Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Departamento de Ingeniería Eléctrica

Sección Computación

Animación de Modelos Deformables

Tesis que presenta

Claudia Magdalena Ramírez Trejo

para obtener el Grado de

Maestro en Ciencias

en la Especialidad de

Ingeniería Eléctrica

Director de la Tesis

Dr. Luis Gerardo de la Fraga

México, D.F. **Diciembre 2005**

Resumen

En este trabajo se ha realizado la animación de modelos deformables elásticos e inelásticos, basados en mallas de simplejos. Este tipo de objetos se deforman siguiendo la ley de Newton del movimiento sobre un sistema mecánico compuesto de resortes, masas y amortiguadores. Para realizar la animación de los objetos se resolvieron numéricamente las ecuaciones de movimiento. En este trabajo se presenta una comparación de cuatro métodos numéricos de la ecuación de movimiento: diferencias finitas, Euler, Heun y Runge-Kuta de orden cuatro, resultando mejor y más simple, la formulación de *diferencias finitas*.

También se desarrolló una interfaz gráfica basada en Qt y OpenGL que permite al usuario definir las característica elásticas de los objetos deformables. Además, se puede interactuar con dichos modelos mediante un dispositivo háptico llamado Phantom Omni. El usuario puede suministrar fuerza externa sobre ciertos puntos del objeto, los cuales son seleccionados a través del Phantom.

Se presentan cuatro ejemplos de aplicación: la animación de la deformación de una esfera a un cubo, la animación del rebote de una pelota contra la pared, una pelota comprimida con dos paredes y la deformación general de una esfera aplicando fuerza en cualquier punto seleccionado con el Phantom Omni. Estos ejemplos se realizaron con el fin de verificar el comportamiento eslástico de las mallas de simplejos.

Los resultados obtenidos en cuanto a la animación de modelos deformables elásticos usando mallas de simplejos son satisfactorios, sin embargo, el sistema no se deforma en tiempo real: primero, el usuario selecciona por medio del Phantom a cuáles puntos de la malla se les aplicará una fuerza externa. Después, un motor de cálculo numérico es el encargado de computar la respuesta de la malla y deformarla de acuerdo a sus características elásticas y la fuerza aplicada.

Abstract

In this work we have generated the animation of deformable objects. These objects has been built with simplex meshes. A deformable object is a kind of object which is reshaped according to the Newton's second law of motion in a mechanical system composed of springs, masses, and dampers.

To perform animation involves numerically solving the law to motion. In this work we present a comparison among four different numerical methods: finite differences, Euler, Heun, and fourth order Runge-Kutta. From our results and for our application, the best method is the simplest one: finite differences.

An graphic interface was designed in Qt and OpenGL which allows users to define the elastic features of the deformable objects. Futhermore, the user can interact with these objects through a haptic device called Phantom Omni. The user can apply external force on object points. These points can be selected for the user, using the Phantom device.

We show four application examples: the animation of a sphere deformed to get a cube, and the animation of a ball bounced against a wall, a ball compressed by two walls, and general deformation, in any point selecting through the Phantom Omni, of a sphere. The purpose of these examples is to verify the elastical behavior of the meshes.

The results we got about the animation of deformable elastic models based on simplex meshes are acceptable, however the deformation of the system is not performed in real time: the user first select trough the Phantom the points on the mesh where the external force will be applied. After that, an numerical calculation engine is used to compute the response of the mesh and then it is deformed, taking acount the elastic features and the applied force.

Agradecimientos

Al CINVESTAV
por darme todas las herramientas necesarias
para la realización de esta tesis.

Al CONACyT
por proporcionar los recursos necesarios
para el desarrollo de este trabajo,
en específico al proyecto 45306.

Al Dr. Luis Gerardo de la Fraga
de quien he aprendido mucho este último año
y quien me alentó a seguir adelante
cuando el camino se tornaba difícil.
Pero sobre todo le agradezco por su tiempo,
paciencia y apoyo.

A mi familia
por tener fe en mí,
por enseñarme a luchar por mis metas
y darme las armas necesarias para salir adelante
brindándome incondicionalmente su amor,
apoyo y comprensión.

A mis amigas: **Abigail Martínez y Grettel Barceló**
que no sólo me han impulsado a superarme
sino que también me han ayudado a ser una mejor persona
en todos los aspectos, y cuyo apoyo fue fundamental
para la culminación de esta tesis.

A Sofía Reza
por su gran paciencia, amabilidad
y por el excelente trabajo que realiza.

A todas aquellas personas que me han enseñado algo
y que han formado parte de mi desarrollo como persona y profesional.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Antecedentes

Los modelos deformables son una clase de primitivas para modelado de curvas, superficies y sólidos generales que obedecen a la dinámica de cuerpos no rígidos, sus leyes se expresan usualmente en forma de ecuaciones diferenciales dinámicas. Estos modelos idealizan amplias gamas de respuestas de materiales bajo diversas condiciones ambientales, son capaces de demostrar una gran variedad de comportamientos naturales incluyendo elasticidad, viscoelasticidad, plasticidad, fractura, transferencia de calor conductiva, termoplasticidad, derretimiento y comportamiento tipo fluido.

La formulación de modelos deformables con algunas restricciones se ha aplicado para simular pelotas sólidas sobre esponjas, modelos de telas como banderas en el viento, alfombras voladoras puestas sobre objetos, ropa, telas que se rasgan al jalarlas, etcétera [2, 3, 4]. Entre algunas de las aplicaciones más importantes se pueden citar las siguientes:

Modelos deformables interactivos en mundos virtuales
Se han implementado varios mundos experimentales 2D y 3D que incorporan modelos elásticos o inelásticos. El usuario puede interactuar con estos modelos aplicando fuerzas simuladas [5, 6, 7].

Modelado y animación humana y facial
El poder expresivo del cuerpo y la cara humana los hacen una meta atractiva pero difícil para animadores y modeladores gráficos. Se han usado modelos deformables para las partes del cuerpo y el cabello. Terzopoulos y Waters [8] han desarrollado un modelo de cara 3D (dinámica, con músculos faciales basados en anatomía) para controlar un modelo deformable del tejido facial.

Modelos deformables para visión
Una aplicación de los modelos deformables es en la segmentación, ajuste y seguimiento de imágenes de objetos. Dentro de los modelos deformables aplicables a visión están las víboras, que son modelos planos deformables de contornos que se comportan como alambre estirable. Las víboras dan una forma sencilla y exacta de localizar y seguir de manera interactiva orillas y otros detalles curvilineos de interés en imágenes naturales. Se han aplicado a seguir el movimiento no-rígido de los labios de la gente al hablar, y extraer los movimientos de otras facciones faciales importantes a partir de videos [9, 10, 11].

1.2. Sistema a desarrollar

Para realizar la animación del comportamiento de modelos deformables se presentan tres problemas principales. El primero consiste en establecer la forma de representación de los objetos (*superquadrics*, mallas de simplejos, etcétera) pues esto determina los algoritmos y ecuaciones que se emplean en la manipulación de los modelos. En segundo lugar, debe especificarse la manera en que se interactuará con los modelos, así como los tipos de comportamiento que serán simulados tales como elástico, inelástico, plástico, entre otros. Una vez establecida la representación de los objetos y el tipo de comportamiento que tendrán, el tercer problema consiste en estudiar y discretizar las ecuaciones de las leyes físicas con las que será calculada la respuesta del sistema.

De forma contraria a otras posibles representaciones las mallas de simplejos permiten deformaciones suaves sobre condiciones locales [12] y aunque la teoría para animar objetos plásticos deformables no es nueva [13, 14], no se conoce que ésta se haya aplicado en la visualización de la deformación de objetos basados en mallas de simplejos. Por lo tanto, probar el esquema de mallas de simplejos para realizar la animación de objetos plásticos deformables resulta un problema interesante.

Así, el propósito de este trabajo es realizar la animación de modelos deformables elásticos e inelásticos basados en mallas de simplejos, con los cuales el usuario pueda interactuar, a través de un dispositivo háptico (Phantom Omni), y aplicar fuerza sobre ellos con el fin de que pueda apreciar las características elásticas de los objetos y observar su comportamiento. Se creará una herramienta de software basada en Qt y OpenGL que permita definir las características elásticas del objeto, la fuerza externa a ser aplicada y visualizar la deformación de la malla. Se integrará el Phantom Omni a esta herramienta para que a través de este dispositivo el usuario pueda seleccionar los puntos de la malla sobre los que se aplicará la fuerza externa especificada.

1.2.1. Representación del objeto deformable

La idea principal es representar un objeto deformable a partir de partículas conectadas entre sí por medio de resortes formando una malla de polígonos. Una característica importante es que cada partícula tendrá una conectividad constante, más específicamente tendrá sólo tres vecinos, a este tipo de mallas se le conoce como mallas de simplejos-2. Se han elegido las mallas-2 ya que en [1] presentan las estructuras para representar este tipo de mallas así como los algoritmos para construir una esfera, un cilindro y un toroide con mallas-2.

Las partículas tienen como atributos principales masa, posición y velocidad así como propiedades de cuerpos rígidos. De esta forma, sólo la fuerza de los resortes de sus vecinos directos o una fuerza externa, afecta la velocidad y aceleración del punto. Cada partícula tendrá sólo tres grados de libertad, es decir que podrá moverse en el espacio pero no rotar. Se asumirá que los modelos no se rompen y no cambian su elasticidad durante la deformación [5].

1.2.2. Interacción

Los objetos deformables podrán manipularse a través del dispositivo háptico Phantom Omni, considerado en la aplicación como un artefacto virtual totalmente rígido, con el cual serán seleccionados los puntos de la malla para aplicarles una fuerza externa, después el sistema calcula la respuesta del modelo de acuerdo a las propiedades del objeto, como se describe a continuación:

Modelos elásticos. Estos modelos mantienen una velocidad y forma constante siempre y cuando una fuerza externa no sea aplicada. En caso contrario, estos se deforman hasta encontrar un estado de reposo donde su energía interna se balancea con la fuerza externa. Si la influencia de la fuerza se detiene, el objeto regresa a su forma inicial [15].

Modelos inelásticos. A diferencia de los modelos elásticos que de inmediato recobran su forma natural no deformada al quitar la fuerza externa, los modelos inelásticos se deforman permanentemente por lo que son usados comúnmente para simular los comportamientos mecánicos de sólidos altamente polimerizados como la plastilina [2].

1.2.3. Repuesta del sistema

Las fuerzas principales que mueven y deforman el objeto son las fuerzas internas de los resortes de sus bordes, las cuales resultan de la fuerza externa aplicada al modelo. Esta fuerza interna es calculada con la ley de Hooke y el sistema se considera no lineal para evitar que los resortes lleguen a su límite elástico o de ruptura. También, ya que los resortes no oscilan siempre, es necesario contemplar la fuerza de amortiguamiento de las aristas e incluir los mecanismos necesarios para simular la pérdida de energía.

La deformación se cuantifica usando las fórmulas básicas de movimiento y las leyes de Newton que balancean las fuerzas elásticas resultantes contra fuerzas inerciales debidas a la distribución de masas del modelo, así como las fuerza de amortiguamiento, de fricción y fuerzas aplicadas externamente sobre la superficie de colisión [16]. Para realizar los cálculos de la deformación se realizará un motor de deformación.

1.3. Resultados esperados

Con el desarrollo de esta tesis se espera lograr los siguientes puntos:

- Realizar la animación de modelos deformables basados en mallas de simplejos para crear una nueva forma de visualizar objetos tridimensionales, como si fuesen objetos plásticos, y estudiar los algoritmos necesarios para poder realizar la animación.

- Determinar la conveniencia del uso de mallas de simplejos con respecto a la ventaja que ofrecen de permitir deformaciones locales, en comparación con otro tipo de representaciones, y el tiempo de respuesta del sistema. Esto será determinado tomando en cuenta que el tiempo requerido para calcular la deformación de éstas se incrementa en proporción a la resolución de los objetos, pues lo más conveniente sería tener objetos con una buena resolución y que además permitieran deformaciones locales de forma muy rápida.

- Ya que Qt ofrece la ventaja de que el programador sólo se enfoque a la funcionalidad y la comunicación entre los diferentes widgets (botones, cuadros de diálogo, menús, etcétera), se espera lograr integrar al dispositivo háptico Phantom con la interfaz gráfica desarrollada con OpenGL y Qt, con el fin de estudiar los problemas, restricciones o inconvientes que se pueden presentar, especialmente en lo que respecta a la concurrencia que se genera entre los hilos para el manejo de eventos de la interfaz gráfica y los hilos para el manejo de eventos del dispostivo.

1.4. Organización de la tesis

En el capítulo 2 se da una breve introducción a los diferentes tipos de representación de los modelos deformables, como *superquadrics*, superficies algebraicas, contornos discretos, mallas de simplejos, etcétera.

En el capítulo 3 se plantea un sistema de resorte amortiguado forzado, se encuentra la solución a éste en distintos intervalos discretos ya que no se puede hacer en forma continua. Se realiza el análisis de eficiencia y robustez de cuatro métodos de discretización para determinar cual es el más adecuado para animar objetos deformables. Por último se presentan las restricciones aplicadas al modelo para que el sistema

tuviera un comportamiento no lineal.

En el capítulo 4 se presentan cuatro ejemplos de animación de modelos deformables cuyo fin fue probar que al integrar el esquema de resorte, masa y amortiguador propuesto en el capítulo anterior a las mallas de simplejos, se lograr que los objetos tengan un comportamiento elástico. Se realizó la deformación de una esfera a un cubo, el rebote de una pelota contra la pared, una pelota comprimida por dos paredes y la deformación general de una esfera, se describe como fueron realizadas las animaciones y se muestran los resultados obtenidos.

En el capítulo 5 se describen las características del Phantom Omni, que es el dispositivo háptico que se utilizó para manipular los objetos y la escena gráfica. Por medio de éste el usuario selecciona los vértices de la malla a los cuales se les aplicará una fuerza externa. Se explica como se logró incorporar el Phantom con OpenGL y Qt por medio de las interfaces gráficas HLAPI y HDAPI que vienen incluidas en la librería del Phantom.

En el capítulo 6 se presentan las conclusiones del trabajo realizado y el trabajo a futuro.

Capítulo 2

Marco Teórico

En este capítulo se introduce a la teoría de los modelos deformables y sus diferentes tipos de representación. Aunque se presentan todos las representaciones, se da mayor atención a las mallas de simplejos pues tal y como se mencionó en el capítulo anterior los objetos serán creados en base a éstas.

El propósito de está sección no es comparar las ventajas y desventajas entre los tipos de representación de los modelos deformables, más bien se pretende definir lo que son las mallas de simplejos y ubicarlas entre los tipos de representación que existen.

2.1. Modelos deformables

Un modelo o superficie deformable está compuesto tanto de la representación de superficie del objeto como de las leyes de evolución que permiten la deformación de éste. La superficie tiene seis grados de libertad: tres para la translación y tres para la rotación [5].

Aunque computacionalmente más demandantes, los modelos deformables son superiores de varias maneras. Son fundamentalmente dinámicos, sus ecuaciones proveen una descripción unificada de formas no-rígidas y sus movimientos complejos a través del espacio.

Como se menciona en [14], los modelos deformables se pueden clasificar por la naturaleza de su representación en dos categorías: discretos y continuos, cada familia a su vez puede ser dividida en diferentes clases de representaciones: las superficies continuas pueden ser definidas a través de una ecuación implícita o explícita mientras que las superifices discretas son principalmente representadas a través de mallas discretas o de sistemas de partículas.

2.1.1. Modelos continuos

Las representaciones continuas deben ser discretizadas para su implementación computacional pero permiten el cálculo de cantidades tales como normales o curvaturas en casi cualquier parte de la superficie. Se clasifican por la ecuación con la cual son representados, explícita o implícita.

Algunas formas de representación explícita son:

1. Funciones de soporte polinomiales finitas.
2. *Superquadrics*, representados por el vector de parámetros de superficie.
3. Descomposición modal.

Para una representación implícita se pueden utilizar:

1. *Superquadrics*, cuya función representa una superficie cerrada.
2. *Hyperquadrics*.

2.1.2. Modelos discretos.

En las representaciones discretas la geometría de la superficie es solamente conocida en un conjunto finito de puntos y se basa naturalmente en el problema de caracterización de las representaciones continuas. Se clasifican por su naturaleza en dos tipos: mallas discretas y sistemas de partículas.

Las mallas discretas se definen como un conjunto con alguna relación de conectividad, entre las más comúnmente usadas son:

1. Contornos Discretos.
2. Triangulaciones.
3. Modelo elástico de masa.
4. Mallas de simplejos.

2.2. Mallas de simplejos

Una malla de simplejos M está definida como un conjunto de vértices p_{ii} y una función de conectividad. Cada vértice de una malla de simplejos-k está conectada exactamente a $k+1$ vécinos como se muestra en la Figura 2.1.

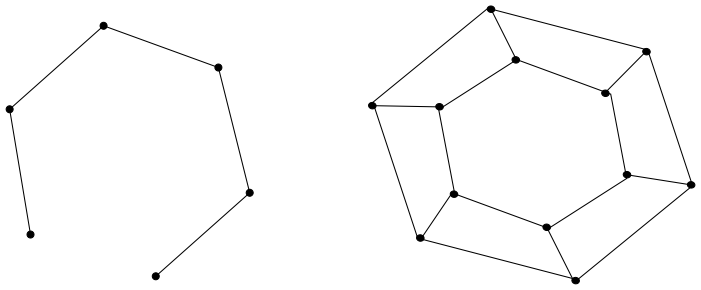


Figura 2.1: A la izquierda se muestra una malla de simplejos-1 y a la derecha una malla de simplejos-2

La función de conectividad define las aristas de la malla de forma que sólo haya una arista entre dos vértices distintos. Una cara de la malla está compuesta por una secuencia de aristas sobre la malla tal que ninguna arista parte una cara en dos más pequeñas.

Las mallas de simplejos-2 son construcciones geométricas usadas para representar superficies en el espacio tridimensional. Cada vértice de una malla-2 esta conectado con tres y sólo tres vecinos. Estas mallas tienen similitudes con la triangulación, de hecho son el dual topológico de la triangulación [17]. Debido a su conectividad constante, la geometría de las mallas de simplejos es bastante simple.

Sea P_i el vector de posición para el vértice i. En una malla-2 de simplejos definimos un vector normal a P_i como:

$$n_i = \frac{(P_{N_1(i)} \times P_{N_2(i)}) + (P_{N_2(i)} \times P_{N_3(i)}) + (P_{N_3(i)} \times P_{N_1(i)})}{\|P_{N_1(i)} \times P_{N_2(i)} + P_{N_2(i)} \times P_{N_3(i)} + P_{N_3(i)} \times P_{N_1(i)}\|} \tag{2.1}$$

donde $P_{N_1(i)}, P_{N_2(i)}, P_{N_3(i)}$ son los vectores de posición de los vecinos uno, dos y tres respectivamnete del vértice i.

2.2.1. Esquema de visualización

En general una malla de simplejos está formada por medio de polígonos, éstos a su vez están formados por aristas y finalmente las aristas están definidas por vértices que son la unidad mínima o elemento básico con los que es construída la malla (Ver Figura 2.2). Las relaciones entre los elementos de una malla de simplejos-2 son las siguientes:

- Cada vértice tiene tres vértices vecinos.
- Un vértice puede pertenecer solamente a tres aristas.
- Un vértice sólo puede pertenecer a tres caras.

- Una arista está formada por dos vértices.
- Una cara puede tener n aristas.

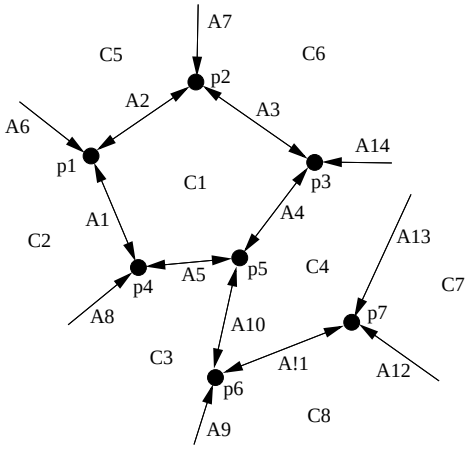


Figura 2.2: Estructura de una malla de simplejos. Los vértices de la malla están etiquetados con la letra p, las aristas con la letra A y las caras con la letra C.

Como se mencionó en el capítulo anterior, la interfaz gráfica será construída en OpenGL y Qt; OpenGL permite desarrollar aplicaciones gráficas en 2D y 3D para lo cual tiene incorporadas primitivas de puntos, líneas, polígonos, imágenes y bitmaps. Para visualizar los modelos construídos con mallas de simplejos debe indicarse qué elementos de la malla serán mostrados, cuáles son los polígonos que forman las caras, el orden que tienen, su normal, color, etcétera. Entonces, para facilitar el dibujado de las mallas con las primitivas que ofrece OpenGL, se requiere una estructura de datos para almacenar y representar los elementos que conforman la malla y las relaciones entre ellos.

Las estructuras de datos que utilizamos son las propuestas en [1], en las cuales los elementos de un mismo tipo están guardados en listas doblemente ligadas, esto facilita la inserción de elementos y agiliza su búsqueda. La estructura de datos principal representa la malla de simplejos y está compuesta por un apuntador al primer elemento de cada una de las listas de puntos, aristas y caras. Estas listas a su vez están ligadas entre sí para representar las relaciones que existen entre los elementos de la malla.

2.2.2. Creación del modelo de un objeto con mallas de simplejos

Como se mencionó en 1.2, se desea realizar la animación de modelos deformables basados en mallas de simplejos para lo cual se deben contruir primero los objetos. En [1] se presentan los procedimientos para la creación de una esfera, un cilindro y un

toroide con mallas de simplejos, estos procedimientos siguen tres pasos principales: la generación de vértices, aristas y caras. Nosotros contamos con la implementación que realizaron en [1] de esos procedimientos, por lo que retomamos dicha implementación para construir los objetos en las animaciones. Sin embargo, para la generación de caras detectamos que el algoritmo codificado en el software de la tesis [1] difiere de los procedimientos que se presentan en el documento.

Identificamos el siguiente inconveniente en la implementación de [1]: para generar las caras, los vértices que conforman cada cara deben estar ordenados en el sentido de las manecillas del reloj para que sean visualizados de forma correcta, es decir, que la normal esté "hacia afuera" del cuerpo geométrico. En la implementación que realizaron, se genera una lista de vértices con los que se forman las caras pero éstos no se encuentran ordenados de manera correcta, por esta razón no les fue posible implementar el algoritmo que propusieron. En su lugar, implementaron un algoritmo recursivo que es mucho más complejo que el que proponen y no se aprovecha la regularidad de las mallas de simplejos.

Para solucionar este problema, en esta tesis desarrollamos un algoritmo de ordenamiento para los vértices, tomando en cuenta que la dirección de la normal debe ser igual al sentido del vector que va del centro de la esfera al vértice, dicho algoritmo se presenta en el procedimiento 1. Además, se codificó el algoritmo presentado en [18], con lo cual se verificó que funciona y que es mucho mejor que el algoritmo recursivo que ellos implementaron.

Procedimiento 1 Método de ordenamiento

Entrada: La lista de vértices

Salida: Lista de vértices con sus vécinos ordenados en el sentido de las manecillas del reloj

for Para cada vértice en la lista de vértices **do**

$v_{proy} \leftarrow P_i - c_m$

$Vizq \leftarrow VuletaIzq(P_i, P_{N_1(i)}, P_{N_2(i)}, v_{proy})$

if $Vizq == 0$ **then**

$P_{N_1(i)} \leftrightarrow P_{N_2(i)}$

$P_{C_1(i)} \leftrightarrow P_{C_2(i)}$

Un polígono tiene dos caras la frontal y la dorsal, los vértices pueden ordenarse de acuerdo a la cara que se este observando. Esto se define por medio de un vector de proyección. Aprovechando que conocemos el centro de la malla c_m podemos definir al vector de proyección como $\overrightarrow{v}_p = P_i - c_{Malla}$, de esta forma los vértices quedarían ordenados por la vista dorsal de las caras.

Ya que los polígonos de la malla son convexos pueden presentarse sólo dos casos: los tres vértices vecinos están ordenados correctamente o las posiciones del segundo y

tercer vecino están intercambiadas (Ver Figura 2.3(a)). Por consiguiente, los vértices vecinos del punto P_i están ordenados en el sentido de las manecillas del reloj si los vectores $\overrightarrow{v_1} = P_i - P_{N_1(i)}$ y $\overrightarrow{v_2} = P_i - P_{N_2(i)}$ forman una vuelta por la izquierda con respecto al vector de proyección. En caso contrario la posición de $P_{N_2(i)}$ y $P_{N_3(i)}$ en la lista de vecinos de de P_i es itercambiada, como se muestra en la Figura 2.3(b). El algoritmo para verificar si dos vectores forman una vuelta por la izquierda es el que se muestra en el procedimiento 2.

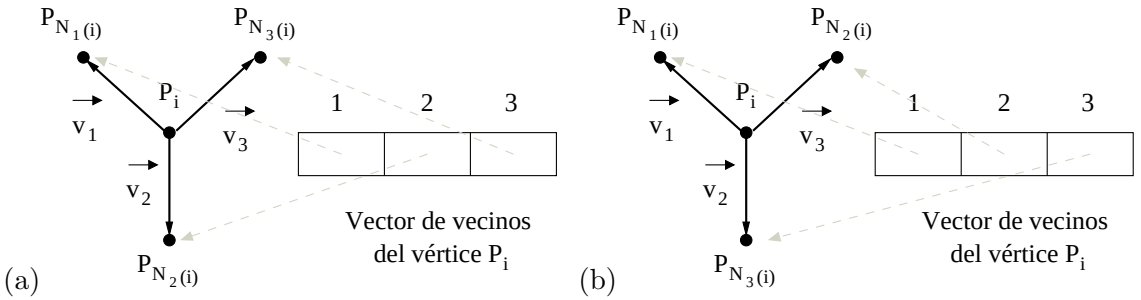


Figura 2.3: (a) Vértices vecinos desordenados con respecto al vector de proyección $z = 1$. (b)Vecinos ordenados después de aplicar el algoritmo 1.

Procedimiento 2 Vuelta por la izquierda

Entrada: P_i, $P_{N_1(i)}$, $P_{N_2(i)}$ y el vector de proyección v_{proy}
Salida: 1 si es una vuelta por la izquierda y 0 en caso contrario

1: $v_1 \leftarrow P_i - P_{N_1(i)}$
2: $v_2 \leftarrow P_i - P_{N_2(i)}$ {v_1 y v_2 son vectores}
3: $Pc \leftarrow v_1 \times v_2$
4: $Pp \leftarrow Pc \cdot v_{proy}$
5: **if** $PP >= 0$ **then**
6: return 1
7: **else**
8: return 0

Para verificar una vuelta por la izquierda entre $\overrightarrow{v}_1$ y $\overrightarrow{v}_2$ con respecto al vector de proyección, obtenemos con el producto cruz de éstos, un vector $\overrightarrow{v}_n$ que por definición es ortogonal a ambos. Por último, si el ángulo entre dos vectores está definido como:

$$\cos\theta = \frac{\overrightarrow{v_1}\cdot\overrightarrow{v_2}}{\|\overrightarrow{v_1}\|\|\overrightarrow{v_2}\|} \quad \text{y} \quad \begin{cases} \cos\theta \geq 0 & \text{si} \quad \theta \leq 90 \\ \cos\theta < 0 & \text{si} \quad \theta > 90 \end{cases}$$

y la magnitud de un vector nunca es negativa entonces los vectores $\overrightarrow{v}_1$ y $\overrightarrow{v}_1$ forman una vuelta por la izquierda si y solo si $v_n \cdot v_p \geq 0$.

Capítulo 3

Sistemas de Resorte y Masa: Movimiento Amortiguado Forzado

En este capítulo se analiza un sistema de resorte y masa con movimiento amortiguado forzado, para lo cuál se presenta una serie de conceptos básicos relacionados, como la ley de Hooke, la fuerza de amortiguamiento y el modo en que se obtiene la solución exacta de dicho sistema.

Para realizar la animación de modelos deformables elásticos es necesario resolver numéricamente las ecuaciones de movimiento del sistema de resorte y masa. En algunos trabajos como [14, 15, 8] proponen usar el método de Diferencias finitas centrales pero no justifican su uso, por lo que también se presenta en este capítulo, el análisis de eficiencia de tres métodos de discretización con el fin de compararlos con el método de Diferencias finitas.

3.1. Ley de Hooke

Supongamos que, como en la Figura 3.1, una masa m_1 está unida a un resorte flexible colgado de un soporte rígido. Cuando se reemplaza m_1 con una masa distinta m_2, el estiramiento, elongación o alargamiento del resorte cambiará.

Según la ley de Hooke, el resorte mismo ejerce una fuerza de restitución F, opuesta a la dirección del alargamiento y proporcinal a la cantidad de alargamiento s. En concreto, $F = ks$, donde k es una constante de proporcionalidad llamada constante del resorte [19, Cap. 5].

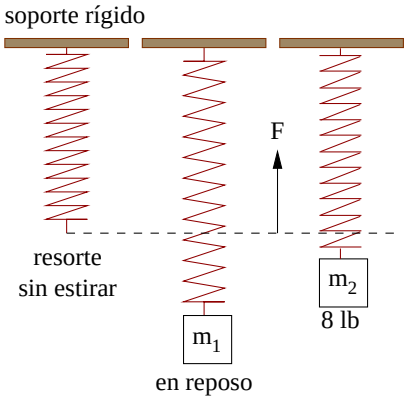


Figura 3.1: Resortes

3.2. Segunda ley de Newton

Después de unir una masa m a un resorte, ésta lo estira una longitud s y llega a una posición de equilibrio, en la que su peso, W, está equilibrado por la fuerza de restauración ks. Recuérdese que el peso se define por $W = mg$. Como se aprecia en la Figura 3.2, la condición de equilibrio es $mg = ks$. Si la masa es desplzada una distancia x respecto de su posición de equilibrio, al aplicarle una fuerza externa f_{ext}, la fuerza de restitución del resorte es $k(x + s)$.

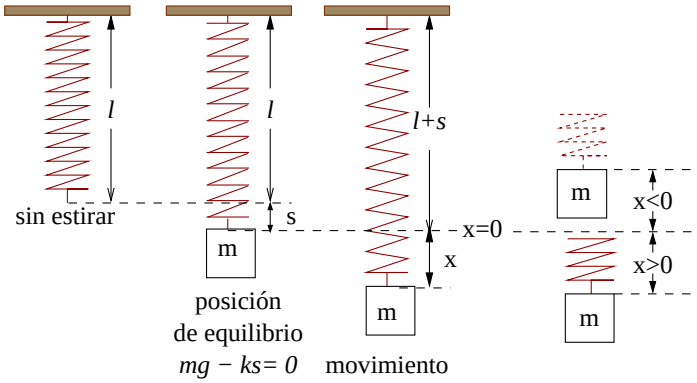


Figura 3.2: Sistema de resortes

Tomando en cuenta que la masa no está colgada en un vacío perfecto, cuando menos habrá una fuerza de resistencia debida al medio que rodea al objeto. Según se advierte en la Figura 3.3, la masa podría estar suspendida en un medio viscoso o conectada a un dispositivo amortiguador. En mecánica, se considera que las fuerzas de amortiguamiento que actúan sobre un cuerpo son proporcionales a alguna potencia de la velocidad instantánea y siempre actúan en dirección opuesta a la del movimiento. En particular, supondremos en el resto de la descripción que esta fuerza está expresada por: $\beta\frac{dx}{dt}$, donde β es una constante de amortiguamiento positiva [19, Cap. 5].

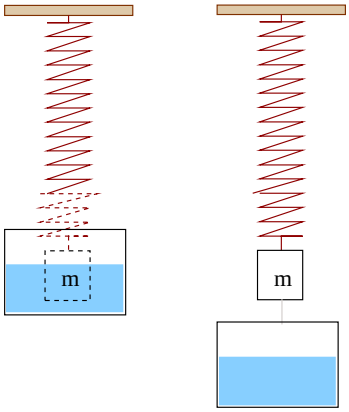


Figura 3.3: Sistema de resortes con amortiguamiento

Al igualar la segunda ley de Newton con la fuerza neta, o resultante, de la fuerza de restitución, la fuerza externa, la fuerza de amortiguamiento y el peso se obtiene

$$-kx - ks - \beta\frac{dx}{dt} + f_{\text{ext}} + mg = m\frac{d^2x}{dt^2}$$

pero $mg - ks = 0$ por lo que

$$-kx - \beta\frac{dx}{dt} + f_{\text{ext}} = m\frac{d^2x}{dt^2} \tag{3.1}$$

El signo negativo de la ecuación (3.1) indica que la fuerza de restitución del resorte y la fuerza de amortiguamiento actúa en la dirección opuesta al movimiento. Además, podemos adoptar la convención que los desplazamientos medidos abajo de la posición de equilibrio son positivos (Ver Figura 3.2)

3.3. Ecuación diferencial del movimiento amortiguado forzado

Al dividir la ecuación (3.1) por la masa m, obtendremos la ecuación diferencial de segundo orden:

$$\frac{d^2x}{dt^2} + 2\lambda\frac{dx}{dt} + \omega^2 x = F_{\text{ext}} \tag{3.2}$$

donde $F_{\text{ext}} = \frac{f_{\text{ext}}}{m}$, $2\lambda = \frac{\beta}{m}$ y $\omega^2 = \frac{k}{m}$. La ecuación (3.2) describe el movimiento amortiguado forzado [19, Cap. 5].

3.3.1. Solución de la ecuación diferencial

La solución general de la ecuación (3.2) es $x(t) = x_c(t) + x_p(t)$, donde x_c es la solución de su ecuación homogénea y x_p la solución particular correspondiente [19, Cap. 4].

Solución particular $x_p(t)$

Aplicando el método de los coeficientes indeterminados [19, Cap. 4], suponiendo que $x_p(t)$ es de la forma $x_p(t) = At^2 + Bt + C$. Obtenemos

$$\frac{dx}{dt} = x_p'(t) = 2At + B \qquad \text{y} \qquad \frac{d^2x}{dt^2} = x_p''(t) = 2A$$

de modo que

$$\begin{aligned} \frac{d^2x}{dt^2} + 2\lambda\frac{dx}{dt} + \omega^2 x = 2A + 2\lambda(2At + B) + \omega^2(At^2 + Bt + C) \qquad &= F_{\text{ext}} \\ \Downarrow \\ = A\omega^2 t^2 + (4A\lambda + B\omega^2)t + (2A + 2B\lambda + C\omega^2) &= F_{\text{ext}} \end{aligned} \tag{3.3}$$

Al igualar los coeficientes de la ecuación 3.3, obtenemos

$$A\omega^2 t^2 = 0, \qquad 4A\lambda + B\omega^2 = 0 \qquad \text{y} \qquad 2A + 2B\lambda + C\omega^2 \qquad = F_{\text{ext}};$$

por consiguiente $\quad A = 0, \quad B = 0, \quad C = \frac{F_{\text{ext}}}{\omega^2} \quad \text{y} \quad x_p(t) = \frac{F_{\text{ext}}}{\omega^2}$

Solución complementaria $x_c(t)$

Ahora resta resolver la ecuación homogénea (3.4) asociada a (3.2) para encontrar $x_c(t)$.

$$\frac{d^2x}{dt^2} + 2\lambda\frac{dx}{dt} + \omega^2 x = 0 \tag{3.4}$$

Observe que $x_c(t) = e^{mt}$ es una solución para la ecuación (3.4), entonces $x_c'(t) = me^{mt}$ y $x_c''(t) = m^2e^{mt}$, de modo que la ecuación (3.4) se transforma en

$$e^{mt}(m^2 + 2\lambda m + \omega^2) = 0 \tag{3.5}$$

Esta ecuación se llama ecuación auxiliar o ecuación característica de la ecuación diferencial (3.4). Las raíces correspondientes a la ecuación (3.5) son:

$$m_1 = -\lambda + \sqrt{\lambda^2 - \omega^2} \qquad \text{y} \qquad m_2 = -\lambda - \sqrt{\lambda^2 - \omega^2}$$

Por lo que

$$x_c(t) = e^{-\lambda t} e^{\pm\sqrt{\lambda^2-\omega^2}t} \tag{3.6}$$

El movimiento que describe la ecuación (3.6) es oscilatorio pero, a causa del coeficiente $e^{-\lambda t}$, las amplitudes de vibración tienden a cero cuando $t \to \infty$ [19, Cap. 5]. De acuerdo al signo algebraico de $\lambda^2 - \omega^2$ pueden presentarse los siguientes tres casos:

CASO 1: $\lambda^2 - \omega^2 > 0$. Aquí se dice que el sistema está sobreamortiguado porque el coeficiente de amortiguamiento, β, es grande comparado con la constante de resorte, k [19, Cap. 5]. La solución complementaria x_c correspondiente a 3.4 es:

$$x_c(t) = e^{-\lambda t}(c_1 e^{\sqrt{\lambda^2-\omega^2}t} + c_2 e^{-\sqrt{\lambda^2-\omega^2}t})$$

Entonces la solución general de (3.2) que describe al sistema sobreamortiguado forzado del resorte es:

$$x(t) = e^{-\lambda t}(c_1 e^{\sqrt{\lambda^2-\omega^2}t} + c_2 e^{-\sqrt{\lambda^2-\omega^2}t}) + F_{\text{ext}} \tag{3.7}$$

Valores iniciales
Al aplicar la condición inicial $x(0)$ en la ecuación (3.7), vemos que

$$x(0) = c_1 + c_2 + F_{\text{ext}} \qquad \Rightarrow \qquad c_1 = x(0) - c_2 - F_{\text{ext}}$$

Diferenciando la ecuación (3.7) y aplicando $x'(0)$ obtenemos:

$$x'(t) = e^{-\lambda t}\sqrt{\lambda^2-\omega^2}(c_1 e^{\sqrt{\lambda^2-\omega^2}t} - c_2 e^{-\sqrt{\lambda^2-\omega^2}t}) - \lambda e^{-\lambda t}(c_1 e^{\sqrt{\lambda^2-\omega^2}t} + c_2 e^{-\sqrt{\lambda^2-\omega^2}t})$$
$$x'(0) = c_1(\sqrt{\lambda^2-\omega^2} - \lambda) - c_2(\sqrt{\lambda^2-\omega^2} + \lambda)$$

Sustituyendo el valor de c_1,

$$x'(0) = (x(0) - c_2 - F_{\text{ext}})(\sqrt{\lambda^2-\omega^2} - \lambda) - c_2(\sqrt{\lambda^2-\omega^2} + \lambda)$$

Reduciendo términos

$$x'(0) = (x(0) - F_{\text{ext}})(\sqrt{\lambda^2-\omega^2} - \lambda) - 2c_2\sqrt{\lambda^2-\omega^2}$$

Finalmente

$$c_2 = \frac{(x(0)-F_{\text{ext}})(\sqrt{\lambda^2-\omega^2}-\lambda) - x'(0)}{2\sqrt{\lambda^2-\omega^2}}$$

CASO 2: $\lambda^2 - \omega^2 = 0$. Se dice que el sistema está críticamente amortiguado puesto que cualquier pequeña disminución de la fuerza de amortiguamiento originaría un movimiento oscilatorio [19, Cap. 5]. La solución general de la ecuación (3.4) es $x_c(t) = c_1 e^{m_1 t} + c_2 t e^{m_1 t}$, es decir, $x_c(t) = e^{-\lambda t}(c_1 + c_2 t)$, entonces $x(t) = e^{-\lambda t}(c_1 + c_2 t) + F_{\text{ext}}$

Valores inciales

$$\begin{aligned} x(0) &= c_1 + F_{\text{ext}} \qquad \Rightarrow \qquad c_1 = x(0) - F_{\text{ext}} \\ x'(t) &= -\lambda e^{-\lambda t}(c_1 + c_2 t) + c_2 e^{-\lambda t} \\ x'(0) &= -\lambda c_1 + c_2 \qquad \Rightarrow \qquad c_2 = x'(0) + \lambda c_1 \end{aligned}$$

CASO 3: $\lambda^2 - \omega^2 < 0$. Se dice que el sistema está subamortiguado porque el coeficiente de amortiguamiento es pequeño en comparación con la constante del resorte [19, Cap. 5]. Ahora las raíces m_1 y m_2 son complejas:

$$m_1 = -\lambda + \sqrt{\omega^2 - \lambda^2}i \qquad \text{y} \qquad m_2 = -\lambda - \sqrt{\omega^2 - \lambda^2}i$$

Entonces, la solución complementaria de la ecuación 3.4 es:

$$x_c(t) = e^{-\lambda t}(c_1 sin\sqrt{\omega^2 - \lambda^2}t + c_2 cos\sqrt{\omega^2 - \lambda^2}t)$$

y su solución general es:

$$x(t) = e^{-\lambda t}(c_1 sin\sqrt{\omega^2 - \lambda^2}t + c_2 cos\sqrt{\omega^2 - \lambda^2}t) + F_{\text{ext}}$$

Valores iniciales

$$\begin{aligned} x(0) &= c_2 + F_{\text{ext}} \qquad \Rightarrow \qquad c_2 = x(0) - F_{\text{ext}} \\ x'(t) &= e^{-\lambda t}\sqrt{\omega^2 - \lambda^2}(c_1 cos\sqrt{\omega^2 - \lambda^2}t - c_2 sin\sqrt{\omega^2 - \lambda^2}t) - \\ &\quad \lambda e^{-\lambda t}(c_1 sin\sqrt{\omega^2 - \lambda^2}t + c_2 cos\sqrt{\omega^2 - \lambda^2}t) \\ x'(0) &= \sqrt{\omega^2 - \lambda^2}c_1 - \lambda c_2 \qquad \Rightarrow \qquad c_1 = \frac{x'(0) + \lambda c_2}{\sqrt{\omega^2 - \lambda^2}} \end{aligned}$$

3.4. Soluciones numéricas a la ecuación diferencial de segundo orden

3.4.1. Diferencias finitas centrales

Al discretizar el tiempo t puede cuantificarse la evolución del sistema descrito por la ecuación (3.2) bajo las leyes de movimiento las cuales definen la velocidad dx/dt y la aceleración d^2x/dt^2 como:

$$\frac{dx}{dt} = \frac{x(t)-x(t-1)}{\Delta t} \qquad \text{y} \qquad \frac{d^2x}{dt^2} = \frac{V_f - V_i}{\Delta t}$$

donde V_f es la velocidad final, V_i es la velocidad inicial y ambas se calculan como:

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

Al substituir las expresiones anteriores en la Ec. (3.2), esquema que se conoce como de diferencias finitas, obtenemos:

$$x(t+1) = x(t)(2 - 2\lambda\Delta t - \Delta t^2\omega^2) - x(t-1)(1 - 2\lambda\Delta t) + \Delta t^2 F_{\text{ext}} \tag{3.8}$$

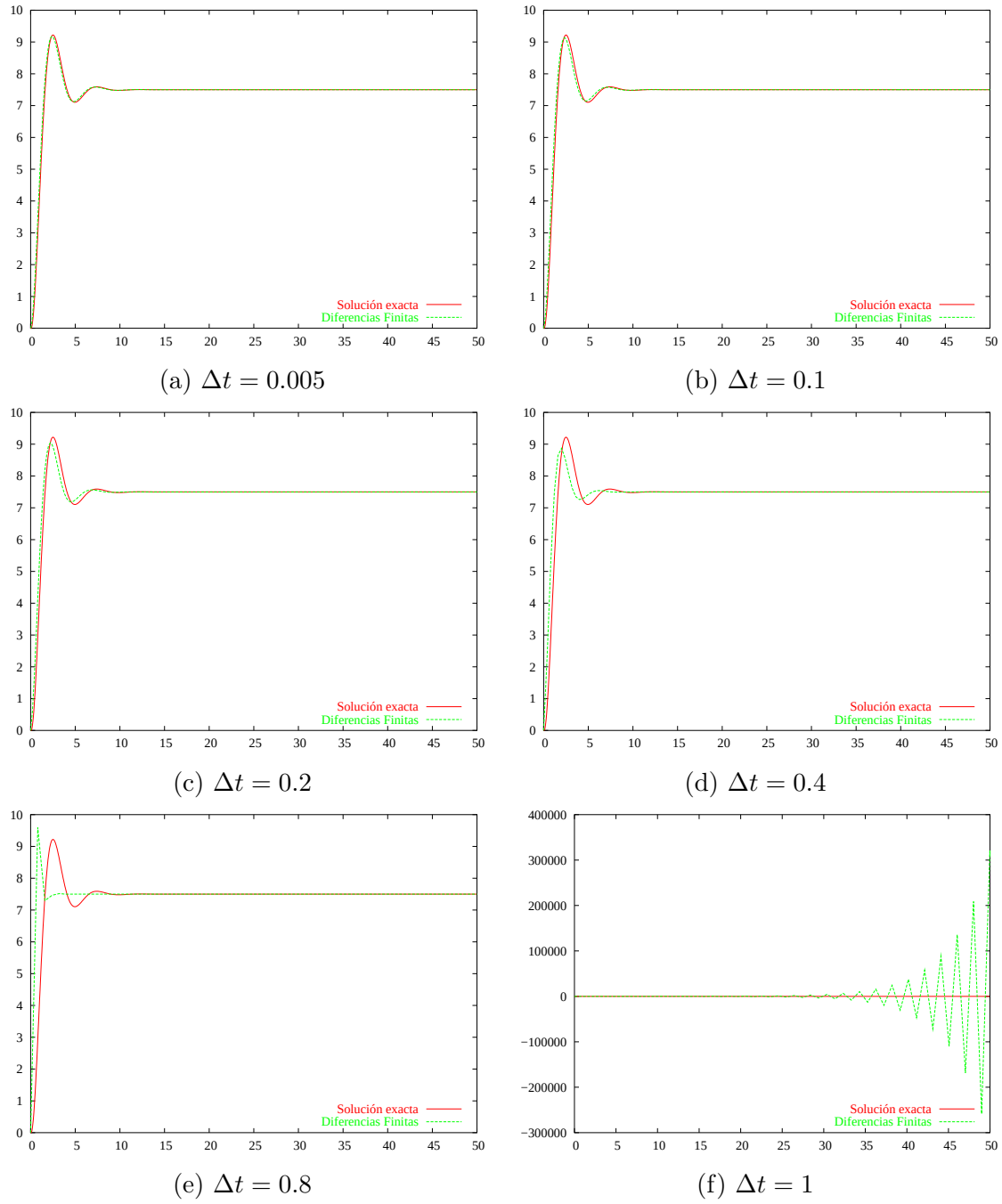
De esta forma el siguiente estado del sitema $x(t+1)$ es calculado partiendo de los valores conocidos del estado actual y el anterior. Este proceso es repetido para obtener una solución aproximada de la ecuación (3.2).

Tomando el valor de m y Δt igual a uno, la Ec. (3.8) cambia a la siguiente ecuación:

$$\begin{aligned} x(t+1) &= x(t)(2 - \beta - k) - x(t-1)(1-\beta) + F_{\text{ext}} \\ &= x(t) + x(t)(1-\beta) - x(t-1)(1-\beta) - kx(t) + F_{\text{ext}} \\ &= x(t) + [x(t) - x(t-1)](1-\beta) + F_{\text{int}} + F_{\text{ext}} \end{aligned}$$

la última expresión es la que viene indicada en [1, 13, 17], F_{int} es igual a $-kx(t)$.

No es conveniente que Δt sea igual uno pues elimina la posibilidad de calcular la evolución del sistema con respecto al tiempo, de hecho entre más pequeño sea el incremento de tiempo se obtienen mejores resultados como se muestra en la Figura 3.4(a)-(f). Sin embargo, es recomendable que m sea igual a uno, como lo hacen en [1], pues simplifica las operaciones.



(a) $\Delta t = 0.005$ (b) $\Delta t = 0.1$

(c) $\Delta t = 0.2$ (d) $\Delta t = 0.4$

(e) $\Delta t = 0.8$ (f) $\Delta t = 1$

Figura 3.4: Movimiento submortiguado. En las gráficas se muestra el error de discretización que se obtiene con el método de Diferencias finitas centrales para diferentes Δt.

3.4.2. Método de Euler, Heun y Runge-Kutta

Los métodos de discretización que se describen a continuación son válidos únicamente para ecuaciones diferenciales de primer orden. Así pues, la ecuación 3.2, que no es una EDO de primer orden sino de segundo, debe ser expresada como un sistema

de dos ecuaciones de primer orden y solucionar a cada una de ellas con los métodos de discretización. Por tanto haciendo un cambio de variable a partir de $\frac{d^2x}{dt^2} = \frac{dv}{dt}$, podemos deducir:

$$\begin{aligned} &\tfrac{dx(t)}{dt} = v(t) \\ &\qquad\qquad\qquad\qquad \text{con} \quad \begin{cases} x(t_0) = x_0 \\ v(t_0) = v_0 \end{cases} \\ &\tfrac{dv(t)}{dt} = -2\lambda v(t) - \omega^2 x(t) + Fext \end{aligned} \tag{3.9}$$

Método de Euler

Sea $[a, b]$ el intervalo sobre el cual queremos encontrar una solución para $y' = f(t, y)$ con $y(a) = y_0$. En lugar de encontrar la función diferencial que satisfaga y' se genera un conjunto de puntos (t_k, y_k) que son usados como una aproximación. Por conveniencia se subdivide el intervalo $[a, b]$ en M subintervalos iguales y se obtiene el conjunto de puntos

$$t_k = a + hk \qquad \text{para} \qquad k = 0, 1, ..., M$$

donde $h = \frac{b-a}{M}$ y es lo que se llama paso (delta o incremento).

Ahora procedemos a resolver aproximadamente

$$y' = f(t, y) \qquad \text{sobre} \qquad [t_0, t_k] \qquad \text{con} \qquad y(t_0) = y_0$$

Asumimos que $y(t)$, $y'(t)$ y $y''(t)$ son continuas y haciendo uso del teorema de Taylor para expandir $y(t)$ con $t = t_0$. Para cada valor de t, existe un valor c, que se encuentra entre t_0 y t tal que

$$y(t) = y(t_0) + y'(t_0)(t - t_0) + \frac{y''(c_1)(t - t_0)^2}{2} \tag{3.10}$$

Cuando $y'(t_0) = f(t_0, y(t_0))$ y $h = t_1 - t_0$ son sustituidos en la ecuación (3.10), el resultado es:

$$y(t_1) = y(t_0) + hf(t_0, y(t_0)) + y''(c_1)\tfrac{h^2}{2}$$

Si h es elegido lo suficientemente pequeño entonces podemos eliminar el término de segundo orden y obtener:

$$y_1 \approx y_0 + hf(t_0, y_0)$$

El proceso es repetido y genera una secuencia de puntos que es una aproximación a la solución de $y = y(t)$ [20, Cap. 9]. La fórmula general para este método es:

$$t_{k+1} = t_k + h, \qquad y_{k+1} = y_k + hf(t_k, y_k) \qquad \text{para} \qquad k = 0, 1, ..., M-1$$

Observe que se evalúa un nuevo estado del sistema partiendo de los valores conocidos del estado anterior. De esta forma sabemos cual es el valor de nuestro sistema un tiempo h después del tiempo anterior t_0, gracias a conocer el estado anterior y_0 y la derivada $y'(t_0)$.

El método de Euler es una de las técnicas más simples para aproximar soluciones de una ecuación diferencial.

Error global de truncación
Si el método de Euler es usado para resolver $y' = f(t, y), y(a) = y_0$ sobre $[a, b]$ y un incremento de tiempo h, el error global de truncación es de orden $O(h)$, esto es,

$$E(y(b), h) = y(b) - y_M \approx Ch$$

donde C es una constante que depende de $f(x, y)$ en el intervalo $[a, b]$ pero no depende de h.

Método de Heun

Sea

$$y'(t) = f(t, y(t)) \qquad \text{sobre} \qquad [a, b] \qquad \text{con} \qquad y(t_0) = y_0.$$

Para obtener el punto solución (t_1, y_1) podemos utilizar el teorema fundamental del cálculo e integrar $y'(t)$ sobre $[t_0, t_1]$ y obtener

$$\int_{t_0}^{t_1} f(t, y(t))dt = \int_{t_0}^{t_1} y'(t)dt = y(t_1) - y(t_0). \tag{3.11}$$

Despejando $y(t_1)$ de (3.11) el resultado es

$$y(t_1) = y(t_0) + \int_{t_0}^{t_1} f(t, y(t))dt. \tag{3.12}$$

Aplicando un método de integración numérica podemos aproximar la integral de (3.12). Si utilizamos la regla trapezoidal con un incremento $h = t_1 - t_0$ entonces

$$y(t_1) \approx y(t_0) + \frac{h}{2}[f(t_0, y(t_0)) + f(t_1, y(t_1))]. \tag{3.13}$$

Note que la parte derecha de (3.13) implica también determinar el valor de $y(t_1)$. Para proceder, usaremos un valor estimado para $y(t_1)$. La solución de Euler será suficiente para este propósito. Después de sustituir en (3.13) la fórmula resultante para encontrar (t_1, y_1) es llamda el método de Heun's

$$y_1 = y(t_0) + \tfrac{h}{2}[f(t_0, y_0) + f(t_1, y_0 + hf(t_0, y_0))].$$

Este proceso es repetido y genera una secuencia de puntos que es una aproximación a la solución de la curva $y = y(t)$. En cada paso el método de Euler es usado como predicción y la regla trapezoidal es usada para corregir el resultado y obtener el valor final [20, Cap. 9]. La fórmula general para el método de Heun's es:

$$\begin{aligned} p_{k+1} &= y_k + hf(t_k, y_k), \qquad t_{k+1} = t_k + h, \\ y_{k+1} &= y_k + \frac{h}{2}[f(t_k, y_k) + f(t_{k+1}, p_{k+1})]. \end{aligned}$$

Error global de truncación
Si el método de Heun es usado para resolver $y' = f(t, y), y(a) = y_0$ sobre $[a, b]$ y un incremento de tiempo h, el error global de truncación es de orden $O(h^2)$, esto es,

$$E(y(b), h) = y(b) - y_M \approx Ch^2$$

donde C es una constante que depende de $f(x, y)$ en el intervalo $[a, b]$ pero no depende de h.

Método de Runge-Kutta, de orden cuatro (RK4)

Suponga que se han calculado aproximaciones $y_1, y_2, y_3, \ldots, y_n$ de los valores reales $y(t_1), y(t_2), y(t_3), \ldots, y(t_n)$ y ahora es necesario calcular $y_{t+1} \approx y(t_{n+1})$. Entonces por el teorema fundamental del cálculo

$$y(t_{n+1}) - y(t_n) = \int_{t_n}^{t_{n+1}} y'(t)dt = \int_{t_n}^{t_n+h} y'(t)dt \tag{3.14}$$

Podemos utilizar la regla de Simpson's como se menciona en [21] para intregrar numericamente la parte derecha de (3.14) y obtener:

$$y(t_{n+1}) - y(t_n) \approx \tfrac{h}{6}\left[y'(t_n) + 4y'(t_n + \tfrac{h}{2}) + y'(t_{n+1})\right]$$

Por lo tanto

$$y(t_{n+1}) - y(t_n) \approx \frac{h}{6}\left[y'(t_n) + 2y'(t_n + \frac{h}{2}) + 2y'(t_n + \frac{h}{2}) + y'(t_{n+1})\right] \tag{3.15}$$

Expresamos el término $4y(t_n + \frac{h}{2})$ en la suma de dos términos porque podemos aproximar la pendiente de $y'(t + \frac{h}{2})$ en el punto medio $t_n + \frac{h}{2}$ del intervalo $[t_n, t_{n+1}]$ de dos formas.

En la parte derecha de (3.15) remplazamos las pendientes reales $y'(t_n), y'(t_n + \frac{h}{2}), y'(t_n + \frac{h}{2})$, y $y'(t_{n+1})$ respectivamente por los siguientes valores estimados:

- $f_1 = f(t_n, y_n)$. Esta es la pendiente en x_n del método de Euler.
- $f_2 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}f_1\right)$. Esta es la pendiente estimada en el punto medio del intervalo $[t_n, t_{n+1}]$ usando el método de Euler para predecir la ordenada ahí.
- $f_3 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}f_2\right)$. Este es un valor mejorado de Euler para la pendiente en el punto medio.
- $f_4 = f(t_n + h, y_n + hf_3)$. Este es la pendiente del método de Euler en x_{n+1}, usando la pendiente mejorada k_3 en el punto medio para el paso en x_{n+1}.

Cuando estos valores son sustituídos en (3.15), el resultado es la fórmula iterativa

$$y_{n+1} = y_n + \frac{h}{6}(f_1 + 2f_2 + 2f_3 + f_4) \tag{3.16}$$

El uso de esta fórmula para calcular la aproximación $y_1, y_2, y_3, ...$ sucesivamente constituye el método de Runge-Kutta. Note que la ecuación (3.16) puede tomar la forma del método de Euler $y_{n+1} = y_n + hf$, si definimos f como $\frac{1}{6}(f_1 + 2f_2 + 2f_3 + f_4)$.

Este método es considerablemente más exacto, mejora mucho comparado con el método de Euler. De hecho Euler es un Runge-Kutta de orden 2. El problema es que Euler avanza el sistema un intervalo de tiempo h usando la información de la derivada sólo al principio del intervalo. Con el RK4 evaluamos cuatro derivadas distintas al principio del intervalo, otra al final y dos en el medio. Después promediamos todo. En otras palabras, nos estamos quedando con más términos de la serie de Taylor y por tanto el error disminuye al aproximar la función [22].

Error global de truncación
Si el método de RK4 es usado para resolver $y' = f(t, y), y(a) = y_0$ sobre $[a, b]$ y un incremento de tiempo h, el error global de truncación es de orden $O(h^4)$, esto es,

$$E(y(b), h) = y(b) - y_M \approx Ch^4$$

donde C es una constante que depende de $f(x, y)$ en el intervalo $[a, b]$ pero no depende de h.

3.5. Análisis de eficiencia de los métodos de discretización

A continuación se presenta el análisis de eficiencia de los métodos de discretización Diferencias finitas centrales, Euler, Heun y RK4 para encontrar la solución de la ecuación 3.2, con $x(a) = x_0$ sobre un intervalo de tiempo $[a, b]$ dividido en M subintervalos iguales.

3.5.1. Diferencias finitas centrales

Para este método, la solución de la ecuación 3.2 es calculada partiendo de los valores del estado actual y el anterior, como sigue:

$$x(t+1) = x(t)\underbrace{(2 - 2\lambda\Delta t - \Delta t^2\omega^2)}_{\text{constante}} - x(t-1)\underbrace{(1 - 2\lambda\Delta t)}_{\text{constante}} \qquad \begin{array}{cc} \text{sumas} & \text{productos} \\ 2 & 2 \end{array}$$

Observe que el número total de operaciones para resolver la ecuación 3.2, sobre un intervalo de tiempo $[a, b]$ dividido en M subintervalos iguales, con el método de Diferencias finitas son: $2M$ sumas y $2M$ productos.

3.5.2. Euler

Como se describe en la sección 3.4.2, encontrar la solución de la ecuación 3.2 de segundo orden con el método de Euler, implica calcular la solución del sistema de ecuaciones diferenciales 3.9, como se muestra a continuación:

$$\begin{array}{lcccll} & & & & \text{sumas} & \text{productos} \\ f(t,x,v) & = & \frac{dx}{dt} & = & v & 0 \quad 0 \\ g(t,x,y) & = & \frac{dv}{dt} & = & -2\lambda y - \omega^2 x + F_{\text{ext}} & 2 \quad 2 \end{array}$$

Total: 2 sumas y 2 productos.

$$\begin{array}{lll} & & \text{sumas} \quad \text{productos} \\ x_{k+1} & = x_k + hf(t_k, x_k, v_k) = x_k + hv_k & 1 \quad 1 \\ v_{k+1} & = v_k + hg(t_k, x_k, v_k) & 1 \quad 1 \end{array}$$

Además, para evaluar $g(t_k, x_k, v_k)$ se requieren: 2 2

Así, el total de operaciones para resolver la ecuación 3.2 en M subintervalos de tiempo utilizando el método de Heun son: 4 sumas y 4 productos.

3.5.3. Heun

Para el método de Heun al igual que en el caso anterior, se resuelve el sistema de ecuaciones diferenciales 3.9 para hallar la solución de la ecuación 3.2, pero el número de operaciones se incrementa para este método, pues se realiza una predicción del valor de $f(t_{k+1}, p_{k+1})$ usando el método de Euler (Véase la sección 3.4.2), como se muestra a continuación:

		sumas	productos
p_{k+1}	$= x_k + hf(t_k, x_k, v_k) = x_k + hv_k$	1	1
q_{k+1}	$= v_k + hg(t_k, x_k, v_k)$	1	1
Para	evaluar $g(t_k, x_k, v_k)$ se requieren:	2	2
x_{k+1}	$= x_k + \frac{h}{2}[f(t_k, x_k, v_k) + f(t_k + h, p_{k+1}, q_{k+1})] = x_k + \frac{h}{2}(v_k + q_{k+1})$	2	1
v_{k+1}	$= v_k + \frac{h}{2}[g(t_k, x_k, v_k) + g(t_k + h, p_{k+1}, q_{k+1})]$	3	1
Para	evaluar $g(t_k, x_k, v_k)$ se requieren:	2(2)	2(2)

Para este método, el número total de operaciones son: $13M$ sumas y $10M$ productos.

3.5.4. RK4

Con el método de RK4, se obtiene una solución más exacata de la ecuación 3.2 pero se evalúan más ecuaciones que en cualquiera de los otros métodos. Además, también se debe resolver el sistema de ecuaciones diferenciales 3.9, incrementando más el número de operaciones en comparación con el método de Diferencias finitas, como se muestra a continuación:

		sumas	productos
f_1	$= f(t_k, x_k, v_k) = v_k$	0	0
g_1	$= g(t_k, x_k, v_k)$	2	2
f_2	$= f(t_k + \frac{h}{2}, x_k + \frac{h}{2}f_1, \frac{h}{2}g_1) = v_k + \frac{h}{2}g_1$	1	1
g_2	$= g(t_k + \frac{h}{2}, x_k + \frac{h}{2}f_1, \frac{h}{2}g_1)$	2	1
Para evaluar	$g(t_k, x_k, v_k)$ se requieren:	2(2)	2(2)
f_3	$= f(t_k + \frac{h}{2}, x_k + \frac{h}{2}f_2, \frac{h}{2}g_2) = v_k + \frac{h}{2}g_2$	1	1
g_3	$= g(t_k + \frac{h}{2}, x_k + \frac{h}{2}f_2, \frac{h}{2}g_2)$	2	1
Para evaluar	$g(t_k, x_k, v_k)$ se requieren:	2(2)	2(2)
f_4	$= f(t_k + h, x_k + hf_3, v_k + hg_3) = y_k + hg_3$	1	1
g_4	$= g(t_k + h, x_k + hf_3, v_k + hg_3)$	2	1
Para evaluar	$g(t_k, x_k, v_k)$ se requieren:	2(2)	2(2)
x_{k+1}	$= x_k + \frac{h}{6}(f_1 + 2f_2 + 2f_3 + f_4)$	4	3
y_{k+1}	$= y_k + \frac{h}{6}(g_1 + 2g_2 + 2g_3 + g_4)$	4	3

El total de operaciones para este método son: $25M$ sumas y $20M$ productos.

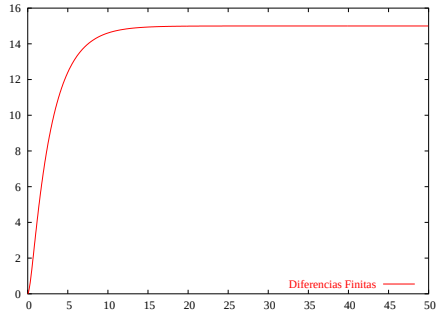
3.6. Restricciones para los valores de k y β

Recuerde que el signo de $\lambda^2 - \omega^2$, dónde $2\lambda = \frac{\beta}{m}$ y $\omega^2 = \frac{k}{m}$, determina si el movimiento del resorte es sobreamortiguado, críticamente amortiguado o subamortiguado, por lo cual la elección de los valores de k, m y β no se debe hacer de forma arbitraria. Para que el sistema sea estable k y β deberan ser mayores a cero y cumplir con las restricciones que se presentan en la tabla 3.1.

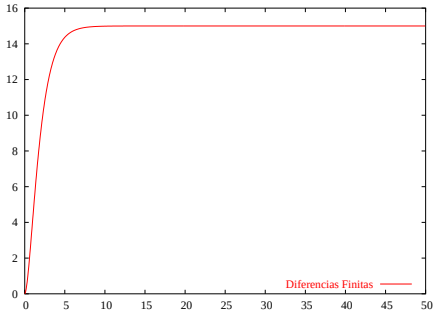
Movimiento	**Restricciones**	**Si** $m = k = 1$
sobreamortiguado	$\beta^2 - 4mk > 0 \quad \Rightarrow \quad \beta > 2\sqrt{mk}$	$\beta > 2$
críticamente amortiguado	$\beta^2 - 4mk = 0 \quad \Rightarrow \quad \beta = 2\sqrt{mk}$	$\beta = 2$
subamortiguado	$\beta^2 - 4mk < 0 \quad \Rightarrow \quad \beta < 2\sqrt{mk}$	$\beta < 2$

Cuadro 3.1: Restricciones para β y k.

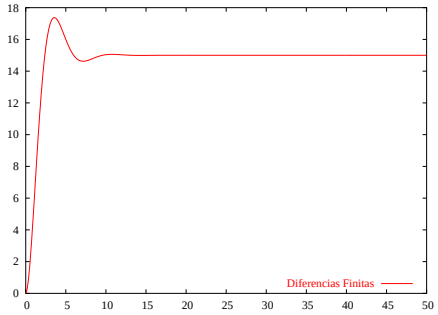
Como ejemplo de este análisis se presentan las gráficas 3.5(a) -(c) de un sistema amortiguado forzado al cuál le fue aplicado una fuerza externa de 15 newtons.



(a) Movimiento sobreamortiguado ($k = 1, m = 1$ y $\beta = 3$)



(b) Movimiento críticamente amortiguado ($k = 1, m = 1$ y $\beta = 2$)



(c) Movimiento subamortiguado ($k = 1, m = 1$ y $\beta = 1$)

Figura 3.5: Solución exacta de los tres tipos de movimiento del resorte: sobreamortiguado, críticamente amortiguado y subamortiguado, los cuales se obtuvieron al variar β.

3.7. Resultados obtenidos

En las figuras (3.6), (3.7) y (3.8) se muestran las gráficas comparativas de la solución exacta de la ecuación (3.2) y los resultados obtenidos al discretizar ésta con los cuatro métodos mencionados en la sección anterior. Las gráficas se obtuvieron para los tres tipos de movimiento que puede tener el resorte: movimiento sobreamortiguado, críticamente amortiguado y subamortiguado. Para el sistema se consideró una fuerza externa de 15 newtons y $h = 0.05$.

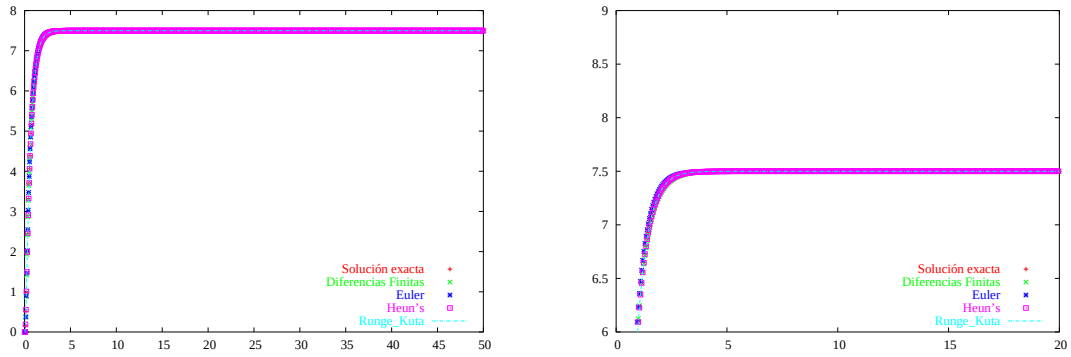


Figura 3.6: Movimiento sobreamortiguado ($k = 2, m = 0.1$ y $\beta = 1.2$)

k	t_k	**Sol. exacta**	**Diferencias finitas**	**Euler**	**Heun**	**RK4**
0	0	0	0	0	0	0
20	1	6.0979345419	6.1222343970	6.2335800909	6.0932475585	6.0979331883
40	2	7.3102320865	7.2871331051	7.3460324693	7.3088977419	7.3102314162
60	3	7.4743177048	7.4671117412	7.4812811427	7.4740440188	7.4743175675
80	4	7.4965242793	7.4949187140	7.4977242239	7.4964745948	7.4965242544
100	5	7.4995296123	7.4992149336	7.4997233187	7.4995211708	7.4995296081
120	6	7.4999363399	7.4998787060	7.4999663620	7.4999349642	7.4999363392
140	7	7.4999913845	7.4999812599	7.4999959104	7.4999911666	7.4999913844
160	8	7.4999988340	7.4999971046	7.4999995028	7.4999988002	7.4999988340
180	9	7.4999998422	7.4999995526	7.4999999395	7.4999998370	7.4999998421
200	10	7.4999999786	7.4999999308	7.4999999926	7.4999999778	7.4999999786
220	11	7.4999999971	7.4999999893	7.4999999991	7.4999999969	7.4999999971
240	12	7.4999999996	7.4999999983	7.4999999998	7.4999999995	7.4999999996
260	13	7.4999999999	7.4999999997	7.4999999999	7.4999999999	7.4999999999
280	14	7.4999999999	7.4999999999	7.4999999999	7.4999999999	7.4999999999

Cuadro 3.2: Movimiento sobreamortiguado

Puede observarse en las tablas presentadas y en las figuras (3.6), (3.7) y (3.8), que el error al discretizar la ecuación (3.2) usando cualquiera de los cuatro métodos es considerablemente pequeño e inlcuso despreciable para los tres tipos de movimiento.

k	t_k	**Diferencias finitas**	**Euler**	**Heun**	**RK4**
0	0	0	0	0	0
20	1	-0.02429985511370	-0.13564554899142	0.004686983455830	1.35365136966e-06
40	2	0.023098981383330	-0.03580038279613	0.001334344651410	6.70311590234e-07
60	3	0.007205963534690	-0.00696343788487	0.000273685937820	1.37258670207e-07
80	4	0.001605565254209	-0.00119994464463	4.96844501398e-05	2.48728992957e-08
100	5	0.000314678755629	-0.00019370640587	8.44154578949e-06	4.21837942354e-09
120	6	5.76339066595e-05	-3.0022065719e-05	1.37571224989e-06	6.86229739699e-10
140	7	1.01246496306e-05	-4.5258572294e-06	2.17866160667e-07	1.08480335825e-10
160	8	1.72939951070e-06	-6.6877535953e-07	3.37883401257e-08	1.67901248460e-11
180	9	2.89542180453e-07	-9.7349680316e-08	5.15723996841e-09	2.54996024295e-12
200	10	4.77590900160e-08	-1.4006530157e-08	7.77339970170e-10	3.80140363631e-13
220	11	7.78815945068e-09	-1.9966899245e-09	1.15989884363e-10	5.95079541199e-14
240	12	1.25868027112e-09	-2.8251978534e-10	1.71604952470e-11	1.06581410364e-14
260	13	2.01960226320e-10	-3.9739767032e-11	2.51976217668e-12	0
280	14	3.22204485314e-11	-5.5599969073e-12	3.70370401014e-13	0

Cuadro 3.3: Movimiento sobreamortiguado: Error de discretización.

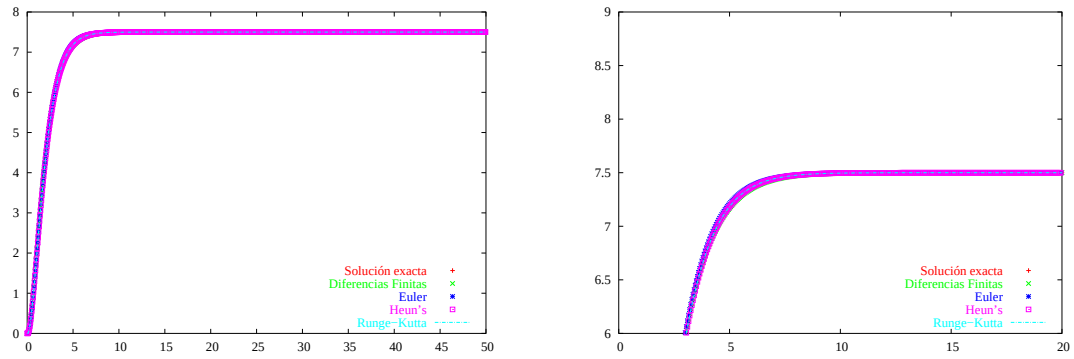


Figura 3.7: Movimiento críticamente amortiguado ($k = 2, m = 2$ y $\beta = 4$)

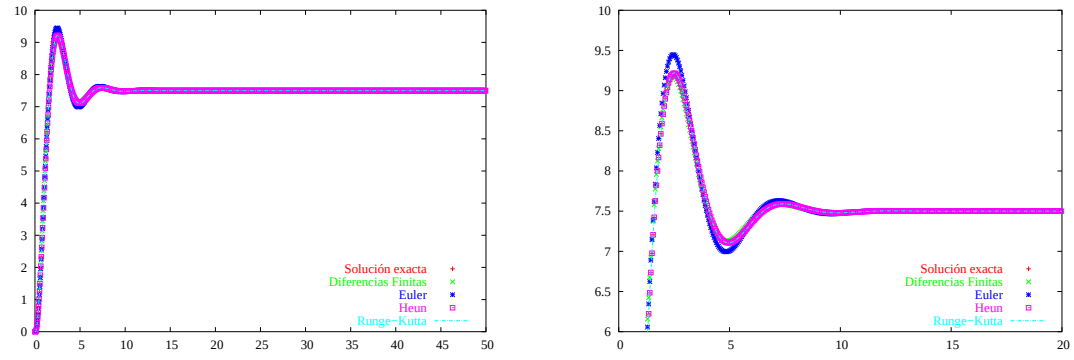


Figura 3.8: Movimiento subamortiguado ($k = 2, m = 1$ y $\beta = 1.2$)

k	t_k	**Sol. exacta**	**Diferencias finitas**	**Euler**	**Heun**	**RK4**
0	0	0	0	0	0	0
20	1	1.8439125319	1.9590664926	1.8396959619	1.8452099722	1.8439129945
40	2	4.3521861661	4.4386363478	4.4028926257	4.3522661703	4.3521864024
60	3	5.9494362112	5.9784629923	6.0068790910	5.9489797935	5.9494362799
80	4	6.7851691965	6.7811551625	6.8274902951	6.7846975090	6.7851691994
100	5	7.1839028765	7.1695276204	7.2097136642	7.1835715656	7.1839028635
120	6	7.3641705629	7.3504258215	7.3782985428	7.3639736887	7.3641705509
140	7	7.4428413789	7.4329238829	7.4500488047	7.4427351272	7.4428413710
160	8	7.4763275521	7.4700870456	7.4798265161	7.4762737243	7.4763275477
180	9	7.4903183644	7.4867054702	7.4919555158	7.4902922943	7.4903183621
200	10	7.4960803680	7.4941037251	7.4968248692	7.4960681604	7.4960803669
220	11	7.4984263628	7.4973883143	7.4987573346	7.4984207946	7.4984263623
240	12	7.4993726470	7.4988441108	7.4995170868	7.4993701601	7.4993726467
260	13	7.4997513880	7.4994886767	7.4998134624	7.4997502963	7.4997513879
280	14	7.4999019845	7.4997738790	7.4999283180	7.4999015121	7.4999019845

Cuadro 3.4: Movimiento críticamente amortiguado.

k	t_k	**Diferencias finitas**	**Euler**	**Heun**	**RK4**
0	0	0	0	0	0
20	1	-0.11515396071224	0.004216570007529	-0.00129744031426	-4.6260450003e-07
40	2	-0.08645018171358	-0.05070645956796	-8.0004249760e-05	-2.3633582024e-07
60	3	-0.02902678104965	-0.05744287972691	0.000456417684169	-6.8648260409e-08
80	4	0.004014034006310	-0.04232109853671	0.000471687588680	-2.8408502217e-09
100	5	0.014375256130220	-0.02581078772511	0.000331310924030	1.29698598527e-08
120	6	0.01374474140471	-0.01412797985014	0.000196874198939	1.20496395084e-08
140	7	0.009917496042720	-0.00720742580008	0.000106251724430	7.89116061383e-09
160	8	0.006240506573310	-0.00349896391767	5.38278870001e-05	4.46250059127e-09
180	9	0.003612894183779	-0.00163715143558	2.60700345204e-05	2.32105001884e-09
200	10	0.001976642914760	-0.00074450122698	1.22076157698e-05	1.14266995865e-09
220	11	0.001038048487940	-0.00033097180299	5.56821303021e-06	5.40910427559e-10
240	12	0.000528536116819	-0.00014443984036	2.48690779969e-06	2.48599363317e-10
260	13	0.000262711349009	-6.2074377420e-05	1.09171114992e-06	1.11639586464e-10
280	14	0.000128105524740	-2.6333477869e-05	4.72383860028e-07	4.92104135219e-11

Cuadro 3.5: Movimiento críticamente amortiguado: Error de discretización.

Podemos concluir que el método de diferencias finitas es el más simple y que el error que ofrece en la solución numérica es un pequeño. Además, es el más eficiente de los cuatro métodos presentados, como se muestra en la tabla 3.8. Esta tabla fue obtenida del análisis de eficiencia realizado en la sección 3.5.

k	t_k	**Sol. exacta**	**Diferencias finitas**	**Euler**	**Heun**	**RK4**
0	0	0	0	0	0	0
20	1	4.1650410891	4.3928855559	4.1784504724	4.1698099702	4.1650407882
40	2	8.7058591931	8.7687171457	8.9640297281	8.7047540110	8.7058605102
60	3	8.8831681971	8.7717779501	8.9868907943	8.8781885958	8.8831691043
80	4	7.5711836355	7.5032154558	7.4496501274	7.5697656646	7.5711832134
100	5	7.1057528356	7.1319879877	7.0060730456	7.1077092136	7.1057522109
120	6	7.3547472994	7.3916310910	7.3760866542	7.3561275384	7.3547472551
140	7	7.5731286187	7.5751257573	7.6241692303	7.5728079542	7.5731288666
160	8	7.5667151817	7.5533704365	7.5752057210	7.5660453032	7.5667152886
180	9	7.4989258642	7.4937863195	7.4812857456	7.4988135396	7.4989258058
200	10	7.4795684430	7.4827029040	7.4704299112	7.4797815043	7.4795683823
220	11	7.4939070303	7.4967529763	7.4976461111	7.4940219241	7.4939070308
240	12	7.5042403807	7.5040700712	7.5088707592	7.5042027282	7.5042404028
260	13	7.5031668510	7.5021301835	7.5032764540	7.5031131728	7.5031668583
280	14	7.4997173656	7.4994411249	7.4980681951	7.4997124512	7.4997173603

Cuadro 3.6: Movimiento subamortiguado.

k	t_k	**Diferencias finitas**	**Euler**	**Heun**	**RK4**
0	0	0	0	0	0
20	1	-0.22784446679293	-0.01340938329773	-0.00476888110318	3.00936910058e-07
40	2	-0.06285795263126	-0.25817053499175	0.001105182117671	-1.3170678805e-06
60	3	0.111390247004509	-0.10372259728183	0.004979601221698	-9.0719944978e-07
80	4	0.067968179736210	0.121533508112241	0.001417970888669	4.22076250394e-07
100	5	-0.02623515218478	0.099679789938120	-0.00195637801562	6.24602780163e-07
120	6	-0.03688379167479	-0.02133935480479	-0.00138023903993	4.42876304518e-08
140	7	-0.00199713861102	-0.05104061161146	0.000320664548949	-2.4789161034e-07
160	8	0.013344745249770	-0.00849053923610	0.000669878571559	-1.0687425966e-07
180	9	0.005139544660339	0.017640118635469	0.000112324557900	5.83639394591e-08
200	10	-0.00313446103460	0.009138531730630	-0.00021306135640	6.06650401024e-08
220	11	-0.00284594600484	-0.00373908078585	-0.00011489383537	-5.4064042132e-10
240	12	0.000170309474849	-0.00463037855586	3.76524290901e-05	-2.2130019594e-08
260	13	0.001036667423799	-0.00010960301933	5.36781623994e-05	-7.3388903842e-09
280	14	0.000276240684240	0.001649170521820	4.91439159944e-06	5.29824983885e-09

Cuadro 3.7: Movimiento subamortiguado: Error de discretización.

Método	**No. de Sumas**	**No. Productos**
Diferencias finitas centrales	$2M$	$2M$
Euler	$4M$	$4M$
Heun	$13M$	$10M$
RK4	$25M$	$20M$

Cuadro 3.8: Número de operaciones realizadas para M subintervalos.

Capítulo 4

Ejemplos de Animación de Modelos Deformables

Para incorporar un comportamiento elástico a las mallas de simplejos, se utilizaron las ecuaciones propuestas en el capítulo anterior para describir la deformación de los objetos, la solución de las ecuaciones se calculó con el método de Diferencias finitas. Para probar el sistema, se realizó la animación de la deformación de una esfera a un cubo, el rebote de una pelota contra una pared, una pelota comprimida con dos paredes y la deformación general de una esfera. Estas animaciones fueron realizadas usando una versión libre de OpenGL llamada Mesa [23] y para la interfaz se utlizó Qt.

4.1. OpenGL

OpenGL es una interfaz de software (API) para hardware gráfico escrita originalmente en *C*. Esta librería se concibió para programar en máquinas nativas *Silicon Graphics* bajo el nombre de GL (Graphics Library). Posteriormente se consideró la posibilidad de extenderla a cualquier tipo de plataforma, con lo que se llegó al término *Open Graphics Library*, es decir OpenGL (los creadores de Mesa pidieron permiso a Silicon Graphics para usar la misma especificación del API).

La interfaz consiste de un conjunto de procedimientos y funciones que permiten producir imágenes en color de alta calidad, especialmente de objetos tridimensionales. OpenGL tiene incorporadas las funciones para el dibujado de un pequeño grupo de primitivas: puntos, líneas, triángulos, cuadriláteros y polígonos en general, a partir de las cuales es posible construir objetos mucho más elaborados. OpenGL también proporciona otras funciones para la manipulación de los objetos dibujados:

- para realizar transfomaciones geométricas (rotación, translación y escalamiento),
- para el dibujado de objetos con color y con textura,

- para dar efectos de iluminación a la escena,
- para dar efectos de sombreado a los objetos.

Toda la geometría que se despliega en las aplicaciones de OpenGL es una geometría proyectiva que está basada en los conceptos de matrices y vectores y las operaciones aritméticas aplicables a estas estructuras.

En OpenGL existen básicamente tres matrices principales: Una matriz de proyección llamada GL_PROJECTION, la cual nos permite determinar la perspectiva que usaremos para observar la escena generada, así como el tipo de proyección a usar. Esta matriz tiene una gran relación con otro concepto también muy interesante llamado clipping, que consiste en recortar u ocultar aquello que "está pero no se ve", es decir lo que queda fuera del foco de la cámara o del campo visual activo.

Una matriz de modelado llamda GL_MODELVIEW, esta es la matriz que se usa para aplicar a la escena operaciones de rotación, traslación o escalamiento, o bien manipular la posición y orientación de la cámara. Y una última matriz llamada GL_VIEWPORT que es la que define la posición y tamaño final de la imagen.

Podemos hacer uso de cada una de estas matrices mediante el procedimiento *GLMatrixMode()*, el cual nos permite seleccionar una de estas matrices para su configuración.

En la figura 4.1 podemos observar el orden en como se aplican las matrices y como se transforman las coordenadas hasta llegar a las coordenadas reales de la venta.

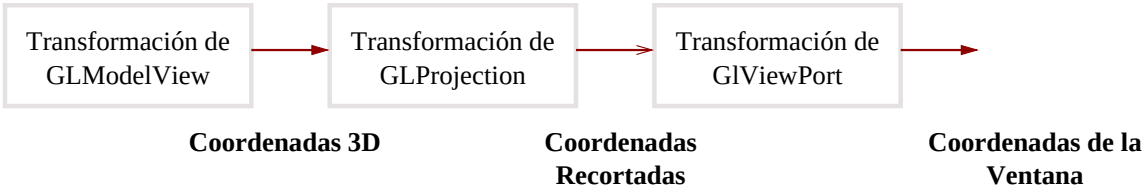


Figura 4.1: Secuencia de transformación de coordenadas

4.1.1. Qt

Qt es una caja de herramientas de C++ para el desarrollo de Interfaces Gráficas de Usuario (GUI, por sus siglas en inglés) multiplataforma. Qt sigue una filosofía del "mismo código, distintos compiladores", del mismo código fuente se generan varios ejecutables dependiendo del compilador que se use en la plataforma. Qt es un producto de *Troll Tech* [24].

Qt permite el desarrollo de prototipos rápidos y es gratuito bajo Linux, además tiene una documentación excelente. Qt introduce el mecanismo de señales−ranuras para el intercambio de mensajes entre objetos que es bastante intuitivo, en comparación al mecanismo de retrollamadas del desarrollo en XWindows y Motif. Las señales son enviadas por un objeto a la ranura de otro. Este mecanismo provoca un sobreuso del procesador porque hay que usar un metacompilador (llamado MOC, Meta Object Compiler, en Qt) para traducir el código de C++ con señales y ranuras a código C++ simple, esta es la única desventaja de Qt pero sólo afecta en tiempo de compilación y no en tiempo de ejecución.

Los componentes para diseñar una interfaz gráfica se llaman widgets. Widgets comunes son: botones, barras de scroll, cuadros de diálogo, menús, botones con imágenes (o iconos), ventanas, etcétera. La librería de Qt provee toda la funcionalidad para el dibujo de widgets, sus estilos de dibujo, y para controlar los eventos de teclado y ratón, además de poder diseñar nuevos widgets inexistentes, como podráa ser una manija circular (como las que controlan el volumen en un radio). El programador se enfoca a la funcionalidad y la comunicación entre los diferentes widgets.

4.2. Sistema

El sistema que se usó para representar la esfera con malla de simplejos en las animaciones es el siguiente:

Para cada vértice de la superficie de la esfera, se conectaron tres resortes que lo unen a sus tres vértices vecinos. También, para que la malla tuviera mayor estabilidad y rigidez se conectó un resorte que une a cada vértice de la superficie al centro de la esfera (Ver Fig. 4.2).

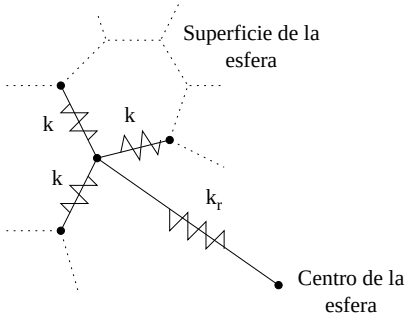


Figura 4.2: Detalle del sistema usado para modelar la esfera deformable

Como mencionamos, las fuerzas principales que mueven y deforman al objeto son la fuerza externa y la fuerza interna de los resortes de las aristas. Para aplicar la deformación a una malla, suponemos que cada arista está compuesta por un sistema

mecánico simple de un resorte y un amortiguador. De este modo el movimiento de cada vértice de la malla al aplicar una fuerza externa F_{ext}, partiendo de 3.2, está descrito por la siguiente ecuación:

$$\frac{d^2P_i(t)}{dt^2} + 2\lambda\frac{dP_i(t)}{dt} + \frac{F_{\text{int}(i)}}{m} = F_{\text{ext}(i)} \tag{4.1}$$

La ecuación 4.1 es una ecuación vectorial y hasta el momento sólo trabajamos con valores escalares sobre el eje x. El cálculo de la velocidad, aceleración y fuerza externa se cálcula exactamente igual que como se venía haciendo sólo que se obtiene por separado para cada una de las componentes de los vectores x, y y z. En el caso de la fuerza interna, aún cuando podemos obtener sus tres componentes, debemos tener cuídado al calcularla. Para el sistema que utilizamos cada vértice tiene conectado no sólo un resorte sino cuatro, así para el vértice i la fuerza interna está definida como:

$$F_{\text{int}(i)} = -k_r\Delta l_i\left[\frac{l_i}{\|l_i\|}\right] - \sum_{j=1}^{3} k\Delta l_{N_j(i)}\left[\frac{l_{N_j(i)}}{\|l_{N_j(i)}\|}\right] \tag{4.2}$$

El primer término de la ecuación 4.2 es la fuerza interna del resorte conectado hacia el centro de la malla cuyo factor de elasticidad es k_r (Ver Fig. 4.2). $\Delta l_i = \|l_i - L_i\|$ y $l_i = P_i - P_{N_j(i)}$, donde L_i es la longitud inicial del resorte. El segundo término es la sumatoria de la fuerza interna de los resortes conectados hacia los tres vecinos del vértice, su factor de elasticidad es k (Ver Fig. 4.2), $\Delta l_{N_j(i)} = \|l_{N_j(i)} - li_{N_j(i)}\|$ y $l_{N_j(i)} = P_i - P_{N_j(i)}$, donde $L_{N_j(i)}$ representa la longitud inicial del resorte conectado de P_i al vecino j.

4.3. Motor de cálculo numérico

Para poder animar objetos deformables necesitamos un motor de cálculo numérico muy robusto, el cual será el encargado de calcular el nuevo valor para todas las posiciones y velocidades de cada una de las partículas.

El sistema irá variando su comportamiento según las fuerzas externas que se le apliquen, para cada partícula de la malla, deberemos evaluar la ecuación diferencial 3.2 utilizando el método de Diferencias finitas (Véase el capítulo 3.4.1) cada vez que se itere el sistema. Por tanto, por cada partícula el motor de cálculo numérico se encargará de calcular 2 valores muy importantes, los cuales representan el "espacio de fase" de una partícula (Figura 4.3) y consiste en un vector de la forma:

$$[P_i'(t), P_i''(t)] = \left[\frac{P_i(t)-P_i(t-1)}{\Delta t}, \frac{V_f - V_i}{\Delta t}\right]$$

para cada una de las N partículas que tiene el sistema. En cada iteración serán evaluados N vectores como este. Tanto V_i como V_f fueron definidas en el capítulo anterior en la sección 3.5.1.

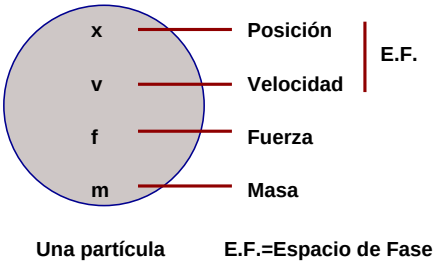


Figura 4.3: Espacio de Fase

Tenemos que mantener una lista de objetos-fuerza para cada sistema de partículas definido. Cada objeto-fuerza afectará a algunas partículas o incluso a todas. Cada vez que el sistema itere tendremos que aplicar cada fuerza a su grupo de partículas asociado (Ver Fig. 4.4).

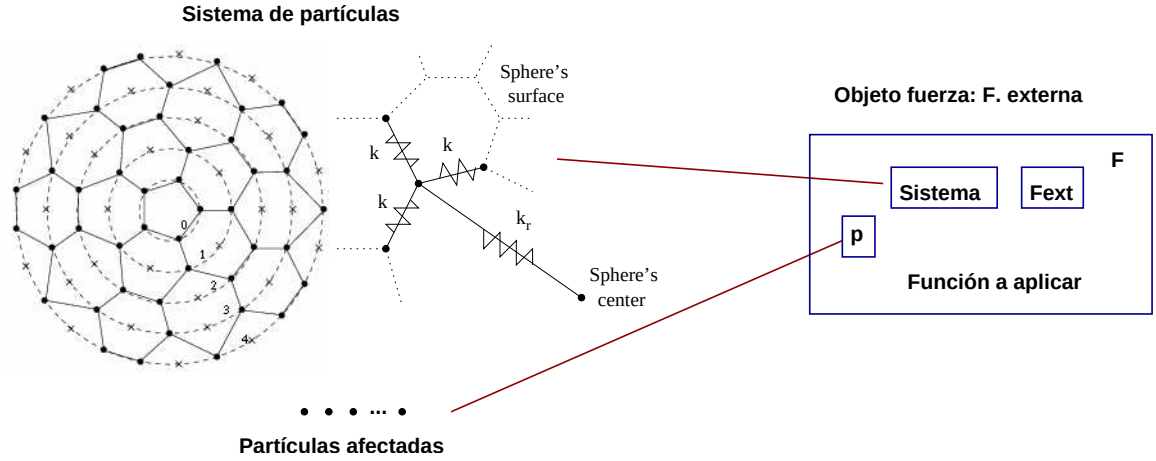


Figura 4.4: Motor de cálculo numérico

Para simular la deformación de la malla empezamos por el "tiempo inicial" e iteramos usando el incremento h cada vez hasta llegar al "tiempo final" que es cuando el movimiento de la malla se ha estabilizado. Los datos que requiere el motor de deformación fueron almacenados en una estructura de datos (Véase el Apéndice B) para facilitar su uso y poder acceder a ellos con rapidez. Los elementos de la estructura de datos son los siguientes:

- El centro de la malla.
- Número de vértices de la malla.
- Factor de amortiguamiento.
- Factor de alasticidad k y k_r.

- Fuerza externa.
- Posición de los vértices en $t-1$.
- Magnitud inicial de los resortes.
- Error del motor de cálculo.

4.3.1. Condición de paro

El sistema es estable cuando la fuerza interna de la malla se iguala a la fuerza externa que se está aplicando a ésta. De este modo definimos el error del motor de cálculo numérico como sigue:

$$e(t) = \text{abs}[MF_{\text{ext}}(t) - MF_{\text{int}}(t)].$$

donde, $MF_{\text{ext}}(t)$ es la magnitud de la fuerza externa total y $MF_{\text{int}}(t)$ la magnitud de la fuerza interna total de la malla en el tiempo t.

Después de analizar los tres tipos de movimiento de un resorte simple determinamos que el sistema es estable cuando el error cumple la siguiente condición:

$$\Delta e(t) < \text{umbral} \tag{4.3}$$

donde,

$$\Delta e(t) = e(t) + e(t-1) + \text{abs}[e(t) - e(t-1)]$$

En la figura 4.5 se observa la curva que describe el movimiento de un sistema de resorte y masa con movimiento subamortiguado forzado. En la gráfica de color rojo se iteró hasta $t = 50$ y visualmente podemos observar que el sistema es estable mucho tiempo antes. Para la curva verde se usó la condición de paro (Ec. 4.3) con un umbral de $1\text{x}10^{-5}$ y puede notarse que el motor de cálculo iteró aproximadamente hasta $t = 25$ que es el tiempo suficiente para que el sistema se estabilizara.

4.3.2. Sistema no lineal

El modelo que consideramos para las animaciones es lineal pero al manipular la malla podríamos obtener un comportamiento extraño pues cabe la posiblidad de que las partículas atreviesen las caras del objeto, lo cual no es realista. Tampoco se consideró que se puede exceder el límite de elasticidad de los resortes o que al aplicar demasiada fuerza sobre las mallas se puede llegar a su punto de ruptura, para lo cual la ley de Hooke ya no se cumple. Por estas razones se cambió el modelo lineal del resorte por uno no lineal, en el que el resorte no se estira más alla del doble de su longitud y no puede comprimirse menos del 20 % de su longitud (Ver Fig. 4.6). El algoritmo para resolver el sistema no lineal usando diferencias finitas se presenta en el procedimiento 3.

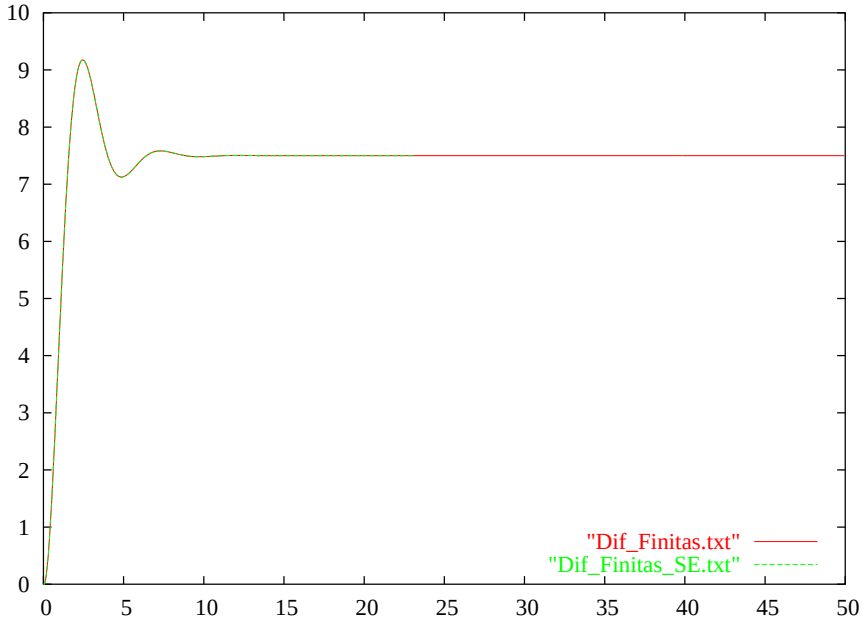


Figura 4.5: Sistema de resorte y masa como movimiento subamortiguador forzado. Condiciones: $F_{\text{ext}} = 15$, $\Delta t = 0.05$, $k = 2$, $m = 1$ y $\beta = 1.2$

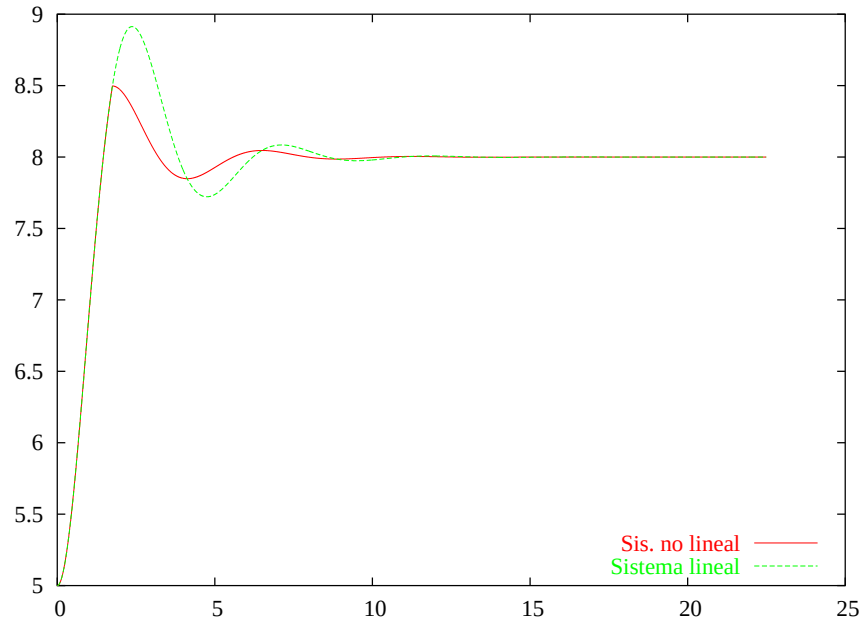


Figura 4.6: Movimiento subamortiguado. Resorte de longitud 5. Condiciones: $F_{\text{ext}} = 6$, $\Delta t = 0.05$, $k = 2$, $m = 1$ y $\beta = 1$

Procedimiento 3 Modelo no lineal resuelto con el método de diferencias finitas.
Entrada: t_0, x_0, v_0, Δt, k, m, β, F'_{ext}, n, l
Salida: $x(i)$, $v(i)$, for $0 < i < n$
  $\omega^2 \leftarrow k/m, \quad 2\lambda \leftarrow \beta/m$
  $x(-1) = x(0) = x_0$, $v(0) = v_0$, $t(0) = t_0$
  $a \leftarrow 2 - 2\lambda * \Delta t - \omega^2 * \Delta t * \Delta t$
  $b \leftarrow 2\lambda * \Delta t - 1$
  $c \leftarrow \Delta t * \Delta t * F'_{\text{ext}}/m$
  $l_{\text{inf}} \leftarrow 0.3 * l$
  $l_{\text{sup}} \leftarrow 1.7 * l$
  for $1 \leq i < n$ **do**
    $x(i) \leftarrow a * x(i-1) + b * x(i-2) + c$
    $l_t \leftarrow l + x(i)$
    if $l_t > l_{\text{sup}}$ or $l_t < l_{\text{inf}}$ **then**
      $x(i) \leftarrow x(i-1)$

4.4. Deformación de una esfera a un cubo

Suponiendo que el centro del cubo es (c_x, c_y, c_z) y que la longitud de sus lados es l entonces las ecuaciones de los planos de sus seis caras son:

$$\begin{array}{ll} x = -(c_x + \frac{l}{2}) & x = -(c_x - \frac{l}{2}) \\ y = -(c_y + \frac{l}{2}) & y = -(c_y - \frac{l}{2}) \\ z = -(c_z + \frac{l}{2}) & z = -(c_z - \frac{l}{2}) \end{array}$$

La deformación de una esfera a un cubo se realizó desplazando cada una de las partículas de la malla $P_i(t)$ al punto $P_i(t+1)$ que es el punto de intersección más cercano entre la normal de la partícula (Ec. 2.1) y las caras del cubo como se muestra en la figura 4.7.

En la figura 4.7 podemos observar como la normal de P_i, con respecto a sus tres vecinos, intersecta a las seis caras del cubo. La normal pude intersectar a los planos de las caras dentro del área del cubo o fuera, pero los puntos más cercanos siempre estarán dentro de ésta.

La fuerza externa aplicada a cada vértice para lograr la deformación es igual y en sentido opuesto a $F_C + F_N$, $\quad F_C$ es la fuerza del resorte conectado al centro de la malla y F_N es la fuerza de los resortes conectados a los vecinos.

Como indica la ley de Hooke la fuerza que opone un resorte a la deformación es proporcional a su factor de elasticidad y a la cantidad de alargamiento. Considerando para este exprimento un mismo factor de elasticidad k para todos los resortes $F_C = -k\Delta l$, donde $\Delta l = [P_i(t+1) - c_m] - L$ siendo L la longitud inicial del resorte.

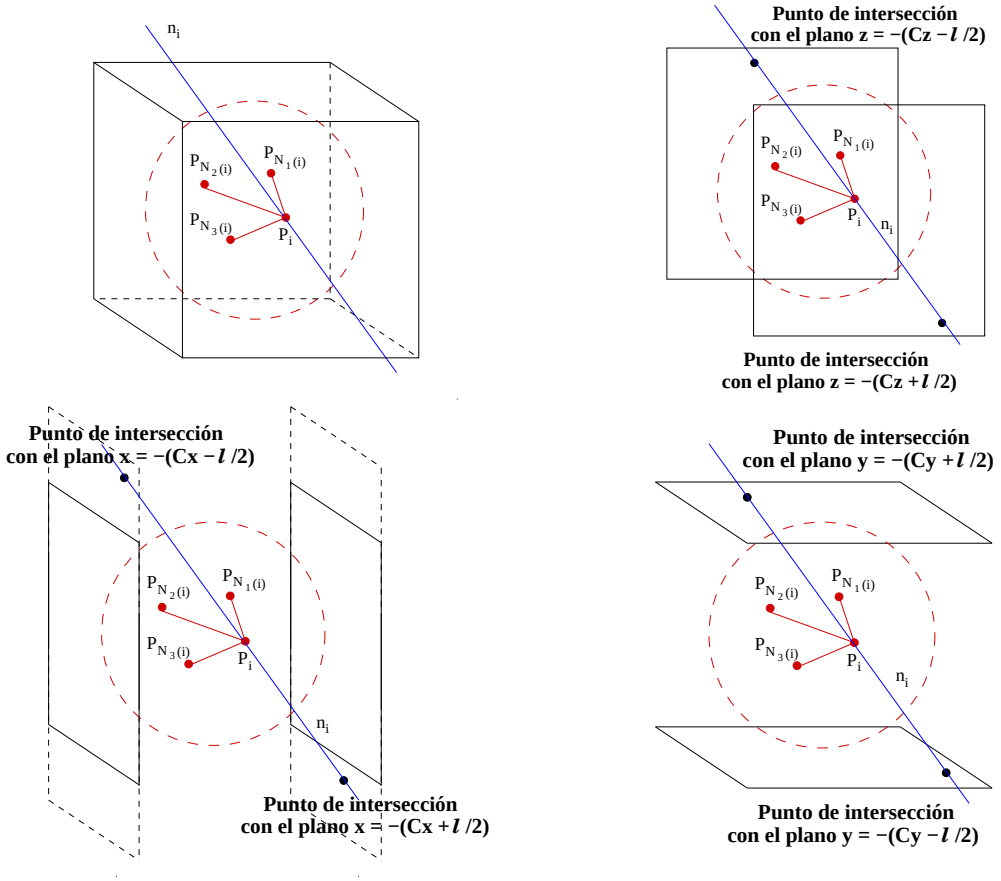


Figura 4.7: Deformación de una esfera a un cubo. Intersección de la normal de la partícula y las caras del cubo.)

Para calcular el valor de F_N tendríamos que anticipar el comportamiento de los vértices vecinos para cada punto lo que resulta sumamente complicado. Pero podemos estimar su valor como una proporción de F_C, así pues $F_N = cF_C$.

Después de realizar una serie de pruebas y considerando que la fuerza de los resortes de los tres vecinos es casi despreciable en comparación con la fuerza del resorte que va conectado hacia el centro de la malla, se asginó a F_N el valor de $-0.2k\Delta l$.

Este experimento se realizó para probar que se incorporó el comportamiento elástico a las mallas, para lo cual se eligieron valores de los parámetros k, m y β de forma que se obtuviera un movimiento subamortiguado, pues los otros dos tipos de movimiento son más estables y no presentan oscilaciones por lo que sería difícil apreciar si los resultados son correctos o no.

El resultado del experimento puede verse en la Fig. 4.8; en la imagen de la Fig. 4.8(a) se observa la esfera original y el la Fig. 4.8(f) el cubo resultante. Tanto a los resortes de la superficie como a los interiores se les asignó un coeficiente de elasticidad

de 0.5 N/m (Newtons/metro). La masa de cada vértice fue de 1 Kg y el coeficiente de amortiguamiento fue de 0.65 N/(m/s). En la Fig. 4.8(d) se ve el pico máximo de la deformación dado que el sistema es subamortiguado como consecuencia de los valores de los parámetros. Las oscilaciones en la deformación de la esfera son similares a las que presenta el movimiento subamortiguado que se muestra en la figura 4.5, por lo que podemos concluir que los resultados que se obtuvieron son correctos.

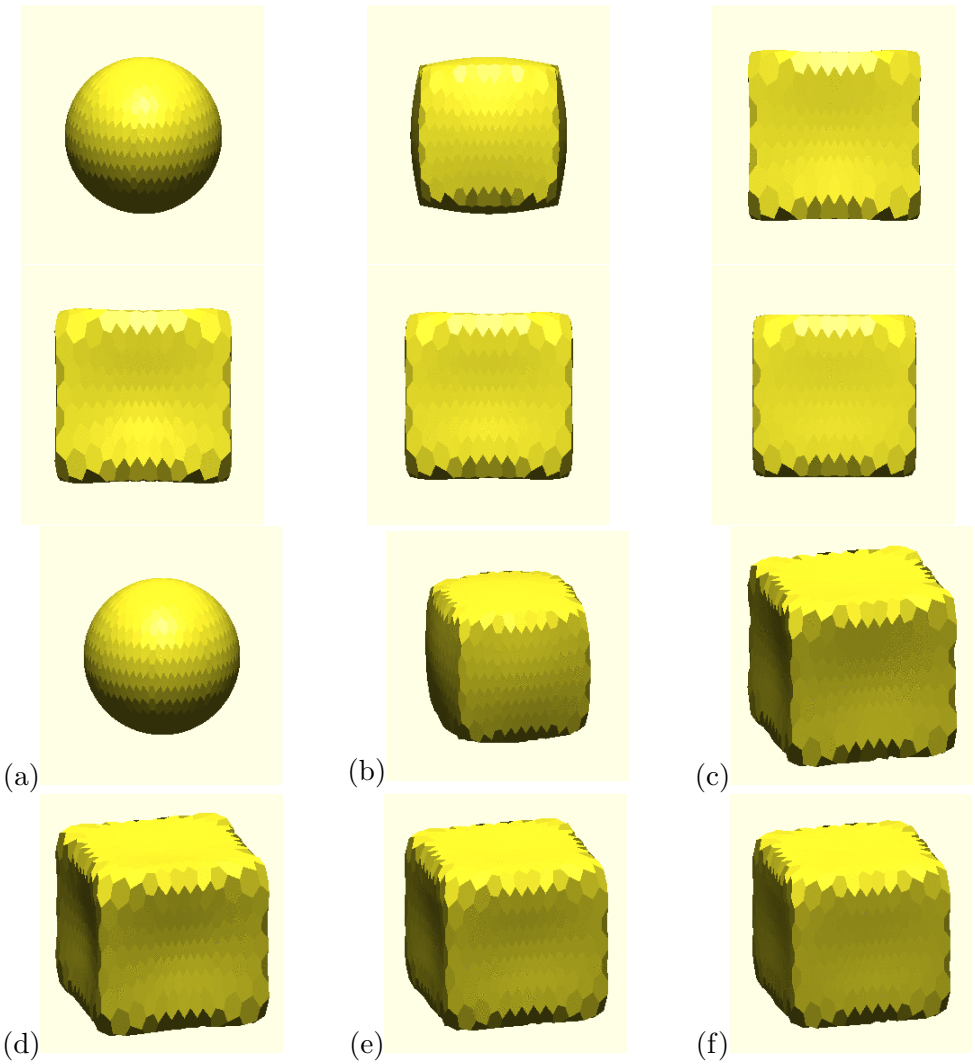


Figura 4.8: Deformación de una esfera a un cubo. La esfera fue construida con 10 etapas y una frecuencia de 8 como se indica en [1]; esto resulta en una malla con 1296 vértices. Las condiciones fueron $\Delta t = 0.05$, $k = k_r = 0.5$, $m = 1$ y $\beta = 0.65$)

4.5. Rebote de una pelota contra una pared

Para el rebote de la pelota contra una pared prácticamente el sistema funciona por sí solo, lo único que se requiere es aplicar una fuerza inicial a la pelota para que

comience a desplazarse, en otras palabras se aplicará la misma fuerza a todas las partículas en $t = 0$. El sistema de amortiguamiento permanece desactivado hasta el momento de la colisión para evitar perdida de energía antes del choque.

La colisión de un punto P_i de la pelota con la pared se detecta cuando su posición en x es igual a la posición de la pared. Para todas las partículas que colisionan, la pared ejerce una fuerza externa sólo en x y está definida como $F_{\text{ext}(i)} = [-f_{\text{int}}, 0, 0]$ pues la pared no puede ni debe ser atrevesada. f_{int} es la componente en x de $F_{\text{int}(i)}$ que se obtiene con la ecuación 4.2. La respuesta del sistema a la colisión se obtiene dejando que las leyes de la física actuen con las ecuaciones planteadas en el capítulo anterior.

Esta animación se desarrolló para observar si la fuerza interna de los resortes, que se genera al momento de la colisión, se propaga de los puntos que colisionan con la pared hacia sus vértices vecinos. Se usaron dos tipos de pelota, una con resolución baja cuya malla de simplejos está compuesta de 54 vértices (Fig. 4.9) y otra con resolución media de 1296 vértices (Fig. 4.10), los resultados que se obtuvieron fueron satisfactorios ya que en las animaciones se observan pequeñas vibraciones en los resortes, lo que nos indica que la fuerza interna de los resortes se está propagando.

La pelota en baja resolución permitió hacer simulaciones en tiempo real, no así la pelota en media resolución. La secuencia de imágenes de la Fig. 4.9 (a)-4.9(i) muestran algunos marcos de la secuencia del rebote de la pelota en baja resolución contra la pared. La pared se dibujó transparente en las Figs. 4.9(j)-4.9(o), también se muestra otra vista de las imágenes. Para esta misma pelota, si el coeficiente de amortiguamiento se hace más grande que 0.09 la pelota presenta un comportamiento gelatinoso.

El choque de la pelota en resolución media se observa en las imágenes de la Figs. 4.10(a) a 4.10(i). La Fig. 4.10(n) es la misma que la 4.10(k) pero en otra posición. La Fig. 4.10(k) está tomada desde detrás de la pared (recuerde que es transparente) y puede observarse la máxima deformación que sufre la pelota.

El coeficiente de amortiguamiento para la pelota en resolución media tuvo que disminuirse un orden de magnitud respecto a la pelota en resolución baja, esto debido al gran aumento en la cantidad de resortes. La masa total de la pelota a resolución baja fue de 54 Kg (un kilogramo por cada vértice) y se consideró la misma masa total para la pelota en resolución media, que tiene 1254 vértices; por ello que le asigno una masa para cada vértice de $54/1296 = 0.0416$ Kg.

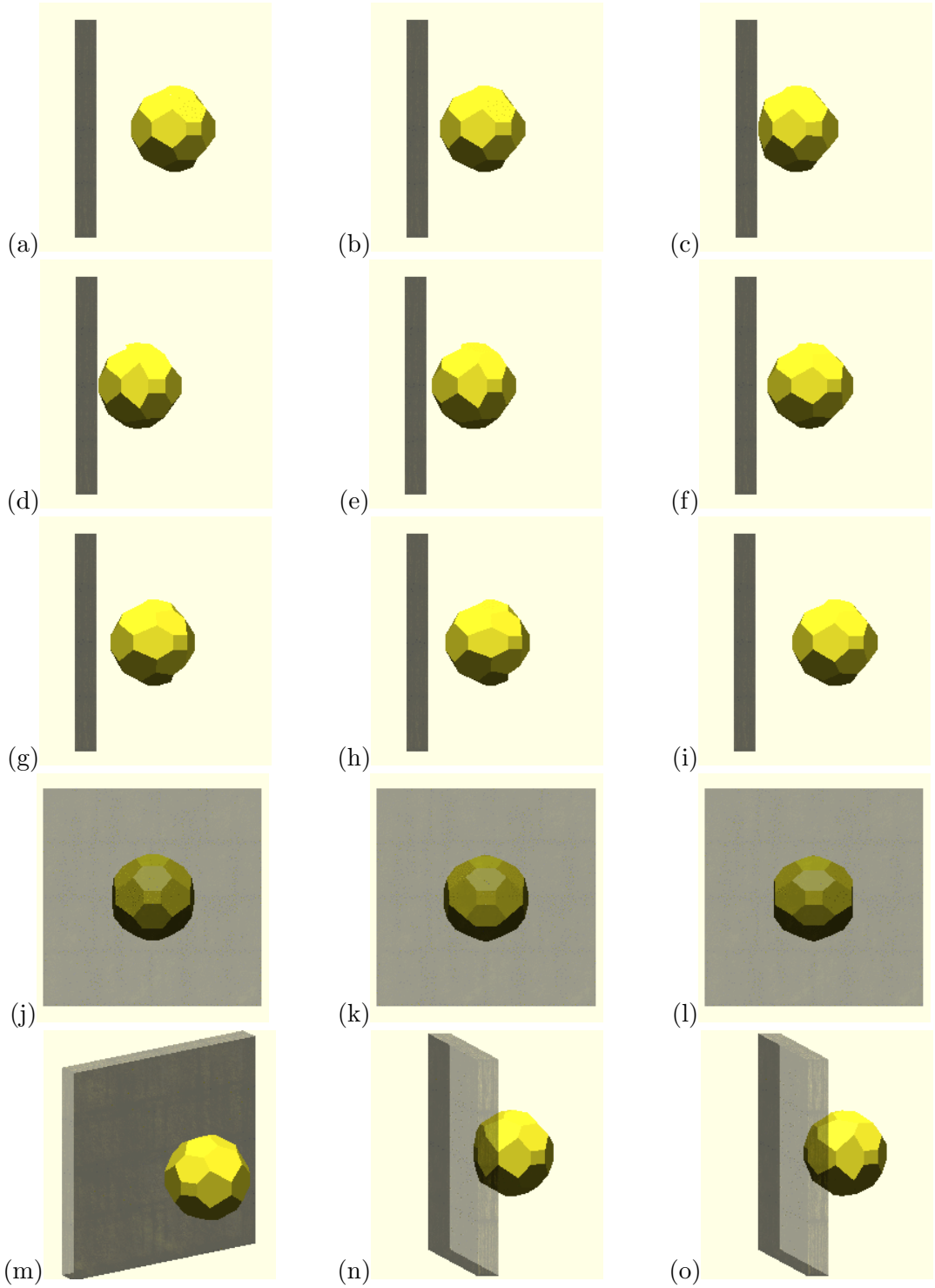


Figura 4.9: Rebote de una pelota de baja resolución contra la pared (una esfera de 4 etapas y frecuencia 3, 54 vértices en total). Condiciones: $\Delta t = 0.05$, $k = 9$, $k_r = 9.2$, $m = 1$ y $\beta = 0.09$

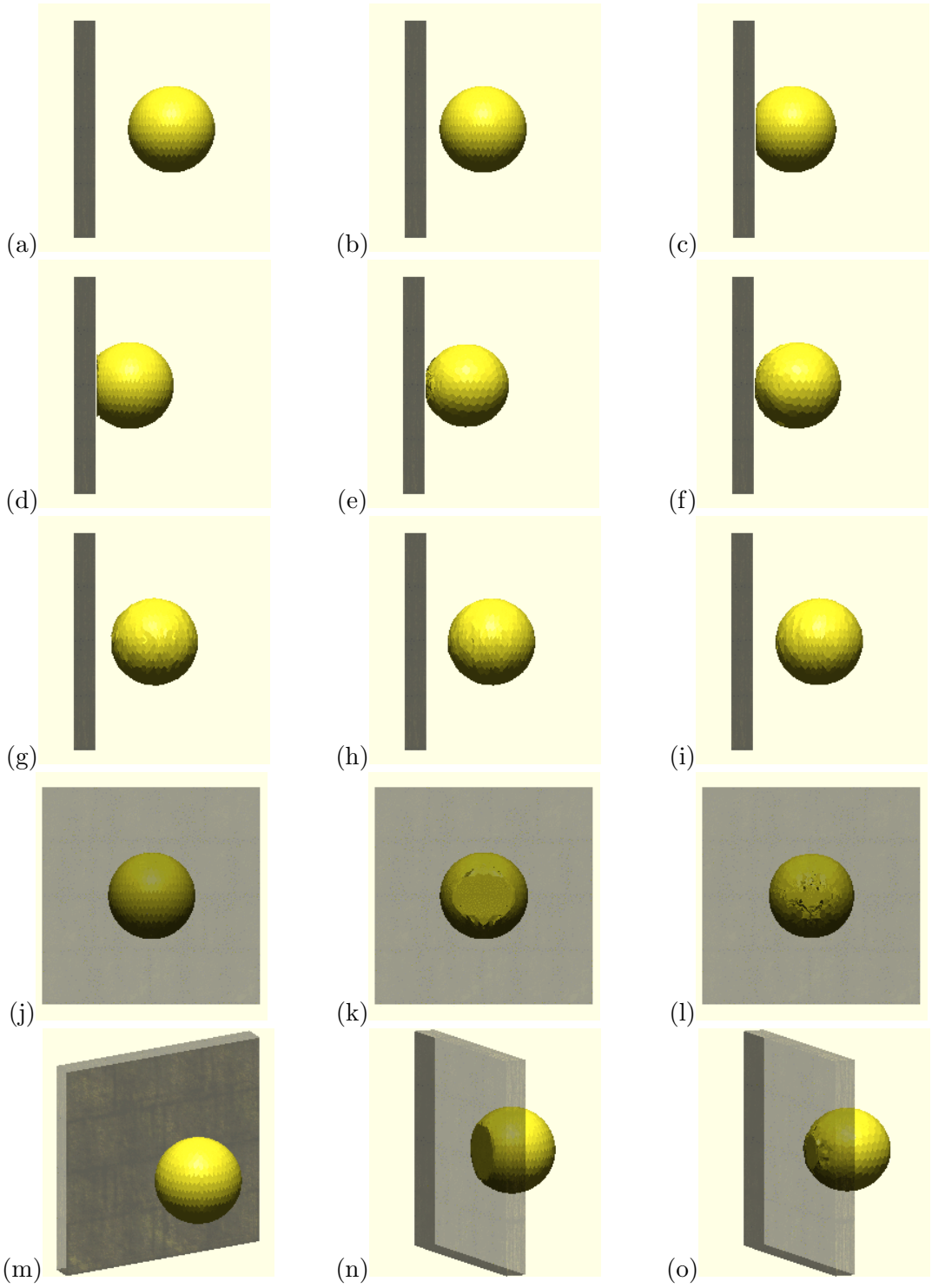


Figura 4.10: Rebote de una pelota de media resolución contra la pared (una esfera de 10 etapas y frecuencia 8, con un total de 1296 vértices). Condiciones: $\Delta t = 0.005$, $k = 9$, $k_r = 1.8$, $m = 54/1296 = 0.0416$, y $\beta = 0.005$)

4.6. Una pelota comprimida con dos paredes

Las paredes mantiene una velocidad constante al acercarse a la pelota y es igual a:

Pared derecha	$x_{der} - \Delta x$
Pared izquierda	$x_{izq} + \Delta x$

x_{der} es la posición de la pared derecha y x_{izq} la posición de la pared izquierda. En este experimento es posible aplicar mayor fuerza sobre más puntos de la mallas y se puede apreciar mejor la deformación elástica de los objetos.

Las paredes no ejercen ninguna fuerza sobre la pelota hasta el momento en que ésta penetra una o ambas paredes. Es decir que si la posición en x de alguna de las partículas de la malla es menor que $x_{izq}(t+1)$ entonces $x_i(t+1) = x_{izq}(t+1)$. En caso de que $x_i(t)$ sea mayor que $x_{der}(t+1)$ entonces $x_i(t+1) = x_{der}(t+1)$. Para ambos casos la fuerza que se ejerce sobre el vértice i es $F_{\text{ext}} = -F_{\text{int}}$ (Ec. 4.2). En cualquier otro caso la fuerza externa es igual a cero y $x_i(t+1) = x_i(t)$ (Ver Fig. 4.11). De este modo las paredes siempre van a ser capaces de comprimir a la pelota sin permitir que ésta las atraviese. Las paredes se mueven sólo hasta que la distancia entre ellas es igual a 0.34. Las componentes de los vectores en y y z no fueron consideradas para este ejemplo ya que las paredes sólo ejercen una fuerza externa sobre la pelota en x.

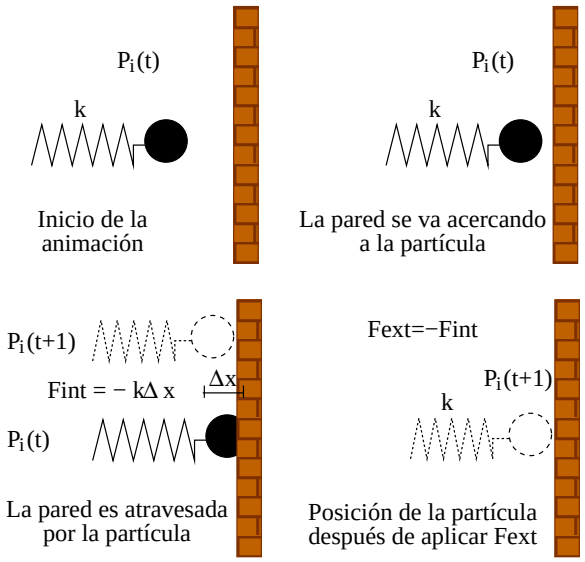


Figura 4.11: Sistema de un resorte, una partícula y amortiguador.

El sistema de la figura 4.11 se desplaza hasta el momento que la partícula penetra la superficie de la pared. En ese momento se cálcula la fuerza externa necesaria para regresar la partícula a la posición donde sólo colisione con la pared sin penetrarla.

En las imágenes de la Figs. 4.12(a) a 4.12(l) se muestran las dos paredes comprimiendo a la pelota. De la Fig. 4.12(a) a la 4.12(i) se roto la escena para observar la deformación de la pelota através de la pared.

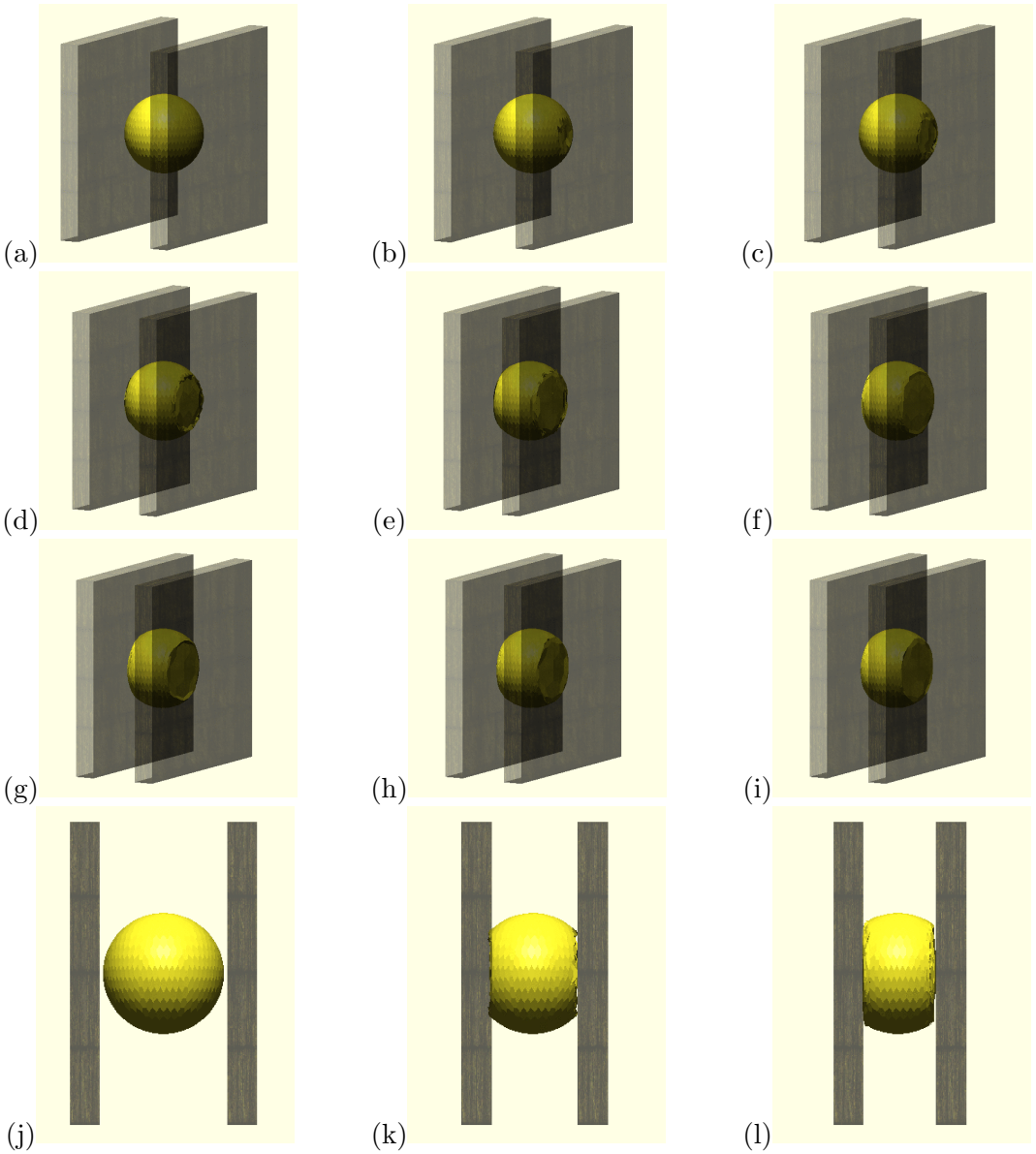


Figura 4.12: Animación de dos paredes comprimiendo una esfera. Ésta fue creada con 10 capas alrededor de un octagono (con un total de 1296 vertices). Condiciones: $\Delta t = 0.005$, $k = 2$, $k_r = 3.2$, $m = 54/1296 = 0.0416$ y $\beta = 0.09$

4.7. Deformación general de una esfera

El último experimento fue aplicar una fuerza externa simulada sobre ciertos puntos de la superficie de la esfera. El usuario proporciona un escalar f_{ext} que es la magnitud del vector de fuerza que se aplicará al objeto. La dirección del vector fuerza va de los puntos seleccionados hacia el centro de la malla como se describe a continuación:

$$F_{\text{ext}} = f_{\text{ext}} \frac{F_{\text{e}}}{\|F_{\text{e}}\|}, \qquad \text{donde} \qquad F_{\text{e}} = Pi - c_m$$

En esta animación los puntos sobre los que se aplica la fuerza externa son fijos, pero este experimento sirvió como base para que posteriormente se permitiera al usuario seleccionar los puntos de la malla sobre los que se aplicará la fuerza externa. Los resultados se muestran en la Fig. 4.13 y 4.14. Para la figura 4.13 (a) se aplicó una fuerza de 0.7 newtons sobre un sólo punto. En la figura 4.13 se observa la deformación de una esfera a la que se aplicó una fuerza de 0.1 newtons sobre un punto y las cinco capas de sus vecinos más próximo. Finalmente, de la figura 4.14(a) a la 4.14(i) podemos ver como se va deformando una pelota al aplicar la misma fuerza que en el caso anterior, pero sólo sobre un punto y sus vécinos directos.

(a)  (b) 

Figura 4.13: Deformación de una esfera aplicando una fuerza externa simulada. Usamos la misma esfera de la Fig. 4.12. Condiciones: $\Delta t = 0.005$, $k = 50$, $k_r = 0.2$,$m = 54/1296 = 0.0416$ y $\beta = 0.09$

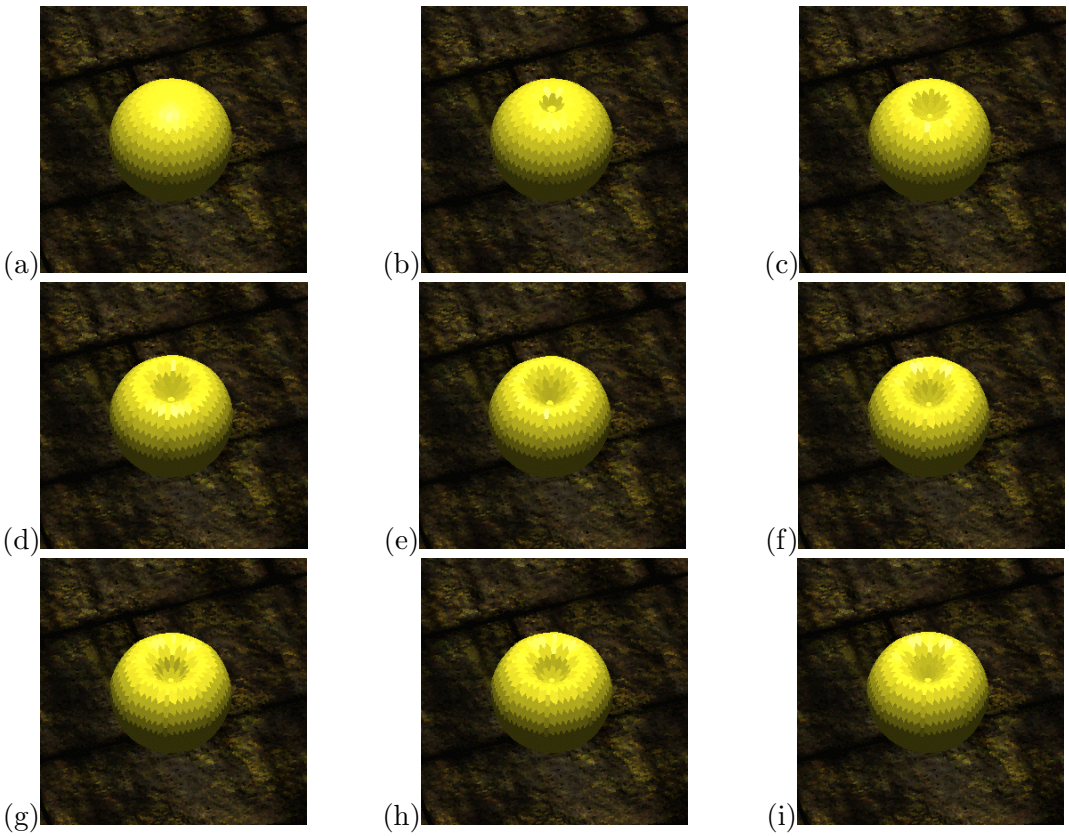


Figura 4.14: Deformación de una esfera aplicando una fuerza externa simulada. Usamos la misma esfera de la Fig. 4.12. Condiciones: $\Delta t = 0.005$, $k = 50$, $k_r = 0.2$,$m = 54/1296 = 0.0416$ y $\beta = 0.09$

Capítulo 5

Interacción Virtual con los Modelos

Después de haber logrado que las mallas tuvieran un comportamiento elástico, el siguiente paso es permitir al usuario seleccionar sobre qué puntos del objeto se aplicaría la fuerza externa. Para este propósito se incorporó a la interfaz gráfica, diseñada en Qt, un dispositivo háptico llamado Phantom Omni (Ver Figura 5.1). El Phantom Omni realiza la función de cursor en nuestra aplicación y a través de éste el usuario puede sentir y manipular los objetos tridimensionales virtuales construídos con mallas de simplejos, así como manipular la escena gráfica.

En el Phantom Omni viene incluida una biblioteca para trabajar con OpenGL y no con Qt, por lo que se realizaron algunas adaptaciones en la parte del manejo de eventos del dispositivo para poder integrar las funciones del Phantom con la interfaz gráfica. A continuación, se explica como se logró incorporar el dispositivo háptico, pero primero se introducen algunos conceptos sobre el Phantom, esta información se ocupó para lograr la selección de puntos y la manipulación de la escena a través del Phantom. La mayor parte de este capítulo fue obtenida del manual [25], pero adaptada a la aplicación que se realizó.

5.1. Phantom Omni

El Phantom Omni es una herramienta háptica que permite incorporar la sensación de tocar y controlar los objetos de la aplicación a través de fuerza o del contacto suministrado [26, 27, 28].

El dispositivo tiene seis grados de libertad de movimiento, sobre los tres ejes x, y, z y las rotaciones sobre esos mismos ejes. Todos los grados de libertad de movimiento tienen limitaciones físicas. Cuando se alcanzan estos límites se siente un alto repentino. El espacio de trabajo es de 160 mm en x, 120 mm en y y 70 mm en z.

Figura 5.1: Imagen del Phantom Omni

El Phantom cuenta con kit de desarrollo (llamado OpenHaptics) que incluye una biblioteca de funciones hápticas, controladores y código de ejemplos. La biblioteca háptica consta de dos partes la HD y la HL.

La HD provee acceso a bajo nivel al dispositivo, permite aplicar fuerza, ofrece control sobre la configuración del comportamiento de los controladores en tiempo de ejecución y provee una utilidad muy conveniente que sirve para modificar las características.

La HL es un API en C de alto nivel para dibujado háptico modelado de acuerdo con la API de OpenGL para dibujado gráficos. La HL permite especificar primitivas geométricas tales como triángulos, lìneas y puntos, con propiedades como rigidez y fricción. El motor de dibujado háptico usa esta información y los datos que lee del dispositivo háptico para calcular la fuerza apropiada que debe enviar a éste.

Para incorporar el Phantom con OpenGL, se utilizó mayormente la HL (la parte de alto nivel del API). El único método que se ocupó de la HD es *hdInitDevice* y está definido de la misma forma para las dos APIs, por lo que sólo nos limitaremos a explicar con detalle la HL en la siguiente sección.

5.2. HL

Así como OpenGL, HL está basada en una máquina de estados. La mayoría de los comandos de HL modifican el estado de dibujado el cual puede ser obtenido usando los métodos de la misma API. El estado incluye información como las propiedades del material, las transformaciones y modos de dibujado. Además incluye la posición y orientación del dispositivo. Por medio de esta API, también se pueden dar de alta funciones de retrollamada para que sean invocadas cuando ocurre un evento, como por ejemplo, tocar una figura o presionar un botón del dispositivo.

5.2.1. Generación de fuerzas

Existen tres formas de generar una retroalimentación háptica usando HL:

Dibujado de figuras.- Permite al usuario definir superficies por medio de primitivas geométricas las cuales son usadas por el motor de dibujado para calcular automáticamente la reacción adecuada de fuerza y simular que se tocó la superficie del objeto.

Efectos de dibujado.- Permite especificar funciones globales de fuerza para generar efectos. Incluye un número de efectos de fuerza estándar, como viscocidad y elasticidad. También el usuario puede especificar sus propias funciones de efectos.

Dibujado del proxy.- Permite al usuario modificar la posición y orientación del dispositivo háptico. El motor de dibujado enviará automáticamente al dipositivo la fuerza adecuada para que se mueva a la posición deseada.

Tanto el dibujado de figuras, comomo los efectos de dibujado y el dibujado del proxy forman parte de lo que se denomina dibujado háptico.

5.2.2. Influencia de OpenGL

El mecanismo primario para especificar la geometría de las figuras con HL usa los comandos de OpenGL, permitiendo reusar código ya existente de programas de OpenGL. Como ya sabemos, los comandos de OpenGL para especificar la geometría de dibujado, que puede usar HL, incluyen primitivas como puntos, líneas y polígonos, asícomo geometrías almaceanadas en la listas de despliegue y arreglos de vértices, todos éstos se especifican usando *glBegin*. La captura de la geometría de OpenGL puede realizarse de dos formas diferentes: por el búfer de profundidad de la figura o el búfer de retroalimentación de la figura.

Se pueden realizar transformaciones invocando funciones de OpenGL como *glTranslate*, *glRotate* y *glScale*. HL aplicará las transformaciones, enviadas por estos comandos, a las primitivas geométricas de dibujado háptico. La API hace ésto obteniendo porciones del estado de transformación de la máquina de estados de OpenGL.

5.2.3. Dibujado del proxy

El dibujado háptico de la geometría es hecho usando el método del proxy. El proxy es un punto el cual sigue muy de cerca la posición del dispositivo. Pero mientras la posición actual del dispositivo háptico puede estar dentro de una figura el proxy estara siempre fuera de la superficie de todas las figuras tocables.

El motor de dibujado háptico continuamente actualiza la posición del proxy intentando moverla a la misma posición del dispositivo. Cuando no se está tocando a una figura, la posición del proxy es actualizada a la misma posición el dispositivo.

Pero cuando el dipositivo está en contacto con una figura y penetra su superficie, el proxy permanecerá fuera. La fuerza enviada al dispositivo es calculada colocando un resorte virtual con amortiguamiento que va de la posición del dispositivo a la posición del proxy (ver figura 5.2).

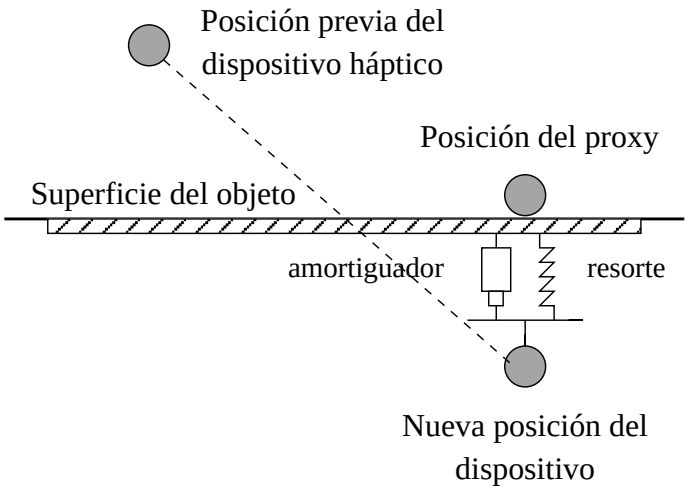


Figura 5.2: Proxy

5.2.4. Hilos

Debido a que el dibujado háptico requiere de una actualización con mayor frecuencia que las aplicaciones clásicas de gráficos, el motor de dibujado de HL crea, además del hilo de la aplicación principal, dos hilos adicionales que usa para el dibujado háptico: el hilo del servomecanismo (o simplemente *hilo servo*) y el hilo de colisión. El hilo de la aplicación principal en un programa típico de HL es referido como el hilo cliente. El hilo cliente es en el que se crea el contexto de dibujado de la HL y se escribe código que no necesita saber nada acerca del hilo servo o el hilo de colisión, aunque un usuario avanzado puede escribir código en alguno de estos hilos.

Hilo servo.- El hilo servo maneja una comunicación directa con el dispositivo háptico. Lee la posición y orientación del dipositivo y actualiza la fuerza que se envía al dipositivo. Corre a una razón constante (usualmnete 1000 Hz) con una prioridad alta y mantiene estable el dibujado háptico.

Hilo de colisión.- El hilo de colisión es el responsable de determinar que primitivas geométricas son tocadas por el proxy. Éste corre a una razón de 100 Hz. El hilo de colisión encuentra cual de las figura especificadas en el hilo cliente están en contacto con el proxy y genera una aproximación local simple de las figuras. Esta aproximación local se envía al hilo servo que utiliza estos datos para actualizar la fuerza del dispositivo háptico.

5.2.5. Diseño de un programa típico de HL

Un típico programa HL tiene la estructura que se muestra en la Figura 5.3.

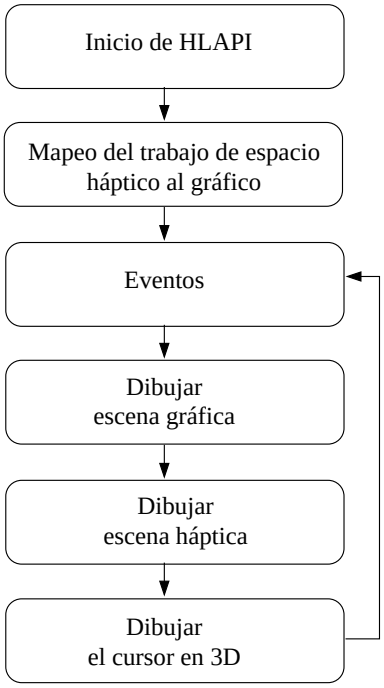


Figura 5.3: Diseño de un programa HL

Primero, el programa inicia OpenGL para crear un contexto de dibujado de gráficos y lo liga a la ventana. Después se inicializa HL creando un contexto de dibujado háptico y ligándolo al dispositivo háptico. En seguida el programa especifica como mapear las coordenadas físicas del dispositivo háptico a coordenadas del espacio usado por los gráficos. HL usa este mapeo para transladar la geometría especificada en el espacio gráfico al trabajo de espacio físico del dispositivo háptico. Después la aplicación dibuja la escena gráfica usando OpenGL. Entonces el programa procesa los eventos generados por el motor de dibujado háptico. Se dibuja el espacio háptico, generalmente se ejecuta casi el mismo código que para dibujar los gráficos, pero se captura la geometríade las figuras con el búfer de profundidad o de retroalimentación. Finalmente, se dibuja el cursor en 3D en la posición del proxy reportada por HL. El ciclo se repite para dibujar los gráficos nuevamente.

5.2.6. Inicio del dispositivo

El primer paso es inicializar el dispositivo por su nombre usando *hdInitDevice*. El nombre es una etiqueta y puede ser modificado con el panel de configuración del Phantom (ver apéndice C, en la página 85).

El segundo paso es crear un contexto para inicializar el dispositivo usando *hlCreateContext*. El contexto mantiene el estado que persiste entro los intervalos del marco y es usado para el dibujado háptico.

```
hHLRC = hlCreateContext(hHD);
```

El tercer paso es activar el contexto que se acaba de crear.

```
hlMakeCurrent(hHLRC);
```

5.2.7. Contexto de dibujado

En HL todos los comandos requieren que esté activo un contexto de dibujado. El contexto de dibujado contiene el estado del dibujado háptico actual y sirve como objetivo para todos los comando de la HL.

Invocando *hlMakeCurrent* el contexto es activado para el hilo actual. Como en OpenGL, todos los comandos de HL hechos en dicho hilo operan en el contexto activo. Es posible usar multiples contextos para dibujar en un mismo hilo haciendo llamadas adicionales a *hlMakeCurrent* para cambiar de contexto activo. También varios hilos pueden activar al mismo contexto, pero al igual que en OpenGL, las rutinas de HL no son hilos seguros. Así que deben usar secciones críticas o candados para sincronizar las invocaciones de *hlMakeCurrent* y aseguar que sólo un hilo a la vez active el contexto de dibujado.

5.2.8. Frames hápticos

Todos los comandos en la HL deben ser usados dentro de un marco háptico. El inicio y fin de un marco háptico se indica con hlBeginFrame y hlEndFrame respectivamente. Explícitamente marcar el inicio y fin del marco permite al API sincronizar apropiadamente tanto los cambios de estado como los cambios en el motor de dibujado.

En un programa habrá un marco de dibujado háptico por cada marco de dibujado gráfico. En la figura 5.4 se muestra la estructura de un programa HL que incorpora marcos hápticos.

hlBeginFrame por lo regular es llamado al inicio del ciclo de dibujado, así cualquier objeto en la escena, que dependa del dipositivo háptico o del estado del proxy, tendrá los datos más recientes. *hlEndFrame* se llama al final del ciclo de dibujado para descargar los cambios del motor de dibujado háptico y los gráficos para que ambos

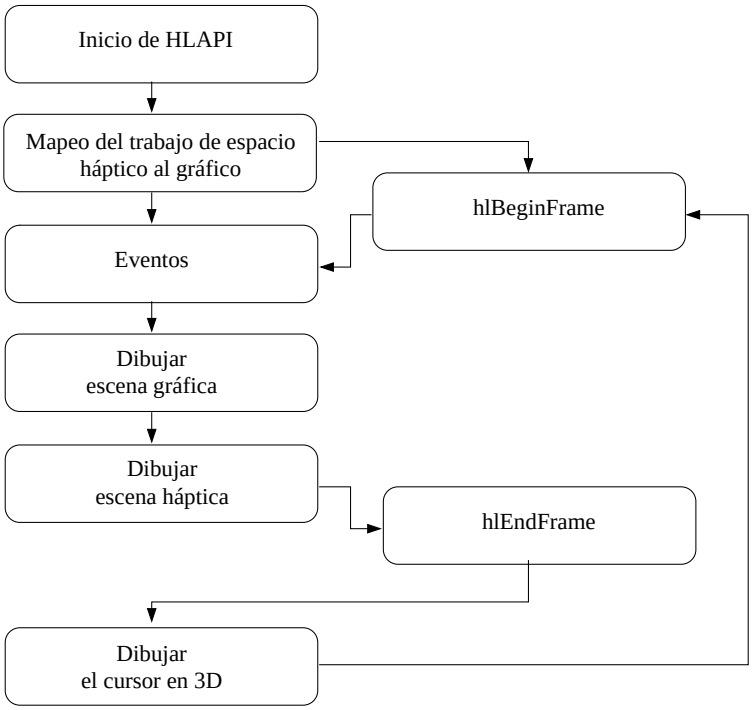


Figura 5.4: Diseño de un programa HL con marcos

estén sincronizados.

Al inicio del marco háptico, *hlBeginFrame* comprueba el estado del dibujado háptico actual apartir del hilo de dibujado háptico. *hlEndFrame* finaliza el dibujado háptico y resuelve de forma síncrona cualquier cambio dinámico con el objetivo de actualizar la posición del proxy.

Además de actualizar el estado de dibujado háptico disponible para el hilo cliente, *hlBeginFrame* también actualiza las coordenadas del marco usadas por el motor de dibujado háptico. Por defecto, *hlBeginFrame* verifica el valor actual de la matriz `GL_MODELVIEW_MATRIX` de OpenGL para proporcionar un espacio de coordenadas para el marco háptico. Todas las posiciones, vectores y transformaciones obtenidas a través de *hlGet* o *hlCacheGet*, en el hilo cliente o de colisión, serán transformadas a ese espacio de coordenadas.

Todos los comandos de HL que requieran al dipositivo háptico o al estado del proxy y que sean invocados entre la misma pareja de primitivas de inicio y fin de un marco, reportarán los mismos resultados. Por ejemplo, para diferentes consultas de la posición del dipositivo en un mismo marco, siempre se reportará exactamente la misma posición, que es la obtenida al momento en que *hlBeginFrame* fue invocado, incluso si la posición actual del dispositivo háptico hubiese cambiado después del inicio del marco. Esto se hace con el fin de evitar problemas donde, por ejemplo, durante un

marco, el programa mueve varios objetos en una escena por medio de la acumulación de movimientos del dispositivo, el reportar distintos movimientos en diferentes partes del marco causaría que el movimiento de los objetos no estuviese sincronizado.

Al final del marco háptico todos los cambios hechos, tales como efectos de fuerza y la especificación de figuras, son cargados al motor de dibujado háptico. Invocar *hlEndFrame* permite a un programa hacer diferentes cambios a la escena durante un marco mientras se dibuja, y hacer que todos estos cambios ocurran simultáneamente al final del marco.

5.2.9. Dibujado de figuras

El dibujado de figuras en HL se usa para definir superficies y objetos sólidos virtuales. Las figuras pueden ser creadas con diferentes primitivas geométricas tales como líneas, puntos y polígonos. El dibujado de figuras es realizado usando el método de proxy descrito en 5.2.3.

La geometría de la figura se especifica usando los comandos de OpenGL invocándolos entre las primitivas *hlBeginShape* y *hlEndShape*. HL captura la geometría especificada por los comandos de OpenGL y usa está geometría para realizar el dibujado háptico. Por ejemplo el siguiente código dibuja un rectángulo de 1×2 en el plano xy como una figura HL:

```
// Inicio de la figura háptica
hlBeginShape(HL_SHAPE_DEPTH_BUFFER, Id_Figura);
glBegin(GL_POLYGON);
glVertex3f(0,0,0);
glVertex3f(1,0,0);
glVertex3f(1,2,0);
glVertex3f(0,2,0);
glEnd();
// Fin de la figura háptica
hlEndShape();
```

Identificador de figuras

Cada figura debe tener un identificador único. El motor de dibujado lo utiliza para detectar los cambios de la figura de marco a marco. Antes de dibujar una figura se genera un identificador con la rutina *hlGenShape* que recibe el número de identificadores que se desea generar.

```
//Hluint es un tipo de dato entero definido en HL
Hluint Id_Figura;
Id_Figura = hlGenShape(1);
```

Tipo de figuras

Hay dos formas diferentes en las que HL captura la geometría definida por los comandos de OpenGL: usando el búfer de profundidad o usando el búfer de retroalimentación. Cuando se dibuja una figura debe especificarse que método va a usarse pasando `HL_SHAPE_DEPTH_BUFFER` o `HL_SHAPE_FEEDBACK_BUFFER`, cualquiera de los dos.

Bufer de profundidad

El búfer de profundidad de las figuras usa el búfer de profundidad de OpenGL para capturar la geometría del objeto. El búfer de retroalimentación almacena los puntos, segmentos de línea y polígonos para realizar el dibujado háptico, mientras que con el búfer de profundidad se realiza el dibujado háptico leyendo una imagen de éste. Cuando se invoca *hlEndShape*, la API lee una imagen del búfer de profundidad de OpenGL. Entonces, esta imagen es pasada al hilo de colisión y se usa para detectar algún contacto con el proxy. Cualquier modificación del búfer de profundidad hecha con los comandos de OpenGL será capturada como parte de la figura y dibujado háptico. Esto incluye cualquier rutina que genere polígonos u otras primitivas que modifiquen el búfer, como sombreado o textura.

Usando el búfer de profundidad sólo son palpables las porciones de geometría de la figura que puedan verse en la escena, es decir que la parte de atrás de la imagen no puede sentirse. Por defecto el dibujado háptico con el búfer de profundidad se realiza usando los parámetros actuales de vista de OpenGL. En este caso sólo pueden setirse las porciones de la figura que estén visibles. Sin embargo, si se habilita la vista de cámara de optimización, la HL automáticamente ajustará los parámetros de vista de OpenGL basándose en el movimiento y mapeo del dispositivo háptico en la escena. Esto permite que toda la figura pueda sentirse independientemente de las porciones de geometría que sean visibles. Ésto funciona para la mayoría de las figuras, sin embargo podría suceder que cuando se toca una figura con profundidad hubiera discontinuidades muy notorias, ranuras estrechas o tuneles. Para tales figuras es mejor utilizar el búfer de retroalimentación.

Bufer de retroalimentación

El búfer de retroalimentación usa el búfer de retroalimentación de OpenGL para capturar la geometría para el dibujado háptico. Cuando se indica que se va usar el búfer de retroalimentación y se invoca *hlBeginShape*, HL automáticamente habilita el búfer y envía a OpenGL el modo de dibujado de retroalimentación. Con este modo, antes de dibujar la geometría en la pantalla, las primitivas geométricas son guardadas en el búfer. Todos los comandos de OpenGL que generen puntos, líneas y polígonos serán capturados. Los comandos de OpenGL como los de textura y materiales serán ignorados. Cuando se invoca *hlEndShape*, las primitivas escritas en el búfer son guar-

dadas por el motor de dibujado háptico y usadas para computar la fuerza en el hilo de dibujado háptico.

5.2.10. Mapeo del dispositivo háptico a la escena gráfica

Se debe definir el mapeo apropiado del espacio de trabajo háptico a la escena gráfica. Con *hlMatrixMode* se indica que matriz se usará en el futuro cuando se ejecuten los comandos de manipulación de matriz. HL provee dos matrices para el mapeo del espacio háptico a la escena: `HL_VIEWTOUCH_MATRIX` y `HL_TOUCHWORKSPACE_MATRIX`.

Con la matriz de espacio de trabajo se realiza automáticamente el mapeo de la posición del proxy a la escena gráfica para detectar la colisión entre el proxy y las figuras hápticas. La matriz `VIEWTOUCH` es usa sólo cuando se desea mapear el espacio de trabajo a coordenadas de vista. Por ejemplo cuando se desea mapear la dimensión x del espacio de trabajo a la dimensión z de la escena. Un mapeo uniforme se indica con *hluFitWorkspace* después de *hlMatrixMode*. De este modo el movimiento del Phantom en la escena gráfica es uniforme aún cuando el espacio de trabajo no lo sea.

5.2.11. Dibujado del cursor

Necesitamos visualizar la posición relativa del proxy en la escena para interactuar con el ambiente virtual. El proxy será representado por un cursor en 3D. Para dibujarlo debemos obtener la matriz de transformación para mapearlo a la escena gráfica y determinar el tamaño adecuado para él. La matriz de transformación puede ser obtenida llamando *hlGetDoublev* con `HL_PROXY_TRANSFORM`.

```
HDdouble transform[16];
transform=hlgetDoublev(HL_PROXY_TRANSFORM,transform);
```

La escala para el cursor se obtiene con *hlScreenToModelScale*, esta función necesita la matriz de modelado, de vista y de proyección de OpenGL para hacer el mapeo adecuado. A continuación se muesta el código para dibujar el cursor en 3D usando la matriz de espacio de trabajo de HL:

```
#define CURSOR_SIZE_PIXELS 20
double gCursorScale = 1.0;

GLdouble modelview[6];
GLdouble projection[6];
GLdouble viewport[6];
HDdouble transform[16];

glGetDoublev(GL_MODELVIEW_MATRIX, modelview);
```

```
glGetDoublev(GL_PROJECTION_MATRIX, projection);
glGetDoublev(GL_VIEWPORT_MATRIX, viewport);

glPushMatrix();
hlMatrixMode(HL_TOUCHWORKSPACE);
hlLoadIdentity();
hluFitWorkspace(projection);
gCursorScale = hluScreenToModelScale(modelview, projection, viewport);
gCursorScale *= CURSOR_SIZE_PIXELS;
hlGetDoublev(HL_PROXY_TRANSFORM, transform);
glMultMatrixd(transform);
glScaled(gCursorScaleg, CursorScale, gCursorScale);
drawCursor();
glPopMatrix();
```

El proxy puede ser construído como un conjunto de puntos, líneas y polígonos pero sólo tiene un punto de contacto, es decir que si colisiona con una figura sólo hace contacto con ella en un punto.

5.2.12. Eventos

HL permite a los programas cliente ser informados vía funciones de retrollamada cuando varios eventos ocurren durante el dibujado háptico. Se debe pasar un puntero (de C) a la función que será llamada cuando el evento ocurra. Los eventos que pueden subscribirse incluyen tocar una figura, movimiento del dispositivo háptico y presionar un botón de la pluma del dispositivo.

5.2.13. Retrollamadas para los eventos

Para dar de alta una retrollamada para manejar un evento se usa la función *hlAddEventCallbak* como se muestra a continuación:

```
hlAddEventCallback(Hlenum event, HLuint shapeId, HLenum thread,
HleventProc fn, void *userdata)
```

A continuación se describen cada uno de los parámetros que usa la función anterior.

event
Los eventos que pueden registrarse son:

Se tocó una figura	`HL_EVENT_TOUCH`
Se dejó de tocar la figura	`HL_EVENT_UNTOUCH`
Movimiento del dispositivo	`HL_EVENT_MOTION`
Se presionó el botón 1 del dispositivo	`HL_EVENT_1BUTTONDOWN`
Se liberó el botón 1 del dispositivo	`HL_EVENT_1BUTTONUP`
Se presionó el botón 2 del dispositivo	`HL_EVENT_2BUTTONDOWN`
Se liberó el botón 1 del dispositivo	`HL_EVENT_2BUTTONUP`

shapeId
Identificador de la figura para la cual será detectado el evento. Se puede utilizar `ANY_OJECT` para que el evento se detecte con cualquier figura háptica.

thread
`HL_CLIENT_THREAD` indica que un hilo del cliente checará los eventos que han ocurrido por medio de la función *hlCheckEvents*.

`HL_COLLISION_THREAD` indica que el hilo de colisión, que está corriendo internamente, checará los eventos ocurridos e invocará a la función de retrollamada correspondiente.

fn
Puntero a la función de retrollamada.

userdata
Apuntador a los datos que serán pasados a la función.

5.3. Incorporación de Phantom con Qt

Como se mencionó, a través del Phantom se podrán seleccionar los vértices a los que se aplicará una fuerza externa. También se podra manipular la escena rotándola en x, y y z. Ya que el Phantom hará la función de cursor, cada vez que se genere algún evento del dispositivo, como cuando se mueva el Phantom, se rote la escena o se seleccionen vértices de la malla, la escena gráfica deberá redibujarse.

5.3.1. Manejo de eventos

En Qt, el hilo de eventos es el encargado de detectar los eventos de la ventana del sistema y envíarselos a los widgets. Cuando se requiere tratar eventos que no sean generados por la ventana del sistema, el método estático *QThread:postEvent* permite despertar al hilo de eventos y enviarle los eventos generados por otros hilos tratándolos de forma normal como si hubiesen sido generados por la ventana del sistema. Por lo tanto, se puede forzar a un widget a redibujar la escena gráfica desde otro hilo haciendo lo siguiente:

```
QWidget *mywidget;
QThread::postEvent( mywidget, new QPaintEvent( QRect(0,0,100,100) ), FALSE );
```

Este código le indica a mywidget que redibuje la escena gráfica de forma asíncrona añadiendo a la escena un rectángulo de 100×100. También se puede pasar `NULL` al constructor de la clase *QPaintEvent* para no añadir nada a la escena.

Como se mencionó, los eventos del dispositivo pueden checarse en un hilo creado por el cliente o el hilo de colisión. Nosotros decidimos aprovechar el hilo de colisión para no tener que crear otro hilo extra. Y los hilos son seguros sin necesidad de usar secciones críticas o candados pues *QThread::postEvent* asegura que sólo un hilo a la vez accese al contexto de dibujado.

5.3.2. Rotación

La escena se rota de acuerdo a la posición del proxy, siempre y cuando el botón 1 de la pluma del dispositivo esté presionado. Cuando el botón 1 está presionado se activa la rotación de la escena y se desactiva cuando éste es liberado. Para detectar estos eventos se usaron las funciones de retrollamada *button1DownCallbak* y *button1UpCallback*.

```
hlAddEventCallback(HL_EVENT_1BUTTONDOWN, HL_OBJECT_ANY, HL_COLLISION_THREAD,
&button1DownCallback, NULL);
hlAddEventCallback(HL_EVENT_1BUTTONUP, HL_OBJECT_ANY, HL_COLLISION_THREAD,
&button1UpCallback, NULL);

void HLCALLBACK Phantom::button1DownCallback(HLenum event, HLuint object,
                        HLenum thread, HLcache *cache, void *userdata){
    B1Down=TRUE; //Bandera que indica si el botón uno está presionado
}

void HLCALLBACK Phantom::button1UpCallback(HLenum event, HLuint object,
                        HLenum thread, HLcache *cache, void *userdata){
    B1Down=FALSE;
}
```

Si el botón 1 del dispostivo está presionado y la posición del cursor cambia la escena se rota como sigue:

1. Si el cursor se mueve en x la escena se rota sobre el eje y.
2. Si el cursor se mueve en y la escena se rota sobre el eje z.
3. Si el cursor se mueve en z la escena se rota sobre el eje x.

Para detectar el movimiento del dispositivo se dio de alta la siguiente función:

```
hlAddEventCallback(HL_EVENT_MOTION, HL_OBJECT_ANY, HL_COLLISION_THREAD,
&motionCallback, NULL);

void HLCALLBACK Phantom::motionCallback(HLenum event, HLuint object,
                        HLenum thread, HLcache *cache, void *userdata){
    if(Ph->B1Down==TRUE){
         hduVector3Dd proxy;
        hlCacheGetDoublev(cache, HL_PROXY_POSITION, proxy);
        Ph->xrot = (float)proxy[2]*180;
        Ph->yrot = (float)proxy[0]*180;
        Ph->zrot = (float)proxy[1]*180;
    }
}
```

5.3.3. Selección de vértices por medio del Phantom

La fuerza externa será aplicada sólo a los vértices de la malla que previamente hayan sido seleccionados por el usuario. El modo de selección se activa cuando el botón 2 del dispositivo háptico es presionado y se desactiva en el momento en que se libera el botón. Para detectar estos dos eventos se usaron las funciones de retrollamada *button1DownCallbak* y *button1UpCallback*.

```
hlAddEventCallback(HL_EVENT_2BUTTONDOWN, HL_OBJECT_ANY, HL_COLLISION_THREAD,
&button2DownCallback, NULL);
hlAddEventCallback(HL_EVENT_2BUTTONUP, HL_OBJECT_ANY, HL_COLLISION_THREAD,
&button2UpCallback, NULL);

void HLCALLBACK Phantom::button2DownCallback(HLenum event, HLuint object,
                        HLenum thread, HLcache *cache, void *userdata){
    Ph->B2Down=TRUE;
}

void HLCALLBACK Phantom::button2UpCallback(HLenum event, HLuint object,
                        HLenum thread, HLcache *cache, void *userdata){
    Ph->B2Down=FALSE;
}
```

Habrá dos tipos de selección virtual, por caras o por capas. En la selección por caras, la fuerza externa sólo afectará a los vértices de las caras seleccionadas. En la selección por capas, el usuario elige los vértices y el número de capas de sus vecinos directos que serán afectados por la fuerza externa. Por ejemplo, cuando seleccione a P_i y el número de capas que eligió es igual a uno, automáticamente también sus tres

vecinos $P_{N_1(i)}$, $P_{N_2(i)}$ y $P_{N_3(i)}$ serán marcados. Para dos capas de selección, se marcará al vértice P_i, sus tres vecinos directos $P_{N_1(i)}$, $P_{N_2(i)}$ y $P_{N_3(i)}$, y a los tres vecinos directos de $P_{N_1(i)}$, $P_{N_2(i)}$ y $P_{N_3(i)}$ como se muestra en en la figura 5.5

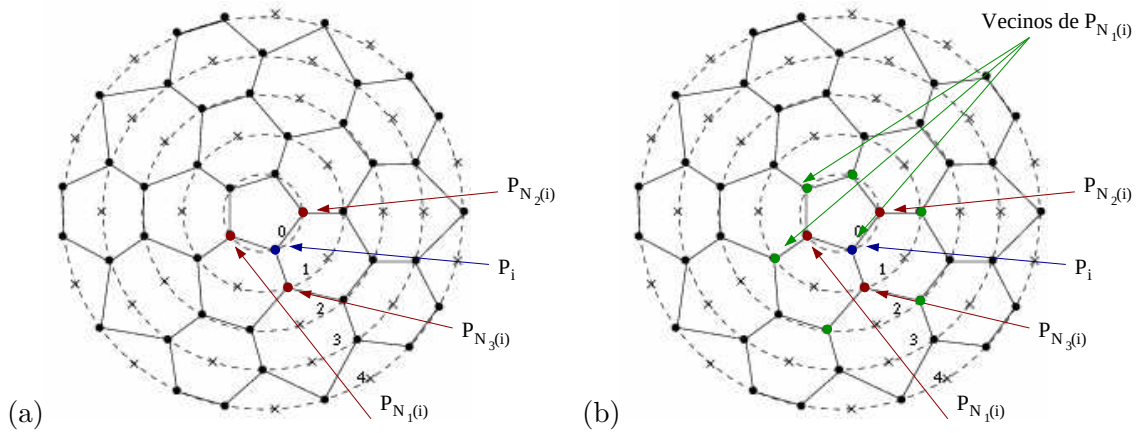


Figura 5.5: Selección de vértices por capas. (a)Selección del punto P_i y una capa de sus vecinos. (b) Selección el punto P_i y dos capas de sus vecinos.

Para distinguir las capas de vecinos de P_i, en la figura 5.5 la primer capa fue pintada de color rojo y la segunda de color verde. Podemos observar en la figura 5.5 (b) que al marcar la segunda capa, uno de los tres vecinos de $P_{N_1(i)}$ es precisamente el vértice P_i que previamente ya había sido seleccionado. Significa que algunos puntos serán marcados dos veces pero ésto no tiene ninguna repercusión en el sistema.

Para la selección por caras, primero se definió a cada una de ellas como una figura háptica con la primitiva para dibujar polígonos que proporciona la HL como se muestra a continuación:

```
// No_Faces es el número total de caras de la malla
for(f=0; f<No_Faces; i++ ){
   ShapeId[f] = hlGenShape(1);
   hlBeginShape(HL_SHAPE_DEPTH_BUFFER, ShapeId[f]);
   drawFace(f);
   hlEndShape();
}
```

De este modo se pueden sentir las caras del objeto. La detección de colisión se realiza de forma automática por el hilo de colisión de la HL por lo que no es necesario implementar un algoritmo de detección de colisión. Finalmente se define una función de retrollamada que responda al evento `HL_EVENT_TOUCH` para saber cuando el proxy toca alguna de las caras hápticas del objeto. Para lo cual se dió de alta la función *touchSahpeCallback* como se muestra a continuación:

```
// No_Faces es el número total de caras de la malla
for(f=0; f<No_Faces; f++){
   hlAddEventCallback(HL_EVENT_TOUCH, ShapeId[f], HL_COLLISION_THREAD,
   &touchShapeCallback, NULL);
}

void HLCALLBACK touchShapeCallback(HLenum event, HLuint object, HLenum thread,
                        HLcache *cache, void *userdata){
   if(B2Down==TRUE){
   // Se almacena la posición del punto de colisión
   hlCacheGetDoublev(cache, HL_PROXY_POSITION, proxy);
   id_selectedFace=object;   // id de la cara seleccionada
   }
}
```

Observe que la función *touchShapeCallback* se registró para todas las caras del objeto, pasándole cada uno de los identificadores de éstas. Así cuando el proxy tiene contacto con cualquiera de ellas, siempre se invoca a la misma función y el parámetro object nos indica el identificador de la cara que fue seleccionada.

Para la selección por capas, por cuestiones de memoria, no se definió a cada vértice como una figura háptica. Se aprovechó que con la función *touchShapeCallback* se identifica la cara que está colisionando con el proxy y suponemos que le usuario desea seleccionar al vértice de la cara que está más próximo al punto de colisión. El punto de colisión se obtiene en la función *touchShapeCallback* con el método *hlCacheGetDoublev*. Una vez seleccionada el vértice se marcan las capas de sus vecinos utilizando el algoritmo recursivo 4.

Procedimiento 4 Seleccion_por_capas(P, nc)

Entrada: El vértice P seleccionado y el número de capas nc de sus vecinos que también se desea seleccionar.

if $nc > 0$ **then**
  $nc \leftarrow nc - l$
  for $1 \leq j \leq 3$ **do**
    $P_{N_j(i)}$ se marca como seleccionado
    Selección_por_capas($P_{N_j(i)}$, nc)

5.3.4. Comunicación entre la interfaz háptica y gráfica

HD y HL fueron diseñados para comunicar los eventos del dispositivo por medio de retrollamadas. En Qt la comunicación entre objetos se realiza por medio de señales. Para que la interfaz háptica pueda indicar a la interfaz gráfica cualquier evento que

ocurra con el dispositivo se invoca la función de retrollamada para dicho evento y ésta envía una señal a la interfaz gráfica.

En Qt para que dos objetos pueden comunicarse primero deben conectarse usando el método *connect* de la clase QObject.

```
connect(emisor, señal, receptor, metodo );
```

El `emisor` comunica al `receptor` que ha ocurrido un evento envíandole una `señal` para qu ejecute un `método`. Para emitir una señal en Qt se usa el método *emit*:

```
emit Re_Es();
```

Para convertir las retrollamadas que generan los eventos del dispositivo háptico a señales de Qt, hacemos la conexión entre la escena háptica y la escena gráfica con la siguiente instrucción:

```
connect(EsHap, SIGNAL(Re_Es()), EsGra, SLOT(ReEsGra( )) );
```

Con esto, la escena gráfica `EsGra` ejecuta el método `ReEsGra()` cada vez que el objeto de la escena háptica `EsHap` emita la señal `Re_Es`. Así, para las funciones de retrollamada que deban redibujar la escena gráfica y para aseguar que sólo un hilo a la vez active el contexto de dibujado, el método `ReEsGra()` debe definirse como sigue:

```
void ReEsGra(){
   QThread::postEvent( this, new QPaintEvent( NULL, FALSE ) );
}
```

Por ejemplo, para que la función de retrollamada *motionCallback*, de la sección 5.3.2 que detecta el movimiento del Phantom, envíe una señal de redibujado al objeto de la escena gráfica de rescribirse como sigue:

```
void HLCALLBACK Phantom::motionCallback(HLenum event, HLuint object,
                        HLenum thread, HLcache *cache, void *userdata){
    if(Ph->B1Down==TRUE){
         hduVector3Dd proxy;
        hlCacheGetDoublev(cache, HL_PROXY_POSITION, proxy);
        Ph->xrot = (float)proxy[2]*180;
        Ph->yrot = (float)proxy[0]*180;
        Ph->zrot = (float)proxy[1]*180;
        emit Re_Es();
    }
}
```

5.4. Resultados

5.4.1. Deformación de una esfera

En la figura 5.6 (a) y (b) se muestra la selección por capas, los vértices marcados son los vértices seleccionados por el usuario y a los cuales se les aplicará una fuerza externa. Para la figura 5.6(a) se seleccionó el punto que se encuentra justo debajo del cursor del Phantom y las dos capas de sus vecinos directos. Para la figura 5.6(b) se marcaron cuatro capas de vecinos y el punto seleccionado con el cursor. En la figura 5.6(c) se muestra la selección por caras, todos los vértices que forman parte de las caras de color azul se verán afectados por la fuerza externa.

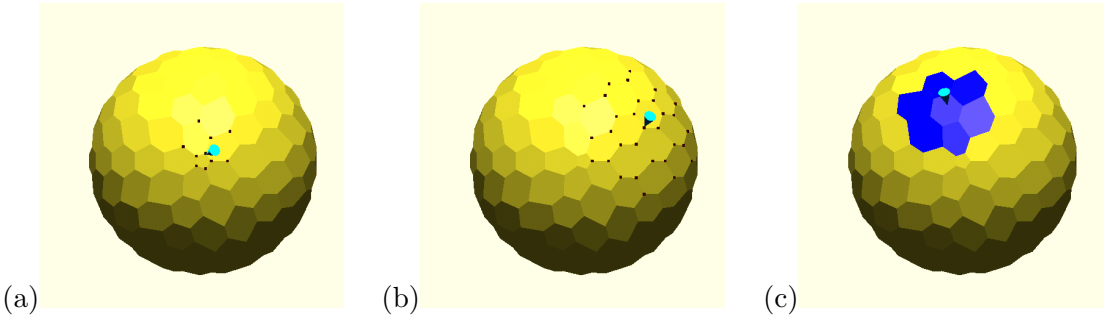


Figura 5.6: (a) Selección de un punto y dos capa de sus vecinos. (b) Selección de un punto y cuatro capas de sus vecinos. (c) Selección por caras.

En la figura 5.7 se muestra la deformación de una esfera construída con mallas de simplejos, se le aplicó una fuerza externa de 0.03 newtons sobre los vértices de las caras marcadas de azul. En la figura 5.7(a) se aprecia la esfera original y las caras seleccionadas. De la figura 5.7(b) a la (h) se muestra la deformación. Por último, en la figura 5.7(i) se rotó la escena con el Phantom para observar mejor la deformación.

5.4.2. Interfaz gráfica

Se utilizó OpenGL y Qt para implementar una interfaz gráfica con la que el usuario puede interactuar con los modelos deformables a través del Phantom (ver figura 5.8). La interfaz permite realizar lo siguiente:

1. Modificar la resolución de las mallas de simplejos.

2. Seleccionar por capas o por caras a determinados vértices de la malla para aplicarles fuerza externa.

3. Visualizar los vértices, las caras y las aritas de la malla, así como los centroídes de las caras y sus normales.

4. Habilitar o deshabilitar las luces de la escena.

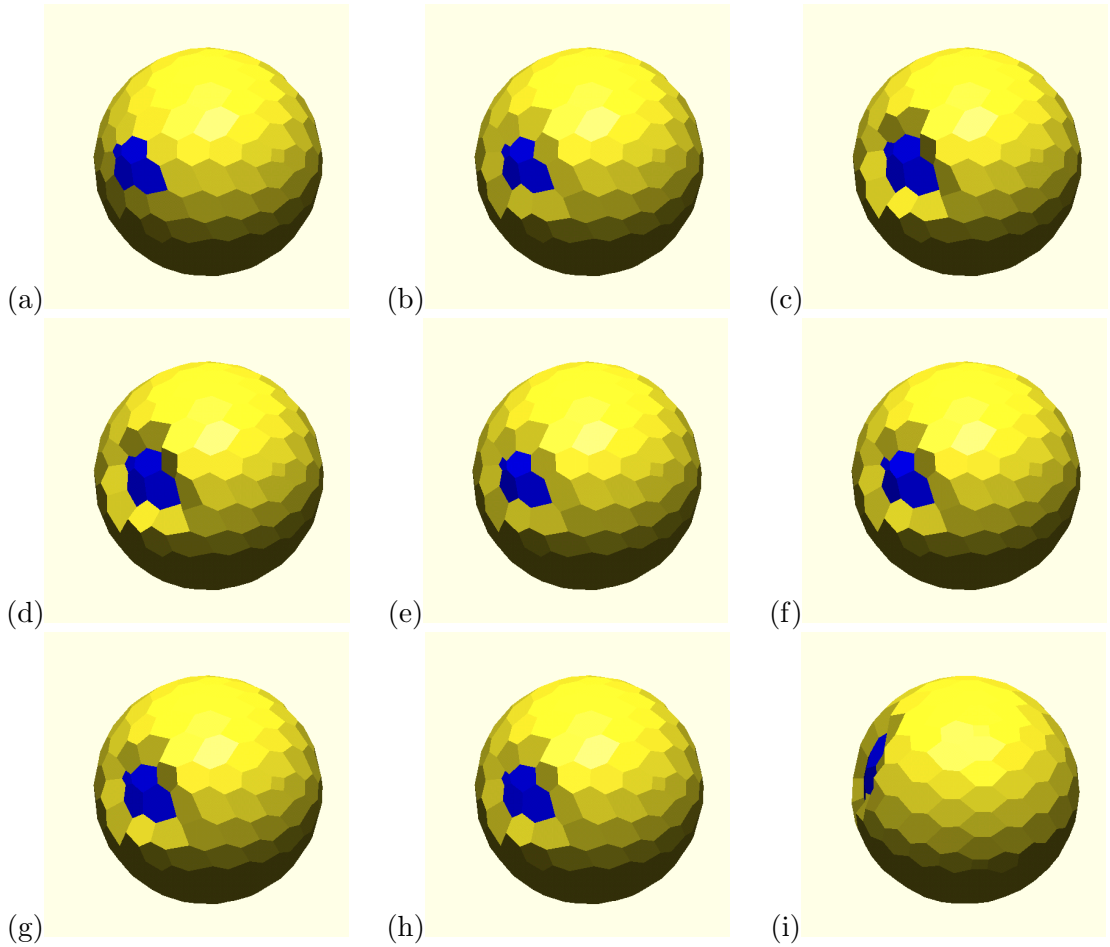


Figura 5.7: Deformación de una esfera. La esfera fue construida con 7 etapas y una frecuencia de 5, como se indica en [1]; esto da como resultado una malla con 360 vértices. Las condiciones para el experimento fueron las siguientes: $F_{\text{ext}} = 0.03, \Delta t = 0.05$, $k = 8, k_r = 0.7$, $m = 1$ y $\beta = 0.09$

5. Modificar las características elásticas de la malla, cambiando el factor de elasticidad k, el factor de rigidez k_r, y el factor de amortiguamiento β.

6. Definir el vector de fuerza externa que se aplica a los vértices seleccionados.

7. Deformar la malla hasta el punto en que se estabiliza su movimiento.

8. Reiniciar el motor de cálculo numérico y regresar la malla a su forma original.

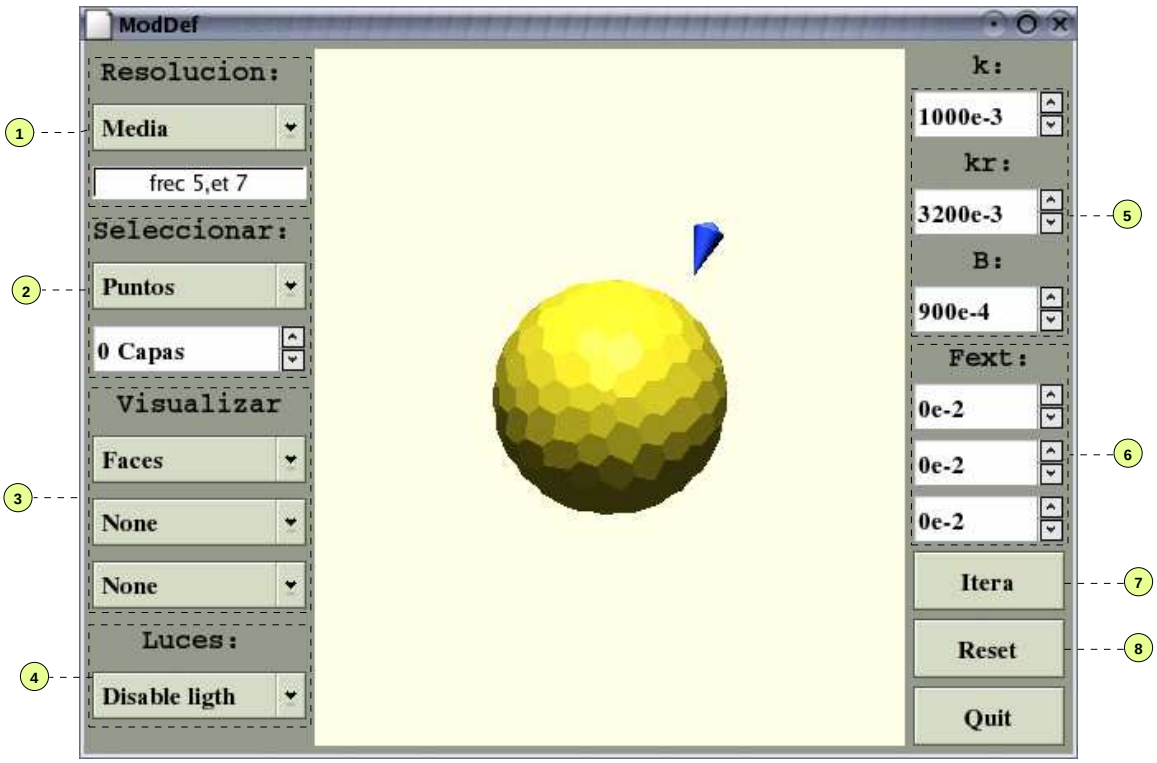


Figura 5.8: Interfaz gráfica

Capítulo 6

Conclusiones y Trabajo a Futuro

6.1. Conclusiones

En el mundo real los objetos están compuestos por un conjunto de partículas por lo que la representación de objetos por mallas de simplejos es muy ventajosa para simular objetos deformables. En la malla de simplejos cada vértice representa una partícula del objeto, para nuestro sistema cada partícula está conectada a sus tres vecinos por medio de resortes. De este modo, cuando se aplica una fuerza externa sobre cualquiera de los puntos de la malla se ven afectados también sus vecinos, dando a la malla una consistencia elástica.

A continuación se presentan lo que se concluyó en cuanto a los métodos de discretización, el sistema propuesto para integrar un comportamiento elástico a las mallas (Véase la sección 4.2), la integración del Phantom, el algoritmo de detección y los resulados obtenidos:

Métodos de discretización
El tiempo que se requiere para calcular la deformación de la malla es directamente proporcional a la complejidad del método de discretización que se utilice, por eso es conveniente utilizar un método muy rápido, pero que además sea preciso. Después de analizar diferentes métodos de discretización se concluyó que, las ecuaciones que describen la deformación de la malla pueden discretizarse de manera simple y eficiente usando el método de diferencias finitas. Siendo que su precisión es suficiente para la animación de objetos deformables ofreciéndonos un margen de error bastante pequeño.

Sistema propuesto

- Por medio de diferentes simulaciones, como el rebote de una pelota, la deformación de una esfera a un cubo, entre otras, se probó la efectividad de usar el sistema mecánico simple de masa, resorte y amortiguador para integrar un comportamiento elástico a las mallas de simplejos.

- Los objetos que simulamos no están huecos, de hecho se incluyó un mecanismo para simular una fuerza de rigidez que se opone a la deformación. Para el sistema que se propuso, esta fuerza es simulada por los resortes conectados del centro de la esfera a cada uno de los vértices de la malla. De esta forma se logró que la malla tuviera mayor estabilidad aunque todavía puede obtenerse una simulación más realista agregando más resortes, por ejemplo, utilizando mallas de simplejos tridimensionales (de orden 3).

- Modificando los valores de k, k_r y β se pueden simular diferentes tipos materiales con distintas características elásticas ya que la consistencia de los resortes controla las propiedades elásticas del modelo. El movimiento de la malla es mucho más estable cuando el sistema es sobreamortiguado o críticamente amortiguado, no así para el tipo de movimiento subamortiguado pues los resortes oscilan. Se debe tener mucho cuidado con los valores que se asignan a los parámetros de la malla pues el sistema se puede desestabilizar, de hecho k y β no pueden valer cero.

- El incremento de tiempo Δt es un factor sumamente importante en las simulación de los modelos deformables, entre más pequeño sea, la deformación de la malla es más estable y precisa, sobre todo cuando se tiene un número muy grande de partículas, como es el caso de la malla de resolución alta.

- Si el factor de amortiguamiento supera al factor de elasticidad, y el valor del factor de elasticidad es muy pequeño, el comportamiento de la malla es muy parecido a la plastilina, es decir, que aunque la fuerza ceda la malla no recupera su forma original.

Incorporación del Phantom
Fue posible integrar el Phantom con OpenGL a través de las bibliotecas HL y HD, a pesar de que la comunicación entre objetos en OpenGL se realiza por señales y tanto HL como HD fueron diseñadas para usar retrollamadas. Para convertir las retrollamdas de los eventos que genera el dipositivo háptico a señales de Qt, fue necesario implementar una clase para el manejo de eventos del dipositivo, así como evitar problemas de concurrencia con los hilos de Qt, que manejan los eventos de la interfaz gráfica, usando el método *QThread:postEvent*. De este forma se logró comunicar la interfaz háptica y la interfaz gráfica.

Algoritmo de detección de colisiones
La eficiencia del algoritmo de detección de colisión es otro punto muy relevante para el tiempo de respuesta del sistema. Cada vez que el Phantom se mueve el hilo

de colisión checa que la posición del proxy no esté tocando la malla o incluso penetrándola. Para este trabajo se utilizó el algoritmo de detección de colisiones que incluye la biblioteca del Phantom, el cual es bueno pero no muy eficiente, ya que para detectar la colisión entre un objeto y el proxy se debe definir la geometría del objeto en la escena háptica, de esta forma el objeto es considerado un objeto sólido y pueda ser tocado. Así, para cada objeto tocable es necesario dibujar su geometría dos veces, una para la escena háptica y otra para visualizar los objetos en la escena gráfica.

Para entender la magnitud de este problema, pongamos el ejemplo de una malla de resolución alta, su número de caras es bastante alto, aproximadamente de 5400 caras, si cada cara se dibuja dos veces tenemos un total 10800 objetos que deben redibujarse cada vez que el Phantom genera algún evento, como presionar un botón o cuando el dispositivo se mueve. El dibujado de la escena se vuelve muy lento y el dibujado del cursor con respecto al movimiento del dispositivo háptico no se realiza con sincronía. Por tanto, para para integrar el dispositivo a la escena gráfica y seleccionar los puntos de la malla a los cuales se les aplicará una fuerza externa, la malla debe tener una resolución prudente para obtener una respuesta del sistema en tiempo real y lograr sincronizar el dibujado del proxy con respecto al movimiento del dispositivo háptico.

Resultados

- Se verificó que es posible incorporar un comportamiento elástico e inelástico a los modelos deformables basados en mallas de simplejos, aplicando la ley de Hooke y la ley de movimiento de Newton.

- Conviene tener una resolución alta para que los objetos tengan una mejor apariencia visual pero esto implica que el motor de cálculo numérico realice mucho más operaciones. Si a ésto le sumamos, que para que el sistema sea estable el incremento de tiempo debe ser más pequeño conforme la resolución de la malla se incrementa, la respuesta del sistema al aplicar una fuerza externa no se puede obtener en tiempo real.

- En el sistema realizado se logró que la detección de colisión para una esfera de baja y media resolución se hiciera en tiempo real, no así para la alta resolución. En cuanto el cálculo de la deformación de la malla al aplicar una fuerza externa, la respuesta del sistema tampoco se pudo obtener en tiempo real: primero deben seleccionarse los vértices y después se aplica la fuerza externa. El motor de cálculo numérico itera hasta que la malla se estabiliza, es decir, cuando las fuerza externa e interna se equilibran (hasta un valor proporcionado de error).

- El Phantom tiene un sólo punto de contacto, por consiguiente la manipulación de los objetos es muy límitada y aún mejorando el tiempo de respuesta del sis-

tema, el dispositivo háptico no permite realizar en tiempo real la manipulación de la malla en diferentes partes de su superficie. Si se desea aplicar fuerza sobre diferentes secciones no adyacentes de la malla, el usuario sólo puede seleccionar una a la vez y después aplicar la fuerza externa al conjunto de regiones seleccionadas.

6.2. Trabajo a Futuro

Consideramos que los siguientes cambios pueden ser realizados al sistema:

Desarrollar un algoritmo eficiente para la detección de colisión, usando estructuras de datos para hallar más rápido el punto de contacto [29, Cap. 2]. Así, se podrá suministrar fuerza externa en tiempo real y obtener de inmediato la respuesta del sistema.

Par mantener la solución dentro de un margen de error aceptable y en los límites que las restricciones marquen, sería adecuado implementar un algoritmo estabilizador de la restricción, con lo cual se asegura que la deformación de la malla converja rápidamente [30, 31, 32].

El Phantom sólo tiene un punto de contacto y como consecuencia la fuerza externa sólo puede aplicarse en una parte muy límitada de la superficie de la malla. Sería muy conveniente integrar un guante virtual para tener cinco puntos de apoyo simulando un punto de apoyo por cada dedo de una mano humana, de esta forma el usuario tendría mayor control sobre el objeto y podría aplicar fuerza en diferentes puntos de la malla.

Los objetos fueron dibujado en 3D pero el monitor de la computadora nos limitó a una visualización en 2D. Integrando unas gafas esteroscópicas el usuario podría visualizar los objetos y la escena en 3D dándole una mejor orientación de ubicación en la escena y facilitar la interacción y manipulación con los objetos.

Sabemos que al aplicar una fuerza externa sobre los resortes éstos tienen un límite de elasticidad para el que se cumple la ley de Hooke, después de este límite el objeto no recupera su forma original. Estas restricciones no fueron consideradas en el sistema y sería interesante incluirlas en las ecuaciones matemáticas. Por último podría mejorarse el sistema, si la estructura de la malla pudiera sufrir fracturas, ya que en la vida real cuando la fuerza externa es excesiva el objeto puede romperse.

Apéndice A

Apendice I

A.1. Algoritmos de los métodos de discretización

Procedimiento 5 Método de diferencias finitas

Entrada: t_0, x_0, v_0, Δt, k, m, β, F'_{ext}, n, l
Salida: $x(i)$, $v(i)$, for $0 < i < n$

$\omega^2 \leftarrow k/m$
$2\lambda \leftarrow \beta/m$
$x(-1) = x(0) = x_0$, $v(0) = v_0$, $t(0) = t_0$
$a \leftarrow 2 - 2\lambda * \Delta t - \omega^2 * \Delta t * \Delta t$
$b \leftarrow 2\lambda * \Delta t - 1$
$c \leftarrow \Delta t * \Delta t * F'_{\text{ext}}/m$
for $1 \leq i < n$ **do**
  $x(i) \leftarrow a * x(i-1) + b * x(i-2) + c$

Procedimiento 6 Método de Euler

Entrada: t_0, x_0, v_0, Δt, k, m, β, F'_{ext}, n
Salida: $x(i)$, $v(i)$, for $0 < i < n$

$\omega^2 \leftarrow k/m$
$2\lambda \leftarrow \beta/m$
$F_{\text{ext}} \leftarrow F'_{\text{ext}}/m$
$x(0) = x_0$, $v(0) = v_0$, $t(0) = t_0$
for $1 \leq i < n$ **do**
  $t(i) = t_0 + i * \Delta t$
  $x(i) \leftarrow x(i-1) + \Delta t * v(i-1)$
  $v(i) \leftarrow v(i-1) + \Delta t * g(x_i(i-1), v(i-1))$

function $g(x, v)$
return $F_{\text{ext}} - 2\lambda * v - \omega^2 * x$

Procedimiento 7 Método de Heun

Entrada: t_0, x_0, v_0, Δt, k, m, β, F'_{ext}, n
Salida: $x(i)$, $v(i)$, for $0 < i < n$

$\omega^2 \leftarrow k/m$
$2\lambda \leftarrow \beta/m$
$F_{\text{ext}} \leftarrow F'_{\text{ext}}/m$
$x(0) = x_0$, $v(0) = v_0$, $t(0) = t_0$
for $1 \leq i < n$ **do**
  $t(i) = t_0 + i * \Delta t$
  $p \leftarrow x(i-1) + \Delta t * v(i-1)$
  $q \leftarrow v(i-1) + \Delta t * g(x(i-1), v(i-1))$
  $x(i) \leftarrow x(i-1) + 0.5 * \Delta t * [v(i-1) + q]$
  $v(i) \leftarrow v(i-1) + 0.5 * \Delta t * [g(x(i-1), v(i-1)) + g(p, q)]$

function $g(x, v)$
return $F_{\text{ext}} - 2\lambda * v - \omega^2 * x$

Procedimiento 8 Método RK4

Entrada: t_0, x_0, v_0, Δt, k, m, β, F'_{ext}, n
Salida: $x(i)$, $v(i)$, for $0 < i < n$

$\omega^2 \leftarrow k/m$
$2\lambda \leftarrow \beta/m$
$F_{\text{ext}} \leftarrow F'_{\text{ext}}/m$
$x(0) = x_0$, $v(0) = v_0$, $t(0) = t_0$
for $1 \leq i < n$ **do**
  $t(i) = t_0 + i * \Delta t$
  $f_1 \leftarrow v(i-1)$
  $g_1 \leftarrow g(x(i-1), v(i-1))$
  $f_2 \leftarrow v(i-1) + 0.5 * \Delta t * g_1$
  $g_2 \leftarrow g(x(i-1) + 0.5 * \Delta t * f_1, v(i-1) + 0.5 * \Delta t * g_1)$
  $f_3 \leftarrow v(i-1) + 0.5 * \Delta t * g_2$
  $g_3 \leftarrow g(x(i-1) + 0.5 * \Delta t * f_2, v(i-1) + 0.5 * \Delta t * g_2)$
  $f_4 \leftarrow v(i-1) + \Delta t * g_3$
  $g_4 \leftarrow g(x(i-1) + \Delta t * f_3, v(i-1) + \Delta t * g_3)$
  $f \leftarrow \frac{1}{6} * (f_1 + 2 * f_2 + 2 * f_3 + f_4)$
  $g \leftarrow \frac{1}{6} * (g_1 + 2 * g_2 + 2 * g_3 + g_4)$
  $x(i) \leftarrow x(i-1) + \Delta t * f$
  $v(i) \leftarrow v(i-1) + \Delta t * g$

function $g(x, v)$
return $F_{\text{ext}} - 2\lambda * v - \omega^2 * x$

A.2. Código en perl del método de Euler para la solución aproximada de la ecuación 3.2

```
#!/usr/bin/perl
$h=0.05;
$k=2;
$m=1;
$B=1.2;
$Fext=15;

$w2=$k/$m;
$dosl=$B/$m;
$x_k=0;
$y_k=0;
$t_k=0;
for($i=1;$i<1000;$i++){
        $x=$x_k + $h*y_k;
        $y_k=$y_k + $h*g($t_k,$x_k,$y_k);
        $t_k=$t_k+$h;
        $x_k=$x;
}

#En perl los parámetros de una función se indican con $
sub g($$$){#t,x,y
        my $t = shift(@_);
        my $x = shift(@_);
        my $y = shift(@_);
        return $Fext/$m - $dosl*$y - $w2*$x;
}
```

Apéndice B

Apendice II

B.1. Estructura de datos para las mallas de simplejos

Una malla de simplejos está compueta de puntos, arista y caras. Las estructuras de datos en C propuesta en [1] para representar las mallas de simplejos y sus elementos se muestran a continuación:

```
typedef struct _Malla{
  struct _Punto   * pri_Punto;
  struct _Arista  * pri_Arista;
  struct _Cara    * pri_Cara;
  float cet[3];
  float cet0[3];
}* Malla;
```

La información que se almacena en esta estructua es la siguiente:

- En `pri_Punto` se almacena el primer elemeto de la lista de puntos.
- `pri_Arista` es la variable que almacena el primer elemento de la lista de aristas.
- En `pri_Cara` se guarda el primer elemento de la lista de caras.
- `cet` representa la coordenada del centro de la malla en coordenadas cartesianas en 3D.
- `cet0` es la coordenada inicial del centro de la malla en coordenadas cartesianas en 3D.

La estructura de datos para almacenar la información de los vértices es:

```
typedef struct _Punto{
  struct _Punto  *vecinos[3];
  struct _Arista *arista[3];
  struct _Cara   *cara[3];
  float cp[3];
  float cp0[3];
  struct _Punto  *ant;
  struct _Punto  *sig;
  /*Manipulación háptica*/
  int sel;
}*Punto;
```

Para cada punto se almacenanan sus tres vecinos en `vecinos`, las tres aristas que están asociadas a él en el arreglo `arista`, y sus tres caras en `cara`. También se almacena la coordenada del vertice en coordenadas cartesianas en 3D en `cp`, y su coordena inicial `cp0`. Con esta estructura se genera una lista de todos los puntos de la malla. La lista es ligada con los campos de la estructura `ant` y `sig`, donde `ant` es el vértice que precede al punto en la lista y `sig` es el vértice que le sigue. A esta estructura se añadio el campo `sel`, que sirve como bandera para saber si el vértice fue seleccionado y va a ser afectado por la fuerza externa.

Para la representación de las aristas se usó la siguiente estructura:

```
typedef struct _Arista{
  struct _Punto   *vecinos[2];
  struct _Cara    *cara[2];
  struct _Arista  *ant;
  struct _Arista  *sig;
}*Arista;
```

- En `vecinos[2]` se guardan los dos vertices de la arista,
- En `cara[2]` se almacenana sus dos caras adyacentes.
- `ant` es la arista precedente en la lista de aristas.
- Por último, `sig` es la arista siguiente en la lista.

Para las caras de la malla se mantiene una lista de todas ellas. La estructura que se usó para representarlas es:

```
typedef struct _Cara{
  struct _Elemento_Cara *pri_elemento;
  float NCcp[3];
  float cent[3];
  float color[3];
  struct _Cara *ant;
  struct _Cara *sig;
  /*Manipulación háptica*/
  HLuint IdFig;
}*Cara;
```

Una cara está compuesta por aristas y vértices, por tanto en su esctructura se mantiene una lista de sus elementos, `pri_elemento` es un apuntador al primer elemento de esta lista. Para dar efectos de iluminación a las caras en OpenGL se necesita definir su normal por medio de un vector en 3D. En este caso, la normal es unitaria y parte del centroíde de las caras. En `NCcp` y `cent` se almacenan la normal de la cara y su centroíde respectivamente. El color de la cara se guarde en el arreglo `color[3]` en formato RGB. Para ligar la lista de las caras se utiliza un apuntador a la cara precedente `ant` y un apuntador a la siguiente cara `sig`. Se añadio a la estrectura el campo `IdFig` para la manipulación háptica.

Cada elemento de la cara representa un vértice de su geometría y su estructura es la siguiente:

```
typedef struct _Elemento_Cara{
  struct _Punto *punto;
  struct _Arista *arista;
  struct _Cara *cara;
  struct _Elemento_Cara *ant;
  struct _Elemento_Cara *sig;
}*Elemento_Cara;
```

- `punto`, es un vértice de la cara.
- `arista`, es la arista compuesta por `punto` y el vértice del siguiente elemento de la cara.
- `cara`, es la otra cara del vértice almacenado en `punto`.
- `ant`, es el elemento precedente en la lista de elementos de la cara.
- `sig`, es el siguiente elemento en la lista.

B.2. Estructura de datos para el motor de cálculo numérico

La deformación de las mallas es idependiente de su esctructura de datos, por tal motivo la infomación que requiere el motor de cálculo numérico (motor de deformación) para computar la respuesta del sistema se almacenó en la siguiente estructura:

```
typedef struct _Deform_engine{
  long int  n_p;
  float     **Fext;
  float     Fe;
  float     B;
  float     k;
  float     kr;
  float     **mag_li;
  float     *mag_li2;
  float     error;
  float     **pi_1;
}*Deform_engine;
```

El número total de vértices de la malla se almacena en `n_p`. Como se mencionó en la sección 1.2.3 eformación de la malla depende directamente de la fuerza externa, el factor de amortiguamiento y el factor de elasticidad de los resortes.

El usuario proporciona la magnitud de la fuerza externa que se aplicará a los vértices seleccionados. Esta fuerza se guarda en `Fe`. Para cada uno de los `np` puntos se cácula su vector de fuerza con magnitu `Fe` y va dirigido hacia el centro de la malla. Ya que este vector tiene tres componentes, x, y y z, se requiere una matriz de npx3 para almacenar los `np` vectores. En la estructura de datos dicha matriz está representada por `Fext`.

El factor de amortiguamiento en la estructura del motor de deformación es `damp_f`. El factor de elasticidad para los resortes entre vecinos es `k` y el factor de elasticidad de los resortes conectados hacia el centro de la malla es `k_r`. La fuerza interna de los resortes no sólo depende de su factor de elasticidad sino también de la variación de su longitud. La longitud inicial de los tres resortes vecinos de cada uno de los `np` puntos se almacenó en una matriz bidimensional de npx3. Esta matriz está representada en la estructura de datos por `mag_li`. Lo mismo sucede para la variación de la longitud de los resorte conectados de los `np` puntos al centro de la malla. La longitud inicial de estos resortes en la estructura se guardó en `mag_li2`.

Para discretizar las ecuaciones que describen el movimiento de la malla utilizando el método de diferencia finitas descrito en la sección 3.4.1, el motor de deformación necesita la posición actual y anterior de los vértices. La posición de los vértices en el

tiepo t está almacenada en la estructura de los puntos y la posición en $t-1$ se almacenó en la matriz `pi_1`, de dimensiones npx3, pues el dominio de las coordenadadas de los vértices es en $\mathbb{R}^3$. La memoria de las matrices utilizadas en esta estructura se reservaron en C de forma dinámica.

El motor de deformación itera hasta que el movimiento de la malla se ha estabilizado, es decir, cuando el error es menor a un umbral previamente establecido. En la sección 4.3.1 se explica como calcular el este error. En la estructura de datos del motor de deformación, el error se guarda en el campo `error`,

Apéndice C

Apéndice III

C.1. Instalación del Phantom

C.1.1. Requerimientos del sistema

Hardware

Procesador intel II o superior o el procesador AMD con puerto paraleo PCI. 30 MB de espacio en el disco duro y 32 MB de RAM.

Software

Red Hat Fedora Core 1, SUSE Linux 9 o superior.

C.1.2. Instalación de los controladores del dispositivo

El proceso de instalación sólo puede realizarse siendo super usuario.

Los controladores del dispositivo vienen en un archivo rpm en el disco de instalación. El paquete se intala con:

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

Además se necesita tener instalado el paquete glut. Puede usarse la versión libre de glut de de Mesa.

C.1.3. Puerto FireWire IEEE-1394

Para utilizar el dispositivo háptico Phantom Omni con el puerto FireWire IEEE-1394 debe cargarse el modulo 1394 del kernel con:

```
/sbin/moprobe raw1394
```

La salida del comando debe ser raw1394 si el modulo fue cargado correctamente de lo contrario ocurrio un erro. Después, debe indicarse la ruta del archivo de la licensia license.lic con la variable de entorno `OH_SDK_LICENSE_PATH`. Por ejemplo, suponiendo que la licencia está en el directorio OHL la ruta del archivo se indica como sigue:

```
export OH_SDK_LICENSE_PATH=$HOME/OHL/license.lic
```

Para checar si la ruta es correcta se utiliza el comando echo.

```
echo $OH_SDK_LICENSE_PATH
```

C.1.4. Configuración del Phantom

Para configurar el Phantom no es necesario ser super usuario. Para comenzar a usar el Phantom es necesario especificar el modelo del Phantom ejecutando:

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

Después aparece la interfaz que se muestra en la imagen C.1 por medio de la cual se configura el dispositivo háptico.

En este caso se está utilizando el modelo Omni del Phantom y su etiqueta es que es la etiqueta por default. Pero puede cambiarse la etiqueta añadiendo una nueva con el boton Add. En la parte inferior izquierda se despliega el puerto que se está ocuapando para el dispositivo y el número de serie. Si no se cuenta con una licencia el número de serie es cero y el dispositivo no puede utilizarse. Se pueden agregar varios dispositivos hápticos, de ser así la sección Dual configuration se habilita. Por último, debe resta reiniciarse la computador.

Para verificar que la instalación de los controladores y completar la configuración del Phantom con se corre el programa PHANToMTest. La interfaz para probar la configuración del Phantom se muestra en la figura C.2

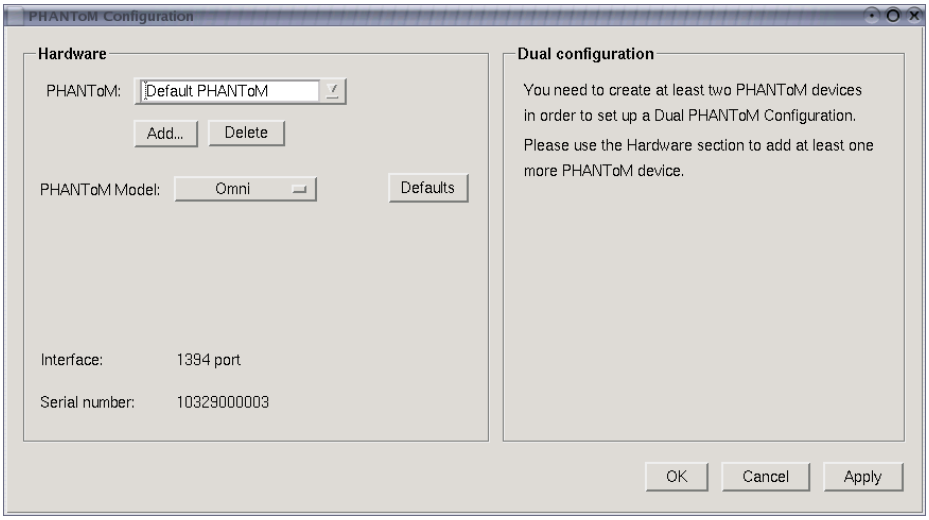


Figura C.1: Pantalla para la configuración del Phantom

`/usr/sbin/PHANToMTest`

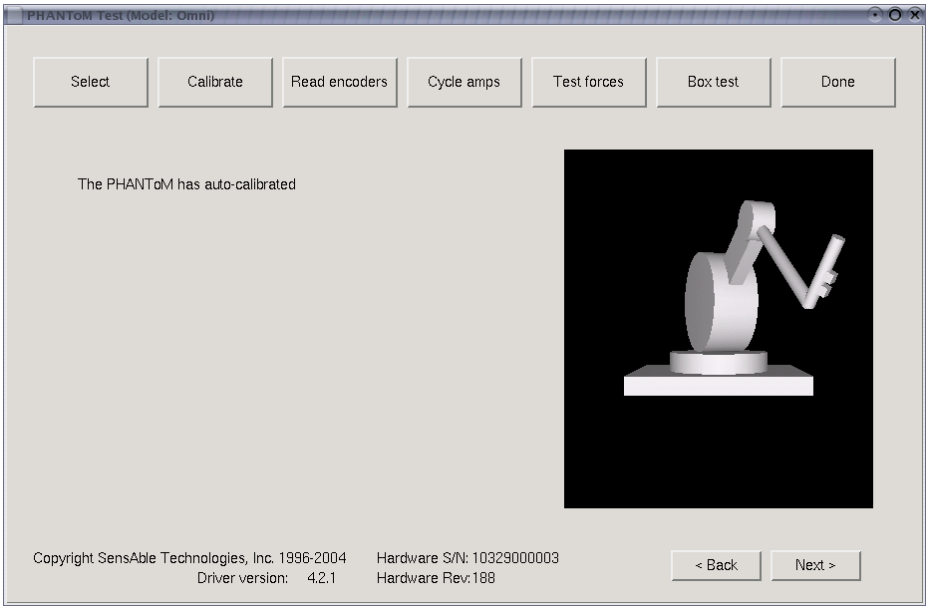


Figura C.2: Pantalla de prueba para la configuración del Phantom

Bibliografía

[1] Jorge Eduardo Ramírez. Modelos deformables para caracterizar macromoléculas biológicas. Tesis de maestría, Centro de Investigación y de Estudios Avanzados del IPN, Departamento de Computación, Junio 2004.

[2] D. Terzopoulos and K. Fleischer. Deformable models. *The Visual Computer*, 4(6):306–331, 1988.

[3] D. Terzopoulos and K. Fleischer. Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. In *SIGGRAPH'88:Proceedings of the 15th anual conference computer graphics and interactive techniques*, volume 22, pages 269–278, 1988.

[4] David Braff and Andrew Witkin. Larg steps in cloth simulation. *Computer Graphics*, 32(Annual Conference Series):43–54, 1998.

[5] Liliya Kharevych and Rafi (Mohammad) Khan. 3D physics engine for elastic and deformable bodies. Universidad de Maryland College Park, December 2002.

[6] Rungun Ramanathan and Dimitris Metaxas. Dynamic deformable models for enhanced haptic rendering in virtual enviroments. pages 31–36. IEEE Computer Society, 2000.

[7] Demetri Terzopoulos and Dimitri Metaxas. Dynamic 3d models with local and global deformations: Deformable superquadrics. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(7):703–714, 1991.

[8] D. Terzopoulos and K. Waters. A physical model of facial tissue and muscle articulation. pages 77–82, 1990.

[9] A. Witkin M. Kass and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988.

[10] C. Xu and J.L. Prince. Snakes, shapes, and gradient vector flow. *IEEE Transactions on Image Processing*, 7(3):359–369, 1998.

[11] E. Bardinet Cohen and N. Ayache. Surface reconstruction using active contour models. Technical Report RR-1824, INRIA, 1995.

[12] J. Montagnat, H. Delingette, and N. Ayache. A review of deformable surfaces: topology, geometry and deformation. *Image and Vision Computing*, 19(14):1023–1040, December 2001.

[13] H. Delingette. Simplex meshes: a general representation for 3D shape reconstruction. In *Proc. of international conference on computer vision and pattern recognition (CVPR'94)*, number RR-3111, pages 856–857, June 1994.

[14] N. Scapel J. Montagnat, H. Delingette and N. Ayache. Representation, shape, topology and evolution of deformable surfaces aplication to 3D medical image segmentation. Technical Report RR-3954, INRIA, 2000.

[15] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. *Computer Graphics (Proc. SIGGRAPH'87)*, 21(4):205–214, July 1987.

[16] Modelos deformables y métodos numéricos, http://www.cc.gatech.edu/dvfx/readings.html.

[17] H. Delingette. General object reconstruction bases on simplex meshes. *International Journal of Computer Vision*, 32(2):111–142, September 1999.

[18] J.E. Ramírez Flores and L.G. de la Fraga. Basic three-dimensional objects constructed with simplex meshes. In *First International Conference on Electrical and Electronics Engineering*, pages 166–171, Acapulco, Mexico, Sep. 8-10 2004.

[19] Dennis G. Zill. *Ecuaciones Diferenciales con aplicaciones de modelado*. International Thomson Editores, 6th edition, 1997.

[20] John H. Mathews. *Numerical Methods for computer science, engineering, and mathematics*. Prentice-Hall, Inc, 1987.

[21] Jr. David E. Penney C.H. Edwards. *Elementary Differential Equations with Boundary Value Problems*. Prentice-Hall, Inc, 3rd edition, 1992.

[22] J. D. Lambert. *Computational Methods in Ordinaty Differential Equations*. Jhon Wiley & Sons, 1972.

[23] The mesa 3d graphics library. http://www.mesa3d.org.

[24] Troll tech - qt overview. http://www.trolltech.com/products/qt/.

[25] 3d touch sdk openhaptics toolkit, programmer's guide version 2.10. SensAble technologies.

[26] K. Salisbury, D. Brock, T. Massie, N. Swarup, and C. Zilles. Haptic rendering: programming touch interaction with virtual objects. In *SI3D'95: proceedings of the 1995 symposium on interactive 3D graphics*, pages 123–130. ACM Press, 1995.

[27] C. A. Mendoza and C. Laugier. Realistic haptic rendering for highly deformable virtual objects. In *VR'01: proceedings of the virtual reality 2001 conference*, page 264. IEEE Computer Society, 2001.

[28] Thomas H. Massie and J. K. Salisbury. The phantom haptic interface: a device for probing virtual objects. In *Proceedings os the ASME winter anual meeting, symposium of haptic interfaces for virtual envioroment and teleoperator systems*, November 1994.

[29] Christer Ericson. *Real-Time Collision Detection.* Elsevier, 2005.

[30] M. Anitescu, A. Miller, and G. D. Hart. Constraint stabilization for time-stepping aproaches for riged multibody dynamics with joins, contact and friction. In *In proceedings of the ASME International Design EngineeringTechnical Conferences*, September 2003.

[31] Dimitri Metaxas and Demetri Terzopoulos. Dynamic deformation of solid primitives with constraints. In *SIGGRAPH'92: proceedings of 19th annual conference on computer graphics and interactive techniques*, pages 309–312. ACM Press, 1992.

[32] Michael B. Cline and Dinesh K. Pai. Post-estabilization for rigid body simulation with contact and constraint. In *ICRA*, pages 3744–3751, 2003.