Modelado Gráfico de un Cuerpo Neumático con OpenGL a Base de Ecuaciones Diferenciales

Jorge Antonio García Galicia

21 de abril de 2008

Índice general

Índice de figuras

Índice de cuadros

Introducción

Uno de los retos más constantes en las ciencias de la computación, y en especial del área de graficación por computadora, ha sido el de alcanzar cada vez mayor realismo. Sin embargo, este objetivo comúnmente se opone al de hacer modelos simples que requieran de poco poder de cómputo.

Es cierto que cada vez tenemos computadoras más poderosas, pero como seres humanos que somos siempre hemos querido muchas cosas más de las que tenemos. Hoy en día el que una simulación se pueda ejecutar rápidamente en una computadora personal es casi tan importante como el que esa simulación se vea real.

Modelos que se sabe son bastante efectivos, como las ecuaciones de Navier-Stokes, también son computacionalmente muy costosos. Por lo tanto, no pueden ser ocupados en simulaciones que requieran de alcanzar tiempo real.

Los videojuegos son un claro ejemplo de simulaciones que requieren de una adecuada combinación de realismo y rapidez de ejecución. Dentro de esta área, se dice que se alcanza tiempo real cuando el programa responde más rápido de lo que el usuario puede darle instrucciones. El objetivo principal de este trabajo es encontrar un modelo que conjunte el mayor realismo posible, sin olvidar alcanzar *tiempo real*.

Hace tiempo, cuando buscaba algún tema interesante para realizar mi trabajo de titulación, no tenía una idea clara de lo que quería hacer. Por un lado sabía qué me gustaba de las materias de mi licenciatura, también sabía que quería hacer una tesis que de alguna manera tuviera que ver con el área de gráficas por computadora.

Afortunadamente para mi, eso fue lo único que necesité para que la maestra Carmen Villar se interesara en asesorar mi trabajo de titulación, aun cuando no tenía un tema bien definido. Después de buscar en diversas fuentes de información llegamos al sitio web de Maciej Matyka, sobre simulación basada en física, en donde pude consultar el artículo [11]. Desde el primer momento este artículo llamó mi atención, el modelo propuesto gozaba de las tres características más deseables en un modelo de simulación: era sencillo de entender, sencillo de implementar y era visualmente muy agradable.

Teníamos entonces una idea de donde comenzar, sólo nos hacia falta un problema que valiera la pena modelar. Discutiendo en el laboratorio de Linux de la FES Acatlán, quizás un poco inspirados en la película American Beauty, tuvimos la idea del sistema que dio origen al presente trabajo. Al final creo que el resultado poco tiene que ver con la inmortal escena, pero estoy convencido de que al menos para mí ha sido un camino igual de inspirador.

Otra meta que quería alcanzar era utilizar Software Libre en todo este trabajo. Sin duda, el utilizar una biblioteca como OpenGL me favoreció para poder alcanzar esta meta, al tratarse de una especificación libre y multiplataforma. La mayor parte del trabajo fue realizada bajo un sistema GNU/Linux y el texto fue escrito en LaTeX. Considero que este objetvo fue cumplido.

A lo largo del desarrollo, tanto del programa como del trabajo escrito, me encontré con muchísimos obstáculos. Algunos se debieron a mi desconocimiento de ciertas áreas de la física, y otros simplemente a mi falta de habilidades de programación. Sin embargo, estoy convencido que en ambos aspectos este trabajo me ayudó a superarme.

La pregunta más importante que ahora trato de responderme es: ¿a quién le sirve este trabajo? Indudablemente me sirvió a mí el escribirlo, y quiero pensar que también le servirá a cualquier persona que lo lea. ¿Quién puede ser este lector potencial? Es un trabajo de animación por computadora, por lo que le debe servir a cualquier persona que desee hacer una animación de cuerpos flexibles. También creo que es un trabajo que modela adecuadamente las leyes físicas del fenómeno, así que se puede beneficiar de él cualquier persona que desee aprender o enseñar cómo funcionan los cuerpos neumáticos. Por último, le sirve a cualquier estudiante que busque alguna aplicación de las matemáticas y la física en la animación.

La manera como decidí organizar este trabajo es la siguiente: En la primera parte se revisó el marco teórico. Éste trata principalmente de aquellas partes de la literatura, que aunque ya están escritas, quizás no sean tan obvias para estudiantes de la licenciatura de Matemáticas Aplicadas y Computación. Este capítulo fue por mucho el más divertido de escribir.

En el siguiente capítulo, ya adentrado en la situación a modelar, traté en mayor detalle las estrategias de implementación del modelo. Es un capítulo en el que intenté de plasmar, sobre todo, las cosas que me fueron más difíciles en el desarrollo de la investigación y cómo fue que encontré soluciones a ellas.

El tercer capítulo trata de cómo se hizo el programa en C++ para esta simulación. Aunque el código que aquí escribo puede ser mejorado de muchas maneras, es una solución que funciona. Y si bien no me preocupé en optimizar el código, si se ganó un poco de claridad. Además, considero dado el propósito de este trabajo, que la programación no es *tan* importante.

El capítulo final fue por mucho el más gratificante de escribir. En él presento las pruebas y juegos

a los que sometí mi programa una vez que terminé el desarrollo. Al escribir este capítulo aprendí dos cosas: que siempre hay más cosas que hacer después de que uno piensa que ha terminado algo y que los resultados no siempre son los que uno espera. De hecho, el desarrollo de esta investigación, de la que fui parte desde su nacimiento, me hizo llevarme unas cuantas sorpresas en su edad madura.

Por último, sólo me queda esperar que los lectores de este trabajo encuentren al leerlo, un poco de la diversión que me dejó a mí escribirlo.

Capítulo 1

Sistemas físicos y modelación matemática

Uno de los principales fenómenos naturales que más se ha estudiado a lo largo de la historia es el del movimiento. La rama de la física que se encarga de estudiar este fenómeno es la *mecánica*. Hay muchos enfoques para estudiar al movimiento. La mecánica puede dividirse a su vez en: *estática*, *cinemática* y en *dinámica*. La estática estudia la ausencia del movimiento. La cinética se encarga de estudiar la manera de describir el movimiento. La dinámica estudia las causas del movimiento, es decir estudia las fuerzas que producen el movimiento.

El movimiento que estamos tratando de modelar puede pensarse como una combinación de movimientos más simples, para los cuales ya existen modelos matemáticos. Por lo que en este capítulo presentaré estos movimiento por separado, con el fin de entender las causas que los producen. Después en el capítulo siguiente uniré estos sistemas simples en uno más complejo, que es el que a fin de cuentas nos interesa modelar.

El primer movimiento que presentaré es el de un cuerpo que se deja caer en el vacío sin imprimirle ninguna fuerza, y que por lo tanto, sólo actúa sobre él la fuerza de gravedad. Este movimiento ha sido muy estudiado y es conocido como *caída libre*.

El siguiente movimiento, es también muy conocido, se conoce como *ley de Hooke*. Éste es el movimiento de algunas masas unidas por un resorte y se modela con una ecuación diferencial.

Por último, analizaré el fenómeno de la presión del gas, con el modelo más simple de todos, que es conocido como la ecuación de estado de la *ley de los gases ideales*. En el cual, se supone un gas hipotético en el que sus moléculas no interactúan entre sí.

En este capítulo, trataré estos sistemas por separado para poder comprenderlos cualitativamente. Así será más clara la manera en que los combinaré en el siguiente capítulo para formar un modelo. Y

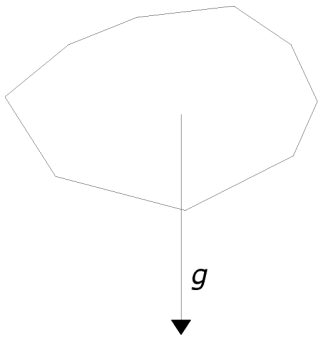

Figura 1.1: Una piedra en caída libre, la única fuerza que actúa sobre ella es la de la gravedad.

finalmente, con este modelo trabajar la situación que deseo analizar.

1.1. Un cuerpo en caída libre

Imagínese una piedra que descansa cerca de la orilla del quicio de una ventana. La piedra es empujada horizontalmente hasta que resbala de la orilla y se cáe, (ver figura 1.1). Éste es un ejemplo de caída libre.

Mas formalmente, la caída libre consiste en modelar un cuerpo sólido, con masa uniformemente distribuida y despreciando la fricción del aire. Es decir, la única fuerza que actúa sobre este cuerpo es la *fuerza de gravedad*.

En la antiguedad, Aristóteles pensaba que este movimiento era directamente proporcional a la masa. Él afirmaba que dada una misma altura, un cuerpo con el doble de masa caería el doble de rápido. Esta idea fue aceptada durante varios siglos. No fue hasta la llegada del método científico que se empezó a cambiar este punto de vista.

Se cuenta una anécdota con ciertos toques de leyenda. Se dice que Galileo dejo caer en un mismo instante una pelota de madera de un kilo de peso y una bala de cañón de diez kilos de peso desde la torre inclinada de Pisa, y la gente pudo apreciar como estos cayeron casi al mismo tiempo. Con el simple hecho de un experimento la teoría aristotélica quedaba refutada. Al parecer antes de los pensadores y filósofos del renacimiento, nadie se había preguntado si los conocimientos que prevalecían eran verdaderos; es decir, nadie había hecho un experimento para probar si algo era de verdad como se decía. La simple autoridad de Aristóteles bastaba para que se diese por hecho.

Fue después de Galileo en 1685 que Newton explicó con más detalle este movimiento, con su *ley*

de la gravitación universal. Newton fundamentó la mecánica con sus leyes del movimiento y la ley de la gravitación universal. En las primeras, se dice que un objeto permanece en su estado de reposo o de movimiento rectilíneo uniforme a menos que una fuerza lo perturbe. Esta fuerza perturbadora le produce al objeto un cambio de velocidad, es decir, una aceleración, en mecánica de Newton las fuerzas producen *aceleración*. La ley de la gravitación universal dice que todos los cuerpos se atraen. También dice que la fuerza de atracción entre dos cuerpos es directamente proporcional al producto de sus masas e inversamente proporcional al cuadrado de sus distancias.

A partir de esta idea, si la fuerza cambia con el tiempo, también lo hará la aceleración. Sin embargo por suerte para nosotros, el movimiento de caída libre imprime siempre *aproximadamente* la misma aceleración, es decir, la fuerza de gravedad actua sobre todas las cosas con *casi* la misma fuerza. Una consecuencia de la ley de la gravitación universal y de que la masa de la Tierra sea muy grande en comparación de las cosas que estamos acostumbrados ver caer.

Gracias a que la fuerza que atrae las cosas a la tierra es casi constante, el problema de caída libre tendrá una solución matemática analítica y hasta cierto punto fácil de obtener. Hay muchas maneras de derivar estas leyes, más adelante cuando se unan todos los modelos se entenderá por qué elegí explicarla de esta manera y no de una manera más natural.

Empiezo planteando las condiciones del modelo: suponemos que se trata de un objeto con masa m, que se encuentra al iniciar su movimiento a una altura y_0, que en el momento de iniciar su movimiento tiene una velocidad inicial v_0 y que durante todo su movimiento sólo va a actuar sobre él una fuerza constante F (figura 1.2), que le imprime una aceleración constante a y que esta fuerza es precisamente la fuerza de gravedad que ejerce la tierra sobre él, es decir: $a = g$.

Además por seguir la convención usual, se moverá en un sistema coordenado derecho, es decir las direcciones positivas serán derecha en el eje x, y arriba en el eje y, por ende los movimientos hacia abajo y a la izquierda serán negativos, respecto a cada uno de los ejes. Además sabemos que la gravedad sólo afecta el movimiento vertical del cuerpo, es decir sólo actúa sobre el eje y. Entonces la segunda ley de Newton nos dice que:

$$F = ma \tag{1.1}$$

Pero nosotros sabemos que la aceleración a, es la derivada de la velocidad v, respecto al tiempo t. Y que la velocidad es a su vez la derivada de la posición y respecto al tiempo t, por lo que podemos escribir la ley de Newton como:

$$\frac{d^2y}{dt^2} = \frac{1}{m}F(t) \tag{1.2}$$

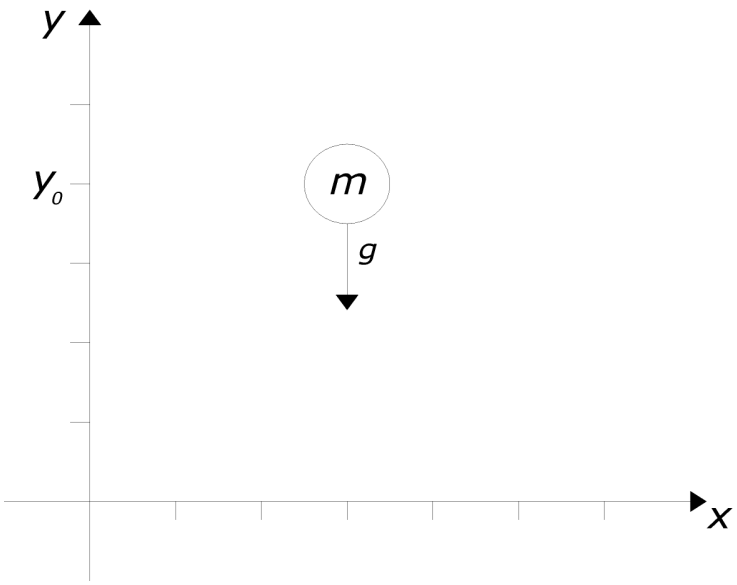

Figura 1.2: El modelo del cuerpo en caida libre, con su sistema de coordenadas.

Además se sabe que tenemos dos condiciones iniciales, para nuestro problema:

$$y(0) = y_0$$
$$v(0) = v_0$$

Tenemos entonces una ecuación diferencial ordinaria, de segundo orden, con dos condiciones iniciales, por lo tanto podemos encontrar una solución única.

Esta ecuación es en particular, fácil de resolver debido a que la fuerza F es constante.

Primero sabemos que $F(t) = -mg$, el signo es porque la gravedad actúa hacia abajo. Y sustituyendo en la ecuación:

$$\frac{d^2y}{dt^2} = \left(\frac{1}{m}\right)(-mg) \tag{1.3}$$

Aquí vemos por qué Galileo encontró que la caída libre *no* depende de la masa del objeto.

Si integramos la ecuación respecto a t.

$$\frac{dy}{dt} = -gt + c_2$$

Donde c_2 es una constante de integración, que puede encontrarse de la primera condición inicial.

$$\frac{dy}{dt} = -gt + v_0$$

Podemos volver a integrar la ecuación respecto a t y volver a obtener la nueva constante c_1 de las condiciones iniciales.

$$\begin{aligned} y(t) &= -\frac{gt^2}{2} + v_0 t + c_1 \\ &= -\frac{gt^2}{2} + v_0 t + y_0 \end{aligned} \tag{1.4}$$

Que es la función que estábamos buscando, esta función nos dice cual es el valor de la coordenada y del objeto cuando ha transcurrido un tiempo t. Esto es precisamente lo que necesitamos saber para hacer una animación: una función que nos diga cómo se mueve algo conforme pasa el tiempo. También cabe señalar que en nuestro caso de la caída libre $v_0 = 0$, recordemos que el cuerpo se dejó caer, por lo que nuestra función se simplifica. Y se puede simplificar aun más, si escogemos un sistema de referencia tal que el movimiento del objeto empiece en el origen de dicho sistema es decir $y_0 = 0$. La forma más simple de nuestra función es entonces:

$$y(t) = -\frac{gt^2}{2} \tag{1.5}$$

Quizás este sistema puede parecer un poco simple, y en efecto lo es, en el sentido que pudo resolverse sin mayor problema y de una manera analítica, esto se debió a que la aceleración, y por ende la fuerza se supuso constante. El valor de este modelo radica más bien en la existencia de una solución analítica.

1.2. El sistema masa-resorte

Para explicar esta situación se partirá desde lo más simple a lo más complejo. El modelo más sencillo se conoce como un *oscilador armónico simple*. Después se tratará de agregar una segunda

fuerza que servirá como un amortiguador, este nuevo modelo se conoce como *oscilador armónico amortiguado*. Por último este modelo se generalizará a un sistema en tres dimensiones y con vectores y finalmente derivaremos la fórmula que ocuparemos durante el resto del trabajo.

La mecánica de vibraciones se debe al físico inglés Robert Hooke, la vida de este científico es muy interesante por diversos motivos.[1] Es uno de los científicos experimentales más importantes de la historia, y sus estudios abarcan campos muy diversos como la biología, la medicina, la cronometría, la física planetaria, la microscopía, la náutica y la arquitectura. Estuvo en disputas con Newton por el descubrimiento de la ley de la gravitación universal.

Aquí estamos interesados en uno de sus descubrimientos en elasticidad y que es conocido como *Ley de Hooke*. Que nos dice:

> La fuerza restauradora ejercida por un resorte es linealmente proporcional al desplazamiento del resorte desde su posición de reposo y está dirigida hacia esa misma posición[2].

1.2.1. Oscilador armónico simple

Imagínese que se tiene una masa sujeta por un resorte sobre la que no actúa ninguna fuerza adicional. Por el momento vamos a suponer que no hay fricción del medio, presumiblemente el aire y que no está sujeta a la fuerza de gravedad. Por estas condiciones ideales la masa no tendrá pérdida de energía, es decir se moverá por siempre.

De nuevo la misma idea que antes; ocuparemos la ecuación de Newton y lo único que cambiará en nuestro modelo será la expresión para calcular la fuerza, en vez de la fuerza de gravedad, se modelará la fuerza del resorte.

Imaginemos el resorte; los resortes se estiran y se comprimen cuando se les aplica una fuerza. Siempre que pasa esto, el resorte en respuesta ejerce una fuerza restauradora, esta fuerza siempre lucha por devolver al resorte a su punto original de reposo es decir a donde no está ni comprimido ni estirado. Es entonces que la ley de Hooke para los resortes nos da una manera de calcular esta fuerza restauradora F_s.

Empecemos a plantear las condiciones del modelo, supongamos que es una masa unida por un resorte a un objeto inamovible, una pared por ejemplo y que todo el sistema descansa en el piso,

[1] Una biografía de Hooke se puede encontrar en la wikipedia en español; una biografía aun más completa, en MacTutor

[2] En el libro [3] en el apartado dedicado a Modelación por medio de Sistemas, se enuncia de esta manera, ignoro si Hooke pensó en resortes al enunciar su ley, o se le descubrió esta aplicación en un momento posterior.

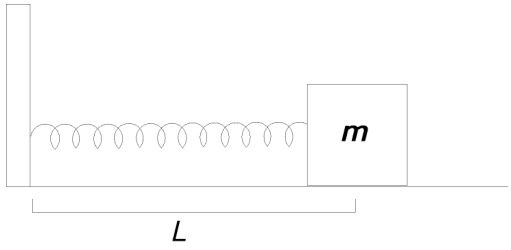

Figura 1.3: Una masa que se mueve solamente afectada por la fuerza de un resorte.

para que no lo afecte la gravedad, y este piso no le da ningún tipo de fricción. En estas condiciones el único movimiento permitido para la masa será el horizontal y a la derecha, de nuevo por convención diremos que es a la dirección positiva del eje x. La figura 1.3 ilustra la situación aquí descrita.

Se trata de modelar el movimiento de la masa a lo largo del tiempo t. La masa del cuerpo de nuevo la denotaremos como m y el origen del sistema será el punto donde el resorte se une a la pared. Supongamos también que cuando el resorte está en reposo tiene un largo L. La expresión que nos dice cual es la fuerza del resorte, que denotaremos como ya se dijo F_s, para un estiramiento razonablemente pequeño es la *ley de Hooke para los resortes* y se escribe como sigue:

$$F_s = -k_s\,(x - L) \tag{1.6}$$

Esta expresión básicamente nos dice que la fuerza es proporcional a la distancia de la masa a su punto de equilibrio donde el resorte esta en reposo. La nomenclatura F_s se debe a que la mayoría de la bibliografía consultada está en inglés y traté de ser consistente con ella, ésta es la fuerza del resorte o *spring force*.

Se observa en esta expresión que cuando el resorte está estirado, es decir, su largo x es mayor a L la fuerza del resorte se vuelve negativa y por ende jala a la masa a su punto de reposo. Como se ve en la figura 1.4.

En el caso que el resorte esté comprimido, es decir, $x < L$, el resorte empuja el cuerpo para que alcance la posición de reposo y de nuevo lo hace con una fuerza proporcional a la distancia que esté alejado del cuerpo de este punto.

También se observa que cuando el largo del resorte es $x = L$ el resorte está en reposo y por lo tanto la fuerza se hace cero, éste es el caso que se ilustro en la figura 1.3.

Esta expresión para la fuerza se vuelve a aplicar a la ecuación de la ley de Newton, quedando la siguiente ecuación diferencial.

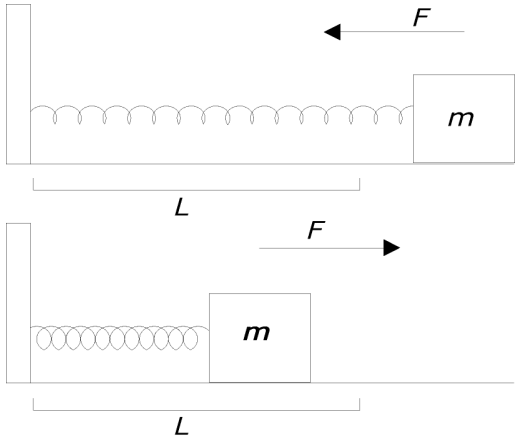

Figura 1.4: En la parte superior el caso donde $x > L$ y en la parte inferior el caso en donde $x < L$.

$$\begin{aligned} m\frac{d^2x}{dt^2} &= F(t) \\ &= F_s \\ &= -k_s\,(x - L) \\ \frac{d^2x}{dt^2} + \frac{k_s}{m}x &= \frac{k_s L}{m} \end{aligned} \tag{1.7}$$

Esta ecuación de nuevo tendrá solución única si se especifican un par de condiciones iniciales de la forma $x(0) = x_0$ y $v(0) = v_0$. La teoría de ecuaciones diferenciales nos dice que esta solución se puede encontrar, mediante la suma de una solución particular x_p y una solución general de la ecuación diferencial homogénea asociada al problema x_h. Es decir $x(t) = x_p + x_h$.

Por simple inspección se puede ver que $x(t) = L$ es una solución que nos puede servir como x_p y afortunadamente la ecuación homogénea es una *ecuación diferencial lineal ordinaria de coeficientes constantes* por lo que es relativamente simple de resolver. Además recuérdese que las constantes k_s, m y L, son todas positivas por lo que una solución es:

$$x_h = c_1 \cos\sqrt{\frac{k_s}{m}}t + c_2 \operatorname{sen}\sqrt{\frac{k_s}{m}}t \tag{1.8}$$

Con estas dos soluciones tenemos:

$$x(t) \;=\; x_p + x_h$$

$$= \quad c_1 \cos\left(\sqrt{\frac{k_s}{m}}t\right) + c_2 \operatorname{sen}\left(\sqrt{\frac{k_s}{m}}t\right) + L \tag{1.9}$$

Y podemos determinar las dos constantes de integración c_1 y c_2 de las condiciones iniciales.

Para este problema en particular la solución que estamos buscando es:

$$x(t) = (x_0 - L)\cos\left(\sqrt{\frac{k_s}{m}}t\right) + \left(v_0\sqrt{\frac{m}{k_s}}\right)\operatorname{sen}\left(\sqrt{\frac{k_s}{m}}t\right) + L \tag{1.10}$$

Como puede apreciarse en la solución el cuerpo nunca deja de moverse, su movimiento está expresado por una función periódica. Esto es debido a que en un medio ideal como éste el sistema no disipa energía por lo tanto, nunca cesa su movimiento, y el resorte se comprime y se estira de la misma manera por siempre.

1.2.2. Oscilador armónico amortiguado

Como es lógico las condiciones ideales antes tratadas son casi imposibles de alcanzar en la realidad, por lo que es necesario agregar a nuestro sistema una manera de perder energía.

Una manera simple de hacerlo es agregándole una resistencia por parte del medio, por ejemplo el aire, o el agua si todo el sistema se haya sumergido bajo ésta.

Esta nueva fuerza en el sistema, a menudo se supone directamente proporcional a la velocidad del cuerpo con respecto al medio en el que se encuentra y se modela con la siguiente ecuación.

$$F_d = -k_d\frac{dx}{dt} \tag{1.11}$$

El signo es negativo debido que la resistencia del medio se opone a la velocidad. Es decir trata de detener la masa poco a poco. De nuevo una aclaración sobre la nomenclatura: se dice que está es la fuerza de amortiguación y se refiere a la palabra amortiguador en inglés: *damping*. Y entonces el nuevo modelo toma la forma:

$$m\frac{d^2x}{dt^2} \quad = \quad F_s + F_d$$

$$\begin{aligned} &= -k_s(x-L) - k_d\frac{dx}{dt} \\ m\frac{d^2x}{dt^2} + k_d\frac{dx}{dt} + k_s x &= k_s L \end{aligned} \tag{1.12}$$

Una vez más sujeto a las mismas condiciones iniciales $x(0) = x_0$ y $v(0) = v_0$.

Esta ecuación se resuelve de una manera similar a la de la sección pasada, donde una solución particular x_p vuelve a ser $x_p = L$. Para encontrar una solución general se requiere resolver la ecuación homogénea asociada:

$$m\frac{d^2x}{dt^2} + k_d\frac{dx}{dt} + k_s x = 0 \tag{1.13}$$

De nuevo la teoría nos dice que esta ecuación tiene una solución de la forma:

$$x(t) = c_1 e^{n_1 x} + c_2 e^{n_2 x} \tag{1.14}$$

Donde las constantes n_1 y n_2 son soluciones de la ecuacion algebraica en n:

$$mn^2 + k_d n + k_s = 0 \tag{1.15}$$

Dado que ésta es una ecuación de segundo grado, puede resolverse con la fórmula general. Además se puede obtener información de la solución analizando cómo es k_d^2 con respecto a $4mk_s$ (lo que vendría a ser b^2 con respecto a $4ac$). Esto nos lleva a tres casos: que n tenga dos valores reales diferentes, que n tenga un solo valor real y que n tenga un par de valores complejos conjugados.

Estos casos son analizados a detalle en [15] y por lo tanto lo único que diré es que los dos primeros casos nos llevan a resultados donde el amortiguador es tan fuerte que no deja a la masa oscilar ni siquiera una vez. Es decir en donde la fuerza del medio es muy superior a la fuerza del resorte.

El único caso que a nosotros nos interesa es precisamente aquel donde el amortiguador es débil, y deja oscilar a la masa unas cuantas veces antes de detenerla, este caso se da precisamente cuando se trata de dos valores complejos conjugados de n.

Este caso es cuando $k_d^2 < 4mk_s$ es decir que el discriminante de la ecuación es negativo, por lo que la solución es de la forma:

$$n_1 = \alpha + \beta i$$

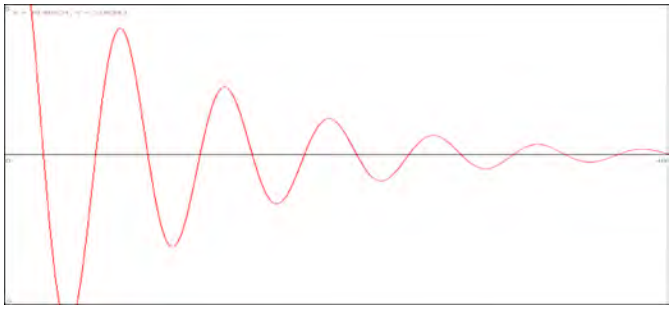

Figura 1.5: Aquí se puede ver la gráfica de $e^{\frac{-1}{10}x}\,(3\cos x + 3\,\mathrm{sen}\,x)$.

$$n_2 = \alpha - \beta i$$

En donde:

$$\begin{aligned} \alpha &= \frac{k_d}{2m} \\ \beta &= \frac{\sqrt{4mk_s - k_d^2}}{2m} \end{aligned}$$

Y por tanto la ecuación diferencial tiene una solución de la forma:

$$x(t) = e^{-\alpha t}\,(c_1 \cos \beta t + c_2\,\mathrm{sen}\,\beta t) + L \tag{1.16}$$

De nuevo por medio de las condiciones iniciales se pueden obtener los valores de las constantes c_1 y c_2, quedando de esta manera:

$$\begin{aligned} c_1 &= x_0 - L \\ c_2 &= \frac{v_0 + (x_0 - L)\,\alpha}{\beta} \end{aligned}$$

De la solucion 1.16 es claro que cuando el tiempo $t \to \infty$, el factor $e^{-\alpha t}$ dominará y la masa se detendrá poco a poco.

Un comportamiento común se puede apreciar en la figura 1.5, en donde se han elegido los parámetros del modelo con el fin de ilustrar el comportamiento de la solución.

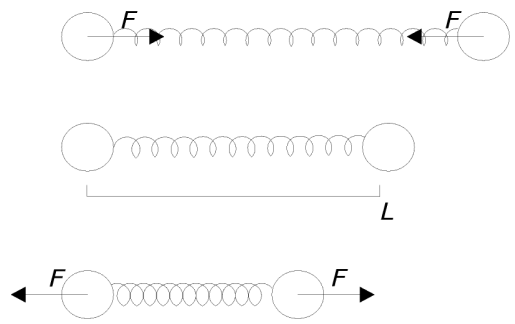

Figura 1.6: Cuando las masas están libres, el resorte aplica sobre cada una de ellas la misma fuerza, pero en sentido contrario.

1.2.3. Masa-resorte en más de una dimensión

Tomaremos el modelo de la sección anterior; pero lo vamos a generalizar de manera que se pueda modelar este mismo fenómeno unidimensional solo que situado en un espacio tridimensional. Por medios geométricos trataré de aplicar las mismas fórmulas de la sección pasada. Por simplicidad en los diagramas, voy a explicar el caso de dos dimensiones, sin embargo, como la explicación se hará con vectores, no se debe de tener problema en generalizar para tres dimensiones.

Primero vamos a considerar que hay dos puntos en el espacio que están unidos por un resorte, este resorte es la única fuerza que actúa sobre ellos. El resorte no tiene masa y ambos cuerpos tienen una masa m. Los puntos no están sujetos, es decir se mueven libremente por el espacio, y la única fuerza que actúa sobre ellos es el resorte.

Hay que notar que a diferencia de la sección pasada, donde el resorte estaba empotrado en la pared, aquí el resorte esta unido a dos puntos sueltos, y por lo tanto aplica la misma fuerza sobre cada uno de los puntos. Esto se puede apreciar mejor en la figura 1.6.

Llamaré estos puntos a y b respectivamente, y vemos que una posible manera de localizarlos en el espacio es el vector de posición de cada uno de ellos, denotados por $\vec{p_a}$ y $\vec{p_b}$.

La fuerza del resorte

Vamos a calcular la fuerza del resorte sobre el punto a; la denotaré como F_a, y por lo anterior ya se sabe que la fuerza que actúa sobre b es su negativo, es decir $F_b = -F_a$

Sabemos que una manera de encontrar el vector que une los puntos a y b es restando los vectores de posición de cada uno de ellos; a este nuevo vector lo llamaremos $\vec{r}$ así que:

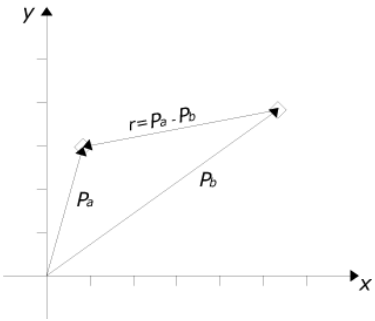

Figura 1.7: Podemos localizar puntos en el espacio por medio de sus vectores de posición.

$$\vec{r} = \vec{p_a} - \vec{p_b} \tag{1.17}$$

Como se ve en la figura 1.7 este vector empieza en la punta de $\vec{p_b}$ y termina en la punta de $\vec{p_a}$. Entonces la distancia que hay de a a b es precisamente la norma de este vector $\vec{r}$, dicho de otra manera $\vec{r} = \vec{p_a} - \vec{p_b} = \vec{ba}$ y $\overline{ab} = |\vec{r}|$.

Ahora que sabemos la distancia de a a b, la podemos restar con el tamaño de resorte que es el escalar L. Si a este resultado lo multiplicamos por la constante del resorte k_s sabremos la magnitud de la fuerza del resorte, es decir calculando $k_s\left(|\vec{r}| - L\right)$.

Éste es un escalar, que representa la magnitud de la fuerza, pero como ésta última es un vector, hace falta que le demos dirección, para hacerlo multiplico el escalar por el vector $\vec{r}$ normalizado.

Analizaremos la acción de la fuerza. Si el factor $k_s\left(|\vec{r}| - L\right)$ es negativo, entonces $|\vec{r}| < L$, significa que el resorte está comprimido, luego entonces la fuerza debe empujar el punto a lejos del punto b, como ya dijimos que $\vec{r}$ representa un vector que va de b a a, entonces la fuerza debe tener la direccion de este vector. Veamos el caso contrario, si $|\vec{r}| > L$ el resorte está estirado, $k_s\left(|\vec{r}| - L\right)$ es positivo y la fuerza debe jalar el punto a en dirección de b, es decir en la dirección contraria que el vector $\vec{r}$. De estas dos observaciones nos damos cuenta de que el signo de $k_s\left(|\vec{r}| - L\right)$ debe ser negativo, es decir:

$$F_a = -k_s\left(|\vec{r}| - L\right)\frac{\vec{r}}{|\vec{r}|} \tag{1.18}$$

Como se dijo, sólo se ha calculado la fuerza que actúa sobre el punto a, sin embargo por la simetría del sistema y que ambos puntos están sueltos en el espacio, la fuerza sobre el punto b, es equivalente pero en sentido contrario, es decir:

$$F_b = -F_a$$

La fuerza del amortiguador

En la sección pasada sólo modelamos la fuerza de un resorte que en esencia presenta la misma carencia del primer resorte que analizamos: le falta una manera de perder energía. Ya habíamos analizado una manera eficaz de hacer que el oscilador disipe energía, sin embargo esto se debía al medio. Aquí la idea es un poco diferente, vamos a pensar que esta resistencia no está presente en el medio, sino en el mismo resorte en sí; es decir nuestro resorte además de ser tal será un amortiguador, y la resistencia actuará por lo tanto en la misma línea de acción del resorte.

Ya dijimos anteriormente que esta resistencia la vamos a suponer proporcional a la velocidad. Y con la misma idea que antes vamos a agregarla al modelo anterior. Ya sabemos que tenemos una manera de localizar a los puntos a y b en el espacio, su vector de posición, sin embargo aún no tenemos una manera de saber la velocidad de los puntos en el espacio, por ello vamos a crear dos vectores $\vec{v_a}$ y $\vec{v_b}$, que contienen la velocidad a la que se mueven estos puntos respectivamente.

Procedamos a calcular la fuerza que actúa sobre el punto a. Esta fuerzas son: la fuerza anterior debida al resorte que los une, y una nueva fuerza ahora debida al amortiguador.

$$F_a = F_s + F_d \tag{1.19}$$

Donde F_s es precisamente la expresión 1.18. Así que sólo nos falta calcular F_d. Esta fuerza, como ya se dijo, es proporcional a la velocidad a la que se mueve un punto respecto al otro $(\vec{v_a} - \vec{v_b})$ con una constante k_d, y además actúa en el sentido contrario al de la velocidad, es decir F_d es proporcional a $-k_d\,(\vec{v_a} - \vec{v_b})$.

Ahora hay que notar también que la posición de un punto no necesariamente tiene que ver con su velocidad, es decir un punto en una misma posición se podría mover bien a una velocidad v u a otra muy diferente, sin tener que cambiar su vector de posición.

Como ya dijimos que para nosotros el amortiguador está en el resorte, es necesario entonces trasladar esta fuerza a donde está el resorte, es decir a $\vec{r} = \vec{p_a} - \vec{p_b}$. Para hacer esto vamos a ocupar la conocida forma de la proyección de un vector sobre otro.

Sabemos que un vector $\vec{A}$ se puede proyectar sobre un vector $\vec{B}$ y se denota como $\text{Proy}_{\vec{B}}\vec{A}$ podemos obtener su magnitud con la siguiente fórmula:

$$|\text{Proy}_{\vec{B}}\vec{A}| = \frac{\vec{A}\cdot\vec{B}}{|\vec{B}|} \tag{1.20}$$

Ahora entonces calculemos la magnitud de la proyeccion del vector $(\vec{v_a} - \vec{v_b})$ sobre el vector donde está el resorte, es decir sobre el vector $\vec{r} = \vec{p_a} - \vec{p_b}$. Quedando de esta manera:

$$|\text{Proy}_{\vec{r}}\vec{v}| = \left[\frac{(\vec{v_a} - \vec{v_b}) \cdot (\vec{p_a} - \vec{p_b})}{|\vec{p_a} - \vec{p_b}|}\right] \tag{1.21}$$

Sin embargo, esta cantidad es un escalar así que necesitamos multiplicarla por un vector unitario para darle dirección. Como esta fuerza es debida al *resorte-amortiguador* y está en su misma línea de acción, la multiplicamos por $\vec{r}$ normalizado. Quedando la expresión final para F_d:

$$F_d = -k_d \left[\frac{(\vec{v_a} - \vec{v_b}) \cdot (\vec{p_a} - \vec{p_b})}{|\vec{p_a} - \vec{p_b}|}\right] \left[\frac{\vec{p_a} - \vec{p_b}}{|\vec{p_a} - \vec{p_b}|}\right] \tag{1.22}$$

Como ya expliqué antes ésta es sólo la fuerza del amortiguador así que la expresión de fuerza para este *resorte-amortiguador*, remplazando $\vec{r} = \vec{p_a} - \vec{p_b}$ y $\vec{v} = \vec{v_a} - \vec{v_b}$ para no hacer muy compleja la notación es la siguiente:

$$\begin{aligned} F_a &= F_s + F_d \\ F_a &= -k_s\left(|\vec{r}| - L\right)\frac{\vec{r}}{|\vec{r}|} - k_d\left[\frac{\vec{v}\cdot\vec{r}}{|\vec{r}|}\right]\left[\frac{\vec{r}}{|\vec{r}|}\right] \\ F_a &= -\left\{k_s\left(|\vec{r}| - L\right) + k_d\left[\frac{\vec{v}\cdot\vec{r}}{|\vec{r}|}\right]\right\}\left[\frac{\vec{r}}{|\vec{r}|}\right] \\ F_b &= -F_a \end{aligned} \tag{1.23}$$

1.3. El modelo del gas ideal

Aquí se va a tratar de agregar al modelo, una fuerza de presión. La idea de esta fuerza es tomada del artículo de Matika [11], y se basa en implementar un modelo simple de gas. Este modelo cuenta con la ventaja de la sencillez, tanto en su implementación como en la rapidez de su ejecución, lo que lo hace un modelo ideal para una animación en tiempo real.

Vamos a empezar por imaginar un cuerpo cerrado en tres dimensiones, por ejemplo una esfera o un cubo, luego vamos a imaginar que la envolvente de ese cuerpo, es decir la superficie que lo cierra, es una superficie elástica, que se puede estirar y deformar, y por último supongamos que dentro de

ese cuerpo, en vez de haber un vacío, hay un gas; ésta es la idea principal del modelo que trataremos de ahora en adelante, y es el motivo de estudio de este trabajo.

En las dos secciones anteriores hemos tratado los modelos que nos pueden servir como la envolvente de este cuerpo (podria ser una envolvente formada por una membrana de puntos unidos por resortes), pero hablaré mas de esto en el siguiente capitulo. Así que sólo falta tratar el modelo de un gas por separado, para terminar nuestro tratamiento de modelos simples. El modelo que vamos a estudiar es el del *gas ideal*.

1.3.1. La termodinámica

Habíamos estado hablando de leyes de mecánica, sin embrago este modelo pertenece a una rama diferente de la física, la *termodinámica*. La termodinámica es la rama de la física que estudia los efectos de los cambios de temperatura, calor y presión, en un sistema, por medio de métodos estadísticos.[3]

El estudio de la termodinámica empieza con el descubrimiento de las maneras de medir el calor, es decir, con la invención de los primeros termómetros, fue así como se notó que existen sistemas que presentan cambios cualitativos, cuando cambian su temperatura.

Uno de estos sistemas es el gas, y por lo tanto existen algunos modelos para estudiar las propiedades de los gases, El más simple de estos modelos es el modelo del gas ideal, que se define por medio de la *ley de los gases ideales*, que no es otra cosa que la ecuación de estado de un gas ideal.

1.3.2. Un modelo simple

Al principio, para poder estudiar un fenómeno del cual no se conocen muchas cosas, es común relajar muchas de sus propiedades e idealizar muchas situaciones, es decir construir modelos simples, en este caso nos abocamos al primer modelo que se construyó para un gas, y por lo mismo está sujeto a muchas suposiciones que en la realidad nunca son alcanzadas por ningún gas, de ahí que lo llamaran *gas ideal*.

En física y en química, una ecuación de estado es una ecuación que describe el estado de agregación de la materia en función de ciertos parámetros, es decir determina la relación matemática entre dos o más funciones de estado asociadas con la materia.

[3] En esta definición de *termodinámica* comúnmente se aplican modelos más complejos que el usado aquí, que sí requieren de métodos estadísticos, pero de hecho nuestro modelo es en esencia determinista.

En nuestro caso queremos describir el estado de un gas en función de propiedades macroscópicas del mismo. Estas propiedades son: la temperatura, el volumen, la presión y el número de moléculas.

Por medio de resultados experimentales se determinó la primera ecuación de estado, una explicación de cómo se hicieron estos experimentos y se llegó a ella se puede encontrar en [14], pero a nosotros nos bastará con conocerla:

$$\begin{aligned} PV &= nRT \\ P &= \frac{1}{V}nRT \end{aligned} \tag{1.24}$$

En donde P es la presión del gas, V es el volumen del gas, n es el número de moles, R es una constante llamada constante del gas ideal y T es la temperatura. La constante R es la multiplicación de la constante de Boltzman y el número de Avogadro [4]. Las unidades de esta constante son Joule (J) sobre mol (mol) Kelvin (K).

$$R = N_A k = 8{,}3145 \ \frac{J}{mol \cdot K}$$

Esta ecuación de estado es un aproximación bastante buena cuando se trata de gases a bajas densidades, y la experimentación nos muestra que en estas condiciones *todos* los gases tienden a comportarse como este gas ideal.

1.3.3. La presión de un gas

La presión no es en sí una fuerza sino es una fuerza aplicada en una unidad de área, y por eso se mide en una unidad llamada *Pascals* (Pa) y no en Newtons (N) que es la unidad en que se mide la fuerza. Entonces, si P es la presion, F es una fuerza y A el área donde se está aplicando esta fuerza:

$$P = \frac{F}{A}$$

De tal manera que si nosotros queremos una *fuerza de presión*, para poder acumularla en la ecuación de Newton; como lo hicimos con las fuerzas anteriores necesitamos multiplicarla por el área de aplicación. Dicho de otra manera:

[4] Esta constante universal aparece en varias ecuaciones importantes de la termodinámica, es equivalente a la constante de Boltzman pero expresada en unidades de energia.

$$F_p = \vec{P} \cdot A$$

Donde F_p es la fuerza de presión que estamos buscando, A es el área de aplicación y $\vec{P}$ es el vector que representa la presión. Ya se dijo que la magnitud de la presión puede calcularse con la ecuación de estado, pero no se ha dicho qué dirección lleva. La presión actúa sobre la superficie de la cara del cuerpo, y es siempre de normal a esta cara, de manera que la presión es:

$$\vec{P} = P \cdot \vec{n}$$

En donde P es el escalar calculado con la ecuación de estado y $\vec{n}$ es un vector unitario normal a la cara sobre la que se está calculando la presión.

Para nuestro modelo se debe tomar en cuenta que sólo nos interesa calcular la fuerza de presión, de esta manera se procede de la siguiente manera.

$$P = \frac{1}{V}\, nRT$$

De esta manera $\vec{P}$ es:

$$\vec{P} = \frac{1}{V}\, nRT\, [\vec{n}]$$

Y este vector $\vec{P}$ se usará a su vez, para obtener el vector fuerza, es decir:

$$F_p = \frac{1}{V}\, nRT\, [\vec{n}]\, A$$

Reacomodando esta ecuación, se tiene:

$$F_p = \left[\frac{1}{V} A\, nRT\right] \vec{n}$$

Donde para *nuestro* modelo el escalar nRT es un parámetro a determinarse experimentalmente, está formado por la temperatura que consideramos constante, el número molar del gas, que debe depender del gas y por la constante del gas ideal. Así que los tres son agrupados en una sola constante, que nosotros por ser consistentes con nuestra nomenclatura llamaremos k_g

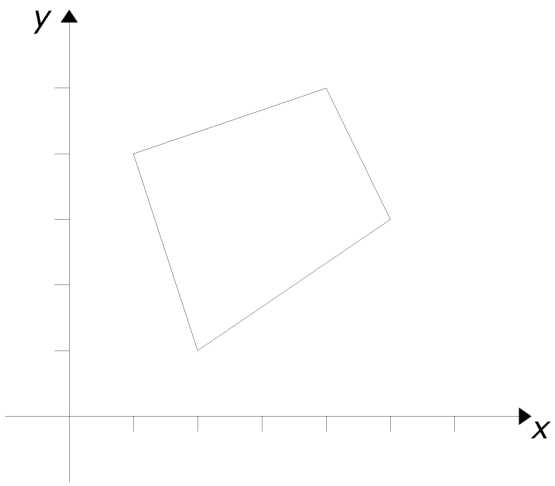

Figura 1.8: La figura cerrada para la cual se harán los cálculos de presión.

De manera que:

$$F_p = \left[\frac{1}{V} A\ k_g\right] \vec{n} \tag{1.25}$$

Donde F_p es la fuerza que deseamos acumular en cada partícula, V es el volumen total de nuestro cuerpo cerrado, A es el área de la cara sobre la que estamos calculando la presión, $\vec{n}$ es un vector unitario normal a esta cara y k_g es una constante positiva a ser determinada experimentalmente. Es la ecuación que nos servirá en nuestro modelo.

1.3.4. Un ejemplo ilustrativo

Supongamos que se tiene una caja cerrada cuya *tapa* tiene el largo y el ancho mostrado en la figura 1.8 la caja tiene una altura igual a 1, es decir, $h = 1$. Esta caja tiene las tapas fijas a lo alto, por lo que la fuerza de un gas solo la puede deformar en sus lados. Calculemos ahora la fuerza debida a la presión de un gas que esté dentro de esta caja.

Vamos a calcular la fuerza correspondiente a la presión, por medio de la ecuación 1.25. Como la caja es de altura unitaria, el volumen V que es igual al área de su tapa por la altura equivale simplemente al área de su tapa. Por la misma razon las áreas de las caras laterales de la caja equivalen a la longitud de el lado de la tapa en el que se encuentran.

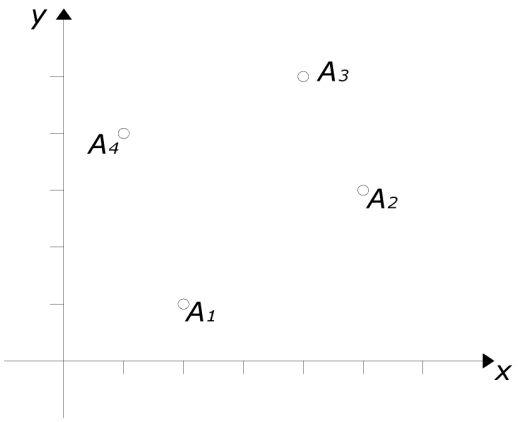

Figura 1.9: Figura con los puntos que representan las esquinas identificados y localizados.

Para calcular la fuerza de presión en este caso se necesita saber entonces: la longitud de cada uno de los lados del cuadrilátero, un vector normal a cada una de sus lados y su área. Por suerte para nosotros lo único que necesitamos para conocer todos estos datos, es ubicar la posición de cada una de sus esquinas.

Identificaremos cada una de las esquinas A_i por medio de sus coordenadas x_i y y_i, dicho de otra manera $A_i = (x_i, y_i)$. Esta situación queda determinada por la figura 1.9.

De donde podemos conocer, gracias al sistema de referencia, las coordenadas de cada uno de los puntos en donde se ubican las esquinas:

$$\begin{aligned} A_1 &= (x_1, y_1) = (2, 1) \\ A_2 &= (x_2, y_2) = (5, 3) \\ A_3 &= (x_3, y_3) = (4, 5) \\ A_4 &= (x_4, y_4) = (1, 4) \end{aligned}$$

Una vez conocidos estos datos podemos calcular la longitud de cada lado simplemente obteniendo la distancia entre dos puntos.

$$\begin{aligned} \overline{A_1A_2} &= \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} = \sqrt{13} \\ \overline{A_2A_3} &= \sqrt{(x_2 - x_3)^2 + (y_2 - y_3)^2} = \sqrt{5} \end{aligned}$$

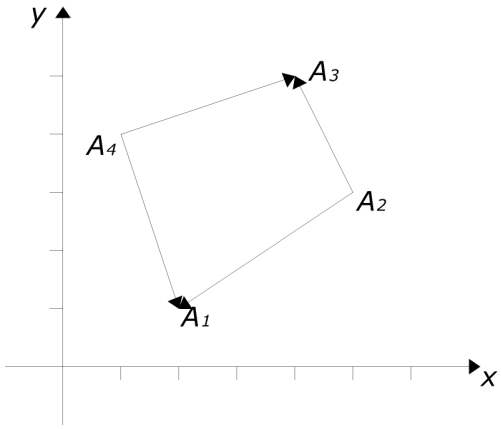

Figura 1.10: Aquí se pueden ver los vectores que representan cada uno de los lados, y que son calculados por la resta de los vectores de posición de los puntos.

$$\begin{aligned}\overline{A_3A_4} &= \sqrt{(x_3-x_4)^2+(y_3-y_4)^2} = \sqrt{10}\\ \overline{A_4A_1} &= \sqrt{(x_4-x_1)^2+(y_4-y_1)^2} = \sqrt{10}\end{aligned}$$

También podemos calcular el área de la tapa con un sencillo truco, podemos trazar la diagonal que divide al cuadrilátero tal que pase por A_1 y A_3, con lo que para calcular el área total del cuadrilátero basta con calcular el área de los dos triángulos $\triangle A_1A_3A_4$ y $\triangle A_1A_2A_3$ y luego sumar ambas áreas (ver la figura 1.10).

Para calcular el área del triángulo $\triangle A_1A_3A_4$ tomamos a A_4 como origen y calculamos el vector que va de A_4 a A_1, luego calculamos el vector que va de A_4 a A_3, y por último calculamos el producto cruz, tomamos su norma y lo dividimos entre dos.[5] De manera análoga para el siguiente triángulo:

El cálculo de los vectores es el siguiente:

$$\begin{aligned}\overrightarrow{A_4A_1} &= \vec{A_1}-\vec{A_4} = (1,-3)\\ \overrightarrow{A_4A_3} &= \vec{A_3}-\vec{A_4} = (3,1)\\ \overrightarrow{A_2A_1} &= \vec{A_1}-\vec{A_2} = (-3,-2)\\ \overrightarrow{A_2A_3} &= \vec{A_3}-\vec{A_2} = (-1,2)\end{aligned}$$

Y por lo explicado anteriormente el área total de la tapa A_T es:

[5]Este es un teorema muy conocido de la geometría vectorial y se puede ver en cualquier libro que incluya una introducción a los vectores, me imagino que no necesito citar alguna fuente en especial

$$A_T = A_{\triangle A_1A_3A_4} + A_{\triangle A_1A_2A_3}$$
$$A = \frac{1}{2}|\overrightarrow{A_4A_1} \times \overrightarrow{A_4A_3}| + \frac{1}{2}|\overrightarrow{A_2A_1} \times \overrightarrow{A_2A_3}|$$
$$A = \frac{1}{2}|10| + \frac{1}{2}|-8|$$
$$A = 9$$

Ahora sólo nos falta calcular la fuerza F_p y acumularla en cada una de las caras laterales de la caja. Para esto y sólo por simplicidad supongamos que escogemos $k_d = 10$ entonces el cálculo de la fuerza debe hacerse por separado para cada una de las caras laterales (que en nuestro caso equivalen a los lados de la tapa):

$$\begin{aligned}
F_{\overline{A_1A_2}} &= \tfrac{1}{V}\overline{A_1A_2}k_g &&= \frac{1}{9}\sqrt{13}\,(10)\\
F_{\overline{A_2A_3}} &= \tfrac{1}{V}\overline{A_2A_3}k_g &&= \frac{1}{9}\sqrt{5}\,(10)\\
F_{\overline{A_3A_4}} &= \tfrac{1}{V}\overline{A_3A_4}k_g &&= \frac{1}{9}\sqrt{10}\,(10)\\
F_{\overline{A_4A_1}} &= \tfrac{1}{V}\overline{A_4A_1}k_g &&= \frac{1}{9}\sqrt{10}\,(10)
\end{aligned}$$

Con lo que tenemos la magnitud de las fuerzas que actúan en cada cara, así que sólo falta calcular un vector unitario normal a la cara correspondiente para poder acumular su fuerza.

Afortunadamente, para calcular un vector normal en dos dimensiones, tenemos un método sencillo, el producto punto de dos vectores normales entre sí debe de dar cero, así que si queremos un vector normal al vector $\vec{v} = (x, y)$ simplemente construimos uno tal que su producto punto con el primero siempre sea igual a cero, por ejemplo $\vec{v^n} = (-y, x)$. Con esto sabemos que el vector $\vec{v^n}$ es siempre normal al vector $\vec{v}$, de hecho en dos dimensiones sólo este vector y sus múltiplos son normales a $\vec{v}$.

Sabiendo todo lo anterior, ya se puede acumular la fuerza correspondiente sobre cada cara. Por ejemplo, sobre la primera cara que está definida por el segmento que une a A_1 y A_2. Como en el paso anterior ya se había calculado el vector $\overrightarrow{A_2A_1} = (-3, -2)$ que en sí representa a la cara, podemos calcular su vector normal con la fórmula anterior. Para esta cara el vector $\vec{n} = (2, -3)$ es el vector normal, para hacerlo unitario lo dividimos entre su norma $\frac{\vec{n}}{|\vec{n}|} = \frac{1}{\sqrt{13}}(2, -3)$. Y finalmente el vector fuerza que estamos buscando para esta cara en particular es:

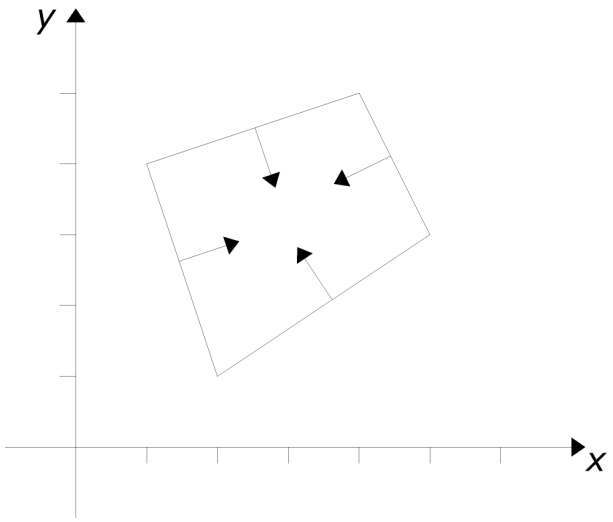

Figura 1.11: Aqui se pueden ver los vectores incorrectos para dirigir la presión pues apuntan al centro de la figura.

$$\vec{F}_{\overline{A_1A_2}} = \frac{1}{V}\overline{A_1A_2}k_g\vec{n} = \frac{1}{9}\sqrt{13}\,(10)\,\frac{1}{\sqrt{13}}(2,-3) = \frac{10}{9}(2,-3)$$

Una consideración más que debe tenerse, al calcular el vector normal, pudiera darse el caso que éste dirija la fuerza al contrario de como se desea, es decir que la dirija, al centro del cuerpo en vez de hacia afuera. Como se ilustra en la figura 1.11. En cuyo caso hay que multiplicar este vector normal por -1. En general la mejor práctica es graficar el cuerpo para así poder calcular el vector normal correcto.

Por ejemplo, al calcular un vector normal para la segunda cara podríamos vernos tentados a repetir el cálculo, ocupando el vector que acabamos de calcular $\overrightarrow{A_4A_1}$ pero al ocupar la fórmula, nos daría $n = (3, 1)$ que en efecto es un vector normal a $\overrightarrow{A_4A_1}$ pero que apunta hacia el centro del cuerpo, y por lo tanto sería incorrecto ocuparlo para dirigir la fuerza de presión. El vector correcto para este caso es $-n = (-3, -1)$. Así que se debe tener cuidado al calcular el vector para dirigir la fuerza de presión.

El resto de las operaciones correspondientes al ejemplo se pueden ver en la tabla 1.1. En la primera columna de la tabla se muestra la cara a la que se estaba calculando la fuerza, después se muestra el valor escalar de la presión, luego el vector asociado a la cara, que ya habíamos obtenido, después el vector normal resultado del procedimiento antes explicado. Hay que tomar en cuenta que el vector podría ser que apunte hacia afuera o hacia adentro, y en la tabla ya fue corregido (multiplicándolo

Cuadro 1.1: Cálculo de la Fuerza de Presión, para el ejemplo

Cara	Fuerza de Presión	Vector de la cara	Vector Normal	Vector Fuerza
$\overline{A_1A_2}$	$\frac{1}{9}\cdot 10\cdot\sqrt{13}$	$\overrightarrow{A_2A_1}=(-3,-2)$	$(\frac{2}{\sqrt{13}},\frac{-3}{\sqrt{13}})$	$(\frac{20}{9},\frac{-30}{9})$
$\overline{A_2A_3}$	$\frac{1}{9}\cdot 10\cdot\sqrt{5}$	$\overrightarrow{A_2A_3}=(-1,2)$	$(\frac{2}{\sqrt{5}},\frac{1}{\sqrt{5}})$	$(\frac{20}{9},\frac{10}{9})$
$\overline{A_3A_4}$	$\frac{1}{9}\cdot 10\cdot\sqrt{10}$	$\overrightarrow{A_4A_3}=(3,1)$	$(\frac{-1}{\sqrt{10}},\frac{3}{\sqrt{10}})$	$(\frac{-10}{9},\frac{30}{9})$
$\overline{A_1A_4}$	$\frac{1}{9}\cdot 10\cdot\sqrt{10}$	$\overrightarrow{A_4A_1}=(1,-3)$	$(\frac{-3}{\sqrt{10}},\frac{-1}{\sqrt{10}})$	$(\frac{-30}{7},\frac{-10}{9})$

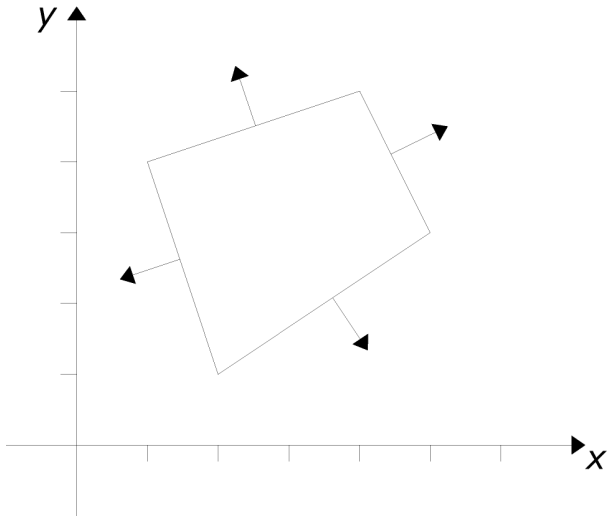

Figura 1.12: Así es como debe de dirigirse la fuerza de presión. Obsérvese que cada esquina debe acomular la fuerza de las dos caras laterales que está uniendo.

por el escalar -1). Y en la última columna aparecen los vectores de fuerza que se estaban buscando, y son los que se requieren para poder acumularse en la ecuación de Newton.

La fuerza de presión es mucho más compleja de calcular que las fuerzas debidas a la gravedad y a los *resortes-amortiguadores*. La principal dificultad es que debe tratar con la geometría del cuerpo para calcularla, además es necesario conocer el volumen total del cuerpo, y por cada cara que se acumule se debe de conocer: el área de la misma y un vector normal, por lo que a diferencia de los casos anteriores no hay una manera general de enunciar las fórmulas sino que debe analizarse el caso particular de la geometría del cuerpo que se esté tratando.

Capítulo 2

Construcción del algoritmo de simulación

En este capítulo nos encargaremos de presentar el algoritmo propuesto por Matyka en [11], que es explicado en más detalle en [10]. Este modelo no es más que la unión de los modelos del capítulo anterior. Además definiremos el experimento que deseamos realizar para probar el funcionamieto del modelo.

Más adelante en este mismo capítulo, explicaremos cómo se pueden enfrentar ciertos detalles sobre el algoritmo de Matika.

Primero veremos cómo se puede calcular el volumen de un cuerpo en tres dimensiones, explicaremos como se puede tener una idea general, y por qué para ciertas geometrías del cuerpo, se prestan mejor ciertos algoritmos.

Luego nos enfrentaremos al problema de cómo integrar la ecuación de Newton, con métodos numéricos, analizando dos alternativas, y se presentan las que se consideran las mejores maneras de implementar el integrador en código.

Por último analizaremos el problema más difícil de la implementación: las colisiones. Esto se analiza en dos partes, la manera como se detectan y la respuesta de las mismas.

2.1. Diseño del experimento

Necesitamos una simulación gráfica (animación) y el principal objetivo será hacer que se *vea* real.

Se quiere modelar un cuerpo neumático. Para hacerlo partimos de un cuerpo flexible y le ponemos

un fluido dentro, en este caso el gas, además queremos una manera de probar cómo es la interacción de éste con otros objetos de la escena.

Con base en esos requerimientos, diseñe la siguiente situación: se modela una caja cerrada, en la cual su cara superior tiene la característica de ser formada como un cuerpo flexible, las otras cinco caras son rígidas. Dentro de esta caja se pone el fluido (gas), de manera que ahora tenemos un cuerpo neumático. Para probar su interacción con otro cuerpo se le deja caer sobre su cara flexible un cuerpo sólido (en este caso una esfera) y se observa el resultado.

Además para poder hacer varios tipos de pruebas se desea que la animación tenga cierta interactividad, es decir que el usuario pueda, en tiempo de ejecución, cambiar algunas opciones tanto de visualización como también algunos parámetros físicos.

Esta simulación es el objetivo principal del trabajo, y el siguiente capítulo está dedicado a cómo se construyó, mientras que en este capítulo se hace énfasis en los detalles teóricos necesarios para escribir el programa.

2.2. El algoritmo de simulación de un cuerpo neumático

Lo primero que debemos de hacer es exponer las condiciones de nuestra simulación de cuerpo flexible. La situación física que deseamos modelar es la de un cuerpo flexible, hueco que contiene un gas. Por ejemplo imagínese un globo, un colchón de aire, la llanta de un automóvil o una burbuja. El comportamiento en el que estamos interesados es en un comportamiento cualitativamente parecido al del fenómeno, es decir que al momento de graficarlo y hacer una animación debe verse *parecido* al sistema real.

El valor de poder graficar de manera realista un modelo como éste, radica en el hecho de que hay muchas cosas que se comportan de manera parecida, y al momento de animar una escena donde hay varios de estos objetos, un modelo físicamente más realista sería muy costoso en tiempo de ejecución.

La forma de atacar este problema será hacerlo con un modelo simple. Imagínese un cuerpo formado por varios puntos unidos entre ellos por resortes.[1] Ahora imagínese que a un cuerpo cerrado formado por esta *tela*, le ponemos dentro una fuente de aire, que le ayuda a mantener su forma más o menos constante durante las deformaciones permitidas por los resortes que lo forman; ésa es la idea principal de nuestro modelo.

En la figura 2.1 podemos ver un pequeño esquema del modelo, en este caso es un hexágono en

[1] Este modelo es muy socorrido cuando se trata de modelar ropa o tela, se pueden ver ejemplos en el último capítulo de [4], que modela una bandera o en [17] y [13], ambos modelan ropa sobre una persona

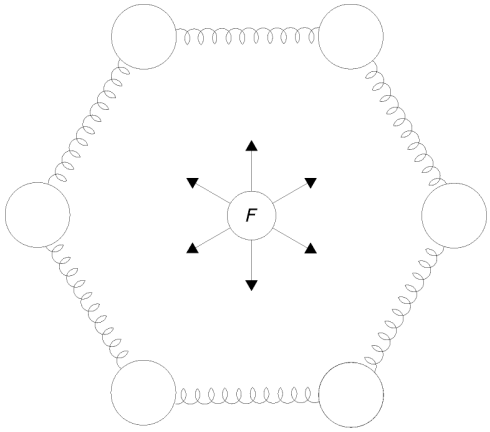

Figura 2.1: Un ejemplo de un modelo de masa resorte, con presión. Se trata de un hexágono cerrado, y F representa la fuerza debida a la presión.

dos dimensiones. Desde luego, mientras más compleja sea la forma de nuestro modelo, mas difícil será trabajar con él, pero se verá más realista

2.2.1. Esbozo general del algoritmo

Una vez conocida la idea, nos empezaremos a preguntar en cómo llevarlo a cabo, es decir cómo podemos implementar este modelo en forma de un algoritmo que seamos capaces de codificar en algún lenguaje de programación.

El diagrama de flujo del algoritmo es el que se muestra en al figura 2.2, en donde se puede apreciar que sólo se tienen bloques de proceso; cada uno de estos procesos puede ser llevado a cabo de muchas maneras, pero tendrá que ser en este orden, en esta sección veremos con un poco más de detalle cada proceso.

Inicializar variables

Lo primero es dar valor a las variables que lo necesiten, me refiero con esto a variables globales, que afectan cómo funciona el programa por el resto de su ejecución. Dentro de estas variables están los parámetros externos del modelo, es decir esos que no pueden ser calculados dentro del programa y que es necesario que se proporcionen por el usuario. Lo parámetros necesarios son m para cada partícula, g, k_d y k_s para los resortes, L para cada resorte en particular, y k_p para la acumulación de la fuerza debida a la presión.

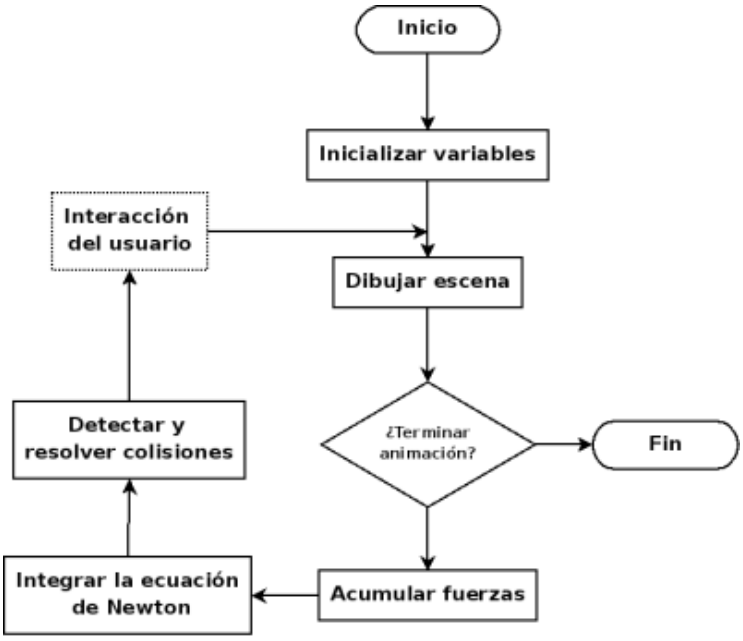

Figura 2.2: Este diagrama de flujo muestra, los detalles generales a seguir en una animación de un cuerpo flexible. Para cada paso del algoritmo se pueden tener diferentes estrategias, pero en general este esquema no debe cambiar.

Otra cosa que debe hacerse en este paso del algoritmo es poner las condiciones iniciales del modelo, por ejemplo, la posición de cada partícula y la velocidad al momento de iniciar la animación. Y de igual manera con los cuerpos no flexibles que deseemos que interactúen con nuestro modelo, se debe de conocer su posición, si se mueven o no, y cómo van a interactuar con el modelo.

Por último las opciones globales, que afecten a la animación también se deben de obtener aquí, por ejemplo el tamaño del paso Δt para mover el modelo, las opciones de los gráficos (si se hará *render*, o se tratará sólo con el *wireframe* [2]) y qué tipo de integrador se ocupará en el modelo. Alguien me podría decir que estos también son parámetros. Y en efecto lo son, quise sin embargo hacer la distinción y llamarlos variables globales, por el hecho de que no se deben al modelo en si, sino más bien a la implementación que cada quien haga del mismo, es decir se deben al programa.

Dibujar escena

Esta parte del algoritmo debe ser la encargada de dibujar en la pantalla toda la escena. Para hacer esto debe contar con la información de donde se encuentran cada punto y donde se encuentra cada cosa que queramos dibujar.

Aunque no hay mucho que decir aquí, esta parte es delicada, depende de nuestro conocimiento de las herramientas que vamos a usar para hacer el programa. Con la ayuda de una biblioteca gráfica, en este caso *OpenGL*, y los datos necesarios guardados de una manera ordenada, como en una estructura de datos, debemos de ser capaces de implementar esta función.

El detalle de la implementación depende del modelo en cuestión, veré a detalle la que yo use. Esto puede ser útil para que alguien que deseé implementar una simulación, se de algunas ideas, pero como ya dije, esta parte depende del fenómeno a modelar, por lo que en cada implementación se debe de hacer un análisis.

Acumular fuerzas

Las fuerzas que actúan sobre cada partícula del modelo, se pueden identificar como *externas*, es decir que se deben al medio, o *internas* que se deben al cuerpo en sí. Ejemplos de fuerzas externas son la gravedad, la viscosidad del medio, la resistencia del aire, por mencionar algunas. Ejemplos de las fuerzas internas son por otra parte: los *resortes amortiguadores* y la debida a la presión del gas.

[2] Cuando sólo se dibujan los bordes de las figuras, como si éstas estuvieran hechas de una malla de alambre se le llama wireframe, cuando dibujamos el área de las figuras y les damos color y efectos de iluminación se dice que se les dio *render*.

Cualquiera que sea el caso, en este paso debemos de calcular todas las fuerzas, que intervengan en nuestro modelo, y debemos de acumulársela a cada uno de las partículas que lo conforman, es decir, cada partícula debe tener asociada la suma de todas las fuerzas que actúan sobre ella. Recordemos que la fuerza en general es un vector, por lo que la acumulación de la que hablo es una suma vectorial de cada una de las fuerzas.

Los detalles de cómo acomular las fuerzas específicas que ocuparé en la simulación se darán en la siguiente sección y con más a detalle en el resto del capitulo.

Integrar la ecuación de Newton

Las fuerzas acomuladas en el paso anterior, junto con la posición y velocidad actuales de la partícula, nos proporcionan (como dice la segunda ley de Newton. Ecuación 1.2) el siguiente estado, nos dan la información suficiente para saber la nueva posición y la nueva velocidad de la partícula en cuestión. Conociendo esto para todas las partículas tendremos determinado el estado completo del modelo en el tiempo siguiente.

Como ésta es una ecuación diferencial, debemos de resolverla, o mejor dicho integrarla, para conocer el siguiente estado del sistema, sin embargo como nuestra acomulación de fuerzas fue hecha a partir de muchas fuerzas, todas ellas de naturaleza distinta, no podremos integrar esta ecuación de manera analítica. Por esta razón debemos de recurrir a un método numérico que nos permita aproximar la solución.

Por suerte hay muchísimos métodos numéricos que se ajustan a nuestras necesidades, aquí he decidido presentar a dos de ellos solamente, en la sección dedicada a integrar la ecuación de Newton explicaré a detalle cómo funcionan. De manera general, se puede decir que todos ellos toman como información la posición y velocidad actual de la partícula, el vector fuerza que actúa sobre ella y el instante de tiempo que pasa entre el estado actual y el nuevo estado que queremos calcular (este instante es Δt). Y después de hacer cálculos con ellos nos devuelven un nuevo estado, en forma de la nueva posición y la nueva velocidad de la partícula.

Detectar y resolver colisiones

En este paso debemos analizar qué pasa cuando nuestro modelo choca o interactúa con otros objetos de la escena. Estos otros objetos pueden ser cuerpos rígidos como el piso o flexibles como otro cuerpo de la misma característica del nuestro.

El segundo paso es la respuesta a esta colisión. Este problema es también delicado, pero en general,

se va a encargar de ver la partícula que chocó, moverla como respuesta al choque y modificar su estado, es decir su posición y su velocidad. En una sección posterior de este capítulo hablaré con mas detalle y también daré algunas ideas generales.

Interacción del usuario

Este paso es opcional, y depende de si deseamos que la simulación tenga una forma de ejecutarse independiente del usuario o si se desea que éste pueda cambiar la simulación en tiempo de ejecución, por ejemplo moviendo alguna partícula, agregando un nuevo cuerpo o modificando algún parámetro de la simulación.

Para enfrentar esto se tiene que recurrir totalmente a la creatividad del programador que vaya a hacer la implementación. Una buena idea es por ejemplo utilizar alguna biblioteca que exista precisamente para este fin (*OpenGL*, nos proporciona a *glut*).

2.3. El sistema de fuerzas

Es momento de hacernos unas preguntas: ¿Qué es lo que compone nuestro cuerpo? ¿Qué necesitamos saber para implementar este modelo? La primera pregunta es sencilla: nuestro cuerpo está formado por puntos, que están unidos por resortes, que a su vez se unen para formar las caras. Resumiendo, nuestro cuerpo es un conjunto de caras, que a su vez son un conjunto de resortes, que a su vez son un conjunto de puntos: ¡Nuestro cuerpo está formado por puntos! La respuesta de la segunda interrogante está ligada a la primera, necesitamos saber todo lo que sea necesario para poder acumular fuerza a los puntos, es decir que para cada punto debemos de poder ocupar las ecuaciones 1.3, 1.23 y 1.25.

Ciertamente cada una de estas fuerzas es de naturaleza distinta, la fuerza de gravedad es externa y se puede calcular para cada punto por separado, la de los resortes depende de la posición de cada punto y de sus vecinos a los cuales está conectado por un *resorte-amortiguador* y por último la de presión depende de la cara en la que esté el punto y del volumen total del cuerpo. Podemos resumirlo en el cuadro 2.1.

Para calcular la fuerza de gravedad es necesario que se conozcan dos parámetros, la masa de cada punto m, y la constante gravitación g. La gravedad debe de acumularse en cada punto.

La fuerza del *resorte amortiguador* es una fuerza interna. Para calcularla necesitamos recibir dos parámetros: k_s y k_d. Luego necesitamos calcular la posición y la velocidad de cada punto, y saber

Cuadro 2.1: Fuerzas a calcular para cada punto y que se necesita saber para hacer los calculos

Fuerza	Símbolo	Parámetros	Datos necesarios
Fuerza de gravedad	F_g	g, m	Ninguno
Fuerza del resorte	$F_s + F_d$	k_s, k_d	Puntos unidos por el resorte Velocidad y posición de cada punto que une el resorte
Fuerza debida a la Presión	F_p	k_p	Puntos que forman la cara Volumen total del cuerpo Area de la cara

por cada resorte qué puntos está uniendo. Cuando tengamos que acumular esta fuerza recorreremos todos los resortes y por cada resorte calcularemos una fuerza, luego la acomularemos en cada unos de los dos puntos a los que este resorte esté conectado.

Por último, para la fuerza debida a la presión, necesitamos un parámetro externo: k_p. Y para poder calcular esta fuerza debemos conocer antes de empezar el volumen total del cuerpo V, luego debemos de saber qué puntos pertenecen a cada una de las caras del cuerpo. Para acumular esta fuerza debemos recorrer cada una de las caras que forman el cuerpo, en cada caso calculamos el área de la cara y acumulamos la fuerza F_p que le corresponda en cada uno de los puntos que la forman.

2.4. Área, volumen y vectores normales

Los parámetros necesarios para calcular la fuerza de presión son el volumen total del cuerpo, el área de cada una de las caras, y un vector normal a dicha cara; como ya se dijo, esto depende enteramente de la geometría del cuerpo que se quiera modelar. Sin embargo, esto no quiere decir que no se puedan dar algunas técnicas generales que en alguna medida puedan ser adaptadas a alguna geometría particular. El propósito de esta sección es precisamente el de explicar esas técnicas generales.

2.4.1. Cálculo de áreas

Para calcular el área de un cuerpo geométrico generalmente se calcula el área de cada una de sus caras y después se suman. Empezando por el caso más simple, supongamos que tenemos un triángulo formado por tres puntos, sabemos las coordenadas de cada uno de los puntos, y queremos su área. Supongamos que los puntos son denotados por P, Q y R. Podemos calcular dos de las aristas del tríangulo si calculamos los vectores que van de P a Q y de P a R. Luego podríamos calcular el producto cruz de los dos vectores que acabamos de encontrar y obtener su norma, finalmente la mitad de la norma sería el área del tríangulo que estamos buscando.

$$\begin{aligned}\overrightarrow{PQ} &= Q - P \\ \overrightarrow{PR} &= R - P \\ A &= \frac{1}{2}|\overrightarrow{PQ} \times \overrightarrow{PR}|\end{aligned}$$

Si queremos calcular el área de un cuadrilátero podemos usar la idea anterior y *triangular*, el cuadrilátero en dos partes, supongamos que queremos calcular el área del cuadrilátero formado por los puntos P, Q, R y S, podríamos calcular el área del tríangulo $\triangle PQR$ y luego sumar el área del tríangulo $\triangle SRQ$. Dicho de otra manera:

$$\begin{aligned}A_T &= A_{\triangle PQR} + A_{\triangle SRQ} \\ A_T &= \frac{1}{2}|\overrightarrow{PQ} \times \overrightarrow{PR}| \times \frac{1}{2}|\overrightarrow{SQ} \times \overrightarrow{SR}| \\ A_T &= \frac{1}{2}\left(|\overrightarrow{PQ} \times \overrightarrow{PR}| + |\overrightarrow{SQ} \times \overrightarrow{SR}|\right) \qquad (2.1)\end{aligned}$$

Este resultado de hecho, puede ampliarse aun más con el enunciado siguiente:

Sea P un polígono simple, sin agujeros, dado por la secuencia ordenada de vértices $P_i = (x_i, y_i)$, $i = 1, \ldots, n$, (por ejemplo en el sentido contrario a las agujas del reloj), entonces el área de P es:

$$A(P) = \frac{1}{2}\sum_{i=1}^{n-1} \det(P_i, P_{i+1}) + \frac{1}{2}\det(P_n, P_1)$$

Este teorema fue tomado de [12] donde también es demostrado. Con la adecuada combinación de estas técnicas es sencillo ingeniárselas para poder calcular el área de una cara.

2.4.2. Cálculo de volúmenes

Calcular volúmenes es usualmente más complejo que calcular áreas, y por ende casi siempre es computacionalmente costoso, de ahí que además de maneras geométricas de calcular un volumen siempre está la alternativa de aproximar un volumen. De nuevo hay que pensar que queremos lograr con la simulación si rapidez en la ejecución, o apego a la realidad física.

No hay que olvidar también que para todo modelo y sus respectivos parámetros, siempre hay diferentes grados de sensibilidad, por ejemplo en el caso de nuestro modelo el volumen sólo es utilizado para poder determinar la fuerza escalar de la presión, por lo que es un modelo *poco sensible* al cálculo del volumen. Es decir, es válido aproximar en este rubro, sin perder mucha realidad en el modelo.

Volúmenes aproximados

Hay varias formas de aproximar el volumen de un cuerpo, una de ellas es con cajas envolventes o *bounding boxes*, la idea es muy simple se trata de aproximar el volumen de un cuerpo complejo mediante el volumen de un cuerpo simple que lo contenga. Es decir, se aproxima el volumen por medio de poliedros regulares ya sean inscritos o circunscritos, tales que den una buena aproximación del cuerpo cuyo volúmen estamos calculando.

Una forma mas simple es aproximando con un *hexahedro regular* cuyo volumen es $l \cdot w \cdot h$, o largo por alto por ancho; otra forma geométrica que se usa de manera muy común es el *elipsoide*, cuyo volumen es $\frac{4\pi}{3} a \cdot b \cdot c$, donde a, b y c, son los largos de sus ejes principales.

Volúmenes Exactos

También se pueden tener ciertas técnicas de cálculo de volúmenes exactos mediante la descomposición de un cuerpo en cuerpos simples y calculando después su volumen, que es la técnica que usaremos aquí. Algunos consejos para el cálculo de estos volúmenes son los siguientes.

El volumen de un *paralelepípedo* construido sobre los vectores linealmente independientes $\overrightarrow{OA}$, $\overrightarrow{OB}$ y $\overrightarrow{OC}$ es $V = |\det(\overrightarrow{OA}, \overrightarrow{OB}, \overrightarrow{OC})|$

El volumen del *tetraedro* determinado por los puntos $A_1 = (x_1, y_1, z_1)$, $A_2 = (x_2, y_2, z_2)$, $A_3 = (x_3, y_3, z_3)$ y $A_4 = (x_4, y_4, z_4)$ es:

$$V = \frac{1}{6} |\det(\overrightarrow{A_1A_2}, \overrightarrow{A_1A_3}, \overrightarrow{A_1A_4})|$$

Como un ampliación a lo hecho en la figura 1.10, podemos calcular el área del *hexaedro irregular* formado por los ocho puntos $P_1, P_2, \ldots P_8$ tales que los vectores $\overrightarrow{P_1P_2}$, $\overrightarrow{P_2P_3}$, $\overrightarrow{P_3P_4}$ y $\overrightarrow{P_4P_1}$, formen la tapa superior y los vectores $\overrightarrow{P_1P_3}$, $\overrightarrow{P_1P_5}$, $\overrightarrow{P_5P_7}$ y $\overrightarrow{P_3P_7}$, formen uno de los lados. Entonces una forma de calcular el volumen es:

$$V = \frac{1}{2}\left(|\det(\overrightarrow{P_1P_2}, \overrightarrow{P_1P_3}, \overrightarrow{P_1P_5})| + |\det(\overrightarrow{P_8P_7}, \overrightarrow{P_8P_6}, \overrightarrow{P_8P_4})|\right) \tag{2.2}$$

2.4.3. Vectores normales

Dado un plano podemos encontrar un vector que le sea normal, sólo necesitamos dos vectores linealmente independientes que se encuentren en el plano y calculando su producto cruz obtenemos un vector normal, es decir en términos mas simples, si conocemos tres puntos del plano y no son colineales, podemos calcular el vector perpendicular a ese plano.

Matemáticemente: dado el triángulo formado por los puntos $A_1 = (x_1, y_1, z_1)$, $A_2 = (x_2, y_2, z_2)$ y $A_3 = (x_3, y_3, z_3)$, un vector normal a este plano, es el vector $\vec{n}$ que se calcula:

$$\vec{n} = \overrightarrow{A_1A_2} \times \overrightarrow{A_1A_3}$$

Un hecho muy importante es que si queremos calcular el vector normal a una superficie, podemos calcular los vectores normales a cada vértice de ella y luego promediarlos. Esto por increíble que parezca funciona y es de hecho la manera como se hacen los cálculos de iluminación en la mayoría de los programas de CAD y de simulación gráfica.

Es decir, si tenemos el polígono de n lados, formado por los puntos $P_1, P_2, \ldots, P_n$, ordenados de alguna manera, por ejemplo en sentido contrario a las manecillas del reloj, y queremos el vector normal a este polígono $\vec{n}$, calculamos los vectores $v_1, v_2, \ldots, v_n$, de la forma $v_i = \overrightarrow{P_iP_{i+1}} \times \overrightarrow{P_iP_{i+1}}$ para $i = 2, \ldots, n$ y $v_1 = \overrightarrow{P_1P_2} \times \overrightarrow{P_1P_n}$, y luego el vector normal $\vec{n}$ es:

$$\vec{n} = \frac{1}{n}\sum_{i=1}^{n} v_i \tag{2.3}$$

Este hecho es fundamental para nuestros cálculos, tanto de la fuerza debida a la presión, como para la iluminación en el render. Una demostración se puede encontrar en [12].

2.5. Integrar la ecuación de Newton

Desde que hay modelos físicos, ha habido interés en la solución numérica de ciertos problemas para los que se sabía de antemano que una solución existía, pero no se contaban con métodos adecuados para encontrarla. Es por esta razón que nace el análisis numérico, sin embrago no fue hasta que se popularizó el uso de las computadoras que este campo tomó más importancia.

Dentro del análisis numérico, es de nuestro interés la solución numérica de ecuaciones diferenciales. El problema comienza con una ecuación de primer orden con condición inicial y que cumple las condiciones de existencia y unicidad.

Actualmente hay muchas maneras de atacar estos problemas, es decir muchas familias de métodos. Por familia quiero decir cuando alguien propone un método y llega otra persona más y le hace correcciones, ahora hay dos métodos pero en escencia funcionan con la misma idea, por eso son dos métodos de la misma familia.

El primer método para obtener una aproximación numérica de la solución de una ecuación diferencial es el método de Euler, y desde entonces se han propuesto muchísimos métodos e ideas más. El cuadro 2.2 es una línea del tiempo que tiene algunos de los acontecimientos más importantes del desarrollo de esta disciplina.

En escencia estos métodos se encargan de resolver la ecuacion diferencial de la forma:

$$\begin{aligned} \frac{dy}{dt} &= f(t,y) \\ y(t_0) &= y_0 \end{aligned} \tag{2.4}$$

Y para hacerlo toman la condición inicial 2.4, como el primer valor de la solución, con ella calculan un valor aproximado para la solución $y(t)$ en el tiempo $t = t_0 + \Delta t$. Ahora este nuevo punto de la solución lo llamamos y_n y nos ayuda a encontrar un nuevo punto dando otro paso hacia adelante en el tiempo $t = y_0 + 2\Delta t$ Y así sucesivamente, de manera que la salida de nuestro método son una colección de parejas $(t + n\Delta t, y(t + n\Delta t))$ la primera de ellas es la condición inicial, y de ahí en adelante se trata de aproximaciones de la solución.

Sin embargo nosotros no queremos resolver una ecuación como ésta, deseamos resolver la ecuación 1.2, que es una ecuación diferencial de segundo orden. Para poder adaptar este problema al método se propone un cambio de variable $\frac{dx}{dt} = v(t)$, por lo que el problema se transforma en resolver un sistema de dos ecuaciones diferenciales.

Cuadro 2.2: Una línea de tiempo que muestra algunos de los acontecimientos mas importantes para la solución numérica de ecuaciones diferenciales

Año	Evento
1768	Leonhard Euler publica su método, el primero en la historia
1824	Agustin Cauchy demuestra la convergencia del método de Euler, emplea el método de Euler implícito
1855	En una carta escrita por John F. Bashforth, se mencionan por primera vez los metodos de pasos multiples de Couch Adams
1895	Carl Runge publica el primer método de Runge Kutta
1905	Martin Kutta describe el popular método de Runge Kutta de orden cuatro
1910	Lewis Fry Richardson anuncia su método de extrapolación.
1952	Charles F. Curtiss y Joseph Oakland Hirschfelder acuñan el término *stiff equations*.
1967	Loup Verlet publica su método, especialmente enfocado a la mecánica de partículas

$$\begin{aligned} \frac{dv}{dt} &= \frac{1}{m}F(x,v,t) \\ \frac{dx}{dt} &= v(t) \\ v(0) &= v_0 \\ x(0) &= x_0 \end{aligned}$$

Los métodos siguientes entonces supondrán que sabemos la velocidad y la posición de la partícula en el tiempo t, así como la fuerza que actúa sobre ella en este tiempo, con esta última podremos calcular la posición y la velocidad de la partícula en un tiempo posterior $t + \Delta d$.

2.5.1. El método de Euler

Este método fue el primero de todos, de ahí que también sea el más simple, sin embargo aún tiene algunas ventajas el usarlo. Básicamente el método de Euler nos dice que si tenemos una ecuación de

la forma:

$$y'(t) = f(t, y)$$

Con una condición inicial de la forma $y(0) = y_0$.

Entonces podemos aproximar el siguiente punto de la solución con la siguiente fórmula de recurrencia:

$$y_{n+1} = y_n + hf(t_n, y_n)$$

En donde h representa el tamaño del paso en el tiempo hacia adelante es decir $h = \Delta t$.

El método de Euler se puede generalizar para sistemas de ecuaciones diferenciales de tamaño n. Estas generalizaciones se pueden ver en casi cualquier libro de Ecuaciones y en cualquier libro de análisis numérico. Aquí sólo daré las fórmulas de recurrencia para el caso de $n = 2$ por ser el que estamos interesados en resolver.

Dado el sistema:

$$\begin{aligned} x'(t) &= f(t, x, y) \\ y'(t) &= g(t, x, y) \end{aligned} \tag{2.5}$$

Con las condiciones iniciales:

$$\begin{aligned} x(t_0) &= x_0 \\ y(t_0) &= y_0 \end{aligned} \tag{2.6}$$

Entonces la solución se puede aproximar con las siguientes fórmulas de recurrencia:

$$x_{n+1} = x_n + hf(t_n, x_n, y_n)$$

$$y_{n+1} = y_n + hg(t_n, x_n, y_n)$$

Ahora vamos a poner nuestro sistema de manera que podamos ocupar estas fórmulas para resolverlo:

$$\begin{aligned} v'(t) &= \frac{1}{m} F(t, x, v) \\ x'(t) &= v(t, x, v) \end{aligned} \tag{2.7}$$

Y nuestras condiciones iniciales son:

$$\begin{aligned} v(t_0) &= v_0 \\ x(t_0) &= x_0 \end{aligned} \tag{2.8}$$

Por lo tanto para resolver nuestro sistema con el método de Euler se tiene:

$$\begin{aligned} v_{n+1} &= v_n + \frac{h}{m} F(t_n, x_n, v_n) \\ x_{n+1} &= x_n + h v_{n+1} \end{aligned} \tag{2.9}$$

El error del método de Euler

Como en todo procedimiento numérico, en el método de Euler se tiene un error, el cual puede encontrarse para nuestro caso por medio de la expansión en la serie de Taylor para la función que determina la posición de una partícula.

$$\begin{aligned} x_{n+1} &= x_n + h v_n \\ x(t_0 + h) &= x(t_0) + h v(t_0) \end{aligned}$$

Pero sabemos que la serie de Taylor de la trayectoria de una partícula es:

$$x(t_0 + h) = x(t_0) + h x'(t_0) + \frac{1}{2} h^2 x''(t_0) + O(h^3)$$

Entonces el error en el método de Euler está dado por la diferencia de ambas expresiones es decir:

$$-\frac{1}{2} h^2 x''(t_0) + O(h^3)$$

Este error es el error de truncamiento, o debido al método en sí. Al momento de implementarlo existe también un error por redondeo, que es debido a que las computadoras operan con un número finito de decimales, sin embargo este error es difícil de estimar y sale del alcance de mi investigación. De aquí en adelante cuando me refiera al error de un método numérico siempre me referiré al *error por truncamiento.*

Ventajas y desventajas del método de Euler

Como todo algortimo este método presenta ventajas y desventajas, éstas se deben tanto al método como a su implementacion en este modelo. Aquí listaré algunas de ellas.

Ventajas:

- Se puede implementar fácilmente.
- Se puede integrar partícula por partícula, dado que sólo requiere de una evaluación de F.
- Es rápido de ejecutarse.

Desventajas:

- Tiende a *explotar* rápidamente.
- Su error es alto.
- Es muy sensible a las variaciones pequeñas, por lo que tarda en estabilizarse.

2.5.2. El método de Runge Kutta

Supongamos que se tiene una ecuación de la forma

$$y'(t) = f(t, y)$$

Con su respectiva condición inicial de la forma $y(0) = y_0$.

El método de Runge y Kutta nos dice que la solución en el siguiente paso de tiempo puede aproximarse con la siguiente fórmula de recurrencia:

$$y_{n+1} = y_n + \frac{h\left(k_{n1} + 2k_{n2} + 2k_{n3} + k_{n4}\right)}{6}$$

En donde h representa el tamaño del paso en el tiempo que se desee aproximar y los coeficientes k_{n1}, k_{n2}, k_{n3}, k_{n4} se pueden calcular de la siguiente manera:

$$\begin{aligned} k_{n1} &= f(t_n, y_n) \\ k_{n2} &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_{n1}) \\ k_{n3} &= f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_{n2}) \\ k_{n4} &= f(t_n + h, y_n + hk_{n3}) \end{aligned}$$

Al igual que en el método de Euler, existe una generalización del método de Runge Kutta para poderse ocupar con sistemas de ecuaciones de primer orden. En casi cualquier libro de ecuaciones se pueden ver estas fórmulas (por ejemplo en [3]); aquí solo pongo el caso $n = 2$ porque es el que voy a ocupar en el modelo.

Supongamos que tenemos de nuevo el sistema 2.5, sujeto a 2.6. Podemos aproximar una solución usando el método de Ruge Kutta con la siguiente fórmula de recurrencia, suponiendo que queremos ir de t_n a t_{n+1}.

$$\begin{aligned} x_{n+1} &= x_n + \frac{h(k_{n1} + 2k_{n2} + 2k_{n3} + k_{n4})}{6} \\ y_{n+1} &= y_n + \frac{h(l_{n1} + 2l_{n2} + 2l_{n3} + l_{n4}))}{6} \end{aligned}$$

Aquí podemos apreciar que ahora tenemos que encontrar cuatro ponderadores, los k_i y l_i significa que tenemos mas cálculos por hacer. Los valores de los ponderadores se calculan con las siguientes fórmulas.

$$\begin{aligned} k_{n1} &= f(t_n, x_n, y_n) \\ k_{n2} &= f(t_n + \frac{h}{2}, x_n + \frac{h}{2}k_{n1}, y_n + \frac{h}{2}l_{n1}) \\ k_{n3} &= f(t_n + \frac{h}{2}, x_n + \frac{h}{2}k_{n2}, y_n + \frac{h}{2}l_{n2}) \\ k_{n4} &= f(t_n + h, x_n + hk_{n3}, y_n + hl_{n3}) \end{aligned}$$

y

$$\begin{aligned} l_{n1} &= g(t_n, x_n, y_n) \\ l_{n2} &= g(t_n + \frac{h}{2}, x_n + \frac{h}{2}k_{n1}, y_n + \frac{h}{2}l_{n1}) \\ l_{n3} &= g(t_n + \frac{h}{2}, x_n + \frac{h}{2}k_{n2}, y_n + \frac{h}{2}l_{n2}) \\ l_{n4} &= g(t_n + h, x_n + hk_{n3}, y_n + hl_{n3}) \end{aligned}$$

El método de Runge Kutta, es mucho más exacto que el método de Euler, de hecho se puede apreciar que el método de Euler, aproxima el siguiente paso con el valor de una pendiente, en cambio el método de Runge Kutta aproxima el siguiente paso con el valor de cuatro pendientes ponderadas, cada una calculada con la aproximación de la anterior.

Supongamos que se tiene de nuevo 2.7 sujeto a las condiciones 2.8, entonces el metodo de Runge Kutta toma la siguiente forma:

$$\begin{aligned} v_{n+1} &= v_n + \frac{h}{6}(k_{n1} + 2k_{n2} + 2k_{n3} + k_{n4}) \\ x_{n+1} &= x_n + \frac{h}{6}(l_{n1} + 2l_{n2} + 2l_{n3} + l_{n4}) \end{aligned} \tag{2.10}$$

Y los ponderadores se pueden calcular de la siguiente manera:

$$\begin{aligned} k_{n1} &= \frac{1}{m}F(t_n, x_n, v_n) \\ l_{n1} &= v_n \\ k_{n2} &= \frac{1}{m}F(t_n + \frac{h}{2},\ x_n + \frac{h}{2}k_{n1},\ v_n + \frac{h}{2}l_{n1}) \\ l_{n2} &= v_n + \frac{h}{2}k_{n1} \\ k_{n3} &= \frac{1}{m}F(t_n + \frac{h}{2},\ x_n + \frac{h}{2}k_{n2},\ v_n + \frac{h}{2}l_{n2}) \\ l_{n3} &= v_n + \frac{h}{2}k_{n2} \\ k_{n4} &= \frac{1}{m}F(t_n + h,\ x_n + hk_{n2},\ v_n + hl_{n2}) \\ l_{n4} &= v_n + hk_{n3} \end{aligned} \tag{2.11}$$

El error del método de Rungue Kutta

Se puede demostrar por medios algebraicos y de la misma manera que se empleó con el método de Euler que el método de Runge Kutta tiene un error del orden de $O(h^5)$. Es decir si partimos el intervalo h por la mitad, y damos el doble de pasos el error por truncamiento disminuye en un orden de 16 veces (es decir $\left(\frac{h}{2}\right)^4$, como el error es del orden de $O(h^5)$ se hace h^4 veces más exacto).

Ventajas y desventajas del metodo de Runge Kutta

Como se puede leer en la bibliografía, el método de Euler y el método de Runge Kutta dependen del tamaño del paso h, y se hacen más exactos mientras el paso es más pequeño, pero también se hacen más cálculos, por lo que aumenta la fuente del otro tipo de error, el error por redondeo. Se acepta de manera general, que de este tipo de métodos, aquel que optimiza esta situación es el método de Runge Kutta de cuarto orden, el método que acabamos de presentar.

Algunas de las ventajas y desventajas para el método de Runge Kutta son las siguientes.

Ventajas

- Es el método estándar más recomendado y no es tan difícil de implementar (métodos más exactos son considerablemente más difíciles de implementar).
- Es recomendado por muchos autores y por lo tanto hay mucha documentación.
- Es rápido de ejecutarse, (Más lento que el Euler, pero para efectos de la animación la diferencia no es apreciable).
- Es un método muy estable, es muy difícil que explote.

Desventajas

- A diferencia de Euler, requiere de muchas evaluaciones de la función en diferentes puntos (lo que para nosotros significa calcular muchas veces la fuerza sobre la partícula).
- Las partículas no se pueden integrar por separado, tendrán que integrarse juntas. (todas las que conforman el cuerpo en un solo paso).

2.6. Cómo enfrentar la colisión de los cuerpos

Por colisión de los cuerpos entenderemos la manera como lidiar cuando dos cuerpos dentro de la escena quedan en contacto uno con el otro. Desde nuestro punto de vista esta respuesta se divide en dos pasos: la detección de la colisión y la respuesta a la colisión. La detección es totalmente un problema geométrico, consiste en que a partir de la información que tenemos de los cuerpos y de la colisión podemos determinar al punto de la colisión y un vector normal a ese punto, este problema es totalmente dependiente de la forma de los cuerpos. Por el otro lado la respuesta a la colisión es un problema físico y generalmente es más sencillo debido que ya existen algoritmos bien definidos para responder a las colisiones y son independientes de la geometría, es decir son algoritmos generales.

2.6.1. La detección de las colisiones

Como ya se dijo antes la detección de las colisiones es uno de los problemas más complejos que nos podemos enfrentar al hacer una animación. Básicamente debemos de poder hacer dos cosas aquí: uno, detectar la colisión, es decir mediante una prueba rápida saber si los cuerpos de nuestra escena entraron en contacto, y después ver si podemos determinar un vector normal al punto (o superficie en 3D) de colisión.

Hay básicamente dos tipos de estrategias para resolver este problema, una es por medio de un *bounding volume*, que se trata de darle la vuelta al problema con una aproximación y otra es por medios estrictamente geométricos es decir tratar de dividir tus cuerpos en figuras de formas elementales, como esferas o paralelepípedos para los cuales las detección es un poco más sencilla.

Un *bounding volume*

Básicamente se trata de imaginar que hay un envolvente de nuestro objeto y este envolvente tiene una forma más sencilla, entonces al probar por la colisión se prueba con el envolvente no con el objeto. Esto tiene la enorme desventaja de ser una aproximación, por lo que la animación se vera un poco saltada, sin embargo una cuidadosa elección del objeto envolvente hará que este efecto sea mínimo.

Objetos como las elipses, los paralelepípedos y los cilindros son buenos objetos para ser un *bounding volume*, porque la detección de la colisión es sencilla.

Por ejemplo, en una esfera podemos determinar si un punto esta dentro de ella con tan sólo comparar el cuadrado de su distancia con respecto al centro de la esfera, con el cuadrado del radio.

Un cilindro con el eje vertical alineado con el eje de la escena es también muy usado. Si queremos saber si dos objetos contenidos en un cilindro, por ejemplo en un videojuego dos personajes caminando, chocan, sólo debemos ver si la proyección de los ejes principales de los cilindros sobre el eje de la escena (líneas rectas) se interceptan y además las proyecciones de los dos cilindros con el plano XZ (dos círculos) también se intersecan.

Un paralelepípedo o rectángulo, o *bounding box*, también se usa mucho, por ejemplo cuando la geometría de los objetos hace a la esfera una mala elección, por ejemplo al ver si dos coches chocan en un juego de carreras el rectángulo es una elección mas sabia.

Un rectángulo usado como *bounding box* generalmente se alinea con las coordenadas del mundo, para así hacer las pruebas de detección triviales, si por el contrario el rectángulo es alineado con las coordenadas del objeto y este tiene la capacidad de rotar, cada vez que lo haga se necesita volver a calcular el *bounding box*.

Uniendo diferentes geometrías

Una manera más exacta de predecir si dos objetos de una escena están en colisión, es descomponer la forma completa de un objeto en varios objetos pequeños, y luego probar contra todos los objetos que componen el cuerpo si es que existió la colisión. Por ejemplo, en el caso de la simulación del cuerpo flexible se podría pensar como que cada partícula que forma el cuerpo es un objeto y luego probar la colisión contra todas las partículas que forman el cuerpo flexible.

Esta técnica es bastante más cara, tanto de implementar como de ejecutarse, sin embargo, da resultados visiblemente más acertados, por ejemplo en un juego de lucha libre, donde la interacción de los luchadores debe de ser bastante creíble, se recomendaría usar este tipo de colisiones.

2.6.2. La respuesta de las colisiones

Como ya habíamos dicho, el trabajo difícil y dependiente de la animación está en la detección de las colisiones, y la respuesta a las colisiones es totalmente física. Para empezar las colisiones se dividen en dos: elásticas e inelásticas. La implementación de ambas no se diferencia mucho, aunque sí son diferentes en concepto.

Básicamente en un algoritmo de respuesta de colisiones, se suministran las posiciones y las velocidades de los dos objetos que coliden, más un vector normal al punto de colisión de los dos objetos, o plano normal en el caso de tres dimensiones. Desde luego que hay dos vectores que cumplen con esa condición, cualquiera de ellos nos servirá siempre y cuando sepamos cual de ellos es el que tenemos.

Ya con esta información debemos ser capaces de responder con dos cosas: una nueva posición de los objetos y una nueva velocidad.

En la figura 2.3 se ven los diferentes pasos de la respuesta a la colisión de dos círculos.

Separar un vector en componentes normal y tangencial

Antes de explicar la forma en que se responden las colisiones quiero hacer énfasis en la manera como separa un vector en componentes ortogonales (figura 2.4), porque es necesario hacerlo en la respuesta a las colisiones.

Para separar a un vector cualquiera $\vec{v}$ en dos componentes: uno normal $\vec{v_n}$, y otro tangencial $\vec{v_t}$ respecto a un vector normal $\vec{n}$, se hace uso de las siguientes fórmula:

$$\begin{aligned} \vec{v_n} &= \frac{(\vec{v}\cdot\vec{n})}{|\vec{n}|}\frac{\vec{n}}{|\vec{n}|} \\ \vec{v_t} &= \vec{v} - \vec{v_n} \end{aligned}$$

Como en los cálculos de detección de colisiones es común que tengamos un vector normal $\vec{n}$, tal que: $|\vec{n}| = 1$, las fórmulas anteriores se simplifican aun más siendo la forma más usada la siguiente[3]:

$$\begin{aligned} \vec{v_n} &= (\vec{v}\cdot\vec{n})\vec{n} \\ \vec{v_t} &= \vec{v} - \vec{v_n} \end{aligned} \tag{2.12}$$

Colisiones elásticas

Una colisión elástica es aquella donde el momento y la energía de los objetos, se conservan después de la colisión; son una abstracción que nunca sucede en la vida real. Aun las colisiones en el espacio exterior son inelásticas aunque están muy cerca de no serlo [9], pero nos sirven bastante para entender el fenómeno, y en algunos casos son suficientes. Por ejemplo, pensemos que estamos modelando un juego de billar en 2D una colisión elástica es suficiente.

[3]Las fórmulas 2.12, se encuentran mal escritas en [2], y esta fue una de las razones que más me retrasó al momento de hacer este trabajo.

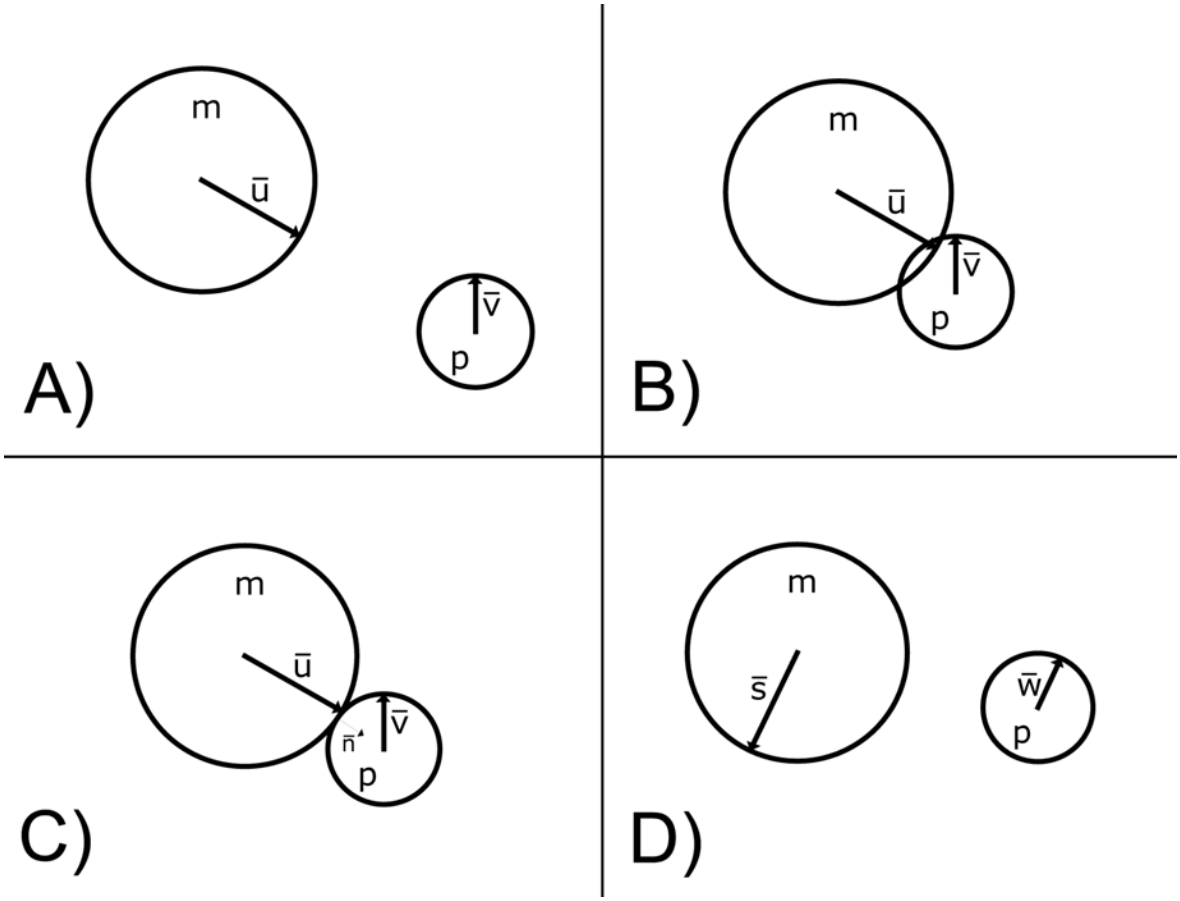

Figura 2.3: A) La situación justo antes de que ocurra la colisión. B)La situación en el momento en que se detecta la colisión. C)La respuesta a la colisión. D)Las velocidades ajustadas después de la colisión

Cuadro 2.3: Condiciones de la respuesta a las colisiones

Objeto	Información conocida	Información por determinar
A	Masa m, Velocidad $\vec{u}$	Velocidad ajustada $\vec{s}$
B	Masa p, Velocidad $\vec{v}$	Velocidad ajustada $\vec{w}$

Supongamos que se tienen dos cuerpos A y B, las formas no importan, y sabemos, gracias a un algoritmo de detección de las colisiones, que están en colisión y un vector normal $\vec{n}$ a la superficie de colisión. Suponemos que este vector apunta del cuerpo A al cuerpo B y que es también un vector unitario.

Suponemos también que conocemos todas las propiedades de los cuerpos, es decir su masa, su velocidad y su posición.

Queremos determinar una nueva posición y una nueva velocidad para los objetos como resultado de la colisión entre ellos. Esto se resumen el el cuadro 2.3.

Para entender el porqué de la respuesta a la colisión, es necesario seguir los siguientes pasos[4]. Primero vamos por el caso más simple: dos objetos A y B, con velocidades $\vec{u}$ y $\vec{v}$ respectivamente chocan, como respuesta a la colisión las velocidades de ambos objetos cambian a $\vec{s}$ y $\vec{w}$. Ver la figura 2.3.

Suponemos que: $\vec{v} = 0$. Es decir el cuerpo B no se mueve, está detenido esperando la colisión del cuerpo A. Sabemos que tenemos un vector $\vec{n}$ normal al plano de colisión. Con este vector podemos dividir los demás vectores en una parte normal al plano y otra tangencial (figura 2.4), es decir:

$$\begin{aligned} \vec{u} &= \vec{u_t} + \vec{u_n} \\ \vec{v} &= \vec{v_t} + \vec{v_n} \\ \vec{s} &= \vec{s_t} + \vec{s_n} \\ \vec{w} &= \vec{w_t} + \vec{w_n} \end{aligned}$$

Ahora veamos que es lo qué sabemos de antemano por la forma como planteamos las condiciones del problema: sabemos que: $\vec{u_t} = \vec{s_t}$ y $\vec{v_t} = \vec{w_t} = 0$, porque esperaríamos que las velocidades sólo se vieran afectadas en su componente normal y porque sabemos que el objeto B estaba inicialmente en reposo. Así que lo único que necesitamos saber es $\vec{s_n}$ y $\vec{w_n}$.

[4]Esta es una reproducción del procedimiento mostrado en [9] con mi nomenclatura

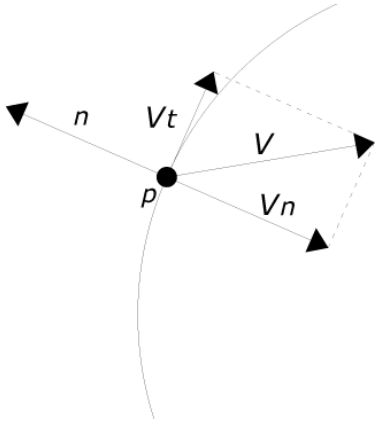

Figura 2.4: El vector V se separa con respecto a n en dos vectores uno normal V_n, y uno tangencial V_t

Se sabe también que, como la colisión es elástica, debe de obedecer la ley de la conservación de la energía y la la ley de la conservación de momento.

$$m\bar{u_n} = m\bar{s_n} + p\bar{w}_n \tag{2.13}$$

$$\frac{1}{2}m\bar{u}^2 = \frac{1}{2}m\bar{s}^2 + \frac{1}{2}p\bar{w}^2 \tag{2.14}$$

Así que tenemos justo las condiciones necesarias para encontrar una solución, pues tenemos dos valores por determinar y dos ecuaciones que las relacionan. Para encontrar la solución hacemos un cambio de variable $r = \frac{m}{p}$. Luego despejamos de la ecuación 2.13 a $\bar{w}_n$, y lo sustituimos en 2.14 y obtenemos el valor de $\bar{s_n}$. En realidad se obtienen dos valores $\bar{s_n} = \bar{u_n}$ y $\bar{s_n} = \bar{u_n}\frac{r-1}{r+1}$, tomamos el segundo (el primero corresponde a la condición inicial, por que las colisiones elásticas son reversibles en el tiempo), y lo sustituimos de nueva en 2.13 para obtener el valor de $\bar{w_n} = \frac{2r\bar{u_n}}{r+1}$.

Con esto se tiene todo lo necesario para resolver una colisión elástica con uno de los dos objetos en reposo. Podemos utilizar el siguiente pseudocódigo:

1. $r = \frac{m}{p}$
2. $\bar{u_n} = parteNormal(\vec{u}, \vec{n})$
3. $\bar{u_t} = \bar{u} - \bar{u_n}$

4. $\bar{s_n} = \bar{u_n}\left(\frac{r-1}{r+1}\right)$
5. $\bar{w_n} = \bar{u_n}\left(\frac{2r}{r+1}\right)$
6. $\bar{s} = \bar{u_t} + \bar{s_n}$
7. $\bar{w} = \bar{w_n}$

En donde la función `parteNormal`, es una función que recibe un vector $\vec{u}$ y un vector $\vec{n}$, devuelve la parte normal de $\vec{u}$ con respecto a $\vec{n}$. Es decir nos sirve para partir al vector $\vec{u}$ en una parte tangencial y una parte normal a la colisión haciendo uso de las fórmulas 2.12.

Ahora veamos el caso más general, donde $\vec{v} \neq 0$. Aquí vamos a ocupar el principio de relatividad y le restamos a todo el sistema $\vec{v}$, lo que lo transforma en el caso anterior. Resolvemos como lo habíamos hecho antes y luego le sumamos a todo el sistema $\vec{v}$. El pseudocódigo es casi idéntico:

1. $r = \frac{m}{p}$
2. $\bar{u} = \bar{u} - \bar{v}$
3. $\bar{u_n} = parteNormal(\vec{u}, \vec{n})$
4. $\bar{u_t} = \bar{u} - \bar{u_n}$
5. $\bar{s_n} = \bar{u_n}\left(\frac{r-1}{r+1}\right)$
6. $\bar{w_n} = \bar{u_n}\left(\frac{2r}{r+1}\right)$
7. $\bar{s} = \bar{u_t} + \bar{s_n} + \bar{v}$
8. $\bar{w} = \bar{w_n} + \bar{v}$

Con esto podemos programar un función general que resuelva colisiones elásticas, sólo debemos asegurarnos de que la función que detecta las colisiones le informe a aquella función de tres cosas: las propiedades del objeto A, las propiedades del objeto B y un vector normal al plano de colisión que vaya de A a B.

Colisiones inelásticas

En una colisión inelástica se tienen un pérdida de energía como respuesta al impacto. En la realidad las colisiones son inelásticas, podemos apreciar parte de la pérdida de la energía, al escuchar el sonido de la colisión. Por ejemplo en el billar.

Para simular un colisión inelástica, vamos también a hacer una suposición: que los objetos tiene una cierta eficiencia, y que ésta se mantiene fija para todas las colisiones que involucren ese objeto. Es una gran simplificación porque en la realidad la eficiencia de una colisión depende de muchos factores, como el medio ambiente, o la velocidad de los objetos al momento de la colisión.

Si dos objetos chocan y uno de ellos tiene *coeficiente de restitución* de 0.9 y el otro un coeficiente de restitución de 0.85, la energía total después de la colisión seria: $0.9 \times 0.85\ (E_b + E_a)$. Donde E_a y E_b es la energía de cada objeto antes de la colisión. Como se puede ver el coeficiente de restitución es una manera de medir la *eficiencia* y significa que el primer objeto después de una colisión transmite por ejemplo el 95 por ciento de su energía.

Para resolver una colisión inelástica se usa el mismo procedimiento que en la sección anterior, sólo que ahora la ecuación 2.14, toma la siguiente forma:

$$\frac{1}{2}em\bar{u}^2 = \frac{1}{2}m\bar{s}^2 + \frac{1}{2}p\bar{w}^2 \tag{2.15}$$

Donde e es el producto de los coeficientes de restitución de ambos objetos. Y se procede de la misma manera que en el caso anterior a calcular los valores de w_n y s_n. Resolviendo el sistema formado por la ecuación 2.13 y la 2.15.

Las nuevas soluciones son:

$$\begin{aligned} s_n &= \frac{-ru - \sqrt{r^2u_n^2 - (r+1)\left((r-e)u_n^2 + (1-e)u_t^2\right)}}{r+1} \\ w_n &= r\left(nu_n - v_n\right) \end{aligned}$$

Y el pseudocódigo, que resuelve la colisión de manera inelástica es el siguiente:

1. $r = \frac{m}{p}$
2. $e = eficiencia(A) \cdot eficiencia(B)$
3. $\bar{u} = \bar{u} - \bar{v}$

4. $\bar{u_n} = parteNormal(\vec{u}, \vec{n})$
5. $\bar{u_t} = \bar{u} - \bar{u_n}$
6. $dis = r^2\,(\bar{u_n} \cdot \bar{u_n}) - ((r-e)(\bar{u_n} \cdot \bar{u_n}) + (1-e)(\bar{u_t} \cdot \bar{u_t}))$
7. $\bar{s_n} = \bar{n}\left(\frac{\sqrt{dis} - r\bar{u_n}}{r+1}\right)$
8. $\bar{w_n} = r\left(\frac{\bar{u_n} - \bar{v_n}}{r+1}\right)$
9. $\bar{s} = \bar{u_t} + \bar{s_n} + \bar{v}$
10. $\bar{w} = \bar{w_n} + \bar{v}$

Capítulo 3

Implementación del código en lenguaje C

Este capítulo está dedicado a cubrir los detalles de la implementación del modelo en código. Dado que la principal finalidad de este trabajo es el modelado físico no entraré en detalles de otras áreas del programa más que las que tienen que ver con el modelo. Sólo hablare de los módulos del programa que tienen que ver con lo tratado en los capítulos anteriores.

La primera parte describe cómo se van a crear las estructuras de datos necesarias para guardar en memoria los datos del modelo.

En la siguiente sección se ve la manera como se implementaron en código los métodos numéricos usados para integrar la ecuación de Newton.

En la tercera sección se explican las rutinas de la física del modelo, concretamente las rutinas que se encargan de aplicar las fuerzas que intervienen en el modelo.

Por último se explica cómo es que se implementaron la rutina tanto de detección como de respuesta a las colisiones.

3.1. Definir la estructura de los datos

Todo programador debería de preocuparse en como debe de guardar los datos en memoria, aquí recurrí a tipos de datos simples, pero siempre teniendo en mente crear un tipo de dato que permitiera hacer las operaciones de acumulación de fuerzas de un manera sencilla.

3.1.1. Las partículas

La parte fundamental del modelo la constituyen los puntos o partículas; de cada una de éstas nos interesa saber básicamente tres cosas: su posición, su velocidad y la fuerza que se le está aplicando. Con esto en mente la primera estructura de datos que se creó fue `Punto`.

```
typedef struct
{
    float x, y, z;
    float vx, vy, vz;
    float fx, fy, fz;
    int fixed;
}Punto;
```

En donde `x`, `y` y `z`, son números reales que representan la posición del punto en el espacio. Los valores `vx`, `vy` y `vz` representan la velocidad del punto en forma también de vector y por ultimo `fx`, `fy` y `fz` representan la fuerza aplicada al punto en un paso de tiempo particular. Y también hay una variable entera, `fixed`, ésta se usa sólo como una bandera que toma sólo dos valores 0 ó 1, y que significa que el punto está fijo en la escena cuando vale 1 (que en lenguaje C se evalúa a verdadero igual que cualquier entero diferente de 0).

3.1.2. Los resortes

Una parte importante del modelo son los osciladores o resortes amortiguadores, la siguiente estructura de datos guarda todo lo necesario con respecto a estos.

```
typedef struct
{
    Punto *a, *b;
    float lenght;
}Resorte;
```

En donde `a` y `b` son apuntadores a variables de tipo `Punto`, que van a ser los dos puntos que están conectados por este resorte, y son referencias porque de esta manera puedo saber y modificar los datos que guardan esos puntos a través de la estructura resorte (lo cual resulta ideal al momento de acumular la fuerza del resorte).

También hay una variable flotante `lenght`, que se utilizará para guardar la distancia entre estos dos puntos, o lo que es lo mismo la longitud del resorte en un momento del tiempo.

3.1.3. Las caras

Para hacer la acumulación de la fuerza debida a la presión, es necesario organizar el cuerpo flexible en un conjunto de caras, justo como se vio en 1.1. Para este fin se hace uso de otra estructura más, la estructura `Cara`.

```
typedef struct
{
    Punto *a, *b, *c, *d;
    float nx, ny, nz;
}Cara;
```

Aquí se tienen cuatro apuntadores de tipo `Punto`, esto por que se decidió hacer caras en forma de cuadriláteros, y también al momento de acumular la fuerza debida a la presión es necesario modificar los valores de estos puntos a través de la `Cara` que forman. Hay también tres variables `nx`, `ny` y `nz`, que se ocupan para guardar el vector normal a esta cara, lo cual es un dato que servirá para hacer la acumulación de la fuerza debida a la presión.

3.1.4. La esfera

Por último, una estructura que utilizaré para guardar información de un cuerpo rígido en la escena: una esfera.

```
typedef struct
{
    float x, y, z;
    float Vx, Vy, Vz;
    float Fx, Fy, Fz;
    float radio;
}Esfera;
```

Los valores `x`, `y`, y `z` guardan el vector de posición, los valores `Vx`, `Vy` y `Vz` guardan la velocidad y la fuerza se almacena en `Fx`, `Fy`, y `Fz`. También hay una variable extra `radio`, que al tratarse de una esfera me dice todo lo necesario para poder dibujarla en la escena.

3.1.5. Arreglos de datos

Por último la estructura de datos mas importante: los arreglos donde se guarda la información acerca del cuerpo flexible que queremos modelar.

Hay básicamente tres arreglos, que son globales a todos los módulos del programa.

```
Punto tela[PARTICULAS];
Resorte esqueleto[RESORTES];
Cara mosaico[CUADROS];
```

En donde las constantes `PARTICULAS`, `RESORTES` y `CUADROS` son enteros previamente definidos.

Hay algunas cosas que decir aquí: primero, que lo que se quiere modelar es una tela cuadrada que servirá como cuerpo flexible, por lo que el número de partículas es en realidad n^2 en donde n es el número de puntos en cada lado de la tela, y por las mismas condiciones el número de resortes totales es $2n(n-1)$ y el número de caras es: $(n-1)^2$. Segunda, que, pese al arreglo cuadrado de la tela, los puntos son guardados en un arreglo unidimensional (lineal), esto por que simplifica y hace mucho más generales las rutinas de acumulación de fuerza.

La rutina que inicializa el arreglo de puntos es la siguiente:

```
primero = -MUNDO / 2;
ultimo = MUNDO / 2;
incremento = (ultimo - primero) / (LADO - 1);
L = incremento;
posX = posZ = primero;

for (i = 0; i < PARTICULAS; i++)
{
  tela[i].x = posX;
  tela[i].y = 0.0f;
  tela[i].z = posZ;
  tela[i].fixed = 0;

  posX += incremento;
  if (i % LADO == (LADO - 1))
  {
     posX = primero;
```

```
      posZ += incremento;
  }

}
```

Se forma el arreglo de la tela, que debe medir la mitad del mundo y debe estar centrada en su origen, también se inicializa la variable `L`, que guarda la distancia entre cada uno de los puntos, o lo que es lo mismo el largo de los resortes en reposo.

Ahora falta inicializar el arreglo de resortes que apunta a los puntos que ya tenemos dentro del arreglo `tela`.

```
int i, k = 0;
/** Ciclo por el arreglo esqueleto, y cada resorte le conecto
  sus dos puntos que le corresponden */
for (i = 0; i < PARTICULAS ; i++)
{
  /* Resorte Horizontal */
  if ((i % LADO) != (LADO - 1))
  {
     esqueleto[k].a = &tela[i];
     esqueleto[k].b = &tela[i + 1];
     k++;
  }
}

for (i = 0; i < (LADO * (LADO - 1)); i++)
{
  /* Resorte Vertical */
     esqueleto[k].a = &tela[i];
     esqueleto[k].b = &tela[i + LADO];
     k++;
}
```

Primero se construyen los resortes que son paralelos al eje x, y después se construyen los resortes que son perpendiculares a los primeros, es decir son paralelos al eje z. También hay que notar que en ningún momento se hace uso de resortes estructurales.

Y por último la rutina que construye las caras, de nuevo se toman como base los puntos guardados en el arreglo tela.

```
int i, j = 0;
/** Cicla por todas las caras */
for (i = 0; i < (LADO * (LADO - 1)); i++)
{
   if ( (i % LADO) != (LADO - 1) )
   {
     /* Haz una cara */
     mosaico[j].a = &tela[i];
     mosaico[j].b = &tela[i + 1];
     mosaico[j].c = &tela[i + LADO];
     mosaico[j].d = &tela[i + LADO + 1];
     j++;
   }
}
```

Esta rutina es especial en el sentido que se debe de tener cuidado de unir cada par de puntos, i, $i+1$, con los puntos que están justo debajo de ellos, por eso se tiene que prevenir que al llegar a la parte inferior de la tela, ya no haya más puntos abajo. Es por eso que el ciclo for itera mas veces que el numero total de caras. Con esto se garantiza que se puedan visitar todos las particulas del modelo de tres formas distintas, ya sea por resortes, o por caras, o accediendo directamente por medio del arreglo de puntos `tela`.

3.2. La física del modelo

Toca el turno de ver las rutinas que tienen que ver con la acumulación de fuerzas en el modelo. Como se ha dicho antes hay básicamente tres fuerzas que se deben acumular, la de la gravedad, la de los resortes amortiguadores y la debida a la presión del gas.

Estas rutinas se encuentran en los archivos: fisica.c y fisica.h

3.2.1. La fuerza de gravedad

La primera y más sencilla de las fuerzas que vamos a poner es la de gravedad. Como se dijo desde 1.3, sólo depende de dos cosas de la masa del objeto y de la constante de gravedad, y además sólo afecta en un componente vectorial el componente y, del vector fuerza.

La fuerza de gravedad se aplica a cada una de las partículas del cuerpo flexible, por lo que es conveniente acumularla por medio del un arreglo que contenga todos los puntos del cuerpo flexible. Aquí está la función que pone la fuerza de gravedad:

```
void putGravity(Punto ptos[PARTICULAS])
{
   int i;
   for (i = 0; i < PARTICULAS; i++)
   {
      ptos[i].fy += M * G;
   }

}
```

La función `putGravity`, recibe como parámetro un arreglo de puntos, y acumula sobre cada uno de los puntos la fuerza de gravedad.

La constante `M` es global y tiene la masa de cada una de las partículas, y la constante `G`, también global contiene la constante de gravedad. En palabras simples acumula la fuerza sobre cada punto usando la ecuación 1.3.

¿Por qué mandar como parámetro el arreglo de puntos, si es una variable global? Bueno eso se debe a que la gravedad no siempre será aplicada sobre el arreglo en el que guardamos puntos con los que se dibuja la escena, debido a que los métodos numéricos de integración (en particular el Runge Kutta) van a requerir copias del arreglo de puntos para hacer cálculos, por lo que no siempre se le mandará el *mismo* arreglo de puntos.

3.2.2. La fuerza de los resortes

Aquí es donde empezaremos a notar el porqué de la construcción de los demás arreglos. Primero vamos a ver qué se debe de hacer en esta función: se debe de ciclar por todos los resortes que se

encuentran en el modelo; cada resorte se debe ocupar la ecuación 1.23, para acumular la fuerza en los dos puntos que son unidos por ese resorte.

```
void putSpringForce(Resorte res[RESORTES])
{
  int i;
  float rX, rY, rZ, vX, vY, vZ, Fd, Fs, Fx, Fy, Fz, largo;

  for (i = 0; i < RESORTES; i++)
  {
    // La distancia entre los dos puntos
    largo = distancia(*res[i].a, *res[i].b);

    if (largo != 0.0)
    {
      // la diferencia de las posisciones
      rX = res[i].a->x - res[i].b->x;
      rY = res[i].a->y - res[i].b->y;
      rZ = res[i].a->z - res[i].b->z;

      // la diferencia de las velocidades
      vX = res[i].a->vx - res[i].b->vx;
      vY = res[i].a->vy - res[i].b->vy;
      vZ = res[i].a->vz - res[i].b->vz;

      /* Calculo de las fuerzas (escalares)*/
      Fd = Kd * (rX * vX + rY * vY + rZ * vZ) / largo;
      Fs = Ks * (largo - L);

      /* Pongo las fuerzas escalares en un vector */
      Fx = -(Fd + Fs) * (rX / largo);
      Fy = -(Fd + Fs) * (rY / largo);
      Fz = -(Fd + Fs) * (rZ / largo);

      //Actualizo con la fuerza que acabo de calcular
      //El primer punto
      res[i].a->fx += Fx;
      res[i].a->fy += Fy;
```

```
      res[i].a->fz += Fz;

      //El segundo punto
      res[i].b->fx -= Fx;
      res[i].b->fy -= Fy;
      res[i].b->fz -= Fz;
    }
 }
}
```

Como se puede ver en el código, ahora ciclamos por todo el arreglo de resortes, y con ellos podemos acceder a los dos puntos `a` y `b` que conectan. Lo primero es saber la distancia entre los dos puntos, si ésta fuera cero, no se hace ningún cálculo. Cabe señalar que por las condiciones del modelo es muy difícil que dicho caso suceda.

Después se procede a calcular la fuerza del amortiguador y del resorte, para cada una de ellas se ocupan sus respectivas constantes `Ks` y `Kd`, que son variables globales.

Por último se le da la dirección a la fuerza del resorte y se actualiza en los puntos, recordando cambiar el signo en el segundo punto. Hay también que hacer énfasis en que la fuerza nunca es actualizada directamente sino más bien es acumulada (sumada a la fuerza anterior).

3.2.3. La fuerza del gas

La siguiente fuerza en ser acumulada es la fuerza debida a la presión del gas. Para esto se ocupa la ecuación 1.25, y se debe de acumular una vez por cada cara. Para poder calcular esta fuerza se necesitan hacer varias operaciones importantes: calcular el volumen total del cuerpo flexible y además calcular el área y el vector normal de cada una de las caras.

Para hacer el cálculo del volumen se hace uso de un arreglo de puntos auxiliares, se simula que el cuerpo está formado por varios pedazos rectangulares. Para cada parte (hexaedro regular) se ocupa la fórmula 2.2 y finalmente la suma del volumen de todas nos proporciona el volumen total del cuerpo.

Para calcular el área de cada cara se ocupa la fórmula 2.1.

Para calcular el vector normal se hace uso de la fórmula 2.3, en donde $n = 4$, dado que cada cara está formada por cuatro puntos.

```
void putPressure (Punto pts[PARTICULAS], Cara msc[CUADROS])
```

```
{
  int i;
  float volumen = 0.0f, area, fuerza, largo, Fx, Fy, Fz;
  float vec1[3], vec2[3], vec3[3], vec4[3], vecPro[3];
  Punto aux[PARTICULAS];

  /* Puntos auxiliaes que me ayudaran a calcular el volumen */
  for (i = 0; i < PARTICULAS; i++)
  {
    aux[i].x = pts[i].x;
    aux[i].y = FONDO;
    aux[i].z = pts[i].z;
  }

  /** Calculo el volumen del cuerpo */
  for (i = 0; i < (LADO * (LADO - 1)); i++)
  {
    if ( (i % LADO) != (LADO - 1) )
    {
      volumen += volumenHexaedro(pts[i], pts[i+1], pts[i+LADO],
      pts[i+1+LADO], aux[i], aux[i+1], aux[i+LADO], aux[i+1+LADO]);
    }
  }

  /** Loop sobre todas las caras para calcular su fuerza de presion */
  for (i = 0; i < CUADROS; i++)
  {
    /* El area de esta cara */
    area = areaCuadrilatero(*msc[i].a, *msc[i].b, *msc[i].c, *msc[i].d);

    fuerza = area * Kp / volumen;

    /* Calculo el vector normal a la cara */
    vectorNormal(*msc[i].a, *msc[i].c, *msc[i].b, vec1);
    vectorNormal(*msc[i].b, *msc[i].a, *msc[i].d, vec2);
    vectorNormal(*msc[i].c, *msc[i].d, *msc[i].a, vec3);
    vectorNormal(*msc[i].d, *msc[i].b, *msc[i].c, vec4);
```

```
    vecPro[0] = (vec1[0] + vec2[0] + vec3[0] + vec4[0]) / 4.0f;
    vecPro[1] = (vec1[1] + vec2[1] + vec3[1] + vec4[1]) / 4.0f;
    vecPro[2] = (vec1[2] + vec2[2] + vec3[2] + vec4[2]) / 4.0f;

    /* Hago unitario al vector normal */
    largo = sqrt(vecPro[0] * vecPro[0] +
    vecPro[1] * vecPro[1] + vecPro[2] * vecPro[2]);

    msc[i].nx = vecPro[0] / largo;
    msc[i].ny = vecPro[1] / largo;
    msc[i].nz = vecPro[2] / largo;

    /* Calculo los vectores de fuerza */
    Fx = msc[i].nx * fuerza;
    Fy = msc[i].ny * fuerza;
    Fz = msc[i].nz * fuerza;

    /* Acomulo la fuerza de presion en cada particula de la cara */
    msc[i].a->fx += Fx;
    msc[i].a->fy += Fy;
    msc[i].a->fz += Fz;

    msc[i].b->fx += Fx;
    msc[i].b->fy += Fy;
    msc[i].b->fz += Fz;

    msc[i].c->fx += Fx;
    msc[i].c->fy += Fy;
    msc[i].c->fz += Fz;

    msc[i].d->fx += Fx;
    msc[i].d->fy += Fy;
    msc[i].d->fz += Fz;
  }
}
```

Para calcular la fuerza de nuevo se hace uso de una variable global `Kp`, que guarda la constante de la presión. Y finalmente la fuerza se acumula en cada uno de los cuatro puntos que forman la cara.

También hay que notar que el vector normal a ésta se quedó guardado en `nx`, `ny` y `nz`; después se puede hacer uso de él para hacer la iluminación de la escena.

La funciones `volumenHexaedro` y `areaCuadrilatero`, sólo hacen uso de las fórmulas 2.2 y 2.1, respectivamente, y devuelven un valor real.

3.3. Los métodos numéricos

Otra de las rutinas mas complicadas son los métodos numéricos. Para integrar la ecuación de Newton, se requiere saber la posición y la velocidad actual de cada una de las partículas del modelo y tener una manera de llamar a la función que se encarga de acumular las fuerzas.

3.3.1. El método de Euler

Básicamente se trata de tomar las ecuaciones 2.9 e implementarlas en el código, aprovechando la gran ventaja de que el método de Euler puede integrar un punto a la vez. La función que se explica recibe de parámetro una referencia a un objeto de tipo `Punto`.

```
void eulerIntegrator (Punto *p)
{
  float drx, dry, drz;

  /* Para las x */
  p->vx += p->fx / M * DT;
  drx = p->vx * DT;
  /* Para las y */
  p->vy += p->fy / M * DT;
  dry = p->vy * DT;
  /* Para las z */
  p->vz += p->fz / M * DT;
  drz = p->vz * DT;


  if (!p->fixed)
  {
    /* Ahora si los tengo que mover */
```

```
    p->x += drx;
    p->y += dry;
    p->z += drz;
  }
  else
  {
    p->vx = 0.0;
    p->vy = 0.0;
    p->vz = 0.0;

    p->fx = 0.0;
    p->fy = 0.0;
    p->fz = 0.0;
  }
}
```

Como puede apreciarse, esta función es muy sencilla, calcula con la fórmula 2.9 y guarda esos valores en variables. Luego pregunta si el punto que le mandaron por referencia tiene el estado `fixed`, de ser así, borra las fuerzas y las velocidades; en caso contrario, simplemente le aumenta el avance en este paso de tiempo. `DT` es una constante real que guarda el tamaño del paso h.

3.3.2. El método de Runge-Kutta

Aquí se explica el que fue probablemente el método mas complicado de implementar, pues no tiene la ventaja de Euler de integrar cada partícula independientemente, así que necesita integrar todas juntas. Además, debe tener espacio para guardar los ponderadores del paso de integración de cada partícula.

Para tener una idea clara de lo que que hace el siguiente código conviene volver a ver las ecuaciones 2.11 y 2.10.

```
void rungeKuttaIntegrator (void)
{
  int i;

  float drx, dry, drz, dvx, dvy, dvz,
  k1[3][PARTICULAS], k2[3][PARTICULAS], k3[3][PARTICULAS], k4[3][PARTICULAS],
```

```
  l1[3][PARTICULAS], l2[3][PARTICULAS], l3[3][PARTICULAS], l4[3][PARTICULAS];
  /*La segunda dimension es el numero de puntos en la tela */

  Punto auxPun[PARTICULAS];
  Resorte auxRes[RESORTES];
  Cara auxCar[CUADROS];

  /*Inicializa los resortes auxiliares, para que apunten a
    los puntos auxiliares y podamos llamar a la funcion
    acumulate forces sobre ellos */
  ponResortes(auxPun, auxRes);
  ponCaras(auxPun, auxCar);

  /* Calculo ponderadores de primer orden K1 y L1 */
  for (i = 0; i < PARTICULAS; i++)
  {
     k1[0][i] = tela[i].fx / M;
     l1[0][i] = tela[i].vx;

     k1[1][i] = tela[i].fy / M;
     l1[1][i] = tela[i].vy;

     k1[2][i] = tela[i].fz / M;
     l1[2][i] = tela[i].vz;
  }
//=====================================================================
  /* Copio en los auxPun para poder estimar una fuerza media */
  for (i = 0; i < PARTICULAS; i++)
  {
     auxPun[i].x  = tela[i].x  + (DT / 2.0f) * l1[0][i];
     auxPun[i].y  = tela[i].y  + (DT / 2.0f) * l1[1][i];
     auxPun[i].z  = tela[i].z  + (DT / 2.0f) * l1[2][i];

     auxPun[i].vx = tela[i].vx + (DT / 2.0f) * k1[0][i];
     auxPun[i].vy = tela[i].vy + (DT / 2.0f) * k1[1][i];
     auxPun[i].vz = tela[i].vz + (DT / 2.0f) * k1[2][i];
  }
```

```
  acomulateForces(auxPun, auxRes, auxCar);

  /* Calculo ponderadores de segundo orden K2 y L2 */
  for (i = 0; i < PARTICULAS; i++)
  {
     k2[0][i] = auxPun[i].fx / M;
     l2[0][i] = auxPun[i].vx + (DT / 2.0f) * k1[0][i];

     k2[1][i] = auxPun[i].fy / M;
     l2[1][i] = auxPun[i].vy + (DT / 2.0f) * k1[1][i];

     k2[2][i] = auxPun[i].fz / M;
     l2[2][i] = auxPun[i].vz + (DT / 2.0f) * k1[2][i];
  }

//=============================================================
  /* Copio en los auxPun para poder estimar una fuerza media */
  for (i = 0; i < PARTICULAS; i++)
  {
     auxPun[i].x  = tela[i].x  + (DT / 2.0f) * l2[0][i];
     auxPun[i].y  = tela[i].y  + (DT / 2.0f) * l2[1][i];
     auxPun[i].z  = tela[i].z  + (DT / 2.0f) * l2[2][i];

     auxPun[i].vx = tela[i].vx + (DT / 2.0f) * k2[0][i];
     auxPun[i].vy = tela[i].vy + (DT / 2.0f) * k2[1][i];
     auxPun[i].vz = tela[i].vz + (DT / 2.0f) * k2[2][i];
  }

  acomulateForces(auxPun, auxRes, auxCar);

  /* Calculo ponderadores de tercer orden K3 y L3 */
  for (i = 0; i < PARTICULAS; i++)
  {
     k3[0][i] = auxPun[i].fx / M;
     l3[0][i] = auxPun[i].vx + (DT / 2.0f) * k2[0][i];

     k3[1][i] = auxPun[i].fy / M;
     l3[1][i] = auxPun[i].vy + (DT / 2.0f) * k2[1][i];
```

```
      k3[2][i] = auxPun[i].fz / M;
      l3[2][i] = auxPun[i].vz + (DT / 2.0f) * k2[2][i];
   }

//============================================================
    /* Copio en los auxPun para poder estimar una fuerza media */
   for (i = 0; i < PARTICULAS; i++)
   {
      auxPun[i].x  = tela[i].x  + DT * l3[0][i];
      auxPun[i].y  = tela[i].y  + DT * l3[1][i];
      auxPun[i].z  = tela[i].z  + DT * l3[2][i];

      auxPun[i].vx = tela[i].vx + DT * k3[0][i];
      auxPun[i].vy = tela[i].vy + DT * k3[1][i];
      auxPun[i].vz = tela[i].vz + DT * k3[2][i];
   }

   acomulateForces(auxPun, auxRes, auxCar);

   /* Calculo ponderadores de cuarto orden K4 y L4 */
   for (i = 0; i < PARTICULAS; i++)
   {
      k4[0][i] = auxPun[i].fx / M;
      l4[0][i] = auxPun[i].vx + DT * k3[0][i];

      k4[1][i] = auxPun[i].fy / M;
      l4[1][i] = auxPun[i].vy + DT * k3[1][i];

      k4[2][i] = auxPun[i].fz / M;
      l4[2][i] = auxPun[i].vz + DT * k3[2][i];
   }
//============================================================================

   for(i = 0; i < PARTICULAS; i++)
   {
      if (!tela[i].fixed)//if que checa que el punto no este fijo
      {
```

```
        /* Calculo los incrementos */
        dvx = (DT/6.0f)*(k1[0][i]+2.0f*(k2[0][i]+k3[0][i])+k4[0][i]);
        drx = (DT/6.0f)*(l1[0][i]+2.0f*(l2[0][i]+l3[0][i])+l4[0][i]);

        dvy = (DT/6.0f)*(k1[1][i]+2.0f*(k2[1][i]+k3[1][i])+k4[1][i]);
        dry = (DT/6.0f)*(l1[1][i]+2.0f*(l2[1][i]+l3[1][i])+l4[1][i]);

        dvz = (DT/6.0f)*(k1[2][i]+2.0f*(k2[2][i]+k3[2][i])+k4[2][i]);
        drz = (DT/6.0f)*(l1[2][i]+2.0f*(l2[2][i]+l3[2][i])+l4[2][i]);

        /* Ahora si los tengo que mover */
        tela[i].x  += drx;
        tela[i].vx += dvx;

        tela[i].y  += dry;
        tela[i].vy += dvy;

        tela[i].z  += drz;
        tela[i].vz += dvz;
     }
     else //esta fijo velocidades y fuerzas a cero
     {
        tela[i].vx = 0.0f;
        tela[i].vy = 0.0f;
        tela[i].vz = 0.0f;

        tela[i].fx = 0.0f;
        tela[i].fy = 0.0f;
        tela[i].fz = 0.0f;
     }
   }
}
```

El truco consiste en ocupar un conjunto de puntos, resortes y caras auxiliares, para con ellos calcular los ponderadores del método de RK4, después la rutina es muy parecida a la de Euler.

3.4. El manejo de las colisiones

Como ya se ha dicho antes, el problema de las colisiones se resuelve en dos partes, primero la detección y luego la respuesta. La forma de responder consiste básicamente en mover los objetos que se colisionan a un lugar donde ya no choquen y ajustar las velocidades como respuesta.

3.4.1. La rutina de las colisiones

La detección es llevada a cabo en dos funciones. Recordemos que nuestra tarea de detección se simplifica muchísimo por el hecho de que uno de los objetos, el objeto incidente es una esfera. Para saber si dicha esfera está en colisión con nuestro cuerpo flexible, lo que hacemos es probar si cualquiera de las partículas está dentro de la esfera; de ser así, empezamos a resolver la colisión entre la esfera y la partícula en cuestión. Después seguimos revisando el resto de las partículas.

El algoritmo tiene la siguiente forma:

```
void colisiones (void)
{
  int i;
  Vector n;

  for (i = 0; i < PARTICULAS; i++)
  {
    if (dentro(tela[i]))
    {
      n = detectaColision(&tela[i]);
      respondeColision(&tela[i], n);
    }
  }
}
```

Recordemos que `tela`, es un arreglo global que contiene todos los puntos, se hace un ciclo por todo este arreglo preguntando con la función `dentro` si está adentro de la esfera; de ser así, se manda llamar la funcion que detecta la colisión, es decir calcula el vector normal y lo devuelve. Luego ese vector normal `n` y el punto son mandados a la función que resuelve la colisión. Y así sucesivamente con todas las partículas que forman parte del cuerpo flexible.

La función `dentro`, sabe si hay una colisión, pues la esfera está guardada en una variable global y el punto se le pasa como parámetro, así que sólo regresa verdadero cuando la distancia del punto al centro de la esfera es menos que el radio de ésta y falso en cualquier otro caso.

3.4.2. La detección de la colisión

Esta función se encarga de calcular el vector normal al lugar de la colisión y de mover el punto fuera de la esfera. Mover el punto fuera de la esfera es en realidad parte de la respuesta a la colisión, pero decidí implementarlo en esta función, para que en la siguiente parte sólo se tenga que ajustar las velocidades.

```
Vector detectaColision(Punto *p)
{
  float vecV[3], distancia;
  Vector resultado, V;
  /* Obtenemos vector V que va del centro de la esfera al punto */
  V.x = p->x - pelota.x;
  V.y = p->y - pelota.y;
  V.z = p->z - pelota.z;
  /* Normalizamos V y lo guardamos en n,
     N me sirve para hacer varios caluclos */
  distancia = sqrt((V.x * V.x) + (V.y * V.y) + (V.z * V.z));
  resultado.x = V.x / distancia;
  resultado.y = V.y / distancia;
  resultado.z = V.z / distancia;
  /* Movemos al punto fuera de la esfera, esto en realidad es parte
   * de la respuesta, pero lo pongo aqui, para que en la respuesta
   * solo ajuste las velocidades */
  if (!p->fixed)
  {
    p->x = pelota.x + (pelota.radio * resultado.x);
    p->y = pelota.y + (pelota.radio * resultado.y);
    p->z = pelota.z + (pelota.radio * resultado.z);
  }
  else
  {
    pelota.x = p->x - (pelota.radio * resultado.x);
```

```
    pelota.y = p->y - (pelota.radio * resultado.y);
    pelota.z = p->z - (pelota.radio * resultado.z);
  }
  return resultado;
}
```

La función primero calcula el vector $\vec{V}$ que va del centro de la esfera al punto, después lo normaliza. Luego pregunta si el punto no está fijo, en cuyo caso, mueve el punto justo afuera de la esfera, cambiándolo de posición al punto $\vec{p} + (r\vec{V})$, donde $\vec{p}$ es la posición de la esfera, r su radio y $\vec{V}$ ya es unitario. Si el punto sí estuviera fijo, lo que hace es mover a la esfera fuera del punto cambiándola de posición al punto $\vec{s} - (r\vec{V})$, donde $\vec{s}$ ahora representa la posición de la partícula..

3.4.3. La respuesta de la colisión

La mayor parte del trabajo se hizo en la función anterior, ahora que sabemos el vector normal $\vec{n}$ a la colisión, sólo nos resta seguir los pasos explicados en la página 56.

```
void respondeColision (Punto *p, Vector normal)
{
  float r;
  Vector U, Un, Ut, Sn, Wn;

  r = Mp / M;

  U.x = pelota.Vx - p->vx;width=7cm
  U.y = pelota.Vy - p->vy;
  U.z = pelota.Vz - p->vz;

  Un = parteNormal(U, normal);

  Ut.x = U.x - Un.x;
  Ut.y = U.y - Un.y;
  Ut.z = U.z - Un.z;

  Sn.x = Un.x * ( (r - 1.0f) / (r + 1.0f) );
  Sn.y = Un.y * ( (r - 1.0f) / (r + 1.0f) );
  Sn.z = Un.z * ( (r - 1.0f) / (r + 1.0f) );
```

```
  Wn.x = Un.x * ( (2.0f * r) / (r + 1.0f) );
  Wn.y = Un.y * ( (2.0f * r) / (r + 1.0f) );
  Wn.z = Un.z * ( (2.0f * r) / (r + 1.0f) );

  pelota.Vx = Ut.x + Sn.x + p->vx;
  pelota.Vy = Ut.y + Sn.y + p->vy;
  pelota.Vz = Ut.z + Sn.z + p->vz;

  p->vx = Wn.x + p->vx;
  p->vy = Wn.y + p->vy;
  p->vz = Wn.z + p->vz;
}
```

En donde las constantes `Mp` y `M` tiene las masas de la esfera y de la partícula, respectivamente.

Capítulo 4

Diseño del experimento

En este capítulo se analizan los resultados que se obtuvieron al haber implementado el modelo, y se divide en tres partes. En la primera de ellas se cuenta cómo es que se decidió implementar el modelo, así como también cuales de sus variables se dejaron fijas (serían constantes en esta implementación) y cuáles es posible variar desde la interfaz de usuario.

En la segunda parte se diseñaron ciertas pruebas con el fin de poner en evidencia características particulares del modelo, específicamente su correcto comportamiento físico.

Y en la última parte se hicieron dos tipos de pruebas para medir el desempeño del programa, variando la forma de ejecución y utilizando diferentes ambientes de ejecución.

4.1. Definición del sistema

En esta primera parte presento el experimento y explico qué parámetros fijé y cuáles pueden ser cambiados por el usuario.

4.1.1. Características del modelo

Tal como se explicó en la seccion 2.1, se modela un hexaedro regular, donde cinco de sus seis caras son rígidas y la cara superior o tapa es flexible: además, asumo que este hexaedro está relleno de gas. A la parte superior de la caja se le deja caer una esfera rígida. Un ejemplo del programa terminado se muestra en la figura 4.1.

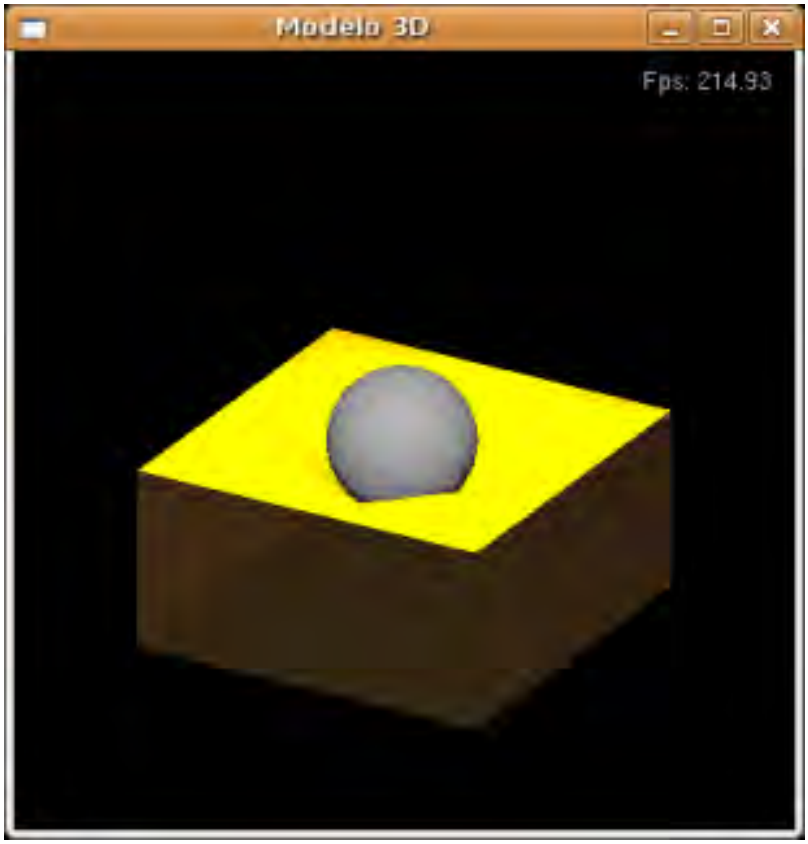

Figura 4.1: Un ejemplo del modelo.

Constantes del experimento

En una situación como la antes descrita hay muchísimas variables del modelo, sin embargo al momento de hacer la implementación en código decidí dejar fijas algunas de ellas, es decir que la única manera de cambiarlas es modificando en el código fuente y recompilando el programa por completo. Aquí está la lista de estas variables y su significado.

Primero las variables del número de partículas que conforman el cuerpo flexible.

LADO: Número de partículas por lado del cuerpo flexible, y es la única variable independiente.

PARTICULAS: Número total de partículas que forman el cuerpo flexible. Utilizamos n^2, en donde n es LADO.

RESORTES: Número de resortes que conectan el cuerpo flexible, es $2n(n-1)$.

CUADROS: Número total de caras que forman el cuerpo flexible, es igual a $(n-1)^2$

Las variables que sirven para hacer la proyección ortogonal y poner el tope de las coordenadas del mundo son las siguientes:

MUNDO: Es el tope de las coordenadas del mundo.

FONDO: Es el lugar donde se sitúa sobre el eje XZ, el fondo de la caja que forma el cuerpo flexible. Además de servir para otros cálculos dentro del programa, la única restricción es que su valor absoluto sea menor que MUNDO.

Una variable para integrar la ecuación de Newton

DT: Es el tamaño del paso en el tiempo por cada vez que se integra la ecuación de Newton. Es decir es lo que en la parte de métodos numéricos llamé Δt.

M: Una única constante física, la masa de cada una de las partículas.

Los valores de estas constantes con las que se hizo este experimento están en la tabla 4.1.

Cuadro 4.1: Valor de las constantes del experimento

Constante	Valor	Comentario
LADO	26	Para que se vea mejor el modelo, se elige un número par
PARTICULAS	676	
RESORTES	1300	
CUADROS	625	
MUNDO	100	La proyección va de $-MUNDO$ a $MUNDO$ en cada eje
FONDO	-50.0	
DT	0.01	
M	0.009	El peso total del cuerpo flexible es $M \cdot PARTICULAS$

Variables del experimento

La variables del experimento pueden ser modificadas en tiempo de ejecución por medio del menú que muestra en la figura 4.2.

La primer categoría, *Esfera*, se refiere al cuerpo rígido. En esta caso la esfera y la única variable que se puede modificar es su masa.

La categoría de *Integrador*, se refiere el método numérico usado para integrar la ecuación de Newton, sólo se implementaron dos métodos el de Euler y el de Runge Kutta. Dado que cambiar de método a mitad de la simulación generalmente lleva a inestabilidad numérica, cada que se cambia el método, todos los objetos regresan a sus posiciones iniciales.

La categoría de variables físicas se refiere a los parámetros del cuerpo flexible que es posible cambiar. Se dividen en tres subcategorias una por cada fuerza que se acumula.

La presión del *gas*, se puede prender o apagar por medio del checkbox. Además, se puede modular su magnitud variando el valor de la constante k_g, por medio del control.

La de la *gravedad*, que al igual que la anterior se puede prender o apagar con el control y también se puede modular variando el valor de la constante g.

Por último la debida al *resorte amortiguador*. Esta fuerza no se puede apagar, pero se puede variar por medio de la modificación de dos parámetros k_s, que, como ya se dijo, controla la rigidez del resorte y k_d que controla el amortiguamiento o pérdida de energía debida al resorte.

Opciones de visualización

Hay otras opciones que se pueden modificar por medio del menú, que se refieren más a cómo se ve el modelo que a la física. Éstas se dividen en dos, las que controlan el render y las que controlan el flujo de la animación

Dentro de las del flujo de la animación, sólo hay dos controles, el de *Iniciar/Pausar*, que mantiene la escena congelada cuando está marcado y el botón de *Reiniciar*, que devuelve a todos los objetos a su posición original.

Dentro de la categoría de render, se encuentran los siguientes controles:

El *wireframe*, si el checkbox está activado, sólo se pintarán las líneas del modelo, así se pueden apreciar todas las partículas que lo componen.

La *luz*, dado que se utiliza una proyección ortogonal. La única fuente de profundidad viene de la luz y las sombras, Y el modelo contiene una única fuente de luz, localizada en $(0, k, 2k)$ en donde $k = MUNDO$. Con este control se puede prender o apagar esta fuente de luz.

La *caja* que dice si se va a pintar o no las caras sólidas de la caja. Las caras sólidas se pintan con un cierto valor de transparencia, para que se aprecie mejor cómo se deforma la parte de abajo de la tela o cuerpo flexible.

Por último un par de controles que manejan el ángulo de rotación del modelo. Sólo se permite rotar sobre el eje X o sobre el eje Y.

Los valores de default de estas variables (el valor que tiene al iniciar la ejecución) son los que muestra la tabla 4.2.

4.1.2. Características del entorno de pruebas

Software

El programa fue hecho en su mayor parte en lenguaje C, el compilador elegido fue gcc por tratarse de un compilador libre y por ser el que se instala en la mayoría de las distribuciones de GNU/Linux (la gran mayoría de este trabajo fue desarrollado bajo este sistema operativo).

Sin embargo, al crear el menú de usuario fue necesario utilizar la biblioteca glui, que está escrita en objetos, por lo que al final se tuvo que compilar en C++, con el compilador g++.

Figura 4.2: Menú de usuario.

Cuadro 4.2: Valor inicial de los parámetros del experimento

Parametro	Valor	Comentario
Masa	0.08	Masa del cuerpo rígido
Integrador	Runge Kutta	Método Numérico con el que se integra.
Gas	Activado	El programa empieza con la fuerza del gas prendida
k_g	780	Constante de presión del gas
Gravedad	Activada	La gravedad está activada
g	-10.0	La gravedad es negativa (jala hacia abajo)
k_s	1.0	La fuerza de los resortes
k_d	0.4	El valor de damping

Para poder compilar el código fuente de este programa es necesario tener un entorno de programación que contemple lo siguiente:

- Un compilador de C++
- La biblioteca OpenGL
- La biblioteca glut
- La biblioteca glui

Todas éstas tienen alternativas libres, y además todas tienen la enorme ventaja de tener equivalentes en cualquier plataforma, por lo que si las bibliotecas están correctamente instaladas, debería de compilar el programa bajo cualquier sistema operativo.

Se ha hecho la prueba en los entornos GNU/Linux y Windows XP (en este último con el IDE Dev-C++, con un compilador Mingw).

4.1.3. Hardware

La mayoría de las pruebas fueron hechas en una estación de trabajo Sun Ultra 20, con las siguientes características.

- Procesador: Opteron 1214 2.2GHz

- Memoria: 2GB DDR2
- Tarjeta de Video: Nvidia FX3500
- Acerleración gráfica: driver de nVidia
- Sistema Operativo: Ubuntu 7.04 64bits (kernell 2.6.20-16)

Esto no quiere decir que este sea el hardware mínimo, sólo que la *mayoría* de las pruebas se realiaron en este hardware. Sin embargo, se ha ejecutado con éxito en equipos mucho más convencionales (las características de estos equipos así como su desempeño al ejecutar el programa se muestran más a detalle en la última sección de este capítulo).

4.2. Ejemplos de las características físicas del modelo

Para poner en evidencia ciertas características del modelo, se hicieron las siguientes pruebas sugeridas, la mayoría para probar la sensibilidad del programa ante la variacion de sus parámetros físicos.

4.2.1. Probando la gravedad

La gravedad es la única fuerza que actúa tanto en el cuerpo flexible como en en el cuerpo rígido. Para entender mejor cómo afecta se sugieren las siguientes pruebas.

Partiendo de los valores de *default*, se espera a que la tela se estabilice (figura 4.1). Ahora se *aumenta* la gravedad a su valor más *pequeño* (recordemos que hacer la gravedad más fuerte es hacerla más negativa), es decir, g =-12.0, se observa ahora como la gravedad es muy fuerte como para que la presión del gas infle la tela, por lo que queda colgando un poco. Las partículas que forman el cuerpo flexible son muy pesadas (son jaladas con más fuerza hacia abajo). Esta situación se muestra en la figura 4.3. Ahora se deja caer la pelota y se espera que se estabilice de nuevo el programa, la gravedad hace que la pelota y las partículas pesen más, sin embargo la fuerza del gas que no se ha tocado compensa de alguna manera y no deja que se hunda más la pelota. Con la pelota estable sobre la tela, apagué la fuerza del gas. Al apagar la fuerza que equilibraba la gravedad todo se va hacia abajo, por la gravedad, pero como además esta fuerza es grande, la rigidez del resorte es poca para evitar un efecto de súper elongación como el que se ve en la figura 4.4.

Figura 4.3: La gravedad aumenta a g =-12.0.

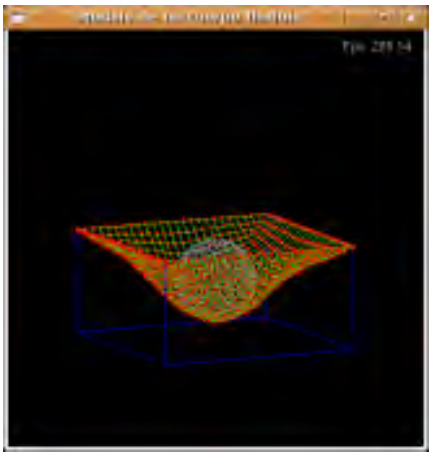

Figura 4.4: La gravedad es grande y se apaga el gas.

Figura 4.5: La gravedad es apagada.

Reinicie la animación y vuelva a los valores de *default*. De nuevo espere a que se estabilice la tela ahora disminuya la gravedad a su máximo valor posible, es decir una gravedad positiva: g =1.0. Ahora verá que la tela se va más hacia arriba, como si se inflara más el cuerpo flexible, esto se debe a que ahora la gravedad no se opone a la presión del gas, sino mas bien le favorece, por lo que el cuerpo flexible, es jalado hacia arriba aún más, como se ve en la figura 4.5. Ahora deje caer la pelota, como la gravedad es positiva y pequeña (su valor absoluto en una décima parte del valor normal) la pelota *sube lentamente*, y se aleja del cuerpo flexible. La constante de gravedad influencía la velocidad de caída de los cuerpos.

Por último vamos a reiniciar la animación y partiendo de los valores de *default* de los parámetros, se apaga la gravedad. Este comportamiento equivale a hacer $g = 0$ con la diferencia de que se hacen menos cálculos, ahora vemos cómo de nuevo la tela se mueve hacia arriba, cómo la gravedad se oponía a la fuerza del gas y ya no está, el gas empuja la tela aún más hacia arriba. Si en esta situación se deja caer la pelota no pasará nada, debido a que la caída de la pelota *depende de la gravedad*, y al no haber, simplemente no hay caída.

Si se apaga la gravedad después de dejar caer la esfera, ésta es empujada afuera del cuerpo flexible por un impulso.

4.2.2. Probando la fuerza del gas

Ahora se harán pruebas sobre la fuerza del gas. La fuerza del gas, por el diseño de nuestro experimento, se opone a la fuerza de gravedad, y actúa sólo sobre el cuerpo flexible, sin embargo tiene

Figura 4.6: Fuerza del gas apagada.

cierto efecto sobre la velocidad de las partículas del cuerpo flexible, que a su vez tienen cierto efecto sobre el cuerpo *rígido* al momento de la colisión.

Inicie el programa con los valores de *default*, ahora apague la fuerza del gas y espere a que se estabilice el modelo, como se ve el la figura 4.6. Como no hay fuerza del gas, la tela o cuerpo flexible cuelga agarrada de las orillas de la caja. Ahora deje caer la pelota sobre el cuerpo flexible y espere a que se estabilice la animación, justo como se ve en la figura 4.7. Prenda la fuerza del gas y observe como la pelota es lanzada hacia arriba súbitamente como se ve en la figura 4.8.

Otra prueba es ver los efectos de la variación de la constante k_g. Inicie la animación con los valores de *default* y haga la constante k_g pequeña, por ejemplo $k_g = 350.0$; verá cómo el gas no es lo suficientemente fuerte para inflar el cuerpo flexible. Ahora aumente poco a poco la constante k_g y observe cómo se infla cada vez más el cuerpo flexible. En la figura 4.9 se ilustran estas situaciones para diferentes valores en aumento de la constante k_g.

4.2.3. Probando los resortes amortiguadores

La siguientes pruebas sirven para mostrar cómo funcionan los resortes amortiguadores. Primero vamos a hacer una prueba con la constante de *damping* k_d. Como ya se dijo, el damping es una forma de perder energía del sistema y, por lo tanto, de eventualmente estabilizarse. Iniciemos la animación con los valores de *default* y hagamos la constante $k_d = 0$, es decir quitemos todo el *damping*. Vemos que la tela empieza a oscilar rápidamente como consecuencia del gas. Ahora quitemos también el gas y esperemos un momento; veremos como la tela oscila sin detenerse ni estabilizarse en ningún

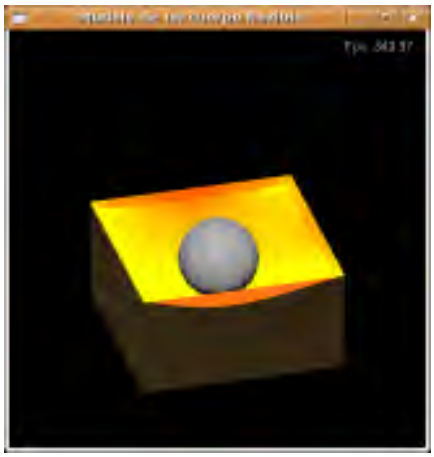

Figura 4.7: La pelota cae mientras está apagado el gas.

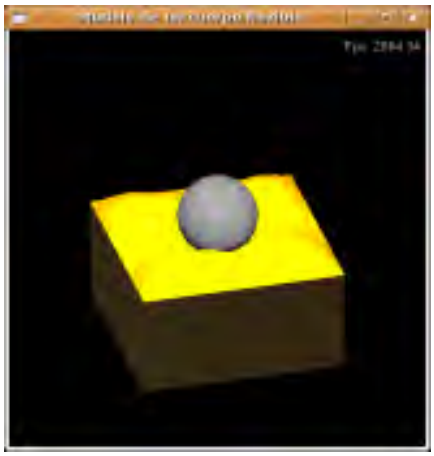

Figura 4.8: La pelota es lanzada por la fuerza del gas.

Figura 4.9: Variando la fuerza del gas: k_g =350.0 (arriba), k_g =730.0 (centro), k_g =970.0 (abajo)

momento, es cierto que cada vez oscila menos, pero ciertamente tardará mucho en detenerse (o nunca se detendrá).

En la figura 4.10, podemos ver diferentes oscilaciones sin control por falta de un amortiguador. Como ya se había dicho, el amortiguador agrega realismo (en la total ausencia de *damping*, el cuerpo flexible se ve poco real). La contra parte es que a mayor *damping*, también hay menos estabilidad numérica, por lo que el modelo es sumamente sensible al aumento en este parámetro.

Podemos ir aumentando el valor de k_d poco a poco y ver cómo el modelo se vuelve inestable, podemos por ejemplo poner el máximo posible k_d =0.6 y reiniciar la animación, esperar a que se estabilice y apagar el gas, el modelo explota (ver figura 4.11).

Ahora analicemos la constante de rigidez k_s. Esta constante hace a los resortes más poderosos, por lo que en general hace que las partículas que están unidas por ellos, se separen menos, es decir hace más estable el cuerpo flexible en general, además de prevenir el efecto de super elongación.

Iniciamos la animación con los valores de default y pongamos la constante del resorte k_s a su máximo valor. Ahora apaguemos el gas y vemos como la caída de la tela es menor, sin embargo al apagar y prender varias veces el gas también nos damos cuenta de que el cuerpo flexible parece oscilar más, como consecuencia de que los resortes en general jalan más fuerte, tanto hacia arriba como hacia abajo.

Ahora con la constante k_s en el máximo y el gas prendido, vayamos poco a poco subiendo la presión del gas, aumentando k_g hasta su máximo k_g =2000.0. Podemos ver cómo aun llegando al máximo de k_g, el cuerpo flexible se expande poco; esperemos a que se estabilice, como se ve en la figura 4.12. Ahora que tenemos el modelo estable y con ambas constantes k_s y k_g al máximo, poco a poco vayamos haciendo k_s más pequeña y podremos apreciar como el cuerpo parece inflarse más (la resistencia de la membrana que lo mantiene unido es menor), como se ve en la figura 4.13, hasta llegar al momento donde explota ($k_s = 0.0$).

4.3. Presentación de resultados

Una manera de medir la eficiencia tanto de la implementación como del modelo es ver cuánto se tarda en hacer los cálculos el programa antes de pintar un frame. Esta medida de rendimiento es calculada comúnmente en las simulaciones gráficas y por ello me decidí a implementarla. Como ya se dijo, esto es dependiente del hardware, así que aquí presento dos tipos de análisis.

En la primera parte, se deja como constante el hardware, hago las pruebas en la misma computadora y se varían los cálculos que se hacen en la ejecución del programa.

Figura 4.10: Oscilación sin control: k_d=0.0

Figura 4.11: La animación explota por inestabilidad numérica

Figura 4.12: El gas está al máximo k_g =2000.0 al igual que la rigidez de los resortes k_s =2.0

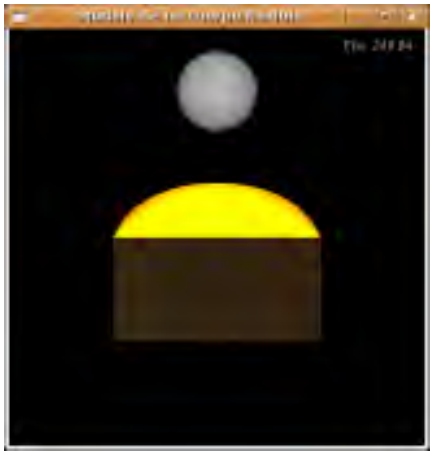

Figura 4.13: El cuerpo flexible se infla como consecuencia del gas la máximo y k_s pequeña

En la segunda parte se mantienen constantes los cálculos en el programa y se ejecuta en varios ambientes de pruebas (diferentes máquinas y distintos sistemas operativos).

4.3.1. Desempeño del programa

La medida mas comúnmente usada es el número de cuadros que la aplicación pinta en un segundo o fps (de abreviar en inglés *frames per second*). La rutina que calcula los fps se implementó con el siguiente código dentro de la función con el registro de callback idle.

```
frame++;
time = glutGet(GLUT_ELAPSED_TIME);
if (time - timebase > 1000) {
  fps = frame * 1000.0 / (time - timebase);
  timebase = time;
  frame = 0;
}
```

En donde la variable global `frame`, ya fue inicializada la primera vez en ceros. Esta rutina mide los frames por segundo, que después son desplegados en la parte superior izquierda de la pantalla.

Estas pruebas fueron realizadas en la máquina ya descrita antes en la sección 4.1.3.

Las variantes consideradas son las siguientes:

Estado de la animación: la animación puede estar en pausa o en movimiento, cuando está en pausa, se pueden cambiar la opciones de render y de la rotación de la cámara, pero el programa deja de hacer los cálculos del método numérico y por ende de la acumulación de fuerzas.

Hay colisiones: la rutina de detección de colisiones siempre es ejecutada, sin embargo sólo en caso de que se detecte una colisión se llama a la rutina de respuesta de las colisiones.

Método Numérico: el programa ejecuta uno de dos métodos numéricos para calcular el estado siguiente de la animación, puede ser por Euler o por Runge Kutta, de donde este último requiere de casi cuatro veces mas cálculos.

Fuerza del gas: calcular la fuerza del gas requiere de hacer cálculos sobre el área y el volumen del cuerpo flexible, cuando esta fuerza está apagada, los cálculos se omiten.

Opciones de *render*: las opciones del *render* hacen que se tengan que hacer cálculos de la iluminación y dar color a los objetos con base a dichos cálculos.

La nomenclatura de la prueba usando estas variables se especifica en el cuadro 4.3.

Cada una de las variantes arriba mencionadas puede tener dos valores. El *render* puede tener muchos valores pero para hacer las pruebas sólo voy a considerar dos: prendido, cuando se dibujan todos los cuerpos en estado sólido con iluminación y apagado, cuando se dibuja en wireframe sin luz y no se dibuja la caja del cuerpo flexible.

Se tienen pues cinco posibles variantes, cada una con dos valores. Se ejecutaron los 18 casos posibles (la animacion apagada implica que haya opciones que no tengan sentido, por eso son sólo 18 en vez de 32) y para cada caso anoté entre qué valores oscilan los fps. Los resultados se resumen en la tabla 4.4.

La razon por la cual los fps, son un rango es que al momento de la ejecución, cada segundo varía el valor de los fps. Es por eso que al hacer el experimento con las condiciones de cada caso, anote tanto el valor máximo que alcanzaron lo fps, como el valor mínimo; esto es lo que llamo el rango.

La columna de contribución, representa cuánto contribuye a los cálculos la prueba en cuestión con respecto a que se hacen *todos* los cálculos, tomando como base el caso en que todas las variables están prendidas. La contribución C es calculada de la siguiente manera:

$$C = \frac{FPS_{min} + FPS_{max}}{2} \div FPS_{base}$$

En donde FPS_{min} es el mínimo de fps en cada categoría y FPS_{max} es el máximo de fps en la misma, FPS_{base} es el promedio de fps de la categoría base, para este caso $FPS_{base} = 210.81$ dado que

Cuadro 4.3: Nomenclatura de la prueba

Variable	Valores	Explicación
Animación	0 ó 1	Se considera 1 cuando la animación esta en marcha, y 0 cuando está en pausa.
Colisiones	0 ó 1	Se considera 1 cuando hay una colisión y se ejecuta la respuesta, 0 cuando no hay colisión y sólo se ejecuta la detección.
Método Numérico	0 ó 1	Se considera 1 cuando se ejecuta Runge Kutta y 0 cuando se ejecuta Euler.
Fuerza del gas	0 ó 1	Se considera 1 cuando está prendida y 0 cuando está apagada.
Render	0 ó 1	Se considera 1 cuando se hace la mayor cantidad render, como se explicó arriba y 0 cuando se hace el mínimo posible.

es el promedio de los FPS de la prueba que más poder requiere y se presenta en el último renglón de la tabla.

Hay algunos datos interesantes con respecto a la tabla 4.4, por ejemplo el hecho de lo poco que contribuye el render al desempeño de la animación (el .03). Esto se explica por que cuando el render está apagado se trazan más objetos gráficos, se dibujan cuatro lineas y cuatro puntos por cada cara, mientras que cuando el render está prendido si bien se hacen cálculos de iluminación sólo se dibuja un cuadro por cada cara.

También de la tabla, puedo concluir que lo que más contribuye al desempeño de la animación es el método numérico usado, pues como dijimos el método de RK4, lleva casi cuatro veces más cálculos que el método de Euler (en la tabla se ve que representa el .56 de los cálculos).

El otro factor que hace considerablemente más lenta la animación es el cálculo de la fuerza del gas con el .48 del tiempo.

4.3.2. Desempeño del programa en diferentes ambientes

La última prueba consistió en hacer constante la ejecución del programa y ver qué desempeño tiene en diferentes tipos de hardware.

Cuadro 4.4: Resultados primera prueba

Anim	Coli	Método	F. Gas	Render	FPS	Contr
0	n/a	n/a	n/a	0	2753.08 - 2768.15	0.08
0	n/a	n/a	n/a	1	2754.01 - 2826.18	0.08
1	0	0	0	0	654.35 - 663.34	0.32
1	0	0	0	1	620.38 - 630.37	0.34
1	0	0	1	0	501.04 - 512.98	0.42
1	0	0	1	1	464.03 - 491.02	0.44
1	0	1	0	0	447.99 - 456.54	0.47
1	0	1	0	1	454.09 - 474.05	0.45
1	0	1	1	0	240.52 - 251.99	0.86
1	0	1	1	1	203.57 - 212.57	1.01
1	1	0	0	0	640.72 - 660.54	0.32
1	1	0	0	1	612.76 - 627.74	0.34
1	1	0	1	0	499.03 - 512.03	0.42
1	1	0	1	1	470.06 - 484.52	0.44
1	1	1	0	0	347.31 - 357.29	0.60
1	1	1	0	1	333.67 - 342.32	0.62
1	1	1	1	0	214.34 - 219.26	0.97
1	1	1	1	1	209.04 - 212.57	1.00

Para esta prueba se ejecutó el programa en diferentes ambientes. En cada ambiente se consideran importantes sólo las siguientes características: la memoria principal, el sistema operativo, el procesador, la arquitectura y si hay o no memoria de vídeo.

La ejecución del programa se hizo de una manera típica, lo que haciendo una equivalencia con la prueba anterior es que esté en marcha la animación, que haya colisiones con el método de Runge Kutta, que haya fuerza del gas y que el *render* se haga completo. También se hizo la ejecución con las siguientes características mínimas: método de Euler y sin fuerza de gas. De ambas pruebas se miden los fps.

Los resultados se listan a continuación.

- Sistema Operativo: Ubuntu 7.04
- Procesador: Opteron 1214 a 2.2GHz
- Memoria RAM: 2.0GB
- Memoria de Video: Tarjeta Nvidia FX3500, 256MB, driver propietario
- Velocidad en prueba mínima: 612.76 - 627.74fps
- Velocidad en prueba normal: 209.04 - 212.57fps

- Sistema Operativo: Windows XP professional edition
- Procesador: AMD Duron a 1.2GHz
- Memoria RAM: 512MB
- Memoria de Video: Tarjeta Nvidia 64MB
- Velocidad en prueba mínima: 72.0 - 102.0fps
- Velocidad en prueba normal: 60.0 - 94.0fps

La última prueba se llevó a cabo en una máquina virtual, con el fin de evaluar los requerimientos mínimos.

- Sistema Operativo: Windows XP professional edition

- Procesador: Dual Core AMD Opteron 999Mhz
- Memoria RAM: 256MB
- Memoria de Video: Ninguna, adaptador VM-Ware SVG 2
- Velocidad en prueba mínima: 65.40 - 71.72fps
- Velocidad en prueba normal: 46.44 - 52.22 fps

Conclusiones

Después de haber hecho pruebas y evaluado los resultados obtenidos presento las siguientes conclusiones.

Lo más importante es que las ecuaciones diferenciales, permiten modelar gráficamente el comportamiento físico de un cuerpo neumático que interactúa con un cuerpo rigido, en tiempo real. También hay que recalcar que este modelo es tiene sustento en la física, es decir es un modelo basado en física y no en geometría.

Creo también que la técnica aquí usada, la de utilizar un gas ideal dentro del cuerpo flexible, es muy buena para enfrentar este tipo de problemas pues nos da todo lo que desearíamos de una animación: es computacionalmente barata, al menos lo suficiente para alcanzar tiempo real y es físicamente adecuada para modelar el gas.

El modelo como se describe depende poco del poder gráfico, es decir el render sigue siendo una operación relativamente barata *en comparación* con los cálculos numéricos. Dentro de los cálculos numéricos, lo que más contribuye es el método numérico de Runge Kutta seguido de la acumulación de la fuerza del gas.

Si bien el utilizar el método de Euler, hace considerablemente más rápido de ejecutarse el programa, recomiendo usar el método de Runge Kutta, por mayor estabilidad ante la variación de las constantes, lo que se traduce en la posibilidad de mayor interacción. También creo que este método no es tan complejo de implementar.

El *damping* es un parámetro que debe de estar presente, sin embargo requiere tener mucho cuidado al determinar un valor adecuado, considero que el valor más adecuado es el más grande posible sin que la animación explote. Este modelo es muy sensible a las variaciones aun *pequeñas* de este parámetro.

Se recomienda ampliamente que al momento de programar se tome tiempo para hacer una adecuada elección de las estructuras de datos. Guardar todo en una estructura de datos lineal, simplifica bastante la programación. De igual manera, recomiendo tener diferentes maneras de acceder a las

propiedades de las partículas, ya sea por los resortes o por medio de las caras.

El modelo del gas ideal es un modelo muy recomendado para simular cuerpo flexibles; con una adecuada elección de los parámetros se puede tener un comportamiento bastante realista. Por lo que considero que el objetivo fundamental de este trabajo se cumple.

Hay muchas mejoras posibles para esta implementación, pienso que hay campo para futuras investigaciones en los siguientes detalles:

- El cálculo del volumen del cuerpo flexible.
- Investigar sobre un método numérico más eficiente, aquel que dé más rapidez sin perder estabilidad.
- El manejo de colisiones más eficiente.

Para hacer el cálculo del volumen en [10] se propone hacer uso del *teorema de divergencia*. Considero que este podría ser un buen camino para hacer más flexible el cálculo del volumen y del área.

Para el método numérico se hicieron pruebas con el integrador de Verlet, sin embargo abandoné ese camino porque en un esquema como este no se toma en cuenta la velocidad de la partícula, por lo que no encontré manera de responder a las colisiones. Creo que investigar la manera de incluir una fuerza de impulso como respuesta a la colisión y utilizar el integrador de Verlet daría resultado.

Hay también que considerar una respuesta a las colisiones que conlleve una pérdida de energía al momento de la colisión. Es decir, eliminar el supuesto de colisiones perfectamente elásticas. Aunque escribí en el segundo capítulo sobre ellas, no fueron implementadas en el programa.

Mi forma de manejar tanto la detección como la respuesta de las colisiones no es la más eficiente, pues pruebo una a una las partículas del cuerpo flexible. Algún algoritmo que probara sólo las partículas cercanas haría mejor la detección.

También hay una mejora posible si se considera que el cuerpo flexible puede colisionarse consigo mismo, es decir probar colisiones de las caras del cuerpo con cada una de las partículas y luego responder la colisión. Considero que en cualquier configuración del cuerpo flexible que no sea un volumen convexo este problema estará presente.

Apéndice A

Sobre el software libre

Un objetivo secundario cuando empecé a hacer esta tesis, fue que toda ella fuera hecha con Software Libre. Al final tengo que admitir que esto no fue llevado a cabo por completo, pues todas las figuras del primer capítulo con la excepción de la 1.5 fueron hechas en AutoCAD (http://www.autodesk.es).

Aun así, pienso que este objetivo se cumplió parcialmente, pues fuera de lo antes mencionado todo se hizo en Software Libre; el desarrollo del programa fue hecha sobre GNU/Linux en particular sobre la distribución Ubuntu (www.ubuntu.com).

Para programar se utilizó gcc (http://gcc.gnu.org/), junto con Mesa (http://www.mesa3d.org/) y freeglut (http://freeglut.sourceforge.net/), la biblioteca glui (http://www.cs.unc.edu/rademach/glui/) también es software libre. Como IDE utilice Geany (http://geany.uvena.de/) y debo decir que estoy muy satisfecho con él.

Cuando hubo necesidad de hacer pruebas en Windows, también se ocuparon programas libres: se compiló con Dev-C++ (http://www.bloodshed.net/devcpp.html) como IDE y con Migwn (http://www.mingw.org/) como compilador junto con glui y freeglut.

El texto de la tesis se escribió en LaTeX, en Ubuntu utilice tetex (http://web.bilkent.edu.tr/History/valley/tetex-index.html)y en Windows miktex (http://miktex.org/).

También fue de muchísima ayuda contar con un repositorio para guardar los avances del proyecto, esto me dio la enorme ventaja de poder trabajar en cualquier computadora que tuviera acceso a internet. Para eso se hizo uso de apache (http://www.apache.org/) y de subversion (http://subversion.tigris.org/).

Bibliografía

[1] Glenn Fielder a.k.a Gaffer. Game phisics articles, 2005. Un artículo con tips sobre física para videojuegos, muy relajado y muy útil para empezar a adentrarse en estos temas, me ayudó en la implementación de los métodos de integración.

[2] David Baraf and Andrew Witkin. Phisically based modeling: Principles and practice. *Sigraph*, 1997. Las notas de un curso que se llevó a cabo en Siggrad, en 1997, bastante completas aunque requieren un buen nivel de programación en C para poder entenderlas Tiene un error en la fórmula para separar velocidades en componentes ortogonales.

[3] Paul Blanchard, Robert L. Devaney, and Glen R. Hall. *Ecuaciones Diferenciales*. International Thompson Editores, 1 edition, 1999.

[4] David M. Bourg. *Physics for Game Developers*. OReilly, California, EU, 1 edition, 2002. Un libro muy curioso que nos ayuda a entender la física y pasarla a código de una manera muy elegante, hay muchísimo material en este libro y para este trabajo sólo se utilizó un par de capítulos.

[5] William E. Boyce and Richard C. DiPrima. *Ecuaciones Diferenciales y Problemas con Valores a la Frontera*. Limusa Willey, 4 edition, 2002. Un clásico de Ecuaciones diferenciales y cubre a detalle muchísimos temas, yo lo utilicé primordialmente para el análisis del error de los métodos numéricos.

[6] Maria del Carmen Villar Patiño. Comparación de implementaciones en hw y sw de métodos de integración numérica para simulación de tela. Master's thesis, Instituto Tecnológico y de Estudios Superiores de Monetrrey, 2003. La tesis de Maestría de mi asesora de tesis, muy útil en la parte de las colisiones y de la integración.

[7] Gabriel Valiente Ferlugio. *Composición de Textos Científicos Con LaTeX*. Alfaomega, México DF, México, 1 edition, 2001. Un libro sobre la edición de textos científicos que fue utilizado para escribir este trabajo.

[8] Tomas Jakobsen. Advanced character physics. *IO Interactive*, 2001. Un artículo muy útil donde explican cómo modelar por medio de resortes y se usa el integrador de Verlet. Además lo escribió uno de los creadores del juego de Hitman, a partir de la técnicas con las que se desarrollo el juego.

[9] Danny Kodicek. *Mathematics and Physics for Programers*. Charles River Media, Massachusetts, EU, 1 edition, 2005. Un libro muy bueno para entender la física necesaria para hacer videojuegos, aunque empieza desde aspectos muy básicos, como álgebra y cálculo, toca muy bien el tema de las colisiones y fue una de las fuentes más importantes al escribir esta tesis.

[10] Maciej Matyka. How to implement a presure soft body model, 2004. Un articulo muy bien explicado sobre como implementar un modelo de presión en un cuerpo flexible, ideal para comenzar a leer sobre el tema y fue punto clave en esta tesis.

[11] Maciej Matyka and Mark Ollila. Pressure model of soft body simulation. *Proc. of Sigrad*, 2003. El primer artículo que leí sobre el tema y que me motivó a hacer esta tesis, muy recomendable junto con el segundo articulo.

[12] Joan Trias Pairo. *Geometría para la informática Gráfica y el CAD*. AlfaOmega, Madrid, Espana, 1 edition, 2005. Libro de geometría, muy útil como referencia trata el tema con mucha seriedad, por lo que incluye numerosas demostraciones, también asume que el lector es proficiente en Álgebra Lineal. Las fórmulas del cálculo de volumen están muy basadas en este texto.

[13] Xavier Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. *INRIA*, 1995. Otro articulo de modelado de ropa, que propone una técnica interesante para prevenir el efecto de súper elongación.

[14] Robert Resnick, David Halliday, and Kenneth S. Krane. *Fisica*. CECSA, México DF, México, 4 edition, 1999. Un clásico de física, nunca puede faltar y fue consultado para entender un poco mas de los fenómenos analizados.

[15] Shepley L. Ross. *Introducción a las Ecuaciones Diferenciales*. Interamericana, 3 edition, 1985. Un libro muy sencillo de Ecuaciones Diferenciales, yo lo utilice en mi primer curso para está materia y aun lo consulto muy a menudo, es sencillo y es excelente para aprender.

[16] Dave Shreiner, Mason Woo, and Tom Davis. *OpenGL Programing Guide*. Addison Wesley, Boston, EU, 4 edition, 2004. El libro clásico de OpenGL, es escrito por los mismos desarrolladores de la API, y nunca está de mas.

[17] B. Spanlang T. Vassiliev and Y. Chrysanthou. Fast cloth animation on walking avatars. *EuroGraphics 2001*, 20(3):8, 2001. Un artículo de implementación de modelos de ropa, lo leí por una

técnica que usan para prevenir el efecto de súper elongación, y para entender por que a mi no me afecta dicho fenómeno.

[18] Richard S. Wright and Benjamin Lipchak. *Programación OpenGL.* Anaya Multimedia, Madrid, España, 4 edition, 1999. El primer libro de OpenGL, que existe en nuestro idioma en el país, es muy extenso y la única queja es que la traducción deja un poco que desear.