatrixMode(GL_MODELVIEW);

        // para tarjetas optimizadas GL_BACK (+ rápido)
        glDrawBuffer(GL_BACK);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        gluLookAt(camara.vp.x,camara.vp.y,camara.vp.z,
                  camara.vp.x + camara.vd.x,
                  camara.vp.y + camara.vd.y,
```

```
                camara.vp.z + camara.vd.z,
                camara.vu.x,camara.vu.y,camara.vu.z);
        MakeLighting();
        MakeGeometry();
    }

    swapBuffers();
```

Es importante señalar que en algunas ocasiones, es apropiado usar la posición del ojo izquierdo cuando no se está utilizando el modo estéreo, en cuyo caso, el código anterior puede ser simplificado.

Capítulo 3

Diseño de la deformación del cuerpo y de la interfaz gráfica

3.1. Sistemas masa-resorte

Los sistemas masa-resorte son uno de los más sencillos e intuitivos entre los modelos deformables. Son un modelo discreto que, como su nombre lo indica, consiste en puntos de masa conectados a través de una red de resortes sin masa (Figura 3.1).

Cuando las fuerzas actúan sobre los nodos de la malla, los puntos de masa son evaluados dentro del sistema, mientras que cada elemento actúa como un resorte generalizado conectando todos lo puntos de masa adyacentes.

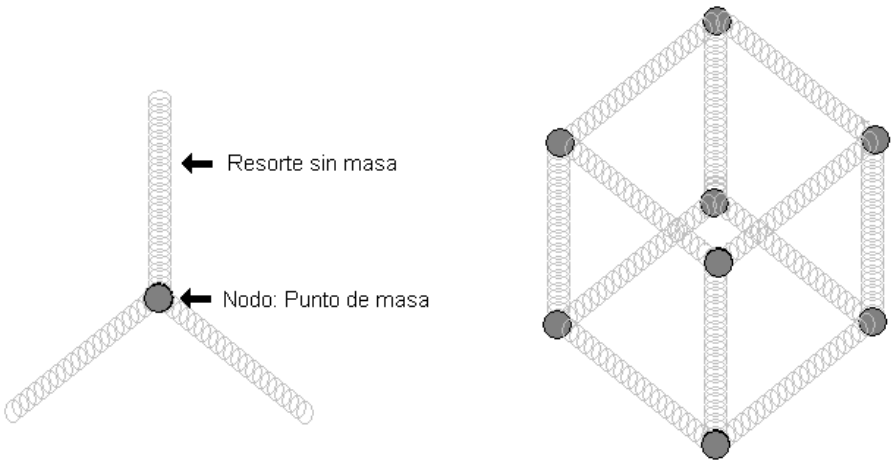

Figura 3.1: Sistema masa-resorte

3.1.1. Movimiento armónico amortiguado

Un objeto deformable está definido por su forma en su estado de reposo también conocido como configuración de equilibrio, forma inicial o forma estacionaria. Además, está definida por el conjunto de parámetros que especifican la deformación cuando se le aplica una fuerza al objeto.

Si se considera un sistema masa-resorte sin fricción que pende verticalmente, al cuál se le aplica una fuerza externa $F_ext = -kx$ (Figura 3.2), donde k es el coeficiente de elasticidad del resorte y x la deformación, tomando en cuenta que el origen de referencia para esta última variable es la posición de equilibrio del resorte; entonces utilizando la segunda ley de Newton, se puede obtener la siguiente igualdad:

$$kx = ma \tag{3.1}$$

Siendo m es la masa y la aceleración es $a = \frac{d^2}{dt^2} = \ddot{x}$, entonces se tiene la ecuación general:

$$\ddot{x} + w_0^2 x = 0 \tag{3.2}$$

donde w_0^2 es la frecuencia de oscilación. Entonces la solución a esta ecuación diferencial está dada por una frecuencia de oscilación constante:

$$x(t) = C \cos w_0 t - \sigma \tag{3.3}$$

de la ecuación anterior, C y σ son constantes.

Lo anterior indica que el movimiento de este sistema continuará indefinidamente debido a que su energía mecánica se mantiene constante.

Entonces, para que el sistema masa-resorte antes descrito llegue a un estado de reposo, se necesita colocar un amortiguador que le reste, de tal manera que permita aproximar éste modelo al mundo real como se muestra en la Figura 3.3.

El modelo para el nuevo sistema genera una fuerza de amortiguamiento que siempre se opondrá al movimiento y será directamente proporcional a la magnitud de la velocidad. De la misma forma que en el sistema anterior, se obtiene la ecuación:

$$m\ddot{x} + b\dot{x} + kx = F_{\text{ext}} \tag{3.4}$$

donde b es la constante de amortiguamiento, m es la masa, $\ddot{x}$ es la aceleración, $\dot{x}$ es la velocidad y k es la constante de elasticidad; y x es la posición.

Las constantes b y k se determinan mediante las condiciones iniciales.

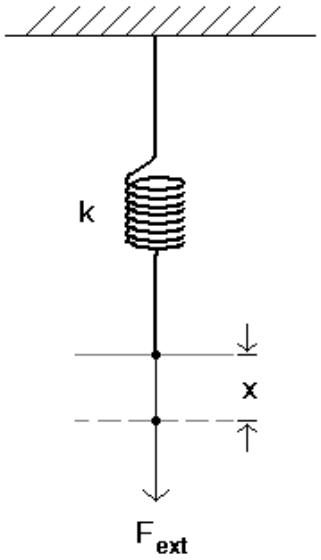

Figura 3.2: Sistema masa-resorte en el movimiento armónico simple.

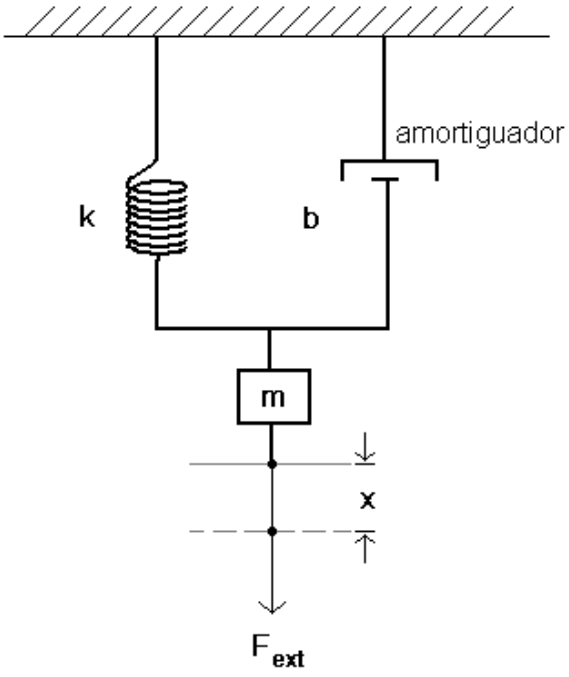

Figura 3.3: Sistema masa-resorte-amortiguador.

De esta forma las características del movimiento resultante del sistema, estarán dadas por la relación entre el factor de amortiguamiento y entonces, la masa va a tender a detenerse debido al decaimiento de energía mecánica.

Utilizando la fórmula general, se pode observar cómo reacciona el sistema ante diversas condiciones, se tienen tres casos: movimiento sobreamortiguado, movimiento críticamente amortiguado y movimiento subamortiguado.

3.1.2. Movimiento sobreamortiguado

Con el sobreamortiguamiento, la masa no tiene oportunidad de oscilar, y sin importar cuáles sean las condiciones iniciales del sistema, la tendencia del movimiento es la de llevar a la masa hacia la posición de equilibrio (Figura 3.4).

Se da cuando el factor de amortiguamiento b es mayor que la frecuencia de oscilación.

$$b^2 - 4mk > 0 \Longrightarrow b > 2\sqrt{mk} \tag{3.5}$$

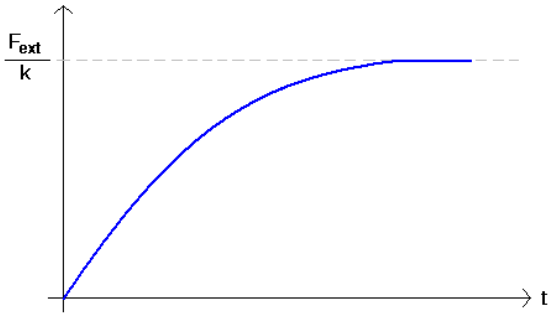

Figura 3.4: Relación tiempo - movimiento para un sistema sobreamortiguado.

3.1.3. Movimiento críticamente amortiguado

En este caso, la masa tiende a la posición de equilibrio debido al decaimiento exponencial y la masa tampoco tiene oportunidad de oscilar a pesar de sus condiciones iniciales (Figura 3.5).

Dicho movimiento da cuando el factor de amortiguamiento b es igual a la frecuencia de oscilación:

$$b^2 - 4mk = 0 \Longrightarrow b = 2\sqrt{mk} \tag{3.6}$$

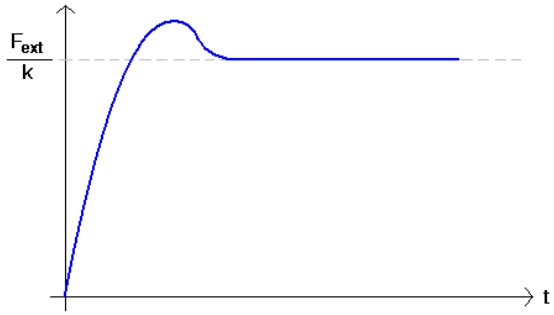

Figura 3.5: Relación tiempo - movimiento para un sistema críticamente amortiguado.

3.1.4. Movimiento subamortiguado

En el caso del movimiento subamortiguado (Figura 3.6), la masa de oscila con una frecuencia determinada antes de llegar a la posición de equilibrio. Esto ocurre si el factor de amortiguamiento b es menor que la frecuencia de oscilación:

$$b^2 - 4mk < 0 \Longrightarrow b < 2\sqrt{mk} \tag{3.7}$$

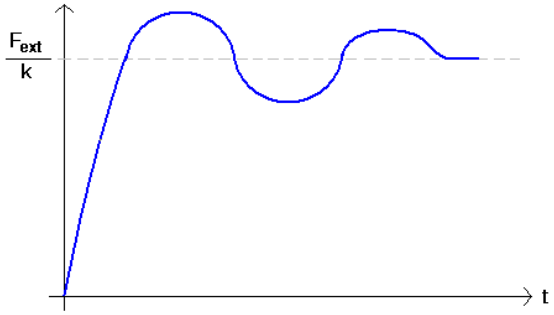

Figura 3.6: Relación tiempo - movimiento para un sistema subamortiguado.

3.1.5. Solución del sistema utilizando diferencias finitas

El método de diferencias finitas, permite resolver la ecuación diferencial de una fuerza elástica ejercida sobre un sistema masa-resorte. Resolviendo para la ecuación 3.4, del movimiento armónico amortiguado:

Definimos:

$$\dot{x} = \frac{x(t) - x(t-1)}{\Delta t}$$

$$\ddot{x} = \frac{V_f - V_i}{\Delta t}$$

Entonces:

$$V_f = \frac{x(t+1) - x(t)}{\Delta t}$$
$$V_i = \frac{x(t) - x(t-1)}{\Delta t}$$

Substituyendo en 3.4:

$$m\left[\frac{\frac{x(t+1)-x(t)}{\Delta t} - \frac{x(t)-x(t-1)}{\Delta t}}{\Delta t}\right] + b\left[\frac{\frac{x(t)-x(t-1)}{\Delta t}}{\Delta t}\right] + k\,x(t) = F_e xt$$

$$m\left[\frac{x(t+1) - 2x(t) + x(t-1)}{(\Delta t)^2}\right] + b\left[\frac{x(t) - x(t-1)}{\Delta t}\right] + k\,x(t) = F_e xt$$

Reduciendo la ecuación anterior, queda la siguiente solución:

$$x(t+1) = x(t) + [x(t) - x(t-1)]\left(1 - \frac{b}{m}\Delta t\right) - \frac{k}{m}(\Delta t)^2 x(t) + \frac{F_e xt}{m}(\Delta t)^2 \quad (3.8)$$

Donde $\frac{k}{m}(\Delta t)^2 x(t)$ es la fuerza interna, x es el desplazamiento, t es el tiempo, k es la constante del resorte y b la constante de amortiguado.

Es importante señalar que si $b = 0$ o es muy pequeño, el sistema se vuelve inestable.

3.2. Solución propuesta: Malla deformable de cuadriláteros

Para una malla bidimensional de cuadriláteros de nxm nodos, se considera que nodos vecinos son aquellos que se encuentran conectados a través de los resortes sin masa (Figura 3.7). Entonces se consideran tres tipos de nodos: con 2 vecinos, con 3 vecinos y con 4 vecinos.

La geometría del objeto deformable de éste caso de estudio tiene la característica de poseer simetría cilíndrica; de esta forma, se facilitó la reconstrucción de su superficie generándose una malla de cuadriláteros al revolucionar su semi-contorno.

Se muestrearon m puntos del semi-perímetro del objeto en un plano 2D (XY) y se definió su eje de rotación en el eje Y del plano. La reconstrucción consta de rotar n veces los puntos y conectarlos con los resortes sin masa (Figura 3.8).

De esta manera, se obtiene una malla tridimensional de cuadriláteros con nxm nodos, donde los m nodos de la primera columna están conectados con los m nodos de la última (Figura 3.9). Por lo tanto, los nodos con 2 vecinos quedan descartados.

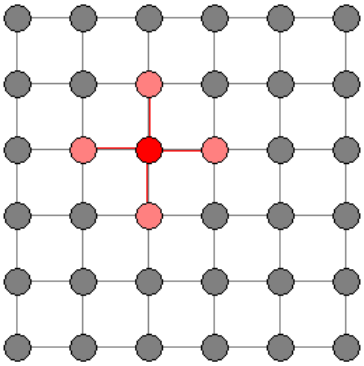

Figura 3.7: Vecindario de un nodo.

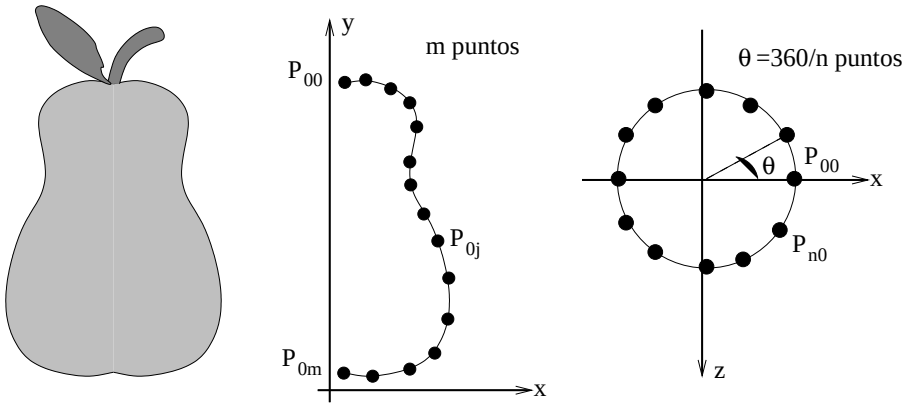

Figura 3.8: Objeto con simetría cilíndrica y su rotación en el eje Y.

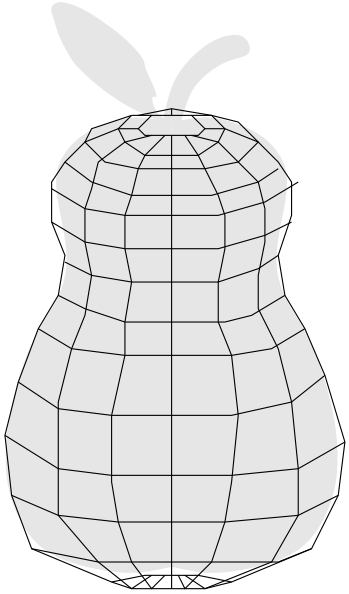

Figura 3.9: Malla de cuadriláteros con simetría cilíndrica.

Finalmente, para calcular la animación del objeto deformable cuando se le aplica una fuerza externa, se deben definir los puntos afectados, recorrer los nodos y aplicar el modelado masa-resorte-amortiguador de la ecuación 3.8. Posteriormente se debe actualizar la malla nodo por nodo con respecto al tiempo.

3.2.1. Programación con Qt y OpenGL

Considerando que se tiene la malla de cuadriláteros del caso anterior, las dimensiones del sistema siempre serán conocidas. Cada nodo o vértice se considera como un punto con propiedades independientes que interactúa con los demás vecinos representando el comportamiento de la malla en general.

Se comienza por definir una matriz de nxm localidades de memoria para una malla de nxm puntos, donde cada localidad consiste de una estructura con las propiedades específicas de cada punto.

```
class MassSpringSystem : public QGLWidget
{
    ...
    ttypedef struct {
        float x, y, z;                  // Point coordinates
        float xt, yt, zt;               // Point at time (t)
        float xt1, yt1, zt1;            // Point at time (t+1)
        float xt_1, yt_1, zt_1;         // Point at time (t-1)
        float forcex, forcey, forcez;   // Force components
        float color[3];                 // Point color
        int num_neighbors;              // Number of neighbors
        int state;                      // 0-NORMAL 1-FORCEAPP
        bool evaluated;                 // FALSE-TRUE
    }Point;

    float x_ini, y_ini, z_ini;          // initial force measure
    float x_end, y_end, z_end;          // final force measure
    float m, b, kr, dt, dt2;            // mass-spring system constants
    float force;                        // Force
}
```

Para la simulación, en la definición de la clase también se declaran variables del modelo masa-resorte-amortiguador: la fuerza externa aplicada al sistema y sus componentes en X, en Y y en Z, la masa y los coeficientes de resorte y de amortiguado; los cuales se inicializan en el constructor.

```
MassSpringSystem::MassSpringSystem( QWidget *parent, const char *name )
: QGLWidget( parent, name )
{
     ForceActing = FALSE;
     HorizontalPoints = 0;
     VerticalPoints = 0;
```

```
        force  = 0;

        x_ini = y_ini = z_ini = 0.0;
        x_end = y_end = z_end = 0.0;

        /* Mass-spring system constants */
        m   = 0.01;        // mass
        b   = 0.25;        // dampling constant [0.01 - 0.4]
        kr  = 2.30;        // spring constant [0.01, 3.0]
        dt  = 0.05;        // delta [0.01, 0.1]

        /* Time */
        sys_timer = new QTimer( this, "my_sys_timer" );
        connect( sys_timer, SIGNAL(timeout()), this,
                            SLOT(nextTime()) );

}
```

Antes de realizar el despliegue gráficos de la malla, se inicializan cada uno de sus nodos y se generan los puntos revolucionados:

```
float theta = 360.0 / HorizontalPoints;

for ( int i=0; i < HorizontalPoints; i++ ) {
    for ( int j=0; j < VerticalPoints; j++ ) {
     /* Rotatin points */
        point[i][j].x = (semi_perim[j][0]) * cos (theta * i * DTOR);
        point[i][j].y = (semi_perim[j][1]);
        point[i][j].z = (semi_perim[j][0]) * sin (theta * i * DTOR);

        // zero initialization
        point[i][j].xt_1 = 0;
        point[i][j].xt = 0;
        point[i][j].xt1 = 0;

        point[i][j].yt_1 = 0;
        point[i][j].yt = 0;
        point[i][j].yt1 = 0;

        point[i][j].zt_1 = 0;
        point[i][j].zt = 0;
        point[i][j].zt1 = 0;

        point[i][j].forcex = 0;
        point[i][j].forcey = 0;
        point[i][j].forcez = 0;

        point[i][j].state = NORMAL;
        point[i][j].evaluated  = FALSE;

        if ( j==0 || j==VerticalPoints-1 )
             point[i][j].num_neighbors = 3;
        else
```

```
                point[i][j].num_neighbors = 4;

        }
    }
```

Una vez que se tiene definido el objeto de acuerdo a un sistema de mallas de cuadriláteros, para simular la deformación del objeto al aplicarle una fuerza puntual, se resuelve el sistema aplicando la ecuación 3.8 en cada una de las componentes de sus nodos.

```
for ( int i=0; i < HorizontalPoints; i++ ) {
    for ( int j=0; j < VerticalPoints; j++ ) {
        if (point[i][j].state == FORCEAPP) {
            expansiveAlgth(i, j);
        }

        point[i][j].xt_1 = point[i][j].xt;
        point[i][j].xt   = point[i][j].xt1;
        point[i][j].xt1  = (point[i][j].forcex/m)*dt2 + point[i][j].xt
                         + (point[i][j].xt-point[i][j].xt_1)*(1-b/m*dt)
                         - kr/m*dt2*point[i][j].xt
                         ;

        point[i][j].yt_1 = point[i][j].yt;
        point[i][j].yt   = point[i][j].yt1;
        point[i][j].yt1  = (point[i][j].forcey/m)*dt2 + point[i][j].yt
                         + (point[i][j].yt-point[i][j].yt_1)*(1-b/m*dt)
                         - kr/m*dt2*point[i][j].yt
                         ;

        point[i][j].zt_1 = point[i][j].zt;
        point[i][j].zt   = point[i][j].zt1;
        point[i][j].zt1  = (point[i][j].forcez/m)*dt2 + point[i][j].zt
                         + (point[i][j].zt-point[i][j].zt_1)*(1-b/m*dt)
                         - kr/m*dt2*point[i][j].zt
                         ;
    }
}
```

Cuando la malla ha sido afectada por una fuerza externa en alguno de sus nodos, el sistema se actualiza de acuerdo al algoritmo `expansiveAlgth` que recorre toda la malla comenzando desde el nodo donde reside la fuerza aplicada, afectando consecutivamente a cada uno de sus nodos vecinos.

```
void MassSpringSys::expansiveAlgth ( int i, int j ) {
    int limLeft=i, limRight=i, limUp=j, limDown=j;
    int counter=0, area = 0;

    neighborsForce(i, j);
    while(area <= HorizontalPoints*VerticalPoints) {
        limLeft  = (limLeft>0)?limLeft-1:0;
```

```
            limRight = (limRight<HorizontalPoints-1)?limRight+1:HorizontalPoints-1;
            limUp    = (limUp>0)?limUp-1:0;
            limDown  = (limDown<VerticalPoints-1)?limDown+1:VerticalPoints-1;

            area = (limRight - limLeft + 1) * (limDown - limUp + 1);

            /* Algorithm for expansive force through neighbors */
            for ( counter = i; counter <= limRight; counter++ ) {
                if ( point[counter][limUp].state==NORMAL )
                    neighborsForce( counter, limUp );
                if ( point[counter][limDown].state==NORMAL )
                    neighborsForce( counter, limDown );
            }
            for ( counter = i-1; counter >= limLeft; counter-- ) {
                if ( point[counter][limUp].state==NORMAL )
                    neighborsForce( counter, limUp );
                if ( point[counter][limDown].state==NORMAL )
                    neighborsForce( counter, limDown );
            }
            for ( counter = j; counter <= limDown; counter++ ) {
                if ( point[limLeft][counter].state==NORMAL )
                    neighborsForce( limLeft, counter );
                if ( point[limRight][counter].state==NORMAL )
                    neighborsForce( limRight, counter );
            }
            for ( counter = j-1; counter >= limUp; counter-- ) {
                if ( point[limLeft][counter].state==NORMAL )
                    neighborsForce( limLeft, counter );
                if ( point[limRight][counter].state==NORMAL )
                    neighborsForce( limRight, counter );
            }

            if( area == HorizontalPoints*VerticalPoints ) break;

        }// end while
    }
```

Para mejorar el rendimiento de la aplicación, se consideró convertir las coordenadas del objeto a coordenadas de pantalla (Figura 3.10).

De esta forma se tienen normalizadas las dimensiones para el despliegue del objeto en OpenGL (Figura 3.11) y se pueden calcular las magnitudes de las fuerzas aplicadas sobre el mismo.

```
void MassSpringSys::PaintSystem ( void ) {
    GLfloat DBlue[3]  = { 0.2, 0.3, 0.5 };   // Mesh lines colors

    /* Drawing points */
    glPushAttrib( GL_CURRENT_BIT | GL_ENABLE_BIT );
        if ( !pointDisplayList ) {
```

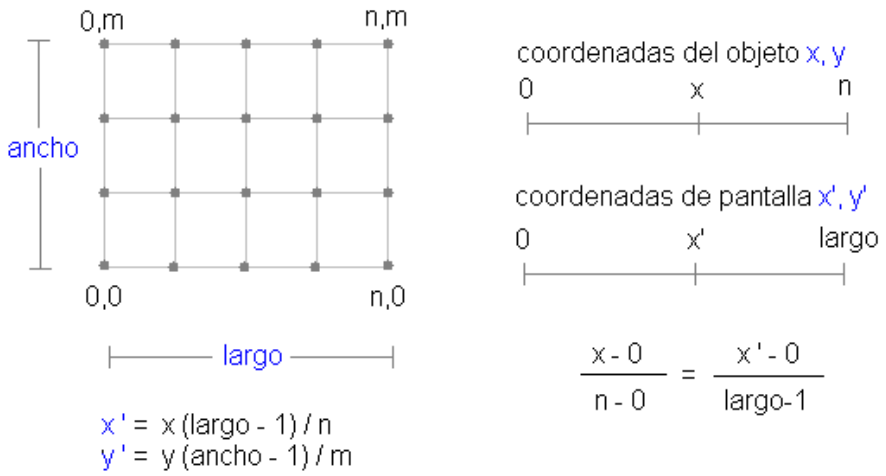

Figura 3.10: Conversión de coordenadas de objeto a coordenadas de pantalla

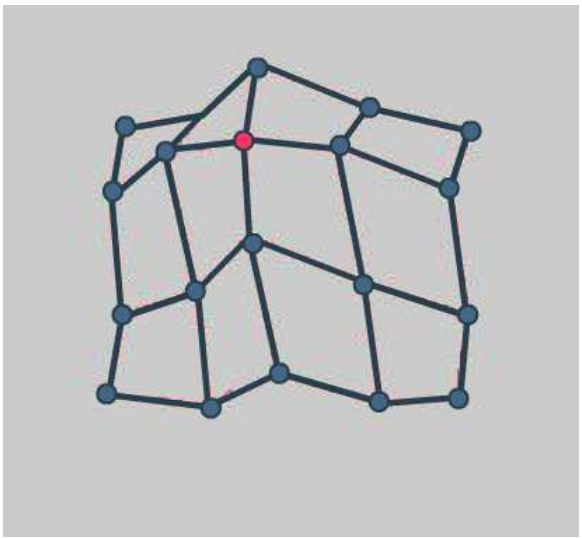

Figura 3.11: Ejemplo de una malla de cuadriláteros en movimiento resuelta con la metodología propuesta.

```
    pointDisplayList = glGenLists(1);

    GLUquadricObj *sphere = gluNewQuadric();
    gluQuadricDrawStyle( sphere, GLU_FILL );
    gluQuadricNormals( sphere, GLU_SMOOTH );

    glNewList( pointDisplayList, GL_COMPILE );
        gluSphere( sphere, 0.01, 10, 10 );
    glEndList();

    gluDeleteQuadric( sphere );
}

glEnable( GL_NORMALIZE );
glEnable( GL_COLOR_MATERIAL );

/* Draw each of the points as lit sphere */
for (int i=0; i < HorizontalPoints; i++) {
    for (int j=0; j < VerticalPoints; j++) {
        glPushMatrix( );
            glColor3fv( point[i][j].color );
            glTranslated( point[i][j].x + point[i][j].xt,
                          point[i][j].y + point[i][j].yt,
                          point[i][j].z + point[i][j].zt);
            glCallList( pointDisplayList );
        glPopMatrix( );
    }
}

/* Draw the line segments that connect the points. */
glBegin(GL_LINES);
    glColor3f( DBlue[0], DBlue[1], DBlue[2] );
    glLineWidth( 2.0 );
    for (int i=0; i < HorizontalPoints; i++) {
        for (int j=0; j < VerticalPoints; j++) {
             if( i < HorizontalPoints-1 ) {
                    glVertex3f( point[i][j].x
                              + point[i][j].xt,
                                point[i][j].y
                              + point[i][j].yt,
                                point[i][j].z
                              + point[i][j].zt );

                    glVertex3f( point[i+1][j].x
                              + point[i+1][j].xt,
                                point[i+1][j].y
                              + point[i+1][j].yt,
                                point[i+1][j].z
                              + point[i+1][j].zt );
            }
            if( j < VerticalPoints-1 ) {
                    glVertex3f( point[i][j].x
```

```
                                        + point[i][j].xt,
                                          point[i][j].y
                                        + point[i][j].yt,
                                          point[i][j].z
                                       + point[i][j].zt );

                                glVertex3f( point[i][j+1].x
                                        + point[i][j+1].xt,
                                          point[i][j+1].y
                                        + point[i][j+1].yt,
                                          point[i][j+1].z
                                        + point[i][j+1].zt );
                            }
                        } // end for
                    } // end for

                    for (int j=0; j < VerticalPoints; j++) {
                        glVertex3f( point[0][j].x + point[0][j].xt,
                                    point[0][j].y + point[0][j].yt,
                                    point[0][j].z + point[0][j].zt );

                        glVertex3f( point[HorizontalPoints-1][j].x
                                  + point[HorizontalPoints-1][j].xt,
                                    point[HorizontalPoints-1][j].y
                                  + point[HorizontalPoints-1][j].yt,
                                    point[HorizontalPoints-1][j].z
                                  + point[HorizontalPoints-1][j].zt );
                    }
                glEnd();

            glPopAttrib();
        }
```

3.3. Descripción general de la aplicación

El diagrama de clases del prototipo de sistema desarrollado se muestra en la Figura 3.12. Las clases principales que describen dicho diagrama permiten tener una vista global del prototipo.

En el diagrama de clases de la Figura 3.12 se encuentran las clases: *Painter*, *StereoDisplay*, *PhantomHandler*, *PatriotHandler*, *HTrackerHandler*, *MassSpringSystem* y *PointsMap* que se encargan del desempeño del prototipo descrito en el presente trabajo y se describen a continuación:

La clase *Painter* se encarga del control de los componentes en la interfaz de usuario, en ella se incluyen los elementos de la ventana como son los botones, las cajas de texto, las etiquetas y demás, de la biblioteca Qt.

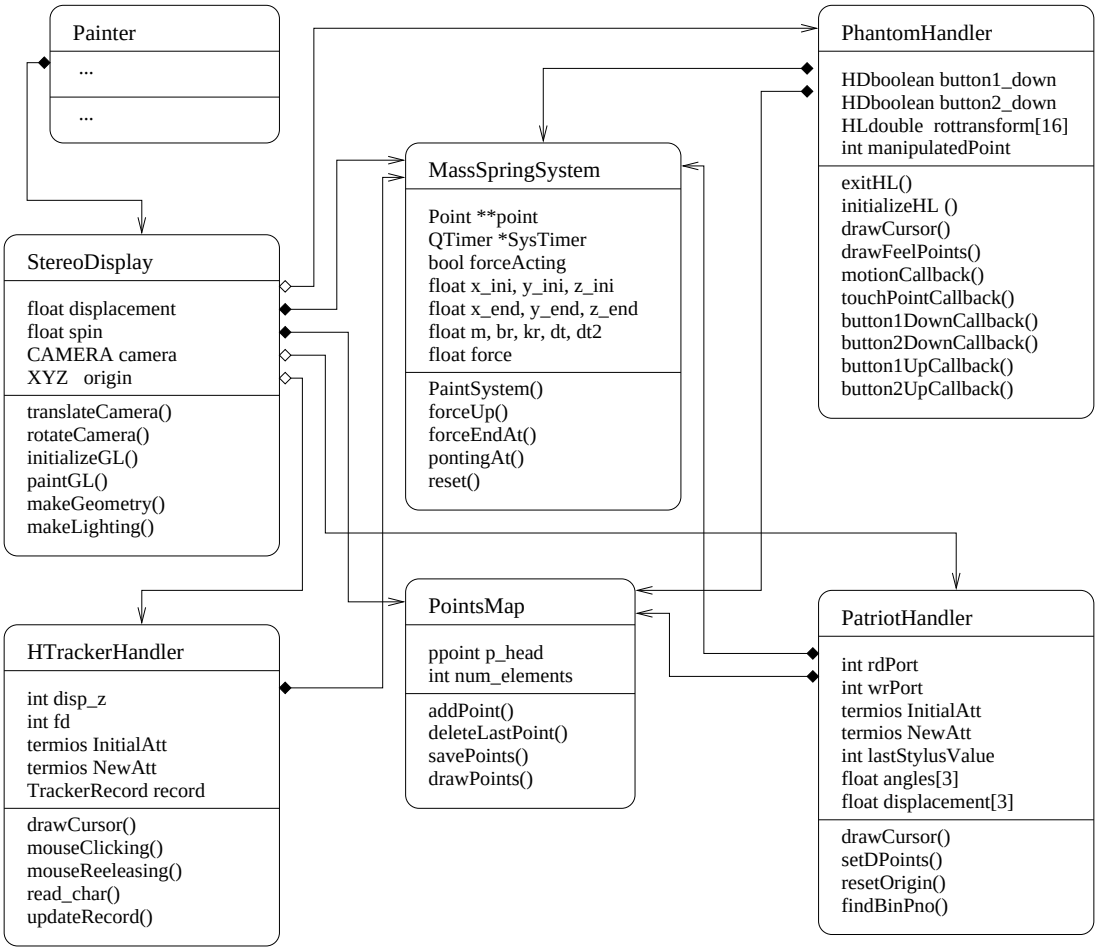

Figura 3.12: Diagrama de clases de la aplicación.

La clase *StereoDisplay* forma parte de la clase *Painter*, se encarga de gestionar el despliegue gráfico del objeto deformable, con o sin visión activa, de acuerdo los parámetros obtenidos de la malla de cuadriláteros y su sistema masa-resorte-amortiguador; es decir, su objetivo es mostrar los resultados de la simulación del objeto deformable y su interacción con los dispositivos manipuladores.

La clase *MassSpringSystem* representa matemáticamente al objeto deformable basándose en una malla de cuadriláteros con un sistema masa-resorte-amortiguador, resuelve la deformación de la malla en cada uno de sus nodos para cada incremento en el tiempo t y define las constantes del sistema.

Las clases *PhantomHandler*, *PatriotHandler* y *HTrackerHandler* tienen como objetivo comunicarse con los dispositivos manipuladores Phantom Omni, Patriot y Head Tracker respectivamente.

3.4. Descripción general de la interfaz

La interfaz desarrollada permite modificar: el estado de los dispositivos manipuladores (activado/desactivado), la visualización del objeto deformable en el escenario y las propiedades del sistema masa-resorte de la malla. Así mismo, en el área de despliegue gráfico permite visualizar un apuntador virtual, el movimiento de la pluma o del ratón de los dispositivos utilizados se convierte en el mismo movimiento escalado dentro del escenario virtual, en el cual, está dibujada la malla de cuadriláteros; cuando dicho apuntador se encuentra sobre un nodo y se le aplica una fuerza, la malla se deformará de acuerdo a los parámetros asignados. En conjunto, todo el escenario se puede apreciar en tres dimensiones para tener una mejor perspectiva del espacio.

La interfaz de usuario se muestra en dos ventanas como se aprecia en la Figura 3.13. La ventana de herramientas, cuenta con tres secciones principales (Figura 3.14) enfocadas, cada una, a facilitar la interacción con la aplicación.

De a cuerdo con la numeración utilizada en la Figura 3.13 y en la Figura 3.14, se puede observar que la interfaz de usuario de la aplicación desarrollada está distribuida en doce áreas específicas, que son:

1. Área de despliegue. Dentro de ésta área se dibuja el escenario tridimensional conteniendo el objetos deformable diseñado, en éste caso, el objeto a través de una malla de cuadriláteros, los ejes y el puntero.

2. Puntero. Éste puntero simula un movimiento en seis grados de libertad conforme se mueve la pluma o el ratón de los dispositivos. Permite intuir la posición dentro del espacio virtual, seleccionar algún nodo de la malla y aplicarle una fuerza.

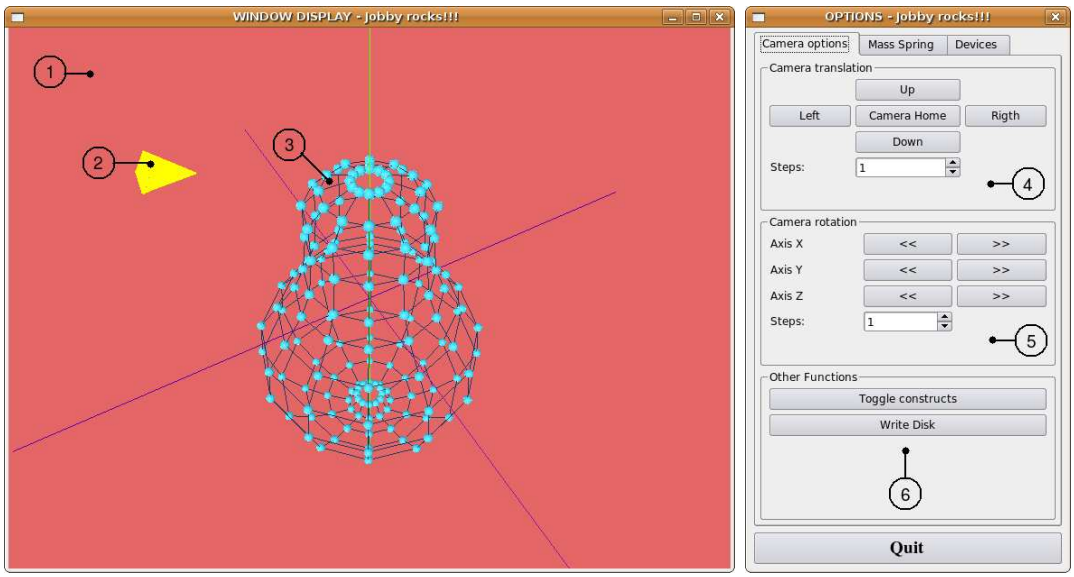

Figura 3.13: Interfaz de usuario de la aplicación.

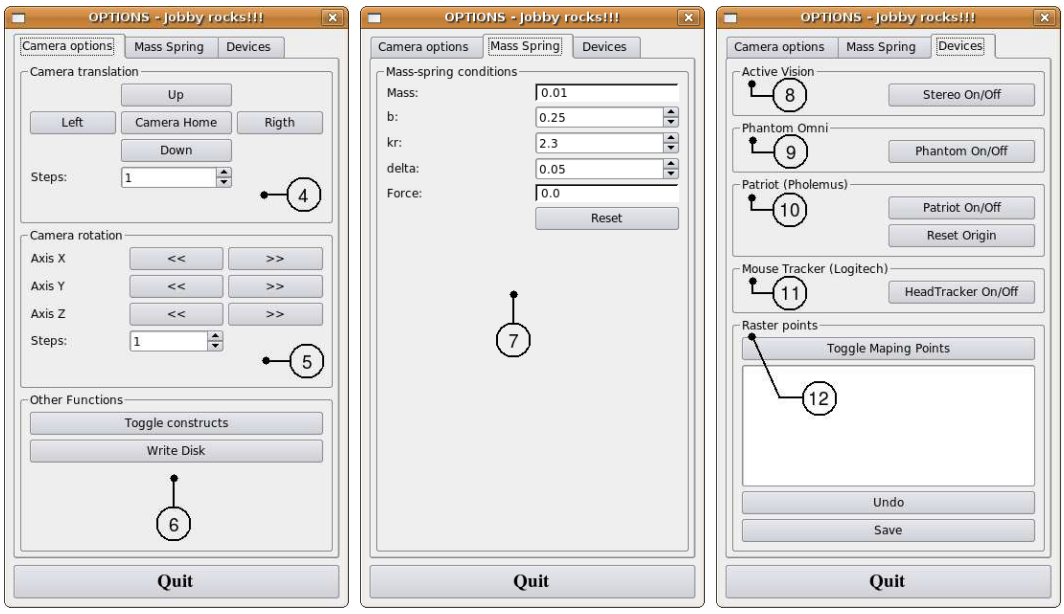

Figura 3.14: Pestañas contenidas en la ventana de herramientas de la aplicación.

3. Malla de cuadriláteros deformable. Es un conjunto de nodos con propiedades específicas, unidos por líneas simulando ser los resortes sin masa del sistema. Cuando se le aplica una fuerza a un nodo, la malla se deforma.

4. Translación de la cámara. Son los botones que permiten desplazar la cámara hacia arriba, abajo, la izquierda y la derecha del escenario, o simplemente llevar la cámara a su punto inicial. Todo desplazamiento se hace de a cuerdo al número de pasos indicados.

5. Rotación de la cámara. Son los botones que permiten la rotación de la cámara alrededor de los ejes x, y ó z el número de pasos indicados.

6. Otras funciones del escenario. El primer botón permite mostrar los ejes situados en el espacio virtual y el segundo permite guardar las imágenes que se muestran dentro del área de despliegue.

7. Condiciones del sistema masa-resorte. Son las cajas de texto que muestran las variables y las constantes aplicadas al sistema masa-resorte-amortiguador de la malla de cuadriláteros. Indican los valores de la masa de cada nodo de la malla y la fuerza aplicada al nodo seleccionado. Permiten modificar el coeficiente b del amortiguador, el kr del resorte y el Δt para la resolución con diferencias finitas del sistema. Y finalmente, también se pueden restablecer los parámetros iniciales del sistema.

8. En esta área se activa o desactiva el despliegue de los pares estereográficos necesarios para simular la visión estereoscópica utilizando el hardware para la visión activa.

9. Este botón permite activar o desactivar tanto el manejo del dispositivo háptico Phantom Omni.

10. Aquí se activar o desactivar el dispositivo Patriot y también se puede mover el origen de referencia del dispositivo.

11. En esta sección, se cambia el modo de trabajo para comenzar la digitalización de puntos; se listan los puntos enviados por el dispositivo en uso y se pueden guardar en un archivo llamado “Puntos.data” o borrar los puntos equivocados.

Todas estas funciones en conjunto completan la aplicación del prototipo de sistema propuesto, simulando el comportamiento específico de un objeto deformable y desplegando la resolución del sistema después y durante su interacción con los dispositivos.

Capítulo 4

Dispositivos

4.1. Dispositivo háptico Phantom Omni

La palabra háptico viene del griego *haptikos*, deriva de *haptesthai*, que significa asir, tocar o percibir [26]. Las ciencias hápticas tratan de incorporar el sentido del tacto y el control, a través de aplicaciones de computadora; para esto se ayudan de dispositivos mecánicos que permiten al usuario sentir y manipular los objetos configurados en un escenario virtual.

El dispositivo Phantom Omni (Figura 4.1) es un sistema computacional que permite retroalimentación de fuerzas, consiste de una punta en forma de pluma, integrada a un grupo de brazos mecánicos y una cabeza giratoria. Trabaja con seis grados de libertad, es decir, permite el desplazamiento en los tres ejes del plano cartesiano y las tres rotaciones sobre los mismos ejes.

El dispositivo cuenta con una extensa biblioteca que proporciona las herramientas suficientes para explotar la funcionalidad del dispositivo, dicha biblioteca está dividida en dos secciones: una es la de bajo nivel, HDAPI, la segunda es de alto nivel HLAPI.

HDAPI proporciona un control directo sobre el dispositivo y sobre la configuración en tiempo de ejecución de los controladores. Así mismo, permite generar directamente las fuerzas del reacción.

HLAPI está diseñada en base a los conceptos gráficos en tres dimensiones, específicamente para ser compatible con OpenGL, simplificando de ésta forma, la sincronización del desempeño háptico y los hilos para el despliegue gráfico.

En general, las herramientas que brindan estas dos bibliotecas se pueden combinar con el código de OpenGL existente para especificar la geometría del objeto deformable en el ambiente háptico. Se agregaron propiedades tales como fricción y rigidez de tal forma que el dispositivo permita ubicar los nodos y manipularlos virtualmente.

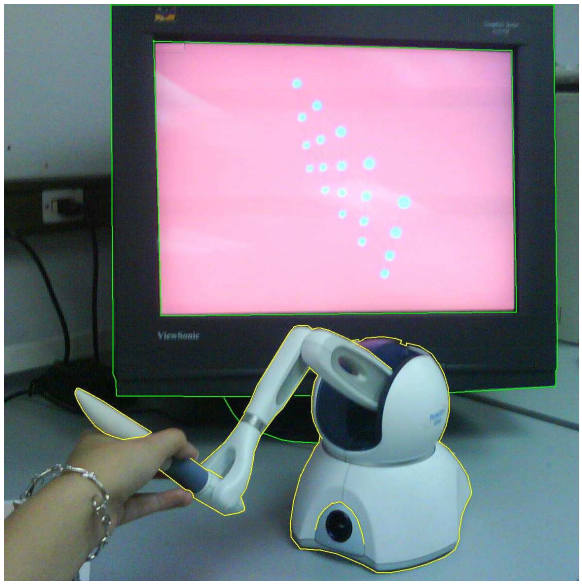

Figura 4.1: Dispositivo Phantom Omni interactuando con la aplicación.

La punta o pluma del Phantom Omni se representa dentro de la aplicación como un puntero a través del cuál, el usuario puede interactuar individualmente con cada nodo de la malla del objeto deformable: manipular la deformación de la malla y rotarla con respecto a los ejes del plano cartesiano.

4.1.1. Instalación

Para trabajar con la interacción háptica, se utilizó el siguiente hardware:

1. Dispositivo háptico Phantom Omni
2. Computadora de escritorio (PC)
   - *a*) Tarjeta madre con puerto FireWire 1394
   - *b*) Procesador de 32 bits
   - *c*) 100 MB de memoria libre
   - *d*) 256 MB en memoria RAM
   - *e*) Sistema Operativo Linux (Red Hat(R), Fedora(TM) Core 1, 2, 3 o SuSE 9.1, 9.2, 9.3)

Antes de comenzar con la programación de la aplicación, se deben tener instalados los controladores del dispositivo. Para instalarlos, es necesario contar con un compilador `gcc 3.x` e instalar el paquete RPM contenido en el disco de instalación:

```
rpm -ivn PHANTOM Device Drivers-4-2-x.i686.rpm
```

Posteriormente, se ha de activar la licencia a través del archivo "license.lic", que debe estar disponible dentro del mismo disco de instalación. La ruta se debe indicar usando una variable de entorno como sigue:

```
export OH_SDK_LICENSE_PATH=$/path/license.lic
echo $OH_SDK_LICENSE_PATH
```

Cada que se requiera usar alguna aplicación en la que intervenga el dispositivo Phantom, es preciso levantar el módulo 1394 para el puerto FireWire IEEE-1394:

```
/sbin/moprobe raw1394
```

Una vez que la instalación ha sido completada, se procede a la configuración del dispositivo. Para esto, es necesario abrir la ventana de configuración de la siguiente forma:

```
/usr/sbin/PHANToMConfiguration
```

La ventana de configuración solicitará el tipo de hardware que en este caso por defecto es el PHANToM, de modelo Omni. Así mismo, debe mostrar el puerto (FireWire 1394) y el número de licencia, si no se ha indicado correctamente la licencia o no se cuenta con una, el dispositivo no funcionará [27].

Una vez configurado el dispositivo, se debe reiniciar la computadora y probar que la configuración haya sido adecuada, por lo que la siguiente aplicación sirve como apoyo:

```
/usr/sbin/PHANToMTest
```

Ésta última es una guía que lleva paso a paso a completar el proceso de prueba.

Es importante señalar que para algunos de los procedimientos anteriores, es necesario contar con privilegios de administrador o encontrarse en modo superusuario, de otra forma no se tendrán los permisos suficientes para la configuración.

4.1.2. Metodología de programación

El código que se muestra a continuación, es un conjunto rutinas para el manejo de fuerzas y del dispositivo en general, permiten definir la funcionalidad del dispositivo, activar y desactivar algunas propiedades, así como consultar otras. El conocimiento sobre el funcionamiento de cada sentencia utilizada, fue consultado en la documentación [28] y [29], así como en los ejemplos que ofrece el kit del Phantom Omni.

Entrando en la descripción de la programación para la interacción con el dispositivo háptico Phantom, es indispensable mencionar las bibliotecas básicas que se deben incluir son:

```
#include <HD/hd.h>
#include <HL/hl.h>
#include <HLU/hlu.h>
#include <HDU/hdu.h>
```

Para dar inicio al trabajo con el Phantom, se ha de inicializar el dispositivo y crear el contexto de trabajo. El contexto háptico, almacena el conjunto actual de primitivas que se deben de tomar en cuenta para el dibujado.

```
HHD hHD = hdInitDevice (HD_DEFAULT_DEVICE);
HHLRC hHLRC = hlCreateContext (hHD);
hlMakeCurrent (hHLRC);
```

Dentro de la inicialización del trabajo del dispositivo, se deben definir los eventos que disparará el objeto virtual, de tal forma que se pueda saber con cuál de esos objetos se estará trabajando y cuáles son las acciones que dispara el Phantom.

```
HLuint núm_figuras = hlGenShapes (número_de_objetos);
HLuint Id = núm_figuras + identificador_del_objeto;

HlAddEventCallback (HL_EVENT_1BUTTONDOWN, Id, HL_CLIENT_THREAD,
            &button1DownCallback, this);
HlAddEventCallback (HL_EVENT_2BUTTONDOWN, HL_OBJECT_ANY,
            HL_COLLISION_THREAD, &button2DownCallback, this);
HlAddEventCallback (HL_EVENT_1BUTTONUP, HL_OBJECT_ANY,
            HL_COLLISION_THREAD, &button1UpCallback, this);
HlAddEventCallback (HL_EVENT_2BUTTONUP, HL_OBJECT_ANY,
            HL_COLLISION_THREAD, &button2UpCallback, this);
HlAddEventCallback (HL_EVENT_MOTION, HL_OBJECT_ANY,
            HL_CLIENT_THREAD, &motionCallback, this);
```

Por otro lado, también se debe inicializar el efecto que tendrá el ambiente virtual, en este caso desea un poco de fricción en el ambiente.

```
hlEffectd (HL_EFFECT_PROPERTY_GAIN, 0.4);
hlEffectd (HL_EFFECT_PROPERTY_MAGNITUDE, 0.1);
hlStartEffect (HL_EFFECT_FRICTION, numero_efecto);
```

Para concluir con el uso del Phantom y salir de la aplicación, primero es necesario detener los efectos del ambiente del dispositivo y quitar el registro de los eventos para los objetos virtuales que se habían inicializado. Después, se deben eliminar las figuras registradas.

```
hlStopEffect (numero_efecto);
hlDeleteEffects (numero_efecto, 1);

HLuint Id = núm_figuras + identificador_del_objeto;

hlRemoveEventCallback (HL_EVENT_1BUTTONDOWN, Id, HL_CLIENT_THREAD,
```

```
                &button1DownCallback);
hlRemoveEventCallback (HL_EVENT_2BUTTONDOWN, HL_OBJECT_ANY,
                HL_COLLISION_THREAD, &button2DownCallback);
hlRemoveEventCallback (HL_EVENT_1BUTTONUP, HL_OBJECT_ANY,
                HL_COLLISION_THREAD, &button1UpCallback);
hlRemoveEventCallback (HL_EVENT_2BUTTONUP, HL_OBJECT_ANY,
                HL_COLLISION_THREAD, &button2UpCallback);
hlRemoveEventCallback (HL_EVENT_MOTION, HL_OBJECT_ANY,
                HL_COLLISION_THREAD, &motionCallback);

hlDeleteShapes (núm_figuras, número_de_objetos);
```

Finalmente, se libera el dispositivo inmediatamente después de liberar el contexto de dibujado háptico, esto se hace de la siguiente forma:

```
hlMakeCurrent (NULL);
hlDeleteContext (hHLRC);
hdDisableDevice (hHD);
```

El puntero, dentro del prototipo desarrollado, es parte importante de la interacción del dispositivo Phantom con la malla de cuadriláteros. Básicamente se define utilizando primitivas de OpenGL combinadas con primitivas del contexto háptico, esto se debe a que el desplegado del puntero debe responder al movimiento del dispositivo.

Se indican los grupos de variables de estado que se deben salvar en la pila de atributos, se genera con código OpenGL la geometría del puntero y se ubica en la posición y ángulo correspondientes al de la pluma del Phantom.

```
glPushAttrib(GL_CURRENT_BIT | GL_ENABLE_BIT | GL_LIGHTING_BIT);
    GLuint punteroDisplayList = glGenLists (1);
    GLUquadricObj *puntero = gluNewQuadric ( );
    glNewList (punteroDisplayList, GL_COMPILE);
        gluCylinder(puntero, 0.0, 2.0, 6.0, 15, 15);
        glTranslated(0.0, 0.0, 6.0);
        gluCylinder(puntero, 2.0, 0.0, 0.7, 15, 15);
    glEndList ( );
    gluDeleteQuadric (puntero);

    glPushMatrix  ( );
        HLdouble punterotransform[16];
        hlGetDoublev (HL_PROXY_TRANSFORM, punterotransform);
        glMultMatrixd (punterotransform);

        glColor3f (Yellow[0], Yellow[1], Yellow[2]);
        glCallList (punteroDisplayList);
    glPopMatrix ( );
glPopAttrib ( );
```

Para el dibujado háptico, es indispensable hacer las declaraciones tanto de los objetos contenidos, como del de comportamiento del ambiente dentro del marco háptico:

```
hlBeginFrame ( );
        // declaraciones
hlEndFrame ( );
```

En el caso del dibujado háptico de la malla, se hicieron los ajustes que permiten sincronizar el dibujado para el dispositivo con la visión en perspectiva que se tiene dentro de la clase *StereoDisplay*.

```
hlMatrixMode (HL_TOUCHWORKSPACE);
hlLoadIdentity ( );
hlWorkspace (left, bottom, back, right, top, front);
hlOrtho (-1000.0, 1000.0, 0.0, 1000.0, -1000.0, 1000.0);
```

Cada punto de la malla puede sentirse a través del dispositivo, para que esto funcione, se especifica las siguientes propiedades para cada uno:

```
hlHinti (HL_SHAPE_FEEDBACK_BUFFER_VERTICES, 5);
hlBeginShape (HL_SHAPE_FEEDBACK_BUFFER, núm_figuras + id_del_objeto));

        hlTouchModel (HL_CONSTRAINT);
        hlTouchModelf (HL_SNAP_DISTANCE, 3.5);
        hlMaterialf (HL_FRONT_AND_BACK, HL_STIFFNESS, 0.5);
        hlMaterialf (HL_FRONT_AND_BACK, HL_DAMPING, 0.0);
        hlMaterialf (HL_FRONT_AND_BACK, HL_STATIC_FRICTION, 0.0);
        hlMaterialf (HL_FRONT_AND_BACK, HL_DYNAMIC_FRICTION, 0.0);
```

Además, se realiza el dibujado de un punto en el espacio, correspondiente a cada nodo de la malla, el cual, contendrá las propiedades anteriormente especificadas:

```
        glMultMatrixd (transformación);
        glScaled (escala, escala, escala);

        glBegin(GL_POINTS);
                glVertex3f (posición_x, posición_y, posición_z);
        glEnd ( );
hlEndShape ( );
```

Es importante asegurarse que dentro del contexto háptico, se disparen los eventos para cada acción del dispositivo, así como los que se registraron para cada objeto del contexto. Para esto se debe tener la primitiva:

```
hlCheckEvents ( );
```

La implementación de las llamadas a los eventos disparados por el dispositivo, se hace de forma general de la siguiente manera:

```
void HLCALLBACK PhantomHandler::Nombre_de_la_llamada (HLenum evento,
             HLuint objecto, HLenum hilo, HLcache *cache, void *puntero) {
        PhantomHandler *PtThis = static_cast<PhantomHandler *> (puntero);
        PtThis->métodos ( );
        // Demás declaraciones
}
```

Un ejemplo de la implementación de la llamada que se dispara con el movimiento de la pluma del Phantom, es la siguiente:

```
void HLCALLBACK PhantomHandler::motionCallback  (HLenum event, HLuint object,
              HLenum thread, HLcache *cache, void *userdata) {
        PhantomHandler *Phant = static_cast<PhantomHandler *>(userdata);

        hduVector3Dd position;
        hlCacheGetDoublev (cache, HL_PROXY_POSITION, position);
        Phant->Mesh->forceEndAt (position[0], position[1], position[2]);
}
```

Es indispensable tener un objeto de la clase *StereoDisplay* que contenga un objeto de la clase *PhantomHandler*, ya que el primero se encarga del despliegue del escenario y por tanto, utiliza las llamadas a los métodos encargados del despliegue háptico y del cursor, contenidos en la implementación del dispositivo háptico. De la misma forma, debe contener un objeto del tipo *MassSpringSystem* para poder trabajar con la malla de cuadriláteros.

4.2. Dispositivo magnético Patriot

El dispositivo Patriot es un sistema de seguimiento que funciona utilizando de un campo electromagnético de baja frecuencia, trabaja con seis grados de libertad y se comunica con las aplicaciones de la computadora a través de una estructura de comandos y respuestas enviados por el puerto USB o el puerto serial RS-232. La comunicación se compone de comandos ASCII y respuestas en formato binario o ASCII [30].

Éste dispositivo cuenta con tres elementos principales para su desempeño que se muestran en la Figura. 4.2):

- Fuente magnética. Es un cubo de 5 cm por lado que genera un campo magnético que será medido por el sensor. Sus ejes X, Y y Z son el origen de referencia inicial para las mediciones correspondientes, por lo que debe fijarse en una posición definitiva.

- Sensor. Se encarga de medir el campo magnético generado por la fuente. Se utiliza para identificar la posición y la orientación del objeto al que se encuentra adherido de acuerdo a su marco de referencia. En éste caso, se utilizó el sensor *Stylus*, que es un tipo de pluma que alberga el sensor.

- Unidad electrónica del sistema. Es una unidad independiente que contiene los conectores y controladores de entrada y salida necesarios para dar soporte a los sensores, la fuente magnética, el puerto USB y el puerto RS-232 por el que se comunica a la computadora..

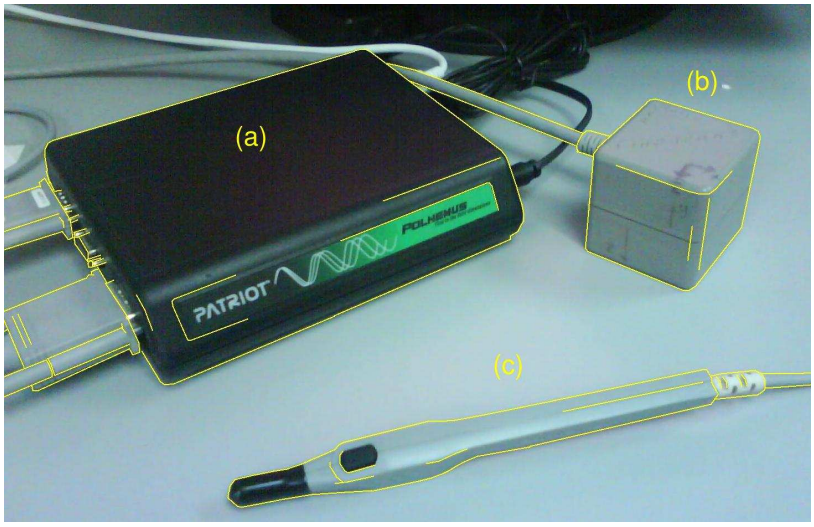

Figura 4.2: Dispositivo Patriot: (a) la unidad electrónica del sistema, (b) la fuente magnética y (c) el sensor.

El sistema inicial de ejes en el medio ambiente se considera congruente al del dispositivo Patriot; la alineación física o matemática del sensor se realiza dentro de éste mismo marco de referencia en el que se mide la posición y orientación del sensor.

El dispositivo puede interpretar la posición del sensor tanto en centímetros como en pulgadas, según sea configurado. Mientras que los ángulos de orientación del sensor: acimut, elevación y rotación propia, están definidos dentro del marco de coordenadas de Euler. El acimut se refiere a la coordenada de orientación medida en el plano horizontal, donde el incremento en su ángulo se da en el sentido de las manecillas del reloj; es decir, es una rotación al rededor del eje Z o del eje vertical. La elevación es la coordenada de orientación medida en el plano vertical cuyo aumento en el ángulo se da hacia arriba de la horizontal. Finalmente, la rotación propia es la coordenada de orientación relativa al eje perpendicular a acimut-elevación, donde el aumento de éste ángulo es en el sentido de las manecillas del reloj conforme a su vista por detrás [30].

En la figura Figura 4.3 se muestra la interpretación gráfica de los ángulos de Euler obtenidos como medición de la orientación del sensor. Los ejes ortogonales x, y, z y X, Y, Z, son dos marcos independientes de coordenadas en tres dimensiones. La triada x, y, z representa al estado del sensor en el marco actual, mientras que la triada X, Y, Z representa el marco de referencia contra la cual la orientación relativa de la estructura del sensor se mide. Por definición, X, Y y Z también representan el marco cero de referencia del sensor.

Los ángulos de Euler: acimut, elevación y rotación propia, son designados por ψ,

θ y ϕ, representan una secuencia de rotaciones que definen la orientación del sensor con respecto al su marco de orientación cero; ésta secuencia es: primero una rotación acimut seguida de una rotación de elevación concluyendo con una rotación propia.

El ángulo de acimut, ψ, se define en la figura como una rotación en los ejes de referencia X y Y alrededor del eje de referencia Z. Los ejes de transición etiquetados como X' y Y' representan la nueva orientación de X y Y después de la rotación azimut.

El ángulo de elevación, θ, se define como una rotación en el eje de referencia Z y el eje de transición X', al rededor del eje de transición Y'. Ahora el eje de transición etiquetado como Z', representa la nueva orientación del eje de referencia Z después de la rotación de elevación. El actual eje x del marco de referencia del sensor, representa la orientación del eje de transición X' después de ésta última operación.

Por último, el ángulo de rotación propia, ϕ, se define como una rotación en los ejes de transición Y' y Z' alrededor del eje x del marco del sensor. De esta forma se obtienen los ejes y y z del actual marco del sensor y representan la orientación de los ejes de transición Y' y Z' después de ésta última rotación.

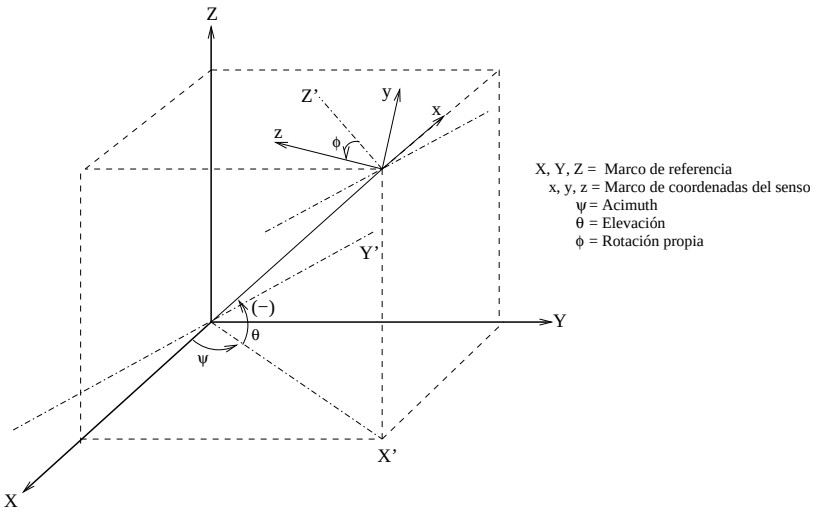

Figura 4.3: Marco de orientación utilizando ángulos de Euler.

En el caso en que los tres ángulos ψ, θ y ϕ son iguales a cero, el sistema de coordenadas quedaría como en la Figura 4.3, que si lo sobreponemos en la pantalla de la computadora, el eje X quedaría fuera de la misma de forma perpendicular.

4.2.1. Instalación

Existen dos componentes para el controlador (*driver*) del dispositivo Patriot, el primero es el cargador del *firmware* y el segundo es el módulo del núcleo USB de Linux que se comunica con el sistema Patriot. Se inicia haciendo la suposición de que el módulo USB está funcionando correctamente en la computadora.

La instalación de éste dispositivo se realizó en una computadora de escritorio (PC) con procesador de 64 bits, bajo ambiente Linux.

Firmware

El *firmware* es cargado por el programa `fxload`, el cual, se encuentra normalmente en la mayoría de las distribuciones de Linux; sin embargo, también se encuentra disponible a través de la página de Internet *http://prdownloads.sourceforge.net/linux-hotplug/*.

El primer paso es localizar el archivo "usb.usermap" (comúnmente ubicado en `/etc/hotplug/usb.usermap` en distribuciones Redhat) y añadirle la siguiente línea:

```
patriot 0x0003 0x0f44 0xef11 0x0000 0x0000 0x00 0x00 0x00 0x00 0x00 0x00 0x00
```

Posteriormente, dentro de la carpeta `Linux` incluida en el disco de instalación del dispositivo Patriot, se encuentra un el archivo "patriot" que se debe copiar a la dirección `/etc/hotplug/usb/`, es un *script* que asume que los archivos "a3load.hex" y "PatriotUSB.hex" también contenidos en el mismo disco, se han colocado en `/usr/share/usb`; sin embargo, se pueden colocar donde se desee y actualizar el archivo "patriot". También, hay que tomar en cuenta que se da por supuesto que `fxload` se encuentra ubicado en `/sbin/`.

Con éstos pasos se busca que el *firmware* se cargue cuando el dispositivo Patriot se conecte. Los mensajes de error se pueden buscar en `/var/log/messages`.

Controlador USB

En el momento en que el controlador genérico USB de Linux se utiliza, funciona, aunque no de la forma ideal. Entonces, se prosigue con las siguientes líneas:

Para un núcleo 2.4 de Linux, se puede ejecutar la siguiente instrucción en línea de comandos o se puede agregar en `/etc/rc.d/rc.local` para que se ejecute al inicio:

```
/sbin/insmod usbserial vendor=0x0f44 product=0xef12
```

Lo anterior, se hace asumiendo que `usbserial` ha sido compilado como un módulo. Sin embargo, bajo el núcleo 2.6 de Linux, se debe utilizar un módulo diferente.

Finalmente, en el archivo `/etc/modules.conf` se debe agregar la línea:

```
options visor vendor=0xf44 product=0xef12
```

y en el archivo `/etc/rc.d/rc.local`, las líneas:

```
modprobe usbserial
modprobe visor
```

Estas últimas, también se pueden ejecutar en línea de comando.

Una vez concluidos estos pasos, ya se puede conectar el dispositivo Patriot y se cargará el *firmware* que se re-conectará y se convertirá en un puerto serial. Ahora la comunicación hacia el dispositivo Patriot se puede hacer directamente por medio del dispositivo `/dev/usb/ttyUSB0`.

4.2.2. Metodología de programación

La comunicación entre el dispositivo Patriot y la aplicación desarrollada se da a través del puerto `/dev/usb/ttyUSB0` utilizando una interfaz de comandos definida. Recordando que la estructura de comandos y respuestas, se compone de comandos ASCII y respuestas en formato binario o ASCII.

En la mayoría de los casos, el formato de los comandos es el siguiente:

```
COMANDO[ESTACIÓN][PARÁMETROS=PARÁMETRO1,PARÁMETRO2,...] < >
```

Por ejemplo, si se desea ejecutar el comando `G` para el sensor conectado a la estación `1`, y sus parámetros son 0, 180 y 270, entonces el comando quedaría de la siguiente forma:

```
G1,0,180,270<>
```

Cuando se desea asignar un parámetro de tipo flotante, se puede usar uno de dos formatos: por ejemplo, para el número 3.0 se especificaría como $3, 3., 3.0$ o $3.0E+00$.

Para dar inicio a la comunicación con el dispositivo Patriot, se definen dos variables de comunicación con el puerto:

```
int rdPort = open( "/dev/usb/ttyUSB0", O_RDONLY | O_NDELAY );
int wrPort = open( "/dev/usb/ttyUSB0", O_WRONLY);
if ( (rdPort == -1) || (wrPort == -1) ) {
        fprintf( stderr, "Error connecting to tracker.");
        exit ( -1 );
}
```

Para definir el modo de trabajo, se consideran tres aspectos iniciales:

- Unidades en centímetros. Su comando es `U[units]` , donde cero son pulgadas y uno son centímetros.

- Poner el seguidor en modo binario. El comando es `U[format]`, donde cero es ASCII y uno es binario.
- Indicar el formato de entrada de datos. Tiene como formato `O[station], [[p1], [p2], ...]`, donde uno es el retorno de carro ASCII, dos indica las coordenadas X, Y y Z, cuatro los ángulos de orientación de Euler ψ, θ y ϕ y diez la bandera de la pluma *Stylus*.

```
/* Put system units in centimeters */
write( wrPort, "u1\r", 3 );
/* Put tracker into binary mode */
write( wrPort, "f1\r", 3 );
/* Put format for input mode */
write( wrPort, "o1,2,4,6,10\r", 12 );

write( wrPort, "\x12\x31\r", 3 );
write( wrPort, "a1,20,20,-20,20,0,-20,20,20,-40\r", 32 );
```

Para solicitar la lectura de la posición y ángulos del sensor y obtener los datos en el formato anteriormente especificado, se utiliza el comando `P`, que no requiere de parámetros:

```
int  br = 0;
char buf[2000];
memset( buf, 0, 2000 );

/* Request data */
write( wrPort, "p", 1 );
br = read( rdPort, buf, 2000 );

/* Checking for proper format: PA for Patriot or LY for Liberty */
if ( strncmp(buf, "LY", 2) && strncmp(buf, "PA", 2) ){
        fprintf( stderr, "Corrupted data received\n" );
}

/* Header is first 8 bytes */
float *pData = (float*)(buf + 8);
int *pFlag = (int*)(buf + 32);

angles[0] = pData[3];
angles[1] = pData[4];
angles[2] = pData[5];

displacement[0] =  pData[2];
displacement[1] =  pData[1];
displacement[2] = -pData[0];

lastStylusValue = pFlag[0];
```

En el caso que se requiera cambiar el origen predefinido del dispositivo Patriot o su alineación, se puede utilizar el comando: `A[station], [[Ox],[Oy],[Oz],[Xx],[Xy],[Xz],`

`[Yx],[Yy],[Yz]]`; del cuál las coordenadas `Ox`, `Oy` y `Oz` se refieren al nuevo origen de referencia, las coordenadas `Xx`, `Xy` y `Xz` son la dirección del eje positivo X y las coordenadas `Yx`, `Yy` y `Yz` son las del eje positivo Y.

```
int  br = 0;
char buf[2000];
char Comando[100];
memset( buf, 0, 2000 );

/* Request data */
write( wrPort, "p", 1 );
br = read( rdPort, buf, 2000 );

/* Checking for proper format: PA for Patriot or LY for Liberty */
if ( strncmp(buf, "LY", 2) && strncmp(buf, "PA", 2) ){
        return;
}

/* Header is first 8 bytes */
float *pData = (float*)(buf + 8);
int *pFlag = (int*)(buf + 32);

/* Initializing zero in axis */
memset( Comando, 0, 100);
sprintf ( Comando, "a1,%g,%g,%g,%g,%g,%g,%g,%g,%g\r", pData[0], pData[1],
                              pData[2], 100.0, 0.0, 0.0, 0.0, 100.0, 0.0 );
int dimCom = strlen(Comando);
write( wrPort, Comando, dimCom );
```

Referente a la interacción con la malla deformable, una vez que se tienen las funciones necesarias para conocer la posición del puntero, entonces, se accede a los métodos del objeto de tipo *MassSpringSystem*, `Mesh`:

```
if( stylusPressed ) {
        Mesh->pointingAt( displacement[0]*scale, displacement[1]*scale,
                                                 displacement[2]*scale );
}
else
if{ manipulatinNode ) {
        Mesh->forceEndAt( displacement[0]*scale, displacement[1]*scale,
                                                 displacement[2]*scale );
}
else
if( !stylusPressed ) {
        Mesh->forceUp( );
}
```

Finalmente, para la representación gráfica del puntero referente a la pluma *Stylus*, que indicará el punto de interacción con la malla se hace de forma similar que con el dispositivo Phantom:

```
glPushAttrib(GL_CURRENT_BIT | GL_ENABLE_BIT | GL_LIGHTING_BIT);

        if ( !cursorDisplayList ) {
                cursorDisplayList = glGenLists( 1 );

                GLUquadricObj *cursor = gluNewQuadric( );
                glNewList( cursorDisplayList, GL_COMPILE );
                        gluCylinder(cursor, 0.0, 2.0, 6.0, 4, 4);
                        glTranslated(0.0, 0.0, 6.0);
                        gluCylinder(cursor, 2.0, 0.0, 0.7, 4, 4);
                glEndList( );
                gluDeleteQuadric( cursor );
        }

        glPushMatrix ( );

                /* Apply the local transform of the haptic device proxy */
                glTranslatef( displacement[0], displacement[1],
                                                displacement[2]);

                float c = cos(angles[0]*DTOR);
                glRotatef( -angles[2], sin(angles[1]*DTOR)*c,
                           -sin(angles[0]*DTOR), cos(angles[1]*DTOR)*c );
                glRotatef( angles[1], 0.0, 1.0, 0.0);
                glRotatef( angles[0], 1.0, 0.0, 0.0);

                glColor3f( SteelBlue[0], SteelBlue[1], SteelBlue[2] );
                glCallList( cursorDisplayList );
        glPopMatrix( );

glPopAttrib( );
```

Para realizar del despliegue del funcionamiento de este objeto, es necesario contar con un objeto de la clase *StereoDisplay* que lo contenga así como tener como miembro a un objeto del tipo *MassSpringSystem* para poder trabajar con la malla de cuadriláteros.

4.3. Dispositivo ultrasónico Head Tracker

El dispositivo Head Tracker, con seis grados de libertad, permite obtener información espacial. Utiliza un transmisor estacionario de señales ultrasónicas que sigue el movimiento del receptor. El receptor se mueve en sus tres dimensiones dentro del área activa localizada al frente del transmisor, recibe la señal ultrasónica y la transmite a la unidad de control. La unidad de control convierte las señales en datos de posición y orientación que se pueden procesar hacia la computadora [31].

Los componentes básicos del sistema Head Tracker, incluyendo los tres ya mencionados son: transmisor, receptor, unidad de control, fuente de energía y cables de computadora (Figura 4.4):

- Transmisor. Está compuesto de tres altavoces que envían señales ultrasónicas de 23 KHz. Estas señales localizan la posición y la orientación del receptor.

- Receptor. Es un triángulo con micrófonos en cada equina; estos muestrean las señales ultrasónicas recibidas desde el transmisor, a una frecuencia mayor a 50 muestras por segundo. Posteriormente, transmite estas señales a la unidad de control de acuerdo a su posición y orientación.

- Unidad de Control. Es el núcleo del sistema Head Tracker, contiene un microprocesador, una interfaz de circuitos y un *firmware* que en conjunto se encargan de decodificar las señales del receptor y calcular su posición y orientación. Además, envía estos datos a la computadora a través de una comunicación serial.

- Fuente de energía. Se encarga de suplir con corriente DC a la unidad de control.

- Cable de computadora. Es un cable RS-232C que se conecta desde la unidad de control hacia el puerto de la computadora. Permite una tasa de transmisión de $19,200$ bps en el modo 3D.

Éste dispositivo trabaja en dos modos: 3D y 2D; para éste caso de estudio únicamente se utilizó la primera. La compañía Logitech también cuenta con un dispositivo similar al Head Tracker, el 3D Mouse, que además tiene incorporado un ratón con tres botones de funcionalidad programable. Tanto el receptor como el emisor del dispositivo Head Tracker cuentan con orificios para tornillo de tal forma que se puedan asir a cualquier objeto que sea necesario. Con el objetivo de simular el 3D Mouse, se utilizaron los orificios del receptor y se unió a un ratón de computadora (Figura 4.5).

El sistema de coordenadas que utiliza el sistema Head Tracker en modo 3D, son las coordenadas rectangulares del plano Cartesiano. La figura 4.6, muestra el sistema de ángulos y direcciones tridimensionales utilizadas el dispositivo. Las rotaciones se obtienen considerando el *sistema de la mano derecha*.

El origen tridimensional y los ejes de referencia tanto del transmisor como del receptor, se definen de acuerdo a la figura figura 4.7, en la que se tiene las siguientes definiciones [31]:

- X_T es un eje de referencia imaginario que cruza a través del centro de los altavoces izquierdo y derecho de la parte de abajo.

- Y_T es un eje de referencia imaginario que cruza a través del centro del altavoz superior y es perpendicular X_T.

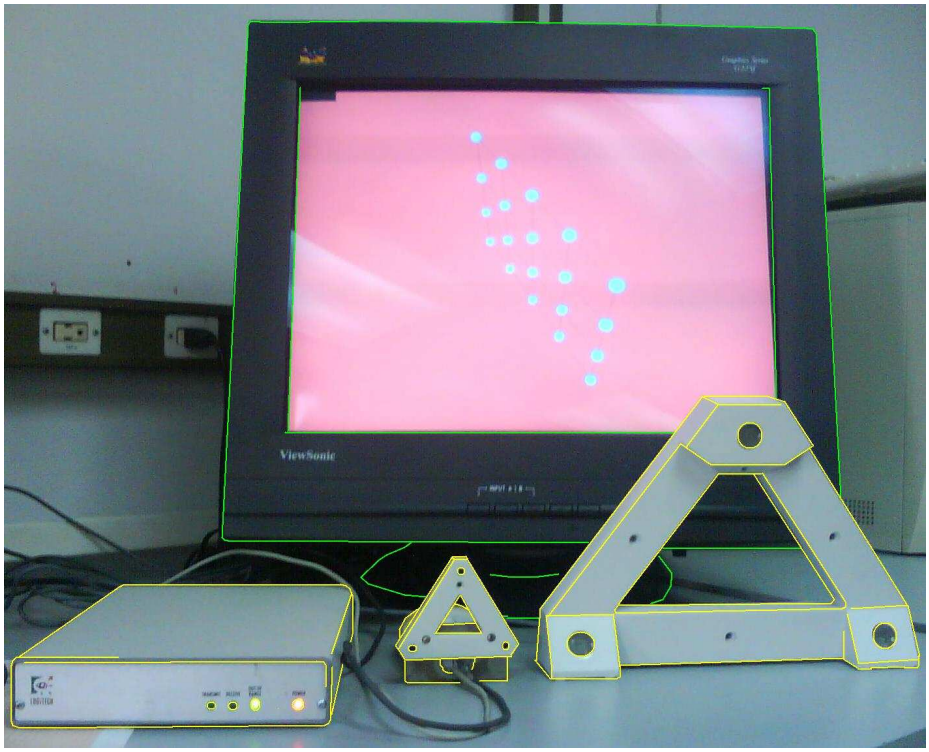

Figura 4.4: Dispositivo Head Tracker interactuando con la aplicación.

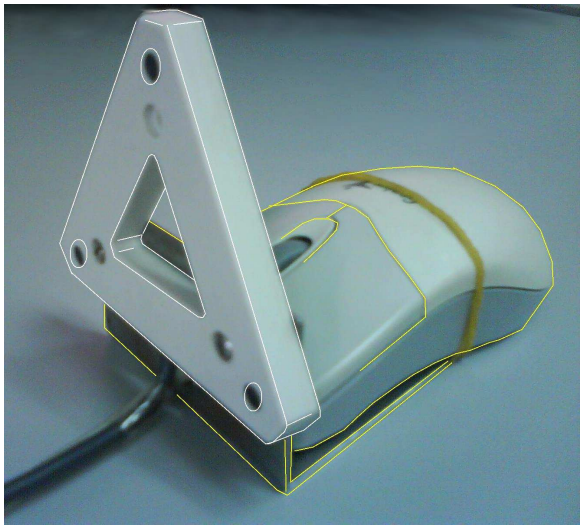

Figura 4.5: Receptor del dispositivo Head Tracker atornillado a un ratón común de computadora.

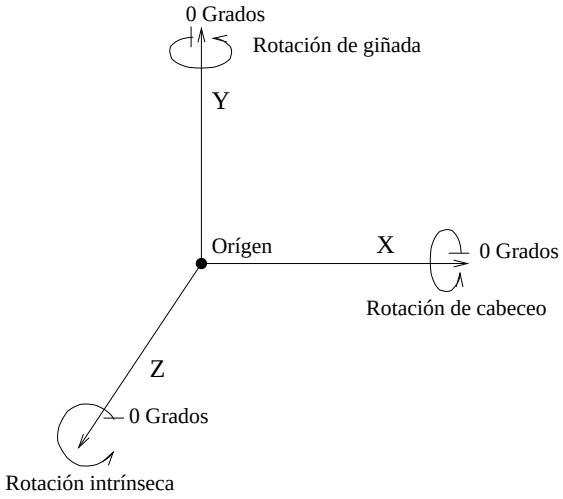

Figura 4.6: Ejes tridimensionales y rotaciones positivas.

- Z_T es un eje de referencia imaginario perpendicular a X_T y Y_T.
- El origen del transmisor es el punto donde se intersecan los ejes X_T, Y_T y Z_T.
- X_R es un eje de referencia imaginario que cruza a través del centro de los micrófonos izquierdo y derecho de la parte de abajo.
- Y_R es un eje de referencia imaginario que cruza a través del centro del micrófono superior y es perpendicular X_R.
- Z_R es un eje de referencia imaginario perpendicular a X_R y Y_R.
- El origen del transmisor es el punto donde se intersecan los ejes X_R, Y_R y Z_R.

La información de posición y orientación generada por la unidad de control, está basada en las definiciones anteriores:

- **Dato X**. Es la distancia del origen del receptor hacia la izquierda o derecha del origen del transmisor, a lo largo del eje X_T.
- **Dato Y**. Es la distancia del origen del receptor hacia arriba o hacia abajo del origen del transmisor menos 12 pulgadas de compensación, a lo largo del eje Y_T,
- **Dato Z**. Es la distancia del origen del receptor que se aleja del origen del transmisor menos 18 pulgadas de compensación, a lo largo del eje Z_T.
- **Rotación de cabeceo**. Del inglés *pitch*, es la orientación positiva o negativa al rededor del eje X_R.
- **Rotación de guiñada**. Del inglés *yaw*, es la orientación positiva o negativa al rededor del eje Y_R.

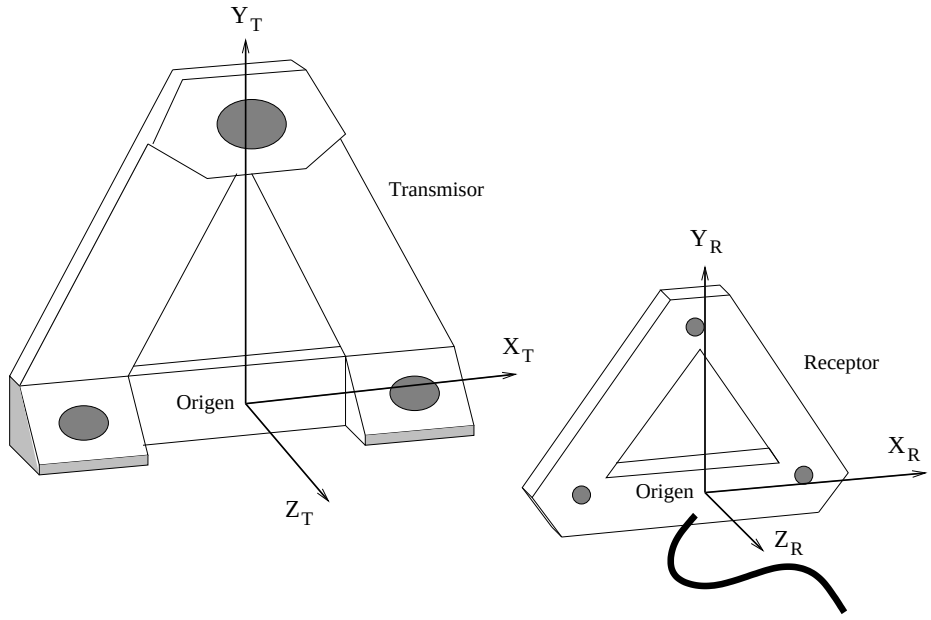

Figura 4.7: Origen y ejes de referencia del receptor y del transmisor.

- **Rotación intrínseca**. Del inglés *roll*, es la orientación positiva o negativa al rededor del eje Z_R.

Al trabajar con el dispositivo, se debe tomar en cuenta que la inclinación del emisor no debe ser mayor a 90° alrededor del eje vertical ni mayor a 90° alrededor del eje horizontal, respecto del plano del transmisor. También, si un objeto de coloca entre el transmisor y el receptor, se genera una interferencia en la comunicación. Así mismo, se debe considerar la forma cónica de área activa del dispositivo Head Tracker (Figura 4.8).

4.3.1. Metodología de programación

Los datos que reporta la unidad de control en modo 3D, tal como la posición y orientación se dan en un reporte de 16 bytes. Mientras que el los comandos enviados desde la computadora consisten de dos ó más caracteres precedidos siempre por un asterisco (*).

Se comienza con abrir el puerto de comunicación y configurarlo:

```
puerto = open(PUERTODEVICE, O_RDWR | O_NOCTTY );
if (fd < 0 ) {
        perror(PUERTODEVICE);
        exit();
}
```

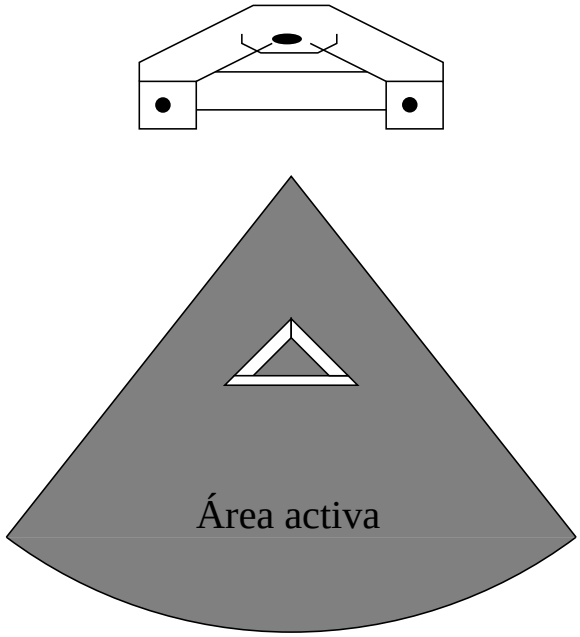

Figura 4.8: Área activa del dispositivo Head Tracker.

```
struct termios newtio;
newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;
newtio.c_iflag = IGNPAR;
newtio.c_oflag = 0; // Raw output
newtio.c_lflag = 0; // Set input mode
newtio.c_cc[VTIME] = 10; // Inter-character timer
newtio.c_cc[VMIN] = 0; // No blocking read

tcsetattr(fd, TCIOFLUSH, &newtio);
```

Posteriormente, se realiza la comprobación de la comunicación con el puerto, se reinicia la unidad de control, se solicita un reporte y se hace un diagnóstico. Si el reporte del diagnóstico, en sus primeros dos bytes entrega `bf` y `3f` respectivamente en hexadecimal, entonces la configuración se ha concluido correctamente.

```
int  res;
char data[2];

/* Begins comunication for configuration */
write (fd, "*D", 2);
usleep(1000000);

/* Reseting control unit */
write (fd, "*R", 2);
usleep(1000000);

/* Request diagnostics */
write (fd, "*\x5", 2);
```

```
usleep(1000000);

/** Reading diagnostics test **/
data[0] = (char)read_char( );
data[1] = (char)read_char( );
```

La lectura de los reportes se hace byte por byte a través de la función `read_char()`, que simplemente asegura que la lectura del byte sea diferente de cero.

```
unsigned char c;
if ( int r = read (fd, (void *)&c, 1) == 0 )
        fprintf( stderr, "ERROR: timeout!\n" );
else return c;
```

Para solicitar un reporte completo, se utiliza el comando `*d` y se hace la conversión de ángulos de Euler a ángulos absolutos en el plano cartesiano.

```
char buf[DATA_SIZE];
int ax, ay, az, arx, ary, arz;

/* Getting Data string */
write (fd, "*d", 2);

for (int i=0; i<DATA_SIZE; i++ ) buf[i] = read_char( );
r = buf[0] & 0x20;

/* Converting euler data to absolute */
ax = (buf[1] & 0x40) ? 0xFFE00000 : 0;
ax |= (long)(buf[1] & 0x7f) << 14;
ax |= (long)(buf[2] & 0x7f) << 7;
ax |= (buf[3] & 0x7f);

ay = (buf[4] & 0x40) ? 0xFFE00000 : 0;
ay |= (long)(buf[4] & 0x7f) << 14;
ay |= (long)(buf[5] & 0x7f) << 7;
ay |= (buf[6] & 0x7f);

az = (buf[7] & 0x40) ? 0xFFE00000 : 0;
az |= (long)(buf[7] & 0x7f) << 14;
az |= (long)(buf[8] & 0x7f) << 7;
az |= (buf[9] & 0x7f);

arx  = (buf[10] & 0x7f) << 7;
arx += (buf[11] & 0x7f);

ary  = (buf[12] & 0x7f) << 7;
ary += (buf[13] & 0x7f);

arz  = (buf[14] & 0x7f) << 7;
arz += (buf[15] & 0x7f);

DatoX = ((float) ax) / 1000.0;
DatoY = ((float) ay) / 1000.0;
```

```
DatoZ = ((float) az) / 1000.0;

Pitch = ((float) arx) / 40.0;
Yaw   = ((float) ary) / 40.0;
Roll  = ((float) arz) / 40.0;
```

Finalmente, para la representación gráfica del puntero que indicará el punto de interacción con la malla, se hace de forma similar que con los dispositivos anteriores:

```
glPushAttrib(GL_CURRENT_BIT | GL_ENABLE_BIT | GL_LIGHTING_BIT);

        if ( !cursorDisplayList ) {
                cursorDisplayList = glGenLists( 1 );

                GLUquadricObj *cursor = gluNewQuadric( );
                glNewList( cursorDisplayList, GL_COMPILE );
                        gluCylinder(cursor, 0.0, 2.0, 6.0, 4, 4);
                        glTranslated(0.0, 0.0, 6.0);
                        gluCylinder(cursor, 2.0, 0.0, 0.7, 4, 4);
                glEndList( );
                gluDeleteQuadric( cursor );
        }

        glPushMatrix ( );
                /* Apply the local position/rotation transforms */
                glTranslatef( DatoX, DatoY, DatoZ);

                glRotatef( Roll, 0.0, 0.0, 1.0 );
                glRotatef( Yaw,0.0, 1.0, 0.0 );
                glRotatef( Pitch, 1.0, 0.0, 0.0 );

                glCallList( cursorDisplayList );
        glPopMatrix( );

glPopAttrib( );
```

Por otro lado, los eventos específicos del ratón integrado al receptor, se disparan cuando éste dispositivo se encuentra activo. Se consideraron dos funcionalidades para los dos botones, la primera es la interacción con un nodo de la malla de cuadriláteros (botón izquierdo) y la segunda es la rotación de la malla (botón derecho). De acuerdo al estado del objeto *MassStringSystem* y los de el manejador del dispositivo, se utilizan sus métodos para interactuar juntos.

```
void HTrackerHandler::mouseClicking( ) {
        if ( isManipulating )
                Mesh->forceEndAt( DatoX, DatoY, DatoZ );
        else
                Mesh->pointingAt( DatoX, DatoY, DatoZ );
}

void HTrackerHandler::mouseReleasing( ) {
        if ( isManipulating ) Mesh->forceUp( );
}
```

En este caso, para este objeto encargado de comunicarse con el dispositivo Head Tracker, también se debe tener en cuenta que depende de la malla deformable predefinida y el despliegue de su comportamiento es realizado por la clase *StereoDisplay*.

4.4. Digitalización de puntos

Al presente prototipo de sistema también se le agregó la funcionalidad que permite digitalizar un punto espacial y de ésta forma poder almacenar una lista de puntos consecutivos. Con esto, se hicieron pruebas para la digitalización de una línea utilizando los dispositivos Phantom y Patriot. No se eligió el dispositivo Head Tracker debido a que no cuenta con una punta o referencia que permitiera tener mayor precisión en el momento de capturar su posición; pero aunque se lee podría acoplar uno, se debe tener en cuenta que no todos los puntos serían accesibles debido a que la comunicación del receptor con el emisor solamente es posible si sus planos XY se encuentran máximo a 90° el uno del otro.

Se digitalizaron los puntos de una línea de 25 cm utilizando como referencia una hoja milimétrica, cada punto con una separación de 0.5 cm. La lista de datos es guardada por el programa en un archivo llamado "puntos.data". Utilizando ésta lista de puntos tridimensionales como entrada para los programas `ls3dline.m` (*Least-squares line in 3 dimensions*) y `ajuste.oct` ver Apéndice A, se hizo el cálculo del error en cada punto y se ajustaron los puntos a la recta correspondiente.

Los resultados de las operaciones mencionadas en el párrafo anterior para el dispositivo Phantom se muestran en las figuras 4.9, 4.10, 4.11 y 4.12.

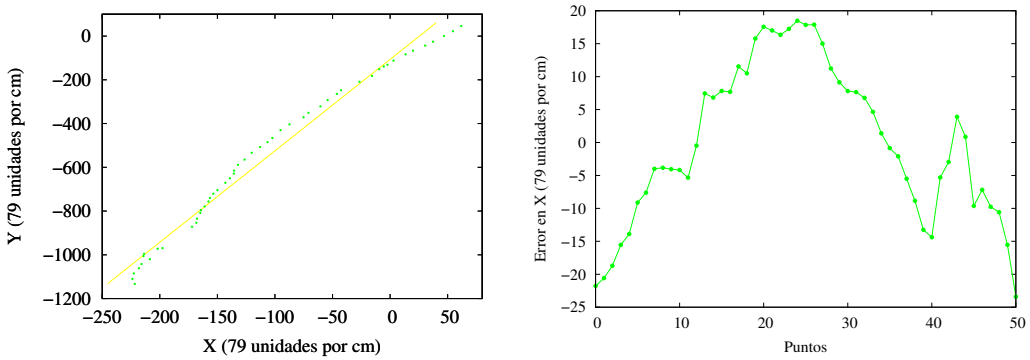

Figura 4.9: Eje XY. Puntos medidos y su ajuste de línea más el error en cada punto (Phantom)

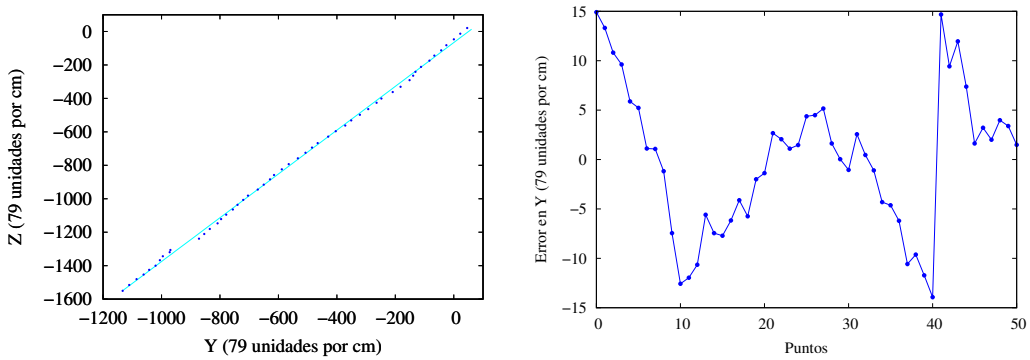

Figura 4.10: Eje YZ. Puntos medidos y su ajuste de línea más el error en cada punto (Phantom)

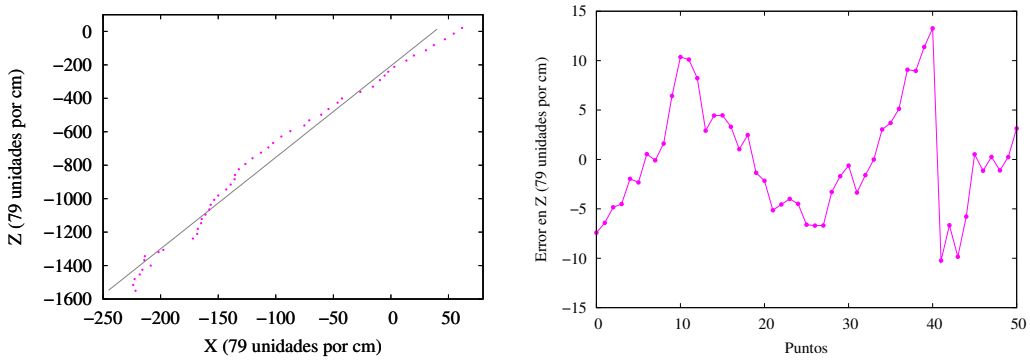

Figura 4.11: Eje XZ. Puntos medidos y su ajuste de línea más el error en cada punto (Phantom)

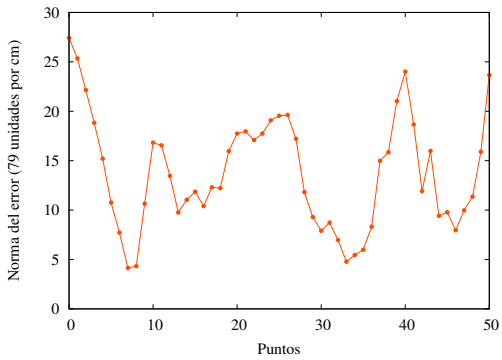

Figura 4.12: Eje XYZ. Error en cada punto (Phantom)

Los resultados de las operaciones mencionadas en el párrafo anterior para el dispositivo Patriot se muestran en las figuras 4.13, 4.14, 4.15 y 4.16.

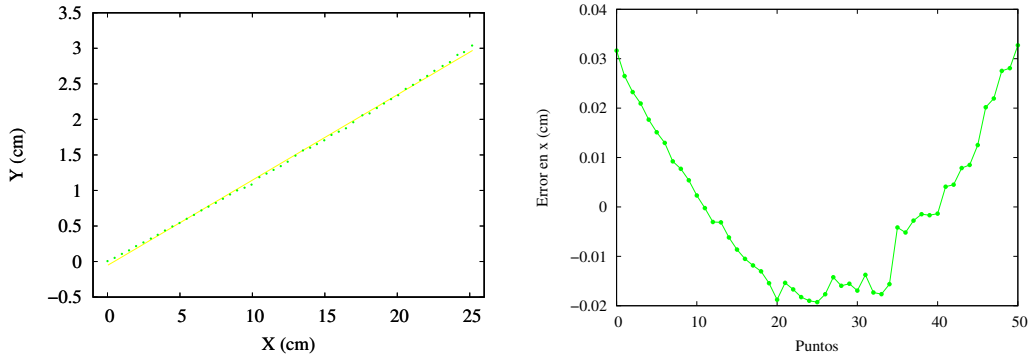

Figura 4.13: Eje XY. Puntos medidos y su ajuste de línea más el error en cada punto (Patriot)

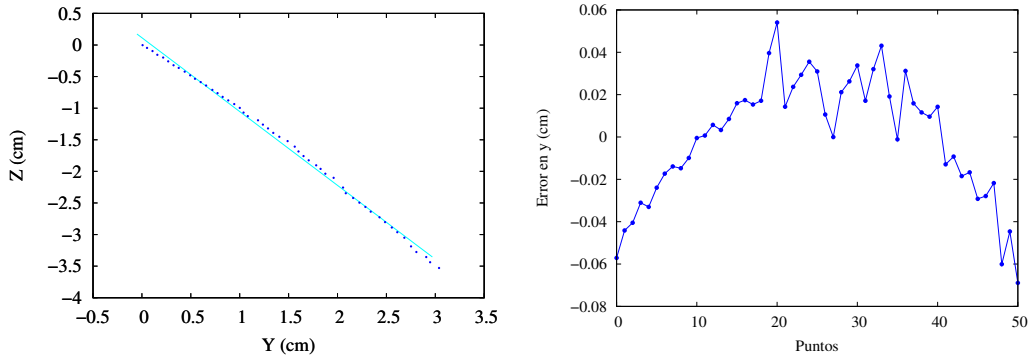

Figura 4.14: Eje YZ. Puntos medidos y su ajuste de línea más el error en cada punto (Patriot)

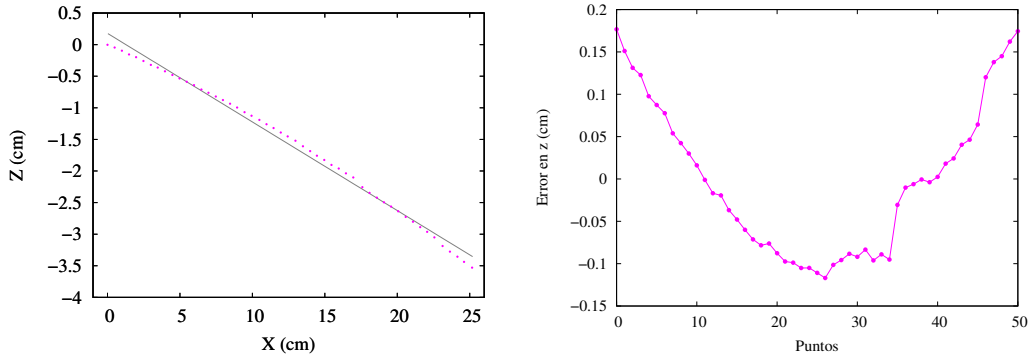

Figura 4.15: Eje XZ. Puntos medidos y su ajuste de línea más el error en cada punto (Patriot)

Con estos resultados podemos observar que la norma del error para el dispositivo Phantom Omni abarca de $[0-0.35]$ centímetros aproximadamente, mientras que para el dispositivo Patriot la norma del error está en un rango de $[0-0.19]$ centímetros

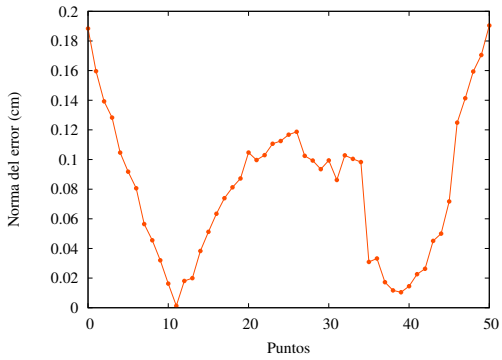

Figura 4.16: Eje XYZ. Error en cada punto (Patriot)

aproximadamente. Así, se encuentra que la norma del error es mayor en el dispositivo Phantom Omni.

4.5. Comparación de dispositivos

De una forma específica, en la Tabla 4.1 se listan las propiedades de cada uno los dispositivos integrados al prototipo de sistema desarrollado. Se puede observar que la principal diferencia se encuentra en la tecnología que cada dispositivo utiliza para su desempeño.

Únicamente el dispositivo Phantom ofrece el despliegue háptico, pues los brazos mecánicos permiten generar fuerzas sobre la pluma simulando texturas y volúmenes.

En cuanto a costos, el precio del dispositivo Patriot fue el más elevado, seguido por el Phantom, siendo el dispositivo Head Tracker el más económico de los tres. Es importante notar que las diferencias son significativas.

En la parte de desarrollo, el dispositivo Patriot requirió de dos y cuatro semanas más que los dispositivos Phantom y Head Tracker respectivamente; esto se debe a que la instalación requirió de considerar las características particulares del ambiente GNU/Linux y además se necesitó hacer un análisis del sistema de coordenadas para conseguir la representación correcta de los ángulos obtenidos con la pluma (*Stylus*) del mismo. Por otra parte, para el desarrollo con el dispositivo Phantom también se debió incluir el despliegue háptico lo que justificaría la diferencia de dos semanas con respecto al dispositivo Head Tracker. Las dimensiones del código no son varían mucho, sólo en el caso del dispositivo Phantom para el que además se implementó la interfaz háptica.

Respecto a las áreas de trabajo, se puede notar que el dispositivo Phantom cuenta con el área de trabajo más pequeña; su manual señala un área cúbica cuyos límites son menores a los que puede alcanzar su pluma si no es indispensable tener dimensiones equidistantes en la aplicación. Por su parte, el dispositivo Patriot, también cubre un área de trabajo mayor a la que el manual indica además de ser el que cubre un espacio de trabajo mayor; pero se debe tomar en cuenta que entre más se aleja el sensor de la fuente magnética, más inestable se vuelve la animación. En el caso del dispositivo Head Tracker, las especificaciones en el manual, concuerdan con su área de trabajo real.

La comunicación de los dispositivos manipuladores con la computadora requiere de conectarse a un puerto específico para cada uno, se puede notar que se tienen opciones diferentes.

El dispositivo Phantom brinda una tasa de muestreo mucho mayor al resto de los dispositivos, lo cuál brinda una mejor calidad en la animación implementada. Así mismo, se debe señalar que a pesar de que los dispositivos Patriot y Head Tracker tienen una tasa similar, la animación con el Patriot es muy inestable; mientras que con el

Head Tracker no existe éste problema si las muestras se toman dentro de su área de trabajo.

Finalmente, de acuerdo a la función de digitalización implementada en el prototipo de sistema, se puede decir que ésta es configurable para los tres dispositivos en cuestión. Sin embargo, dado que el dispositivo Head Tracker no cuenta con un punto de referencia visible ni funcional, no se incluyó en las pruebas de digitalización de puntos.

	Phantom Omni	**Patriot**	**Head Tracker**
Compañía	SensAble	Pholemus	Logitech
Tecnología	Brazos mecánicos	Magnético	Ultrasonido
Dispositivo háptico	Sí	No	No
Tiempo de desarrollo	6 semanas	8 semanas	4 semanas
Líneas de código	500 líneas	300 líneas	310 líneas
Área de trabajo (según manual)	X : 16 cm Y : 15 cm Z : 6 cm	Hasta 90cm (útil hasta 152cm)	Cono de 100° con 152cm ($5ft$) de longitud
Área de trabajo (real)	Arco horizontal en revolución 120°, 35cm de altura y 15cm de radio	X : 160cm $-X$: 90cm Y : 100cm $-Y$: 70cm Z : 70cm $-Z$: 60cm	Cono truncado de 90°, radio menor de 32cm y radio mayor de 152cm
Puerto	FireWire	USB y Serial RS-232	Serial RS-232D
Muestras por segundo	1000	60	> 50
Digitalización	Sí	Sí	No en su estado actual

Cuadro 4.1: Tabla comparativa de los dispositivos utilizados en la implementación.

Capítulo 5

Conclusiones y trabajo a futuro

5.1. Conclusiones

Se implementó un prototipo de sistema que permitiera la visualización y animación de un objeto deformable cuya estructura se modifica al interactuar con alguno de los dispositivos manipuladores: Phantom Omni, Patriot o Head Tracker. Se hizo una comparación entre dichos dispositivos. La implementación se hizo en lenguaje C/C++ utilizando las bibliotecas de Qt y OpenGL en un ambiente de GNU/Linux.

La estructura deformable del objeto simulado, se construyó con un sistema masa-resorte-amortiguador acoplado a una malla de cuadriláteros; sus nodos fueron conectados utilizando una técnica de sólidos en revolución. La interfaz gráfica de éste prototipo brinda las opciones de activar y desactivar los dispositivos, activar y desactivar el despliegue estereoscópico utilizando la visión activa, modificar las constantes del sistema masa-resorte-amortiguador del objeto deformable y además permite cambiar la apreciación del escenario desplegado.

Las características propias de cada uno de los dispositivos, especifican su comportamiento esperado y resaltan las ventajas y desventajas dentro de un ambiente virtual específico. A través de ésta implementación, se encontraron diferentes problemas y soluciones en la integración de cada dispositivo.

El trabajo con el dispositivo Phantom permitió trabajar en un ambiente con fricción y una malla de nodos (puntos, para el dispositivo háptico) que carecían de todo tipo de fuerza, de tal manera que se percibiera cuando que el dispositivo embonaba al pasar sobre uno de esos puntos. Esta propiedad y sus demás características que permiten la retroalimentación de fuerzas, hacen que el dispositivo brinde una interacción realista de la simulación que se tiene en pantalla. Además de que la velocidad de respuesta del dispositivo no genera conflicto con el despliegue gráfico.

Por su parte, el dispositivo Patriot cuenta con un área de trabajo más amplia en

comparación con la de los tres dispositivos manipuladores utilizados y permite una mayor libertad de movimiento con respecto a la pluma. Otra de sus ventajas es que tiene la opción mover el origen de referencia sin la necesidad de mover físicamente la fuente magnética. También, la tasa de envío de datos es muy rápida, lo que permite tener una simulación casi en tiempo real. Sin embargo, tiene la desventaja de que cualquier objeto metálico produce interferencia y, además, las mediciones se vuelven más inestables mientras más se retira el sensor del origen.

El dispositivo Head Tracker, tiene la desventaja de tener un área cónica, restringiendo el área de acción entre más se acerca el receptor al transmisor. Así mismo tiene la desventaja de tener sus rotaciones limitadas a 180° funcionales en el eje X y Y. Sin embargo, a pesar de que si se interpone un objeto entre el receptor y el transmisor se genera interferencia, evitándolo, la transmisión de datos es constante y suficientemente rápida para tener una animación casi en tiempo real.

Al final, se tiene un prototipo de sistema funcional capaz de expandirse hacia diferentes áreas de investigación.

5.2. Trabajo a futuro

Dentro del prototipo de sistema desarrollado, se pueden agregar muchas mejoras en el dibujado háptico, como centrar los objetos de tal forma que se aproveche enteramente el espacio de trabajo del dispositivo.

Hasta el momento, el comportamiento de la malla es específico y está delimitado por las constantes básicas del sistema masa-resorte-amortiguador, constantes que pueden modificarse únicamente dentro de sus rangos específicos. También, se pueden agregar otras variables como una fuerza interna que permita simular una malla más rígida o más holgada.

Respecto a la visión activa, el despliegue gráfico con visión estereoscópica se calcula considerando un punto fijo frente del monitor, si el observador se mueve en torno al escenario desplegado, la vista apreciada es la misma; se podría incluir un dispositivo encargado de seguir la posición del usuario y modificar el despliegue del escenario.

Por otra parte, también faltó probar los dispositivos con métodos de detección de colisiones que permitirían deformar la superficie que representa la malla en cualquier punto cuando se detecta que hay contacto. Con los datos obtenidos en esta tesis, se podría realizar una detección de colisiones eficaz con el Phantom y posiblemente con el Head Tracker. Esto debido a que se necesita una frecuencia de muestro de al menos 100 Hz para que la sensación de tocar un objeto sea eficaz.

Se necesita agregar la creación automática de la malla cuando se están digitalizando los puntos de una superficie (por ahora solo se capturan los puntos, sin estructura alguna).

Actualmente la interacción con la malla es sobre un solo punto o nodo, se pueden agregar dispositivos que permitan deformar la malla en varios puntos al mismo tiempo.

Apéndice A
Programas del ajuste de linea

La biblioteca general [32], desarrollada por el NPL Centre for Mathematics and Scientific Computing, consiste en funciones de MatLab para ajustar formas geométricas como: circulo, cono, esfera, línea y plano, por mínimos cuadrados a los datos dados. En nuestro caso, se utilizó el ajuste a una línea en tres dimensiones, donde el promedio de los datos es el centro de la línea y se calcula la dirección según el valor singular más grande de los residuos.

Programa `ls3dline.m`

De sus siglas en inglés "*Least-squares line in 3 dimensions*", su código es el siguiente:

```
function [x0, a, d, normd] = ls3dline(X)
% ------------------------------------------------------------------
% LS3DLINE.M   Least-squares line in 3 dimensions.
%
% Version 1.0
% Last amended   I M Smith 27 May 2002.
% Created        I M Smith 08 Mar 2002
% ------------------------------------------------------------------
% Input
% X         Array [x y z] where x = vector of x-coordinates,
%           y = vector of y-coordinates and z = vector of
%           z-coordinates.
%           Dimension: m x 3.
%
% Output
% x0        Centroid of the data = point on the best-fit line.
%           Dimension: 3 x 1.
%
% a         Direction cosines of the best-fit line.
%           Dimension: 3 x 1.
%
% <Optional...
% d         Residuals.
```

```
%          Dimension: m x 1.
%
% normd    Norm of residual errors.
%          Dimension: 1 x 1.
% ...>
%
% [x0, a <, d, normd >] = ls3dline(X)
% ------------------------------------------------------------

% check number of data points
  m = size(X, 1);
  if m < 3
    error('At least 3 data points required: ' )
  end
%
% calculate centroid
  x0 = mean(X)';
%
% form matrix A of translated points
  A = [(X(:, 1) - x0(1)) (X(:, 2) - x0(2)) (X(:, 3) - x0(3))];
%
% calculate the SVD of A
  [U, S, V] = svd(A, 0);
%
% find the largest singular value in S and extract from V the
% corresponding right singular vector
  [s, i] = max(diag(S));
  a = V(:, i);
%
% calculate residual distances, if required
  if nargout > 2
    m = size(X, 1);
    d = zeros(m, 1);
    for i = 1:m
      d(i) = norm(cross((X(i, 1:3)' - x0), a));
    end % for i
    normd = norm(d);
  end % if nargout
% ------------------------------------------------------------
% End of LS3DLINE.M
```

Programa `ajuste.oct`

Programa en octave que se encarga de la representación de los datos, apoyándose del programa `ls3dline.m`. Calcula los residuos en cada uno de los ejes, como se mostró en la Sección 4.4, en la pág. 70.

```
if ( margin != 1 )
        printf ("Args: file\n");
        exit (1);
endif
```

```
file = char( argv() );
printf ( "# %s\n", file );

[FILE1, msg] = fopen( file, "r" );
if  ( FILE1 < 0 )
        printf ( "%s\n", msg );
        exit (1);
endif

[n, count] = fscanf (FILE1, "%d", 1 );

X = zeros( n, 3);
for i=1:n
        [x, count] = fscanf (FILE1, "%f %f %f", 3);
        # printf ( "%f %f\n", x(1), x(2) );
        X(i,1) = x(1);
        X(i,2) = x(2);
        X(i,3) = x(3);
endfor

# X
[x0, a ] = ls3dline(X);

# normd
# x0
# a
# norm(a)
# d

d = zeros( n, 3);
for i=1:n
        Pi = X(i,:)';
        u = Pi - x0;
        t = dot( u, a );
        d(i,:) = ((x0 + t*a) - Pi)';
endfor


## AJUSTE ###################################################
printf ( "# Error en x y z \t\t # Norma del error \n" );
for i=1:n
        dist = norm( d(i,:) );
        printf ( "%f %f %f %f\n", d(i,1), d(i,2), d(i,3), dist);
endfor


## DISTANCIAS ###############################################
printf ( "# Distancias \n" );
dists = zeros(3);
for i=1:n
        norma_d = norm( d(i,:) );
```

```
        dists(1) = d(i,1)/norma_d - X(i,1)/norma_d;
        dists(2) = d(i,2)/norma_d - X(i,2)/norma_d;
        dists(3) = d(i,3)/norma_d - X(i,3)/norma_d;
        printf ( "%f %f %f %f\n", dists(1), dists(2), dists(3),
                                                    norm(dists) );
endfor


## LA LÍNEA CORRECTA ##############################################
printf ( "# Distancias  y la norma\n" );
line = zeros(3);
for i=1:n
        dists(1) = X(i,1) + d(i,1);
        dists(2) = X(i,2) + d(i,2);
        dists(3) = X(i,3) + d(i,3);
        printf ( "%f %f %f %f\n", dists(1), dists(2), dists(3),
                                                    norm(dists) );
endfor
```

Bibliografía

[1] Gerard Jounghyun Kim. *Designing Virtual Reality Systems: The Structured Approach*. Springer-Verlag, 2005.

[2] A. Naelen, M. Muller, R. Keiser, E. Boxeman, and M. Carlson. Physically based deformable models in computer graphics. *Computer Graphics Forum*, 25(4):809–836, 2006.

[3] F. García. Objeto deformable inmerso en un fluido. Master's thesis, CINVESTAV del I.P.N., Av. Instituto Politécnico Nacional No. 2508 Col. San Pedro Zacatenco México, D.F. 07360, Dec 2007.

[4] M. Chover. Representación poligonal. Universidad Politécnica de Valencia Website, Material docente, 2005. http://provadep.uji.es.

[5] A. Agarwala. Interactive furniture design: Using deformable models to create chairs. Technical report, Department of Electrical Enginering and Computer Science, Media Laboratory, MIT, 20 Ames Street Cambridge MA 02139, 1999.

[6] I. Boier-Martin, H. Rushmeier, and J. Jin. Parameterization of triangle meshes over quadrilateral domains. In R. Scopigno and D. Zorin, editors, *Eurographics Symposium on Geometry Processing*, 2004.

[7] Gino van den Bergen. *Collision Detection in Interactive 3D Environments*. Morgan Kaufmann Publishers - ELSEVIER, 2004.

[8] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raughpathi, A. Fuhrmann, M.-P. Cani adn F. Feaure, N. Magnenat-Thalmann, W.Strasser, and P. Volino. Collision detection for deformable objects. *Computer Graphics Forum*, 24(1):61–81, 2005.

[9] D. Zhang and M. M.F. Yuen. A coherence-based collision detection method for dressed human simulation. *Computer Graphics Forum*, 21(1):33–42, 2002.

[10] M. Overmars O Schwarzkopf M. de Berg, M. van Kreveld. *Computational Geometry, algorithms and applications*. 2000.

[11] V.Q.H. Huynh, T. Kamada, and H.T. Tanaka. An adaptative 3d surface mesh cutting operation. *Articulated Motion and Deformable Objects (AMDO 2006), 4th International Conference*, pages 366–374, 2006.

[12] L. Kavan and J. Zara. Fast collision detection for skeletall deformable models. *Computer Graphics Forum*, 24(3):363 372, 2005.

[13] J. Mezger and W. Straber. Interactive soft object simulation with quadratic finite elements. *Articulated Motion and Deformable Objects (AMDO 2006), 4th International Conference*, pages 434–443, 2006.

[14] J. L. Schoner, J. Land, and H.-P. Seidel. Measuremente-based interactive simulation of viscoelastic solids. *Computer Graphics Forum*, 23(3):547–556, 2004.

[15] R. Turner and E. Gobetti. Interactive construction and animation of layedred elastically deformable characters. *Computer Graphics Forum*, 17(2):135–152, 1998.

[16] J. Lanquentin, R. Raffin, and M. Neveu. Generalizad scodef deformations on subdivision surfaces. *Articulated Motion and Deformable Objects (AMDO 2006), 4th International Conference*, pages 132–142, 2006.

[17] A. Greb, M. Gurhe, and R. Klein. GPU-based collision detection for deformable parametrerized surfaces. *Computer Graphics Forum*, 25(3):497–506, 2006.

[18] J. Lovisach. Wrinkling coarse meshes on the GPU. *Computer Graphics Forum*, 25(3):467–476, 2006.

[19] J.G. Moctezuma. Manipulación tridimensional de objetos deformables virtuales. Master's thesis, CINVESTAV del I.P.N., Av. Instituto Politécnico Nacional No. 2508 Col. San Pedro Zacatenco México, D.F. 07360, 2006.

[20] Judd Bowman and Laurence Edwards. *Mars Exploration Rover (MER) Project*, 2002.

[21] Jonathan David Pfautz. Depth perception in computer graphics. Technical report, University of Cambridge - Computer Laboratory, 2002.

[22] M. McKenna and D. Zelzer. *Three Dimensional Visual Display Systems for Virtual Environment*. Morgan Kaufmann Publishers - ELSEVIER, 1992.

[23] P. Bourke. Calculating stereo pairs. Western Australian Supercomputer Program Website, Jul 1999. http://wasp.uwa.edu.au.

[24] StereoGraphics Corporation. *Setup and Installation Guide*. www.StereoGraphics.com.

[25] StereoGraphics Corporation. *CrystalEyes Software Development Kit*, December 1997. ftp://ftp.stereographics.com/developers/ce_sdk.pdf.

[26] Samuel Thomas McJunkin. *Transparency Improvement for Haptic Interfaces*. PhD thesis, Rice University, Houston, Texas., May 2007.

[27] C.M. Ramírez. Animación de modelos deformables. Master's thesis, CINVESTAV del I.P.N., Av. Instituto Politécnico Nacional No. 2508 Col. San Pedro Zacatenco México, D.F. 07360, Dec 2005.

[28] SensAble Technologies, Inc. *OpenHaptics(TM) Toolkit, API Reference*, Jul 2002. http://www.sensable.com.

[29] SensAble Technologies, Inc., 15 Constitution Way Woburn, MA 01801. *OpenHaptics(TM) Toolkit, Programmer's Guide*, Aug 2005.

[30] Alken, Inc., dba Pholemus, Colchester, Vermont, U.S.A. *Patriot(TM), User Manual*, Nov 2005. http://www.polhemus.com.

[31] Logitech Inc., Logitech Inc., Fremont, CA 94555. *3D Mouse and Head Tracker, Technical Reference Manual*, Nov 1992.

[32] NPL Centre for Mathematics and Scientific Computing. Lsge: The least squares geometric elements library. EUROMETROS Website. http://www.eurometros.org/gen_report.php?category=distributions&pkey=14&subform=yes.