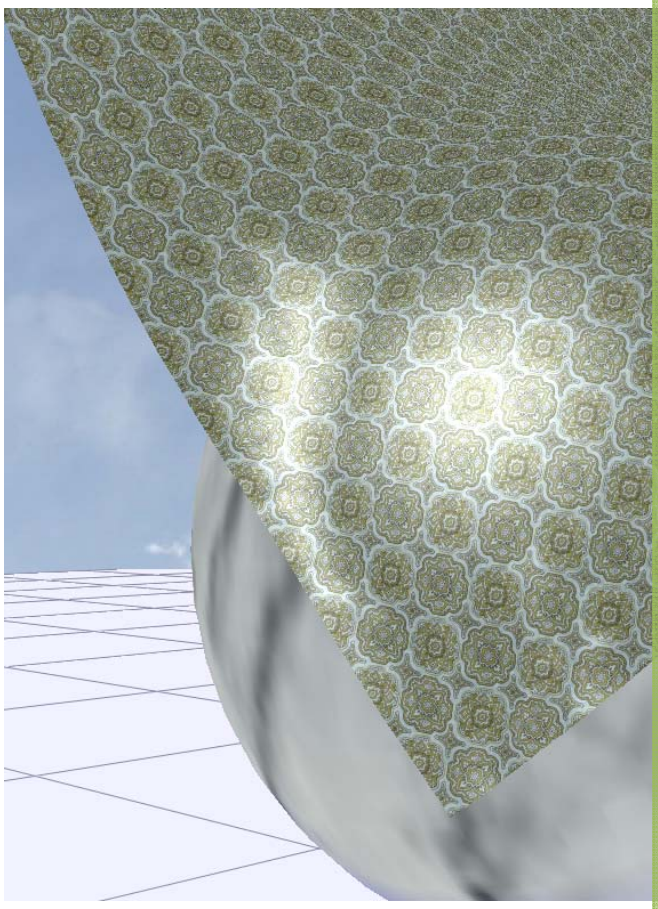
Sistemas informáticos

Simulación 3D de superficies deformables - Telas



Sebastián Breit Foncillas

Iñigo Sainz Rivera

Enrique García Conde - Corbal

Los autores de este proyecto, autorizamos a la Universidad Complutense de Madrid a utilizar y difundir con fines académicos, no comercials y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Sebastian Breit Foncillas Iñigo Sainz Rivera Enrique García Conde-Corbal

Contenido

0 – Resumen

Este proyecto trata del estudio, diseño, implementación y aplicación de la simulación 3D de superficies deformables, concrétamente en el campo de las telas.

A partir de la concepción actual de los sistemas de medida del movimiento de las partículas de estas superficies, hemos creado un método de simulación basada en muelles, que implementa, en escencia, una idea similar pero con un rendimiento mas óptimo.

La memoria explica el modelo actual de representación de telas, desarrolla nuestra idea de representación con mueles, muestra la implementación llevada a cabo, nuestra metodología de trabajo, un catálogo de escenas y un conjunto de conclusiones y aplicaciones de nuestro trabajo.

Abstract

This project handles the study, design, implementation and aplication of deformable surfaces 3D simulation, concretly into the subject of the clothes.

Starting from the actual conception of particle movement measure systems, we have created an alternative method based on springs that implements a similar but optimal idea.

This document explains the actual cloth-representation model, developes our springs-representation idea, shows our proposed implementation, the work methodology we followed, a scenes catalogue and a set of conclusions and aplications of our project.

0.1 – Palabras clave para su búsqueda bibliográfica

1. Kawabata
2. Sólidos deformables
3. Simulación 3D
4. Resortes
5. Telas
6. Motor de física
7. Colisiones
8. OpenGL
9. LWJGL

1 - Motivación - Por qué

La simulación gráfica del movimiento de sólidos deformables es una "ciencia", que por momentos cobra más importancia en el mundo de la informática, los videojuegos o la industria cinematográfica de animación. Incluso es un recurso cada vez más utilizado por la misma industria textil, con el objetivo de estudiar el comportamiento de las telas que van a producir, sin necesidad de tener que fabricarlas para comprobarlo.

En la actualidad, los videojuegos han llegado hasta puntos en los que el jugador dispone de un abanico enorme de movimientos y distintos modos de visualización en cada escena por la que se mueve. Esto es así tan a gran escala en todos los videojuegos, que una de las cosas más importantes que ahora demandan los jugadores, es el realismo en los gráficos que componen cada una de las escenas del videojuego. Esta demanda erxige a las compañías invertir mucho tiempo y dinero en el estudio del movimiento de todos los objetos y elementos que forman cada escena. Esto es, cuidar que la red de una portería de un videojuego de fútbol muestre un movimiento realista cuando entre la pelota; que una tela que cuelga en una escena 3D se mueva con fiabilidad al paso de un personaje que entra en contacto con ella; que la existencia de un río otorgue al jugador sensación de autenticidad por su movimiento natural o al entrar en contacto con un obstáculo; o que la vegetación se mueva por el viento o el contacto con los distintos elementos de la escena.

Similar importancia tiene la calidad de los gráficos en la industria cinematográfica. Tras haber llegado en las películas de animación a una precisión suprema en cada detalle de cada elemento que aparece en pantalla, la industria ha comenzado cada vez más a apostar por la evolución del 3D. Esto obliga a invertir continuamente en estudios y tecnologías para dar más fiabilidad a las escenas.

Por otro lado, la industria textil también aboga mucho por simuladores gráficos a la hora de manufacturar sus telas. Disponer de un software fiable que muestre el comportamiento de una tela antes de ser fabricada permite a la industria ahorrar mucho tiempo y dinero. Con esta intención nace Kawabata, un sistema japonés de cálculo del movimiento de las partículas que componen una tela. Este sistema, en el que ahondaremos en el siguiente punto, es el fundamento de nuestro proyecto.

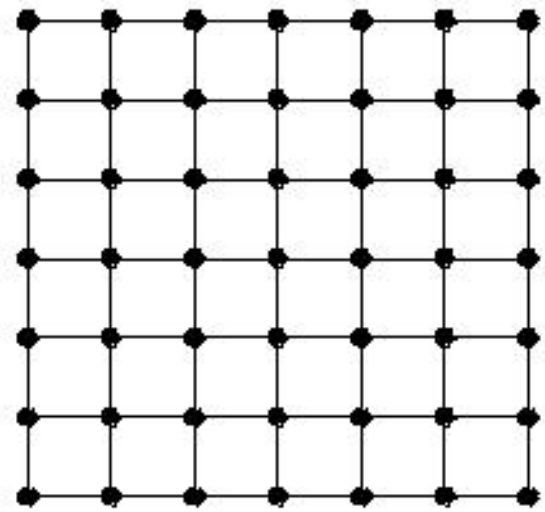
Nuestra motivación u objetivo en el proyecto de sistemas informáticos es estudiar y adentrarnos en este mundo de simulación del movimiento de superficies deformables, concretamente en telas, para, si cabe, poner nuestro granito de arena, o por lo menos ser conocedores de los entresijos de esta especialidad que tan interesante nos resulta.

2 - Modelos deformables 2D

Para llevar a cabo el proyecto de la simulación del comportamiento de las telas nos hemos basado en el sistema antes comentado: Kawabata.

Kawabata es un sistema de medida japonés que fundamenta la simulación del movimiento de estos sólidos deformables a través de las cinco energías que actúan sobre las telas, sin tener en cuenta las interacciones de éstas con otras telas, consigo mismo o cuerpos del espacio. Para entender de una manera más comprensible la interacción de estas energías, cabe destacar el modo en que Kawabata plantea la estructura de una tela.

Kawabata discretiza el movimiento de la tela entendiendola como una rejilla bidimensional de partículas, colocadas cada una de ellas de manera equidistante con respecto las de sus cuatro lados:



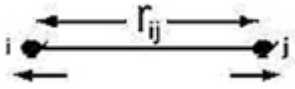
Volviendo a las cinco energías que producen las distintas reacciones de la tela, podemos catalogar cuatro de ellas como internas, ya que son energías resultantes de las fuerzas con las que interactúan las partículas entre sí unas con otras, y la otra como externa, que no es otra que provoca la fuerza de la gravedad sobre cada una de las partículas de la tela. De este modo, podemos determinar que en un instante concreto, cada partícula tendrá una tendencia a un movimiento como resultado de la suma de las cinco fuerzas, que con su módulo, dirección y sentido son engendradas por las cinco energías.

Pasemos a estudiar cada una de estas energías más a fondo. Entendemos la energía total que se aplica sobre cada partícula de la malla como:

$$U_{total} = U_{repel} + U_{stretch} + U_{bend} + U_{trellis} + U_{gravity}$$

2.1 - Repelling

El repelling es la energía provocada por la tendencia de dos partículas contiguas, separadas por una distancia inferior a la del equilibrio, a separarse la una de la otra, provocando una fuerza de repulsión ente ellas.



El cálculo de esta energía entre dos partículas contiguas viene dado por la siguiente ecuación:

$$R(r_{i,j}) = \begin{cases} C\left[\dfrac{(w - r_{ij})^s}{r_{ij}}\right] & r_{ij} \le w \\ 0 & r_{ij} > w \end{cases}$$

donde **C** es un parámetro escalar, **w** la distancia de equilibrio entre dos partículas, y **rij** la distancia actual entre ellas.

La energía Repelling total que actúa sobre una partícula debido a sus partículas vecinas es la suma de las cuatro que se producen por sus vecinas:

$$U_{repel_i} = \sum_{j \neq i}^{n} R(r_{ij})$$

2.2 - Stretching

El stretching es la energía provocada por la tendencia de dos partículas contiguas, separadas por una distancia superior a la de equilibrio, a unirse la una con la otra, provocando una fuerza de atracción ente ellas.

El cálculo de esta energía entre dos partículas contiguas viene dado por la siguiente ecuación,

$$S(r_{i,j}) = \begin{cases} 0 & r_{ij} \le w \\ C\left(\frac{r_{ij} - w}{w}\right)^5 & r_{ij} > w \end{cases}$$

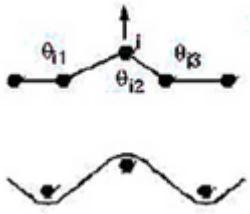
donde **C** es un parámetro escalar, **w** la distancia de equilibrio entre dos partículas, y **rij** la distancia puntual entre dos partículas.

La energía Stretching total que actúa sobre una partícula debido a sus partículas vecinas es la suma de las cuatro que se producen por sus vecinas:

$$U_{stretch_i} = \sum_{j \in N_i} S(r_{ij})$$

2.3 - Bending

El bending es una energía que se produce sobre una partícula a causa de la posición de las partículas contiguas y las contiguas a estas últimas debido a un doblamiento de la tela. Es decir, al doblar la tela, en una fila de cinco partículas, la del centro se verá afectada por las contiguas en función del ángulo que forma con ellas, y por las contiguas a estas últimas, debido también a los ángulos que a su vez forman éstas entre sí:



El cálculo de esta energía sobre una partícula se realiza del siguiente modo:

$$U_{bending_i} = \sum_{j \in M_i} B(\Theta_{ij})$$

siendo **Mi** el conjunto de los seis ángulos que forma una partícula con las contiguas en horizontal y en vertical.

A su vez para calcular el valor puntual de **B**:

$$B = \frac{MK}{2} w^2$$

donde **M** es el momento por unidad de longitud y **K** la curvatura:

$$K(\Theta) = \begin{cases} -\left(\frac{\pi}{4}\right)^2 \frac{b}{\Theta} + a + \frac{\pi}{4} b & 0 \le \Theta \le \frac{\pi}{4} \\ \frac{2}{w} \cos\left(\frac{\Theta}{2}\right) & \frac{\pi}{4} \le \Theta \le \pi \end{cases}$$

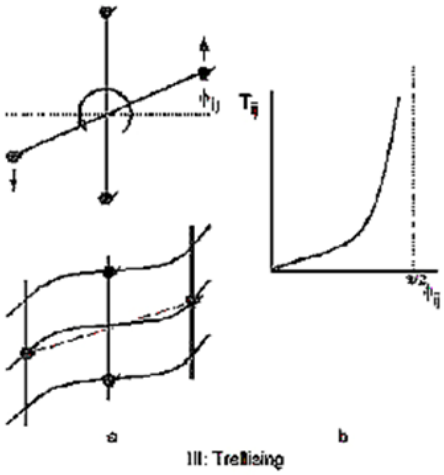
$$a = \frac{2}{w}\cos\left(\frac{\pi}{8}\right), b = \frac{1}{w}\sin\left(\frac{\pi}{8}\right)$$

2.4 - Trellising

En una posición de equilibrio de las partículas, la unión de una partícula con sus cuatro vecinas forma dos segmentos que a su vez generan cuatro ángulos de 90 grados cada uno. Cuando se produce una torsión en la tela, dichos segmentos pierden esos ángulos rectos respecto de los de equilibrio. Cuando esto ocurre, se produce lo que se llama la energía trellising, y no es otra cosa que la tendencia entre las cinco partículas implicadas a recuperar los cuatro ángulos rectos antes indicados. La energía total calculada es calculada como la suma total de las energías sobre los cuatro ángulos.

$$U_{trellis_i} = \sum_{j \in Q_i} T(\Phi_{ij})$$

Q_i - set of four trellising angles



III: Trellising

2.5 - Gravedad

A parte de las energías antes indicadas, que podríamos considerar como internas, la gravedad, aun no siendo interna, podríamos incluirla en el conjunto de energías que por defecto actúan sobre una tela, dado que toda tela se encuentra (a no ser que se encuentre en el espacio)

sometida a dicha energía. Esta energía se calcula puntualmente sobre cada una de las partículas que compone la tela, independientemente de sus partículas vecinas:

$$U_{gravity_i} = m_i g h_i$$

m_i - mass
h_i - height
g - acceleration due to gravity

3 - Implementación – Diseño

3.1 - Nuestra idea

La idea de llevar a cabo la implementación de acuerdo a las ecuaciones de Kawabata era muy atractiva, ya que estábamos seguros de que obtendríamos una representación fiel de una tela sin muchas complicaciones. Sin embargo, cuando teníamos casi todo el sistema de partículas implementado (menos el trellising), quedamos decepcionados con el rendimiento de nuestros computadores a la hora de renderizar una escena. Nos planteamos dos opciones: seguir adelante con Kawabata y proponer nuestro diseño para aplicaciones que no necesitaran de renderizado en tiempo real, ó reinventar el comportamiento de una tela y utilizar otras ecuaciones/métodos que nos brinden un resultado aceptable y más eficiente.

Después de unas semanas pensando en otros métodos factibles, llegamos a una idea interesante y relativamente sencilla: implementaríamos la malla de partículas a base de muelles.

A pesar de que el hecho de aplicar un modelo totalmente diferente para representar nuestra simulación significaba desechar gran cantidad del código implementado, el estudio de Kawabata nos dio unas nociones muy útiles en cuanto al comportamiento de telas. Nociones que facilitaron el desarrollo de nuestra idea.

3.2 – Implementación con muelles

Se conoce como muelle o resorte a un operador elástico capaz de almacenar energía y desprenderse de ella sin sufrir una deformación permanente cuando cesan las fuerzas o la tensión a las que es sometido.

La manera más sencilla de analizar un resorte físicamente es mediante su modelo ideal global y bajo la suposición de que éste obedece la Ley de Hooke. Se establece así la ecuación del resorte, donde se relaciona la fuerza *F* ejercida sobre el mismo con el alargamiento/contracción o elongación **x** producida, del siguiente modo:

$$F = -kx$$

siendo

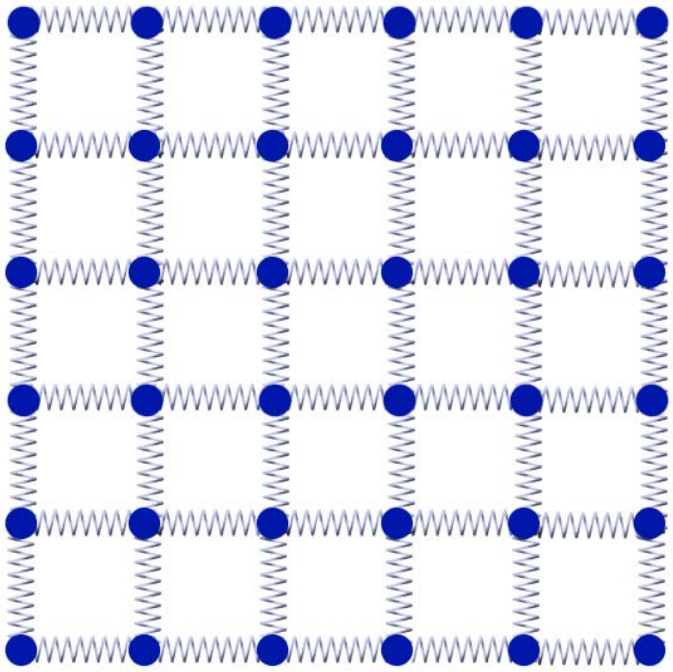
$$k = \frac{AE}{L}$$

Donde **k** es la constante elástica del resorte, **x** la elongación (alargamiento producido), **A** la sección del cilindro imaginario que envuelve al muelle y **E** el módulo de elasticidad del muelle.

En esencia, y de manera más trivial, la característica principal de los muelles es la fuerza que oponen a fuerzas de compresión o de alargamiento con el objetivo de recuperar su posición inicial o de equilibrio. Basándonos en este concepto, hemos implementado las distintas energías de Kawabata con “muelles”.

3.2.1 - Stretching / Repelling

Para las energías de Stretching y Repelling, la implementación con muelles es bastante trivial. Basándonos en las definiciones propias del Stretching, del Repelling y de los muelles, hemos decidido unir cada par de partículas contiguas con un muelle.

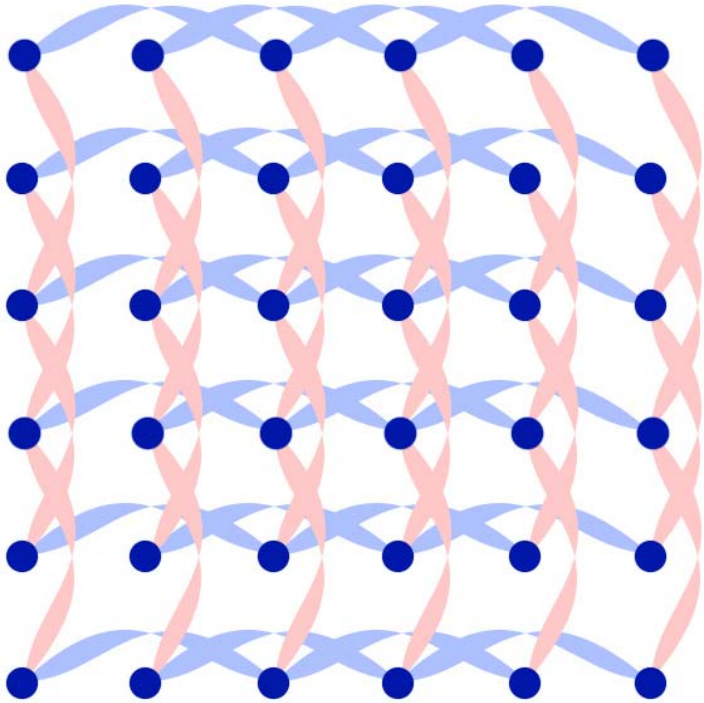


De este modo, cuando dos partículas se encuentran a una distancia mayor de la de equilibrio, el muelle que las une, simulando el Stretching, tenderá a aproximarlas tratando de recuperar la distancia de equilibrio.

De la misma manera, cuando dos partículas se encuentran a una distancia menor de la de equilibrio, el muelle que las une, simulando el Repelling, tenderá a repelerlas tratando de recuperar la distancia de equilibrio.

3.2.2 - Bending

En cuanto al Bending, como la intención de esta fuerza es devolver la forma original a la tela cuando ésta es doblada de forma vertical u horizontal, pensamos que la mejor manera de implementarla con muelles sería uniendo cada partícula con las partículas contiguas a sus vecinas, tanto verticales como horizontales.

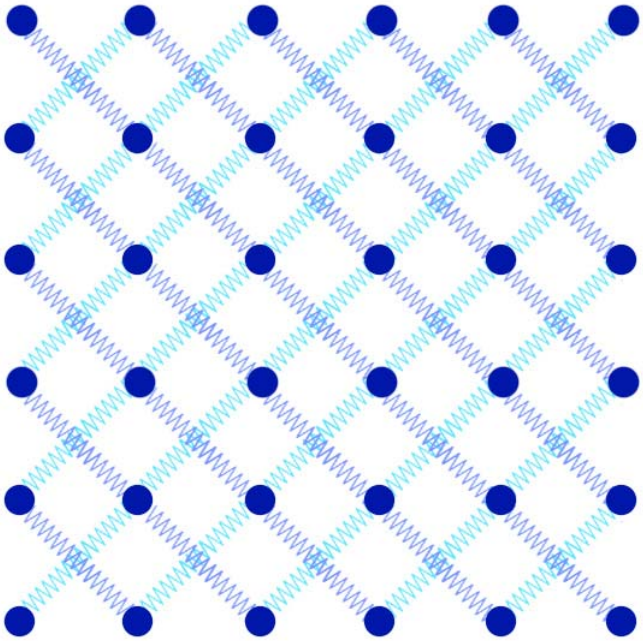


Intuitivamente, por efecto del Repelling, si en un trío de partículas alineadas, las partículas de los extremos se acercasen, la partícula de en medio tendería a escapar de ellas. Tratando de recuperar la posición de equilibrio, el Bending opone una fuerza opuesta a la de "escape" de la partícula con el objetivo de tratar de acercarla de nuevo.

Uniendo las partículas de los extremos de cada trío de partículas alineadas con un muelle de longitud doble de la distancia de equilibrio entre dos partículas contiguas, se consigue una manera muy similar de Bending, ya que cuando estas partículas se encuentran a una distancia menor que la longitud del muelle, dicho muelle tratará de repelerlas, desencadenando la consecuente fuerza de atracción de la partícula central a su posición original.

3.2.3 - Trellising

Como la propiedad Trellising de la tela añade una fuerza a las partículas con el fin de que éstas recuperen los ángulos rectos con sus contiguas cuando la malla es sometida a una torsión, la implementación que hemos encontrado coherente, es la de unir cada partícula con sus cuatro vecinas diagonales a través de muelles.



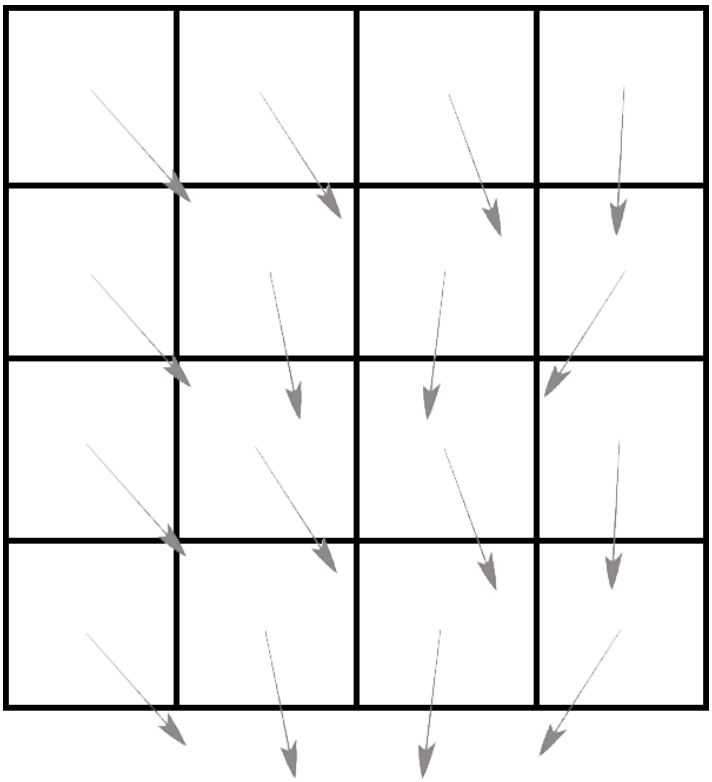
Aplicando esta implementación, imaginemos una sub-malla de la malla anterior, de 9 partículas (3x3). Al mover las partículas de dicha malla, los muelles de las diagonales tratarán de recuperar su elongación de equilibrio. Dado que los cuatro muelles de las diagonales tratan de conseguir el mismo resultado, la estructura tenderá a recuperar los ángulos de 90 grados entre las partículas centrales de cada arista exterior de la malla con la partícula central.

3.3 – Iluminación de la tela

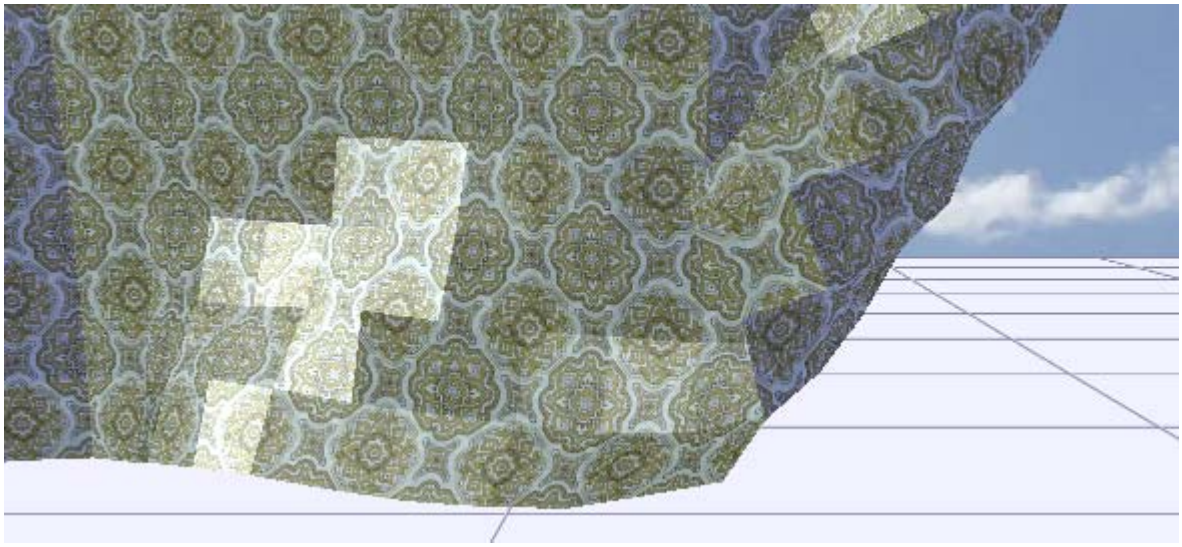
Para dar mayor realismo a la tela, y dado que nuestras escenas tienen lugar en escenarios exteriores, resultaba importante otorgar a la tela efectos de iluminación. Esto consiste en dar a cada parte de la tela la intensidad de luz que le corresponda en función de la posición que ocupe ésta en el espacio con respecto a una fuente de luz que hemos colocado en cada escena.

Para conseguir este efecto, utilizamos un cálculo de normales. Esto nos permite medir la cantidad de luz que debe incidir sobre cada región de la tela en función del ángulo entre la normal y la fuente de luz.

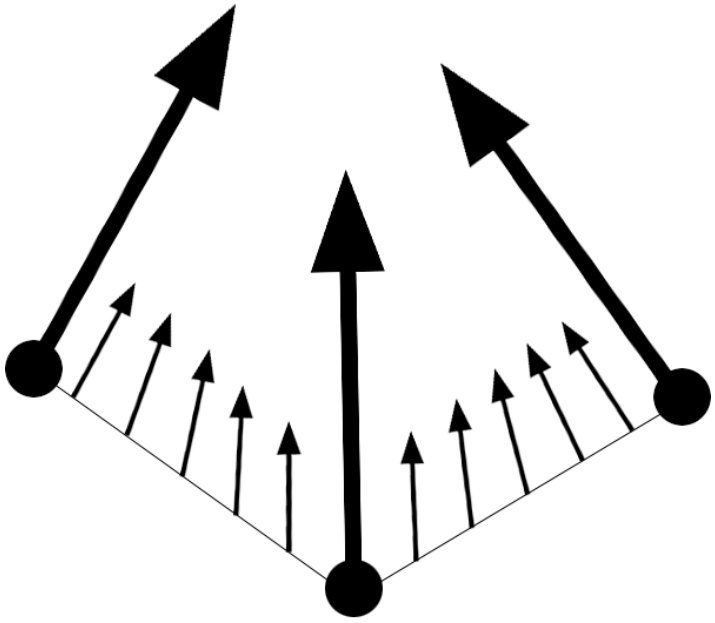
Inicialmente este cálculo de normales lo realizábamos para cada cuadrilátero formado por cada cuatro partículas con sus aristas. El cálculo se realizaba sobre el centro de dicho cuadrilátero, sacando su vector normal y luego renderizándolo con su iluminación correspondiente.



Este método no es muy realista porque la iluminación es la misma para todos los puntos del cuadrilátero. De este modo, si el cuadrilátero contiguo tiene una normal distinta, se notará el cambio de iluminación al marcarse las aristas, dando un efecto pixelado y poco real.



La manera de solucionar este problema es utilizando un método de interpolación de normales. De este modo, se calcula una normal para cada pixel de la tela.



Para realizar el cálculo de dicha interpolación, calculamos el vector normal media de las cuatro normales de los cuatro planos que confluyen en una partícula de la malla, es decir, calculamos la normal en cada partícula que compone la tela.

Este método permite que independientemente del número de partículas que componga la tela representada, cada píxel de la tela tenga la intensidad de luz en función de su posición respecto de la fuente de luz.

Cabe matizar que nuestro programa realiza el cálculo de las normales sobre cada partícula, y que la interpolación es llevada a cabo a través del Smooth Shading que proporciona OpenGL.

3.4 – Interacción con el medio

Como la representación debe ser lo más realista posible, debemos simular ciertas propiedades del medio. La gravedad es una de ella, que añadimos fácilmente gracias a la interfaz Fuerza que será explicada en detalle en el punto 3.6 de este documento.

El rozamiento con el medio lo conseguimos a través de una constante que actuará frenando el movimiento de las partículas independientemente de su posición, velocidad o fuerza. De esta manera, si la constante es grande, la malla de partículas parecerá sumergida en un líquido viscoso, y si la constante es casi nula, no existirá fricción alguna. Probando diferentes valores para esta constante, hemos logrado simular el rozamiento de una tela con el aire.

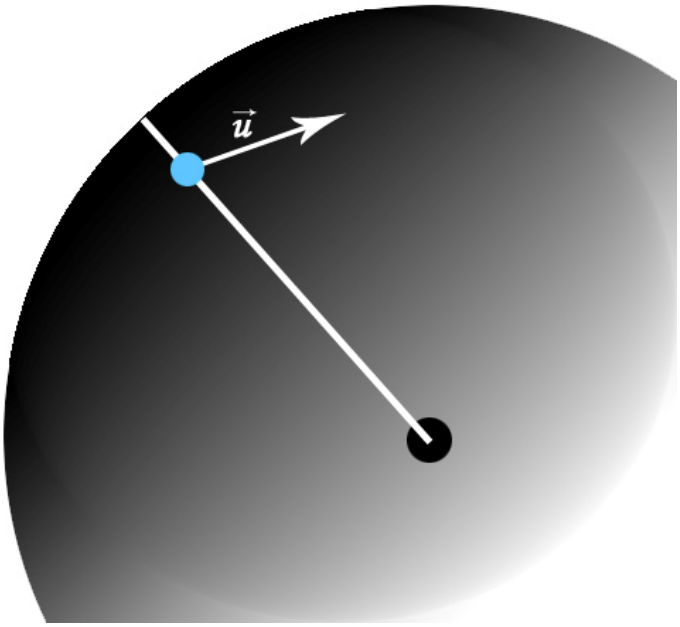
3.5 – Colisiones con esferas

3.5.1 – Modelado

Una vez modelada nuestra tela, se nos ocurrió la idea de hacerla interactuar con esferas. Para ello, previamente debíamos implementar y renderizar las esferas. Una esfera la concebimos como una partícula, que es el centro de la misma, y su radio. Este modo de implementar una esfera resulta interesante por su buen rendimiento. Además al tratar a la esfera de manera similar a una partícula e incluirla en el ´mundo´ creado por el motor de físicas, esta tambien es afectada por las propiedades del medio (gravedad, rozamiento).

Para implementar las colisiones de la tela con la esfera, hemos tomado como principal fundamento el hecho de que ninguna partícula de la tela puede encontrarse en ningún momento, respecto del centro de la esfera, a una distancia menor del radio. Entendemos entonces que existe colisión entre una partícula de la tela y la esfera si se incumple dicha condición.

Teniendo en cuenta este concepto para implementar las colisiones, tras mover las partículas de la tela en cada ciclo, calculamos para cada una de ellas su distancia respecto del centro de la circunferencia. Si esta distancia es mayor, no se produce colisión, por lo que no se hace nada. Si por el contrario es mayor, se ha producido una colisión.



Partícula de la tela que ha invadido la esfera

3.5.2 – Tratamiento de las colisiones

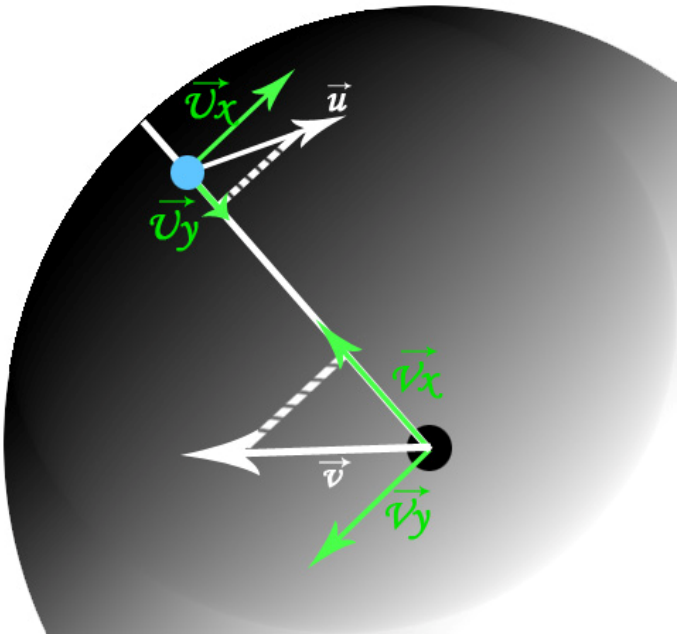
Para tratar una colisión se debe tener en cuenta la dirección del movimiento de la partícula que entra en contacto con la esfera. Para expulsar la partícula de la esfera y respetar su movimiento con su correspondiente velocidad, colocamos la partícula en la superficie de la esfera y anulamos las componentes del vector que se dirigen hacia su interior. Procedemos de la siguiente manera:

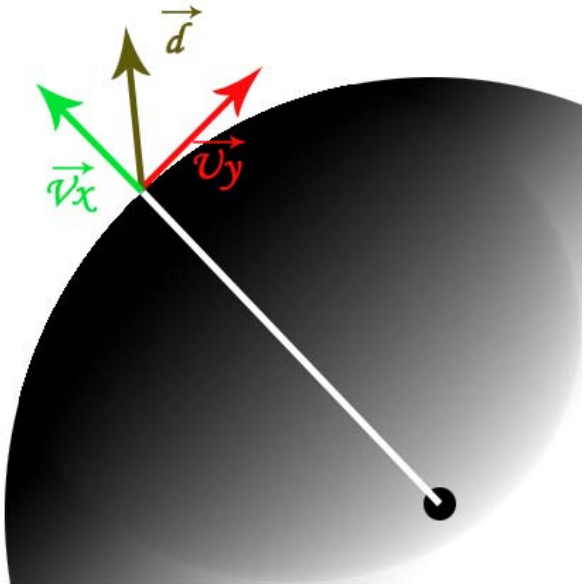
1. Calculamos el vector con origen el centro de la esfera y con destino la partícula.
2. Hallamos la proyección del vector de velocidad de la partícula sobre el vector hallado en el punto 1.
3. Cambiamos su sentido multiplicándola por -1.
4. Sumamos la proyección invertida al vector original de la partícula para de este modo anular las componentes que la dirigen hacia el interior de la esfera.

5. Calculamos el punto donde corta a la superficie de la esfera el segmento que nace en el centro y pasa por la partícula.
6. Colocamos la partícula en dicho punto.

Se debe tener en cuenta también que la esfera puede encontrarse en movimiento, lo que consecuentemente produciría una reacción distinta sobre la partícula, ya que ésta a su vez recibiría una fuerza complementaria en el sentido del movimiento de la esfera. Para ello, hemos aplicado los siguientes puntos en el cálculo del movimiento de la partícula colisionada.

7. Calculamos la proyección del vector velocidad de la esfera sobre el vector hallado en el punto 1.
8. Sumamos el vector proyección resultante al vector velocidad de la partícula.





3.6 - Motor de física

Para poner todas estas ideas en orden, implementamos una serie de objetos que dieron forma a un motor de física aplicable a la tela.

En un principio necesitamos:

- Vectores + operaciones necesarias (tamaño, normalización, producto vectorial, etc.)
- Interfaz para la implementación de fuerzas (atracciones, colisiones, etc.)
- Partículas con su respectiva posición, masa, velocidad y fuerza.
- Muelles con punteros a las partículas que une, constante de elasticidad y constante de oscilación. Los muelles son a efectos prácticos una fuerza, por lo que implementan la interfaz Fuerza.
- Malla (ó sistema de partículas) que contiene a todas las partículas de un sistema junto con sus muelles. La malla también contendrá a la fuerza de gravedad.

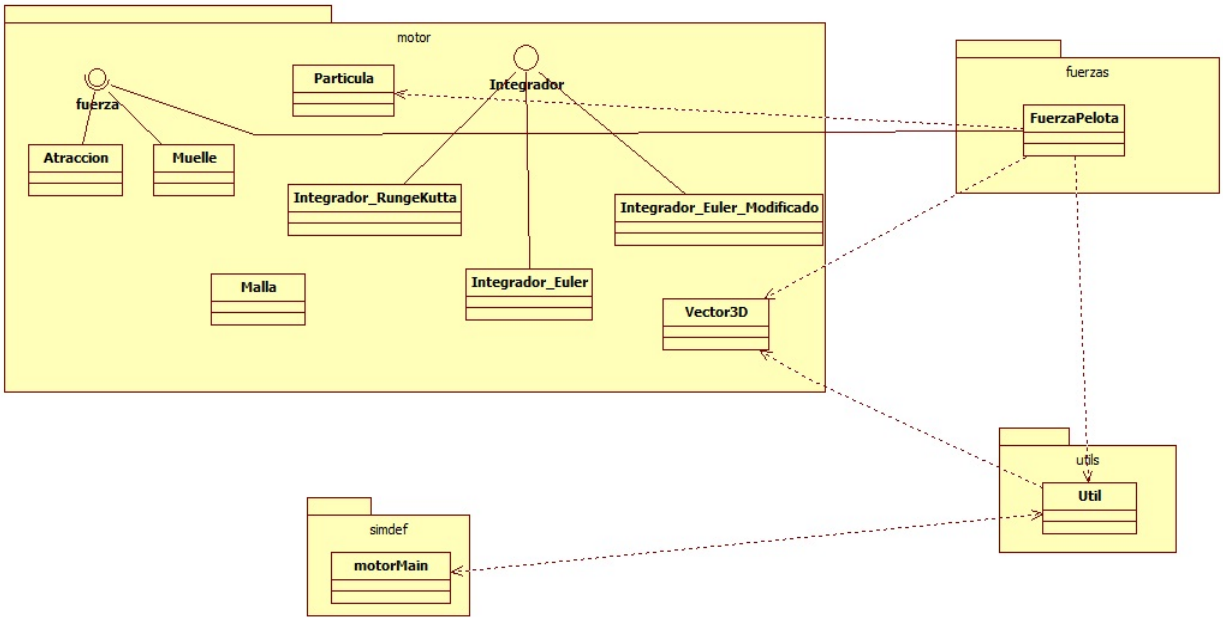
La implementación de este motor se explica más detalladamente en el apartado 3.7.2.

3.7 – Implementación

Tomamos la decisión de utilizar el lenguaje Java para la implementación de nuestro proyecto ya que nos simplificaba el problema de trabajar en diferentes plataformas.
Para poder utilizar OpenGL, utilizamos la librería LWJGL 2.7.1 y slick-utils para la carga de texturas.

3.7.1 - Diagramas

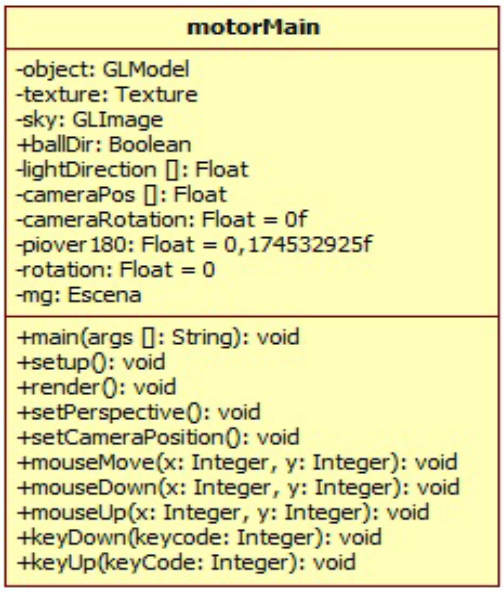
3.7.1.1 – Diagrama de paquetes



3.7.1.2 – Diagrama de clases paquete simdef

El paquete *simdef* está formado por la clase *motorMain*, que es el main de nuestro programa.

En esta clase se inicializan los parámetros que forman parten de la escena. Incluye el método de renderización, así como las diferentes operaciones que nos permiten interactuar con la escena desde el teclado para ir cambiando de posición la cámara o bien "jugar" con la fuerza y posición de algunos de los objetos.



3.7.1.3 – Diagrama de clases paquete motor

El paquete *motor* está formado por las clases que dan forma al motor de física que hemos hecho para la representación de telas. En realidad este motor está implementado de forma que sirva para la simulación de cualquier otro tipo de sistema de partículas. Debido a esta ambición de crear un motor independiente y reutilizable, las colisiones fueron colocadas fuera, ya que de lo contrario aumentaba considerablemente la complejidad de la implementación de este paquete.

Las clases que forman el paquete Motor son:

- Clase Vector 3D: Esta clase define todas las operaciones necesarias para trabajar con vectores de tres dimensiones.

- Interfaz Fuerza: Interfaz necesaria para las distintas implementaciones de fuerzas de nuestro proyecto. Tanto las clases FuerzaViento como Muelle implementan esta interfaz. Así mismo también implementa a FuerzaPelota del *paquete fuerzas*.

- Clase Atraccion: Clase que implementa a la interfaz Fuerza. Esta clase finalmente no fue utilizada en la simulación de telas pero la dejamos en el motor ya que su aplicación podría ser interesante en el futuro.

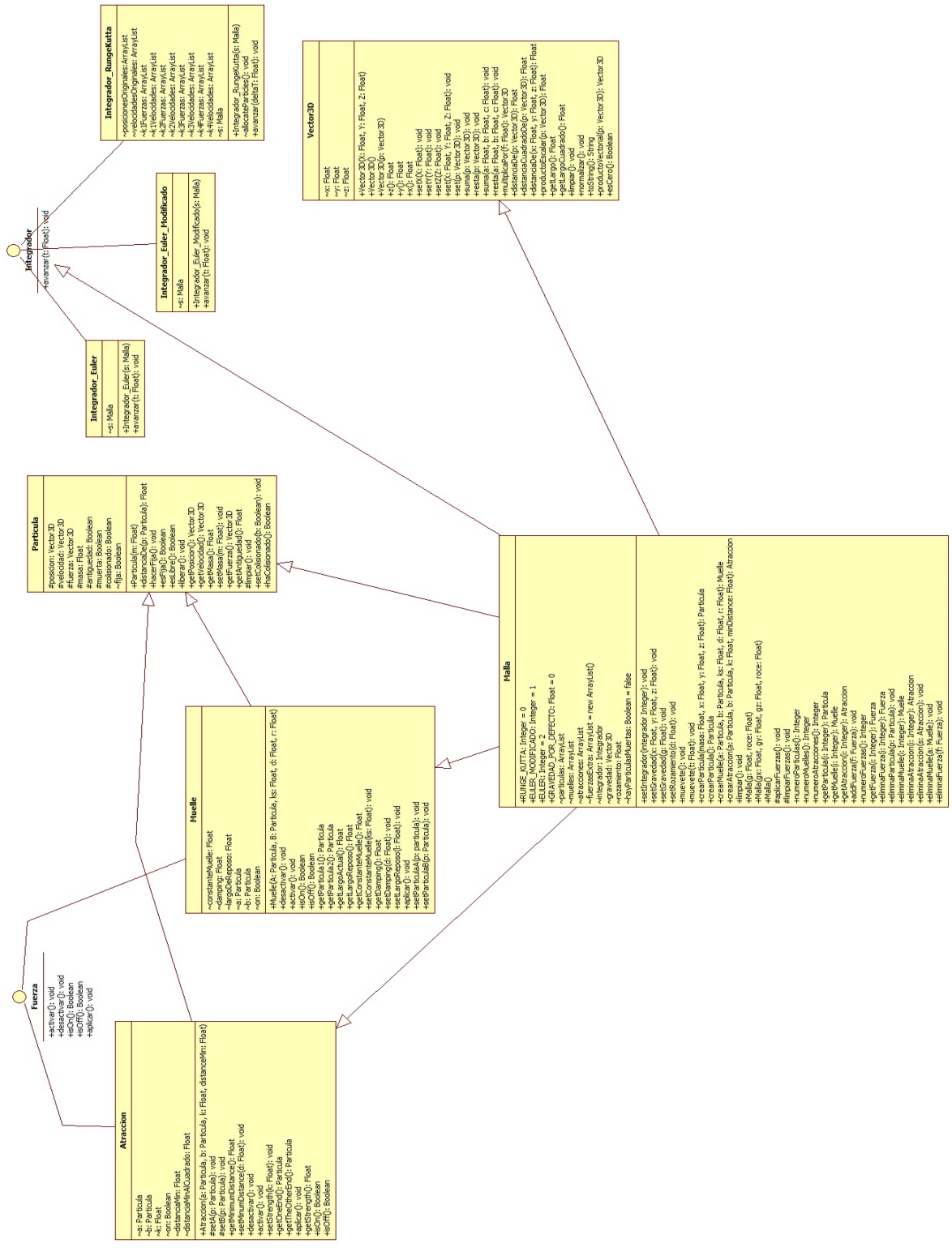
- Clase Muelle: La clase Muelle es fundamental para la idea de implementación final de nuestro proyecto. Esta clase une dos partículas a través de la aplicación de la ley de Hooke. Tiene como atributos dos punteros a sendas partículas, la constante elástica del muelle, la distancia de reposo y el factor de oscilación (damping). Como al fin y al cabo un muelle simplemente ejerce fuerzas entre partículas, esta clase también implementa la interfaz Fuerza.

- Clase Particula: En esta clase se definen la posición, velocidad y fuerza de las partículas. Al tratarlas de manera vectorial se apoyan en la clase Vector3D en sus distintas operaciones.

- Interfaz Integrador: Para poder animar la malla de partículas, necesitamos un integrador que junte todas las fuerzas/velocidades/masas y calcule la posición de cada partícula pasado un tiempo **n**.

- Clase Malla: La clase Malla contiene todas las partículas que forman el sistema, junto con los muelles que las unen y el resto

de fuerzas que entran en juego. Al tratarse de la clase que debe formar el sistema de partículas, utiliza la mayoría de las clases del paquete *motor* como las clases Particula, Muelle, Atraccion, Vector3D, y la interfaz Integrador.



La clase Particula no hace ningún cálculo que merezca mención en esta memoria. Sin embargo alberga una serie de propiedades importantísimas para el desarrollo del comportamiento de la tela.

```
public Particula( float m )
 {
        posicion = new Vector3D();                  //Posición de la partícula
        velocidad = new Vector3D();          //Velocidad de la partícula
        fuerza = new Vector3D();                    //Fuerza de la partícula
        masa = m;                           //Masa de la partícula
        fija = false;                  //Determina si la partícula está fija o se puede mover
        antiguedad = 0;              //Número de iteraciones que ha existido la partícula
}
```

En la clase Muelle, el método más importante es el que aplica la fuerza entre ambas partículas respecto a su distancia.

```
public final void aplicar()
 {
       if ( on && ( a.esLibre() || b.esLibre() ) )  //Chequea la si las dos partículas están fijas
       {
              float a2bX = a.getPosicion().x - b.getPosicion().x;
              float a2bY = a.getPosicion().y - b.getPosicion().y;
              float a2bZ = a.getPosicion().z - b.getPosicion().z;

              //calcula la distancia entre las 2 partículas
              float distanciaAB = (float)Math.sqrt( a2bX*a2bX + a2bY*a2bY + a2bZ*a2bZ );

              if ( distanciaAB == 0 ){ //evitar división por 0
                     a2bX = 0;
                     a2bY = 0;
                     a2bZ = 0;
              }
              else{ //normalización
                     a2bX /= distanciaAB;
                     a2bY /= distanciaAB;
                     a2bZ /= distanciaAB;
              }

              // fuerza proporcional a la distancia (Ley de Hooke)
              float fuerzaMuelle = -( distanciaAB - largoDeReposo ) * constanteMuelle;

       // velocidad es el vector entre a y b... el damping es proporcional a la velocidad
              float Va2bX = a.getVelocidad().x - b.getVelocidad().x;
              float Va2bY = a.getVelocidad().y - b.getVelocidad().y;
              float Va2bZ = a.getVelocidad().z - b.getVelocidad().z;

              float fuerzaDamping = -damping * ( a2bX*Va2bX + a2bY*Va2bY + a2bZ*Va2bZ );

       // La fuerza para la partícula B es igual que para A en la dirección contraria.
              float r = fuerzaMuelle + fuerzaDamping;

              a2bX *= r;
              a2bY *= r;
              a2bZ *= r;

              if ( a.esLibre() )
                     a.getFuerza().suma( a2bX, a2bY, a2bZ );
```

```
                if ( b.esLibre() )
                        b.getFuerza().suma( -a2bX, -a2bY, -a2bZ );
        }
}
```

La clase Malla alberga a todas las partículas de un sistema (una tela), llama a las diferentes fuerzas existentes para que sean aplicadas a las partículas, determina que integrador será utilizado (por el momento sólo tenemos un integrador Runge-Kutta que será explicado a continuación) y llama al integrador para efectuar un paso de la animación. Esta clase también proporciona la funcionalidad de crear y eliminar partículas y muelles para ese sistema.

Uno de los métodos más interesantes de esta clase es el de aplicar fuerzas:

```
protected final void aplicarFuerzas()
 {
        if ( !gravedad.esCero() )  // aplicar fuerza de gravedad
        {
                for ( int i = 0; i < particulas.size(); ++i )
                {
                        Particula p = (Particula)particulas.get( i );
                        p.fuerza.suma( gravedad );
                }
        }

        for ( int i = 0; i < particulas.size(); ++i ) // modificar el vector fuerza de cada
                                                      partícula respecto a su velocidad
        {
                Particula p = (Particula)particulas.get( i );
                // el parámetro rozamiento se define en la constructora de la escena sobre
                     la que se realizan los cálculos
                p.fuerza.suma( p.velocidad.x() * -rozamiento, p.velocidad.y() * -rozamiento,
                p.velocidad.z() * -rozamiento );
        }

        for ( int i = 0; i < muelles.size(); i++ ) // aplicar la fuerza de los muelles existentes
                                                   en el sistema
        {
                Muelle f = (Muelle)muelles.get( i );
                f.aplicar();
        }

        for ( int i = 0; i < fuerzasExtra.size(); i++ ) // aplicar las fuerzas extra que han sido
                                                        añadidas al sistema
        {
                Fuerza f = (Fuerza)fuerzasExtra.get( i );
```

```
                f.aplicar();
            }
        }
```

La clase Integrador_RungeKutta implementa la interfaz Integrador y es el que hemos utilizado para el avance en el tiempo de las partículas. En algunas simulaciones de cálculo de física se necesita un integrador, que es un método numérico para la integración de las trayectorias de las fuerzas (y por tanto las aceleraciones) que sólo se calcula en intervalos de tiempo discretos. El método de Euler nos resultó poco eficiente y en ocasiones provocaba la rotura de la malla (no llegamos a descubrir el trasfondo matemático del problema). Es por esto que investigando llegamos al método Runge Kutta de orden 4 que nos dio fantásticos resultados.

Definiendo un problema inicial como:

$$y' = f(x, y), \quad y(x_0) = y_0$$

Entonces el método Runge Kutta de orden 4 para este problema está dado por la siguiente ecuación.

$$y_{i-1} = y_i + \frac{1}{6}(k_1 + 2k_2 - 2k_3 + k_4)h$$

donde:

$$k_1 = f(x_i, y_i)$$

$$k_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right)$$

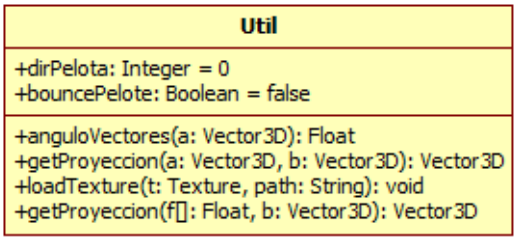
$$k_3 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2 h\right)$$

$$k_4 = f(x_i + h, y_i + k_3 h)$$

La implementación de la clase Integrador_RungeKutta es fiel a la definición del método de orden 4.

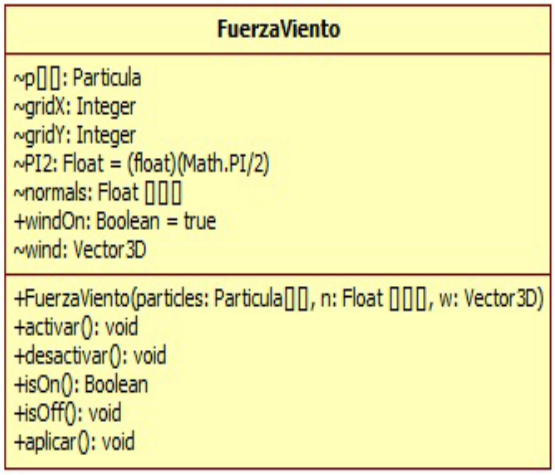
3.7.1.4 – Diagrama de clases paquete utils

El paquete *utils* contiene a la clase *Util* que hemos implementado para calcular una serie de operaciones auxiliares que se utilizarán para calcular las fuerzas que se dan en el impacto del objeto Pelota con la tela.



3.7.1.5 – Diagrama de clases paquete fuerzas

El paquete *fuerzas* contiene la clase FuerzaViento que implementa la interfaz Fuerzas del paquete *motor.* Esta clase aplica la fuerza del viento a cada una de las partículas de la tela.



3.7.1.5 – Paquete Objetos

Este paquete contiene el conjunto de clases que definen las escenas, tras las que se esconden los paquetes y clases antes definidos.

El principal componente de este paquete es la interfaz Escena. Esta interfaz contiene los métodos que todas las escenas deben implementar, para llevar a cabo la simulación. Son métodos de renderizado, de carga de texturas o imágenes para las telas o esferas, métodos para aplicar fuerzas sobre las partículas de la tela, y otro método para ejecutar un ciclo completo del movimiento de las partículas de la escena.

Por otro lado se encuentran las clases que definen cada escena. Estas clases implementan los métodos de la clase escena y configuran la simulación que se llevará a cabo.

En el siguiente punto se explica más detalladamente la forma de implementar una escena.

3.7.2 - Implementación de una escena

El paquete *simdef* está formado por la clase *motorMain* que es el main de nuestro programa.

En esta clase se inicializan los parámetros que forman parte de la escena y se configura la ventana de OpenGL que vamos a utilizar. Incluye el método de renderización, así como las diferentes operaciones que nos permiten interactuar con la escena desde el teclado para ir cambiando de posición la cámara o bien "jugar" con la fuerza y posición de algunos de los objetos. Una vez inicializada la ventana y el resto de objetos que necesitamos, creamos una instancia de alguna clase que implemente la interfaz Escena. De este modo podemos crear infinidad de escenas diferentes y simplemente variando el nombre en nuestro motorMain obtenemos el renderizado deseado.

```
mg=(Escena) new Banderas(1000,800,0,0);
mg=(Escena) new Telon(1000,800,0,0);
...
```

La escena en cuestión define las constantes del sistema (o sistemas) de partículas que requiere, como la fuerza de los muelles, la constante de elasticidad, bending, trellising, viscosidad del medio, tamaño del grid, y aquellos objetos que interactúan con la tela en la escena, como la esfera, el suelo, etc.

Una vez creados los objetos e inicializadas las variables necesarias, se crea la malla, fijando mediante parámetros la fuerza de gravedad y el rozamiento con el medio que la rodea:

```
physics = new Malla(-9.82f, 0.02f);
```

A continuación creamos las partículas pertenecientes a este sistema:

```
for (int i = 0; i < gridX; i++) {
        for (int j = 0; j < gridY; j++) {
        particles[i][j] = physics.crearParticula(0.8f, i * pasoX + (screenWidth / 4), -j * pasoY, 0f);
            if (j > 0) {
               Particula p1 = particles[i][j - 1];
               Particula p2 = particles[i][j];
               physics.crearMuelle(p1, p2, fuerzaMuelle, elastMuelle, pasoY);
            }
            if (i > 0) {
               Particula p1 = particles[i - 1][j];
               Particula p2 = particles[i][j];
               physics.crearMuelle(p1, p2,fuerzaMuelle, elastMuelle,
                     gridStepY);
            }

        //Bending
            if (j > 1) {
               Particula p1 = particles[i][j - 2];
               Particula p2 = particles[i][j];
               physics.crearMuelle(p1, p2, bendingStrength, bendingDamping,
                     gridStepY * 2);
            }
            if (i > 1) {
               Particula p1 = particles[i - 2][j];
               Particula p2 = particles[i][j];
               physics.crearMuelle(p1, p2, bendingStrength, bendingDamping,
                     gridStepY * 2);
            }

            //trellesing
            float dist = (float) Math.sqrt(2f * gridStepY * gridStepY);
            if (j > 0 && i > 0) {
               Particula p1 = particles[i - 1][j - 1];
               Particula p2 = particles[i][j];
               physics.crearMuelle(p1, p2, trellesingStrength, trellesingDamping,
                     dist);
               p1 = particles[i][j - 1];
               p2 = particles[i - 1][j];
               physics.crearMuelle(p1, p2, trellesingStrength, trellesingDamping,
                     dist);
            } ...
```

De esta manera hemos unido cada partícula con sus vecinas en horizontal y vertical y del mismo modo creamos los muelles en diagonal para el bending y trellising.

Si queremos que una partícula se encuentre fija y no reaccione a fuerzas, basta con cambiar su atributo *fija*.

Para añadir fuerzas externas a nuestro sistema, gracias a nuestro motor es muy sencillo:

```
physics.addFuerza(new FuerzaExterna(new Vector3D(0.0f, 0.5f, 1.0f)));
```

En este caso FuerzaExterna implementa a la interfaz Fuerza de nuestro motor.

Las fuerzas pueden actuar de diferentes maneras. Por ejemplo, para simular un viento, haremos lo siguiente:

```
physics.addFuerza(new FuerzaViento(particles, normals, new Vector3D(0.0f, 0.5f, 1.0f)));
```

Aquí la clase FuerzaViento implemente a la interfaz Fuerza, pero extiende su funcionalidad, ya que utilizará las normales de cada partícula para aplicar una fuerza mayor o menor. Es así como hemos logrado simular exitosamente el comportamiento de una tela al viento.

En este caso, el método *aplicar* de la clase FuerzaViento sería así:

```
public void aplicar(boolean over) {
        for (int i = 0; i < gridX; i++) {
                for (int j = 0; j < gridY; j++) {
                        Vector3D proyeccion = Util.getProyeccionf(normals[i][j], wind);
                        if(proyeccion.x()>0)
                                proyeccion.multiplicaPor(-1f);
```

```
                    p[i][j].getFuerza().suma(proyeccion);
                }
            }
        }
```

Las clases que implementan la interfaz Escena, también serán las encargadas de efectuar el cálculo de las normales para la correcta iluminación de la tela. Este cálculo es trivial por lo que no lo cubriremos en esta memoria.

El paquete *utils* contiene la clase *Util* que incluye una serie de operaciones auxiliares que se utilizan para calcular las proyecciones de un vector sobre otro, la carga de texturas, etc.

El *paquete fuerzas* contiene a la clase FuerzaViento que implementa a la interfaz Fuerza del motor y representa la fuerza efectuada por el viento sobre una malla determinada. El viento actúa de manera pseudo-aleatoria por intervalos de tiempo, lo que nos permite obtener resultados inesperados cada vez que ponemos en marcha una escena.

4 – Animación

Para la correcta visualización y animación de nuestras escenas hemos utilizado la librería LWJGL 2.7.1. Esta librería nos ha permitido dedicar el mínimo tiempo posible al proceso visual de nuestro proyecto para así poder enfocarnos en la física y comportamiento implícito en una tela.

Para conseguir una ventana de OpenGL en java con LWJGL, el método que hemos utilizado ha sido el siguiente:

- La clase que crea la ventana debe extender la clase GLApp (proporcionada por LWJGL).

- Implementamos los métodos relevantes:

```
public void setup(){
        GL11.glEnable(GL11.GL_LIGHTING);
        GL11.glEnable(GL11.GL_COLOR_MATERIAL);
...
public void render(){
        Escena.render();
...
public static void setPerspective(){
        GL11.glMatrixMode(GL11.GL_PROJECTION);
...
```

La interfaz GLApp tiene un método run() que crea un thread encargado de renderizar lo más rápido posible, es decir, llama al método render() en cada iteración.

El resto ha sido seguir la documentación de OpenGL.

Para el renderizado de los mástiles de las banderas, otra vez LWJGL nos ha facilitado el trabajo ya que disponemos de la librería GLUT, la cual proporciona diferentes tipos de objetos. En nuestro caso utilizamos el objeto Cylinder, que representa un cilindro de las dimensiones que deseemos.

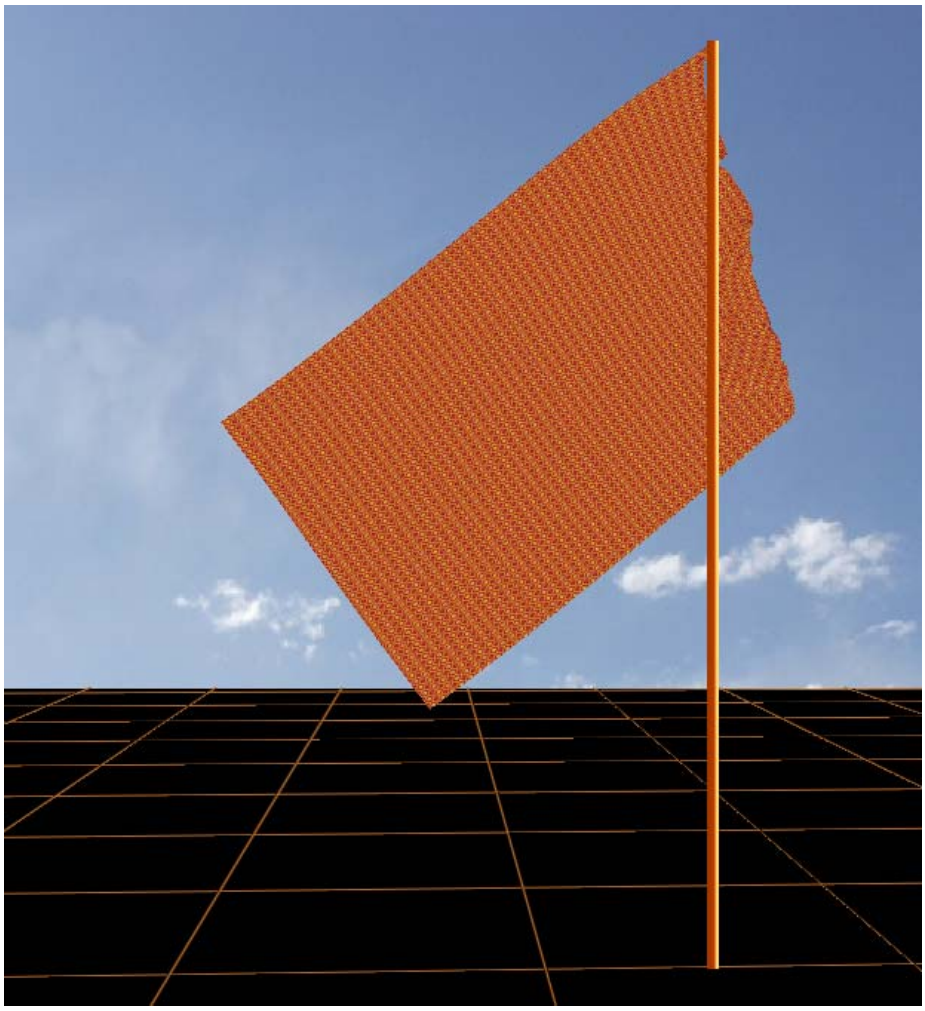
5 - Catálogo de escenas

Hemos llevado a cabo la creación de varias escenas distintas sobre las que poner a prueba nuestro motor de física.

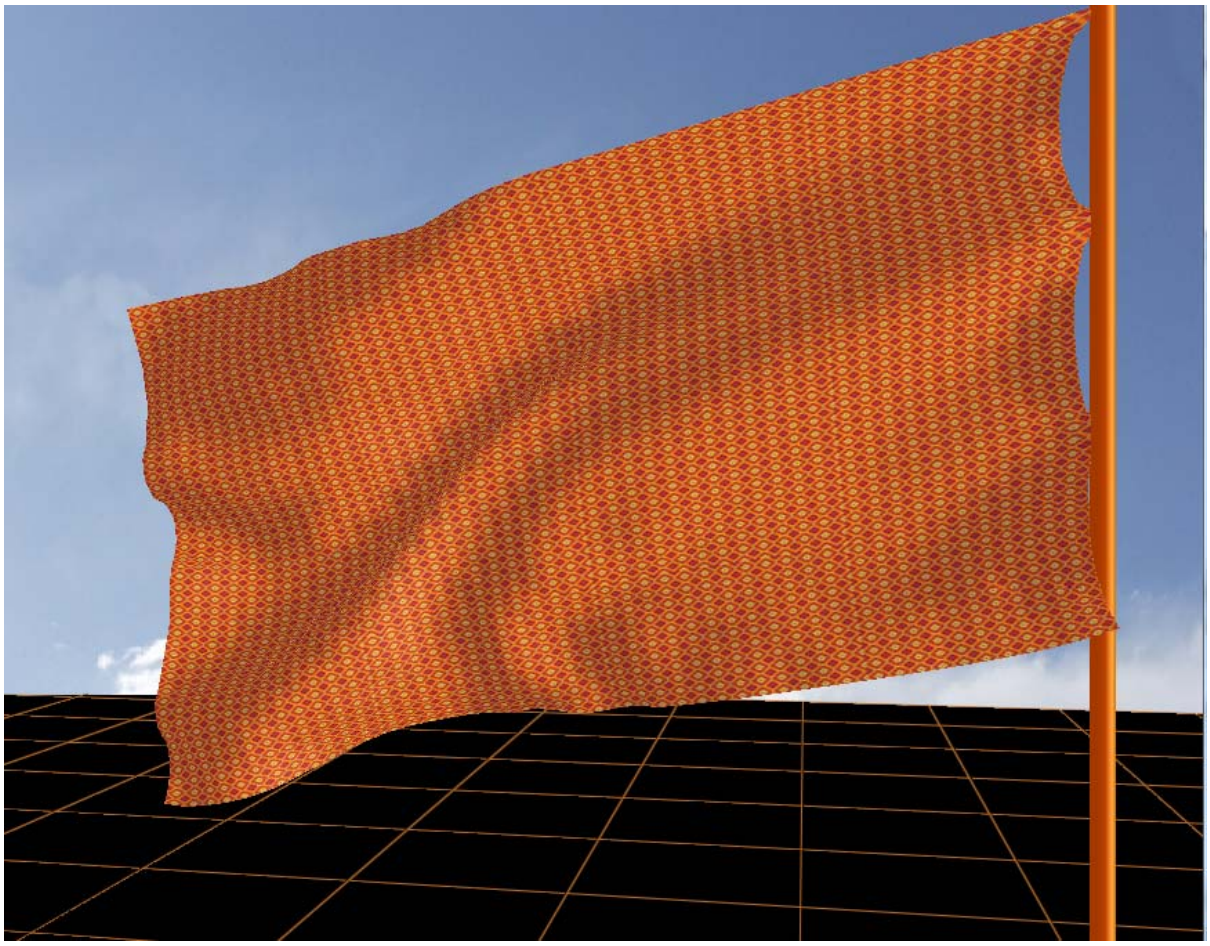
5.1 - Bandera

Esta escena realiza a cabo la simulación del efecto que el viento provoca sobre una bandera.

En dicha escena, disponemos un mástil sobre el suelo, perpendicular a él, y situamos sobre él cuatro puntos fijos de la tela. La tela inicialmente sólo se encuentra sometida a la fuerza de la gravedad.

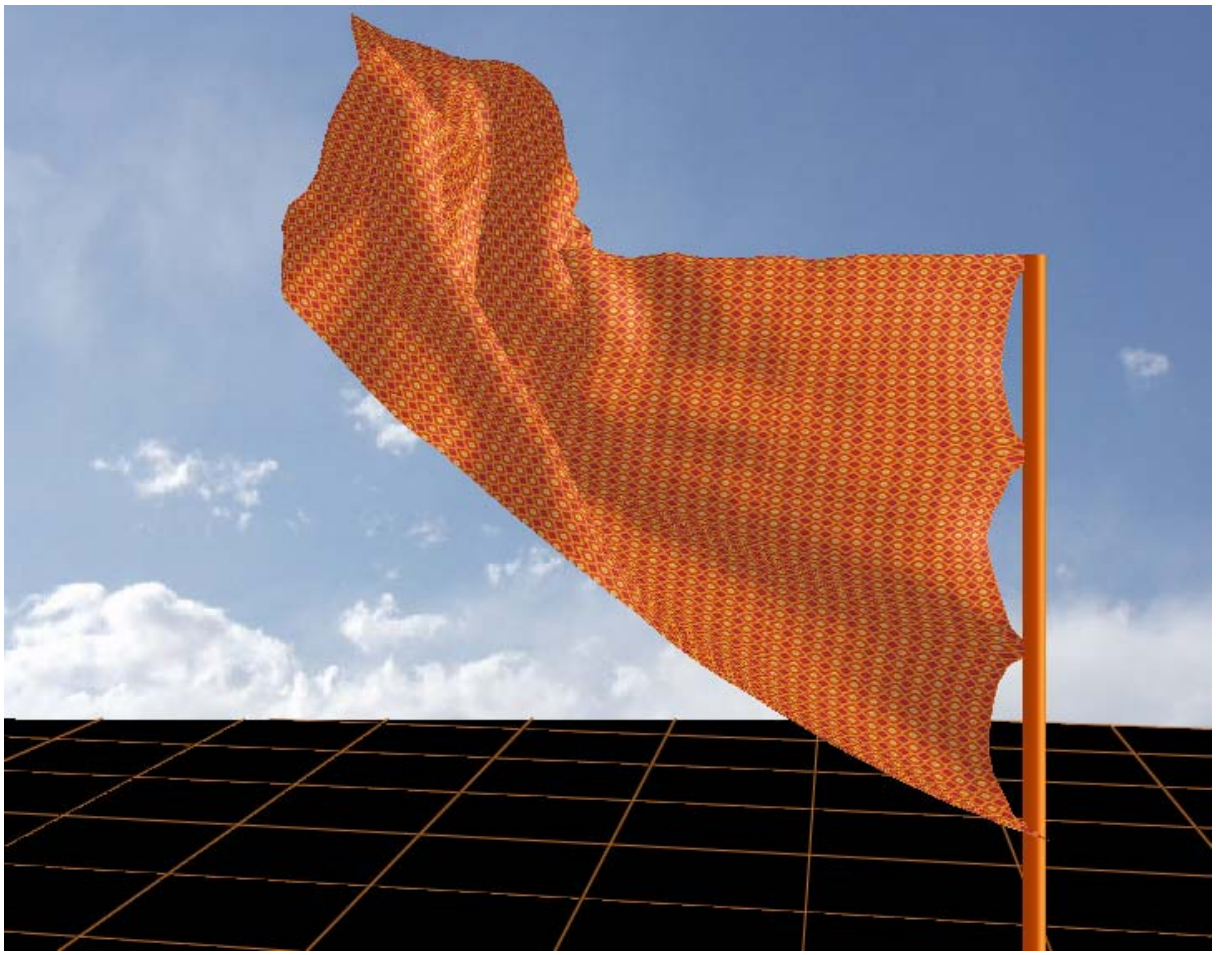


A continuación le aplicamos un viento gradual hasta alcanzar una cierta velocidad. Adoptada esa velocidad, se deja fija.



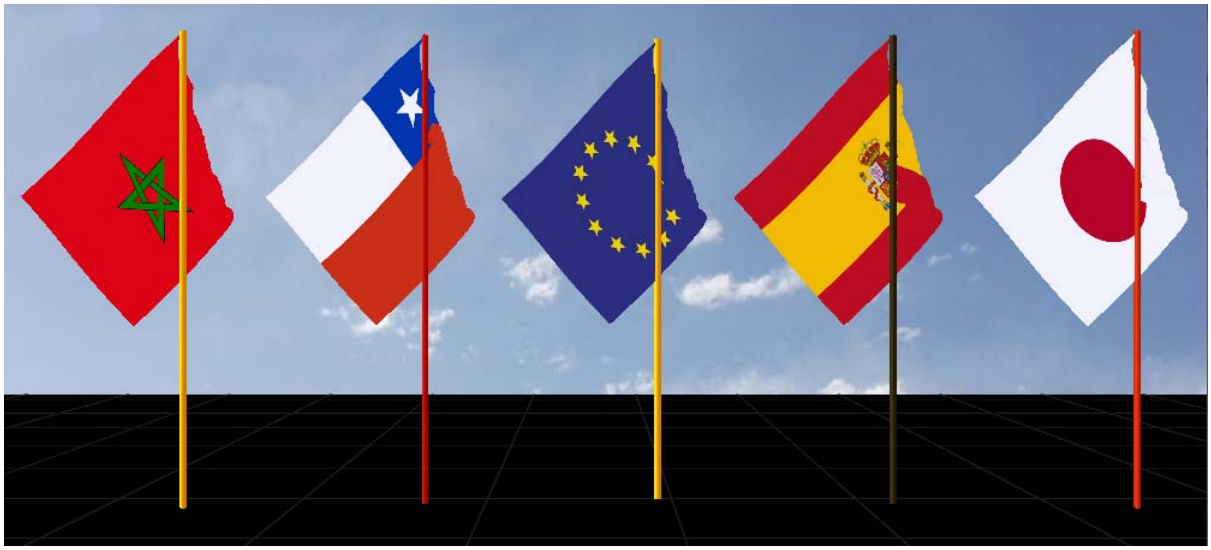
Este viento constante lo hemos implementado como una fuerza constante perpendicular a la de la gravedad. Para otorgar mayor realismo a la imagen, este viento lo racheamos cada cierto tiempo. De este modo conseguimos que la tela caiga un poco pasado ese tiempo y acto seguido se vuelva a recuperar.

Además, hemos configurado el teclado para que manteniendo pulsada una tecla, el viento se corte de raíz permitiendo que la tela caiga completamente y al ser de nuevo liberada retome de nuevo y de golpe la fuerza con la que ondeaba, consiguiendo un movimiento brusco muy realista.



5.1.2 Varias Banderas

Utilizando el modelo de bandera del apartado anterior, hemos decidido crear una escena con varias banderas.



La escena consta de cinco banderas distintas en apariencia, pero con las mismas propiedades. En un momento dado, el usuario puede pulsar sobre un botón que hemos configurado para someter a la bandera a los efectos de la fuerza del viento.

Para solventar el grado de irrealismo que surge cuando al estar sometidas las cinco banderas a la misma fuerza, y teniendo las mismas propiedades, se mueven las cinco del mismo modo debido a que el viento no llega al mismo tiempo con la misma fuerza o sentido a cada punto de ellas, hemos creado un efecto de viento racheado. De este modo cada bandera se mueve de manera independiente de las demás. Esto quiere decir que cada cierto tiempo se detiene la fuerza del viento sobre una bandera durante unos instantes, mientras que en las otras, esto se hace en otros instantes distintos.

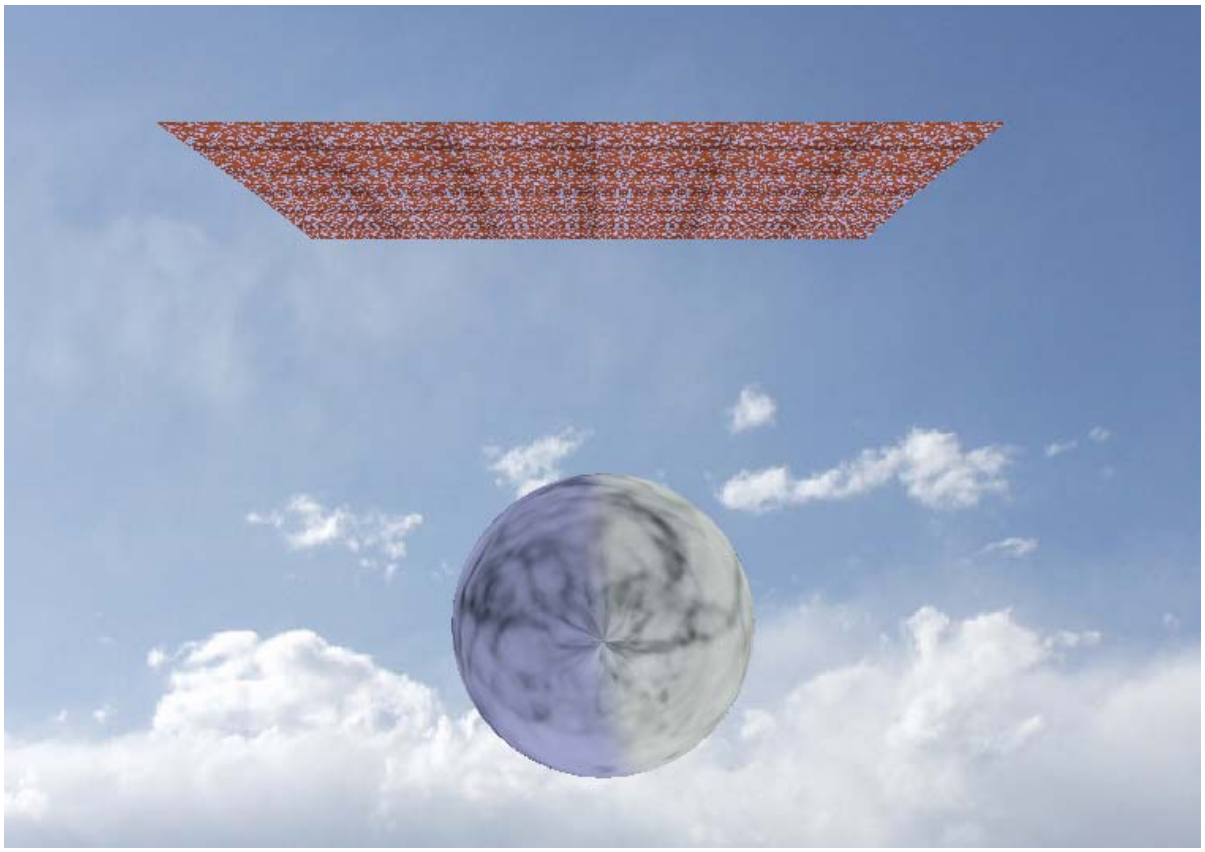
De este modo, hemos conseguido representar una escena muy común, como la que se puede apreciar en cualquier reunión diplomática, en dependencias militares o gubernamentales, ayuntamientos, centros públicos...

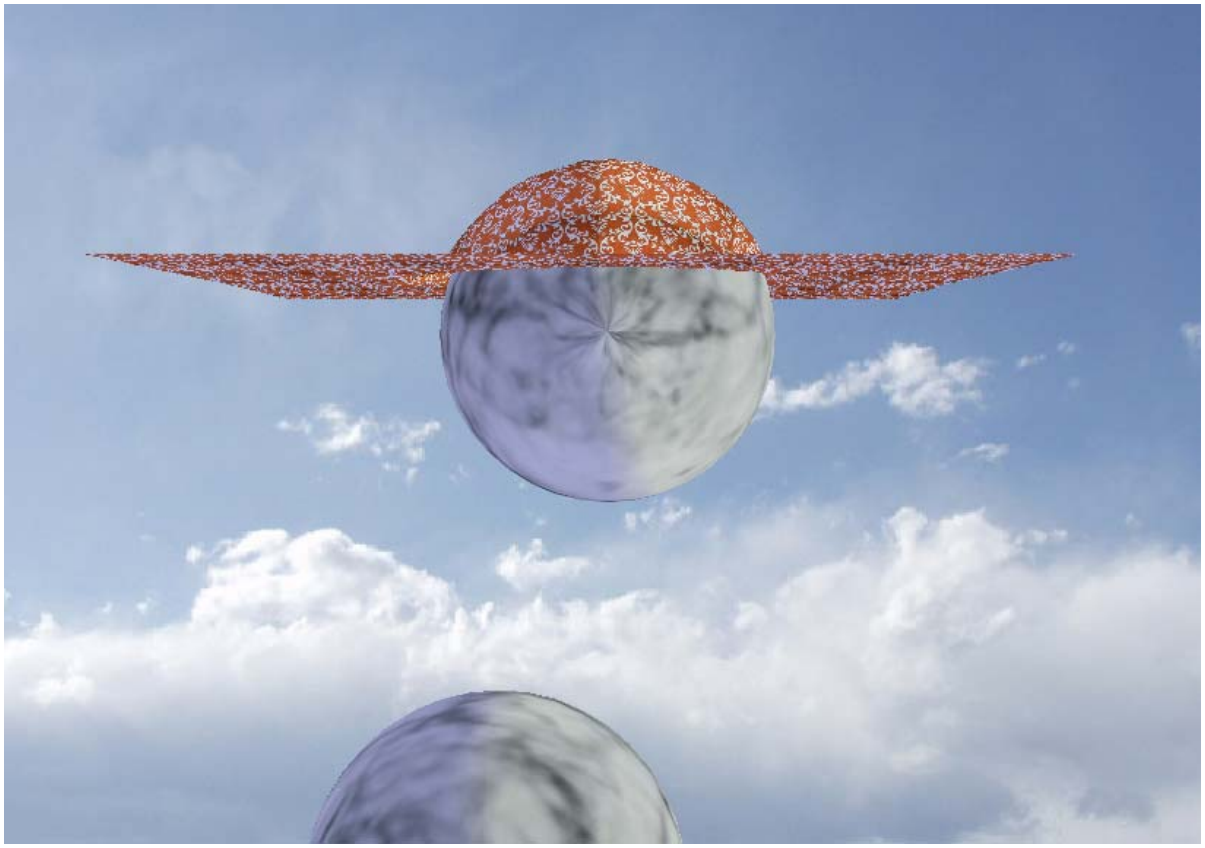
5.2 - Caída de una tela sobre esferas

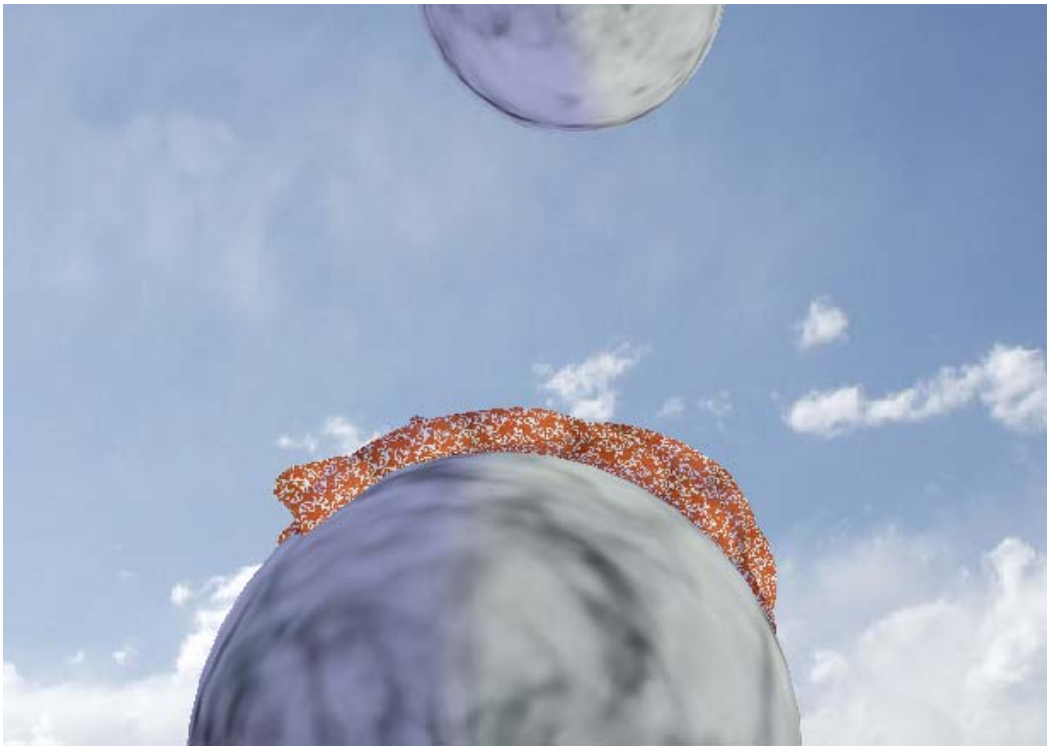
En esta escena entran en juego las esferas, y consecuentemente las colisiones de la tela con ellas.

La escena consta de una tela que se suelta en caída libre desde una posición inicialmente horizontal. Debajo de ella y a distintas alturas, se encuentran varias esferas fijas situadas estratégicamente para que la tela colisione contra ellas y su rumbo se vea afectado por dichas colisiones.

En esta escena cobra gran importancia la gravedad, ya que es la única fuerza que actúa sobre la tela mientras no se tope con ninguna esfera. Es la principal precursora de la continua caída de la tela.





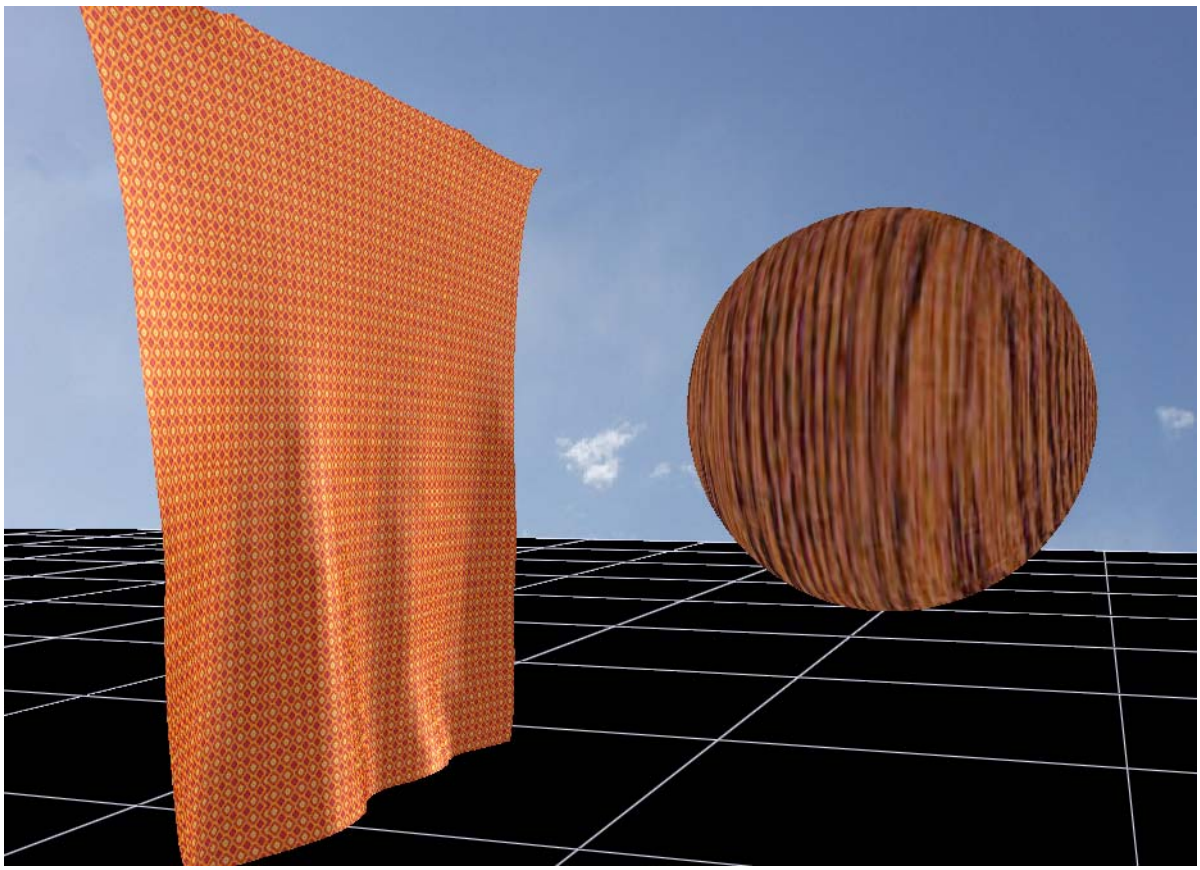


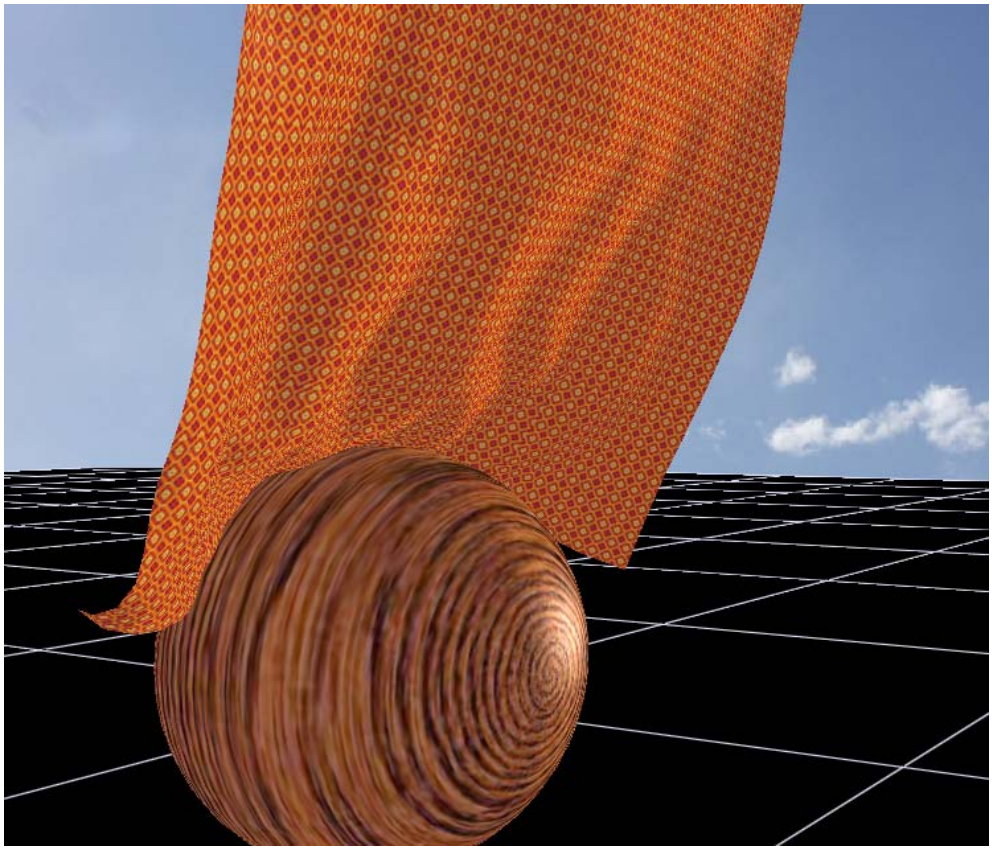
5.3 - Colisión de una esfera con una cortina

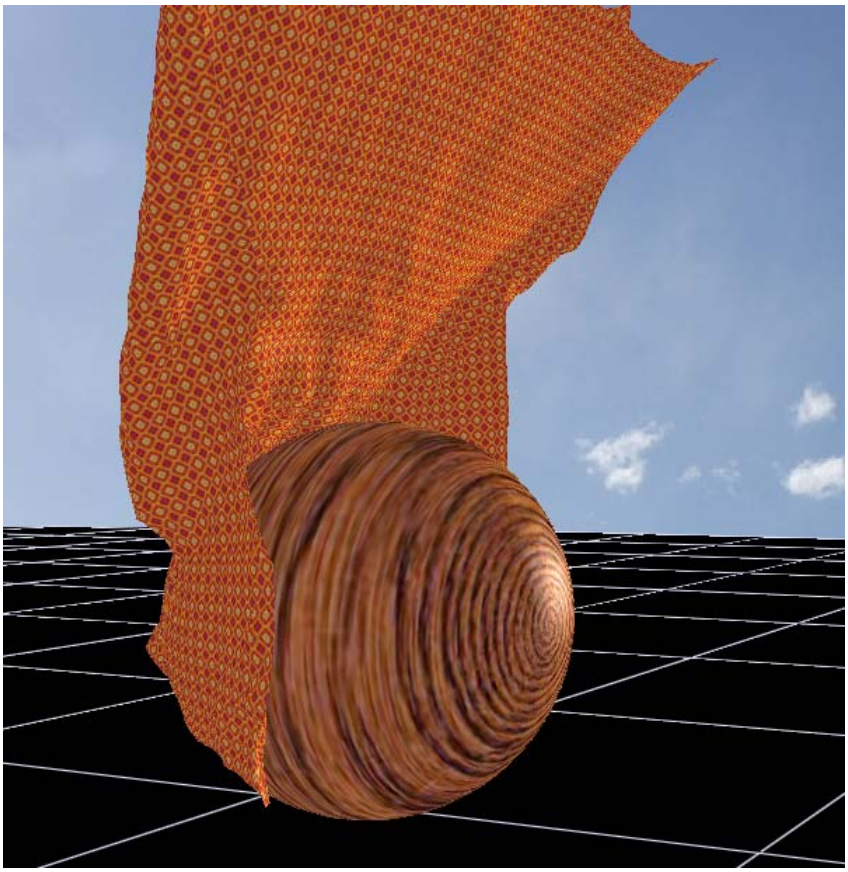
Esta escena la hemos creado con el objeto de probar las colisiones de una tela con una esfera en movimiento.

La escena consta por un lado de una tela colgada por la parte superior a modo cortina (fijamos varias de sus partículas superiores). La tela se ve afectada por la fuerza del viento. Por otro lado se encuentra una esfera que dispone de movilidad.

Además, en esta escena se otorga al usuario la posibilidad de manejar la esfera con el teclado. Dispone de teclas para hacerla avanzar, retroceder, y elevarse. Cabe destacar que la esfera también se ve afectada por el efecto de la gravedad y por el rozamiento con el medio.

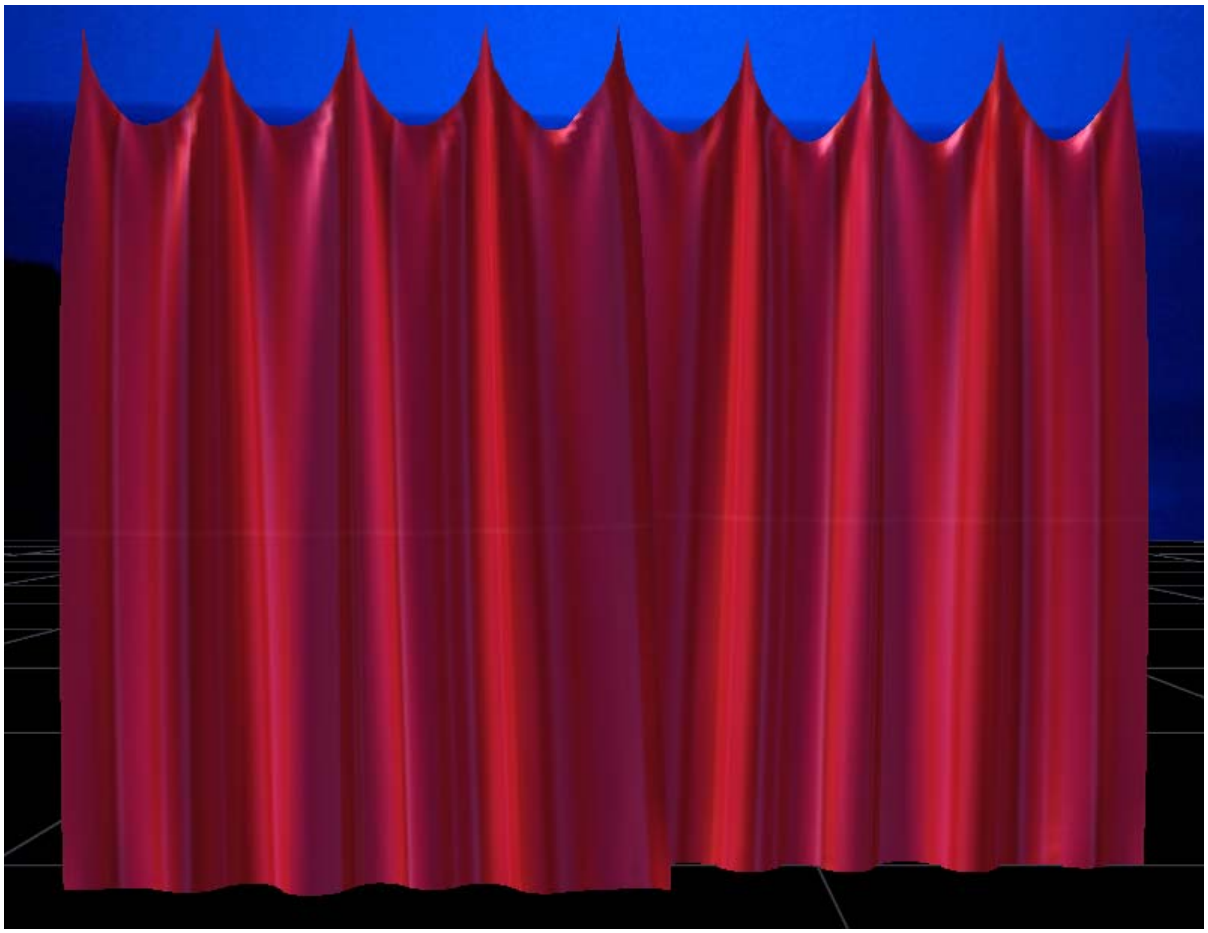






5.4 – Telón

Hemos realizado una escena que simula la apertura y cierre de un telón.

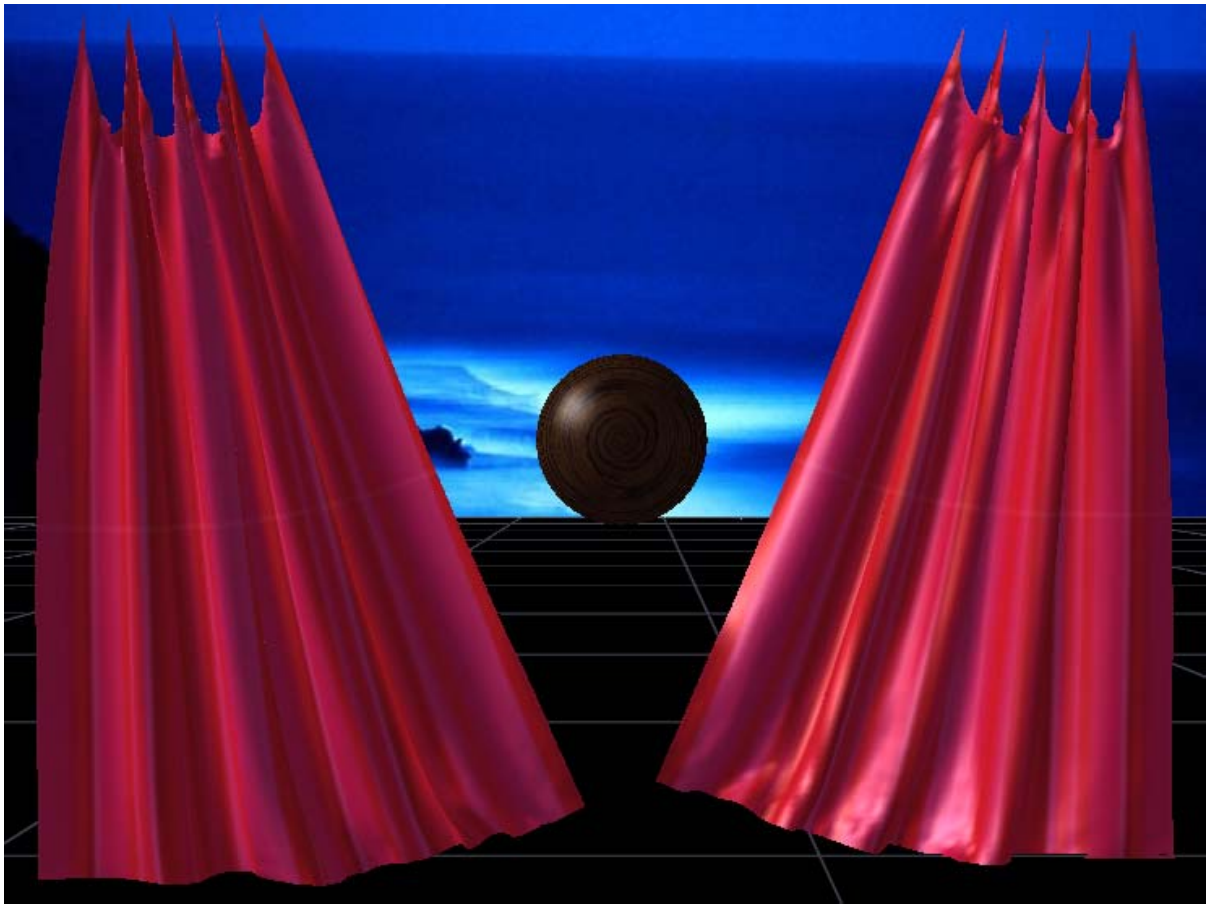


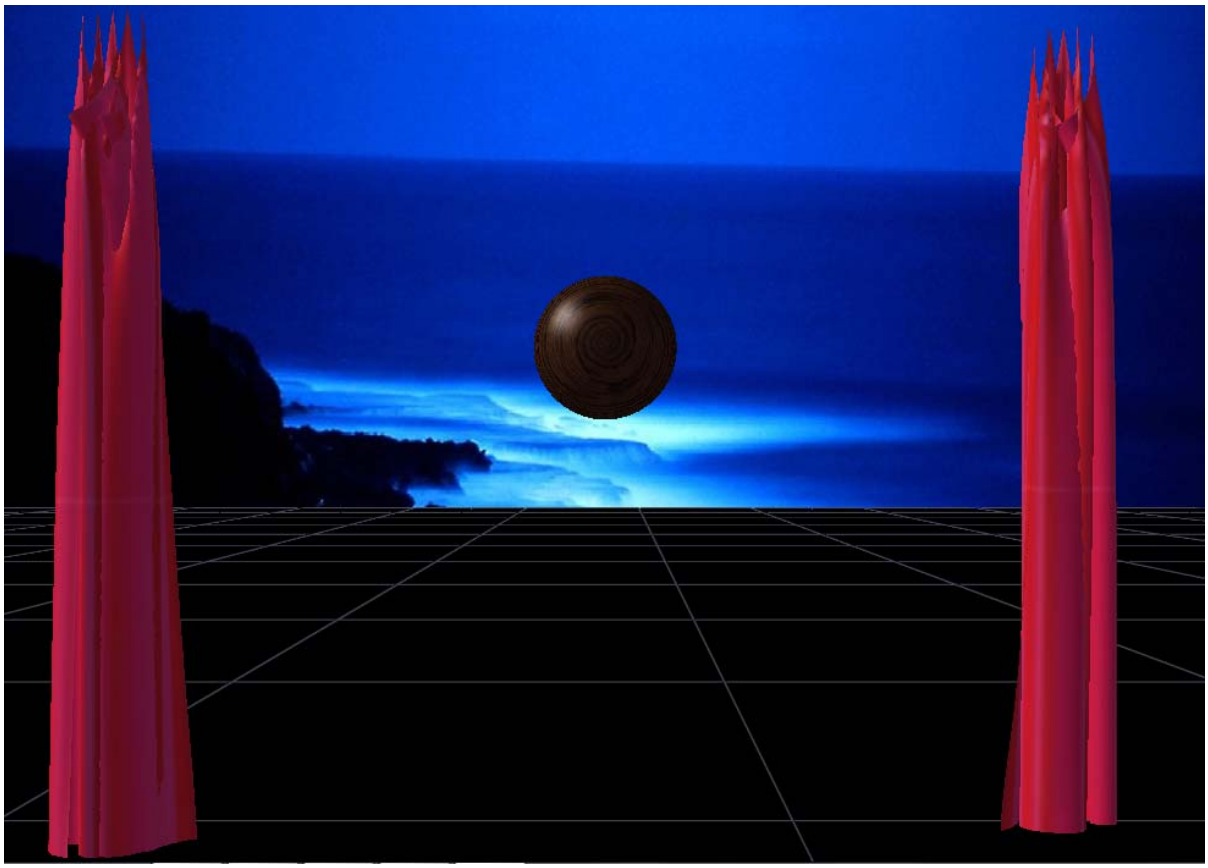
Para la realización de esta escena, hemos generado una tela fijada por cinco partículas, situadas a una distancia menor que la que les correspondería si la tela estuviese estirada.

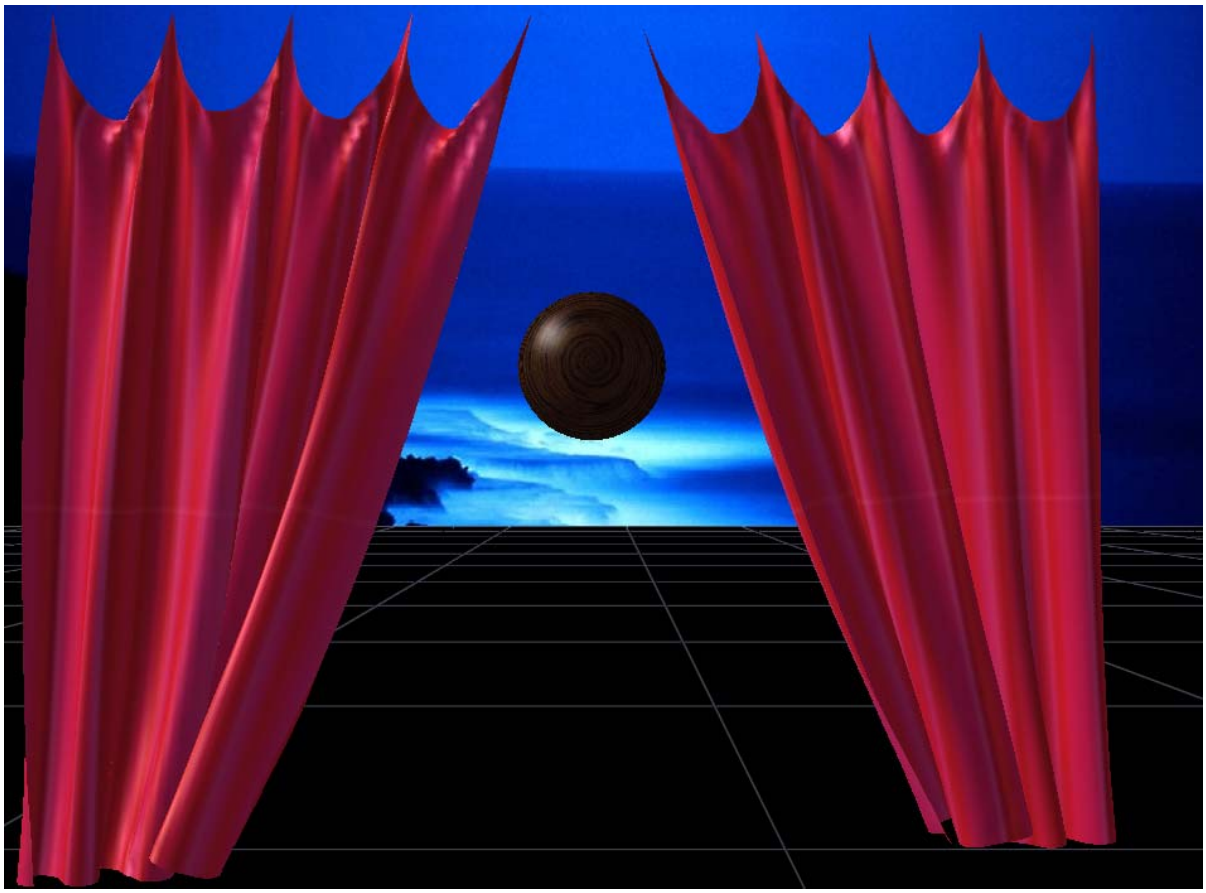
Hemos replicado esa misma tela, pero la hemos girado 180 grados, con el objeto de crear la segunda tela del telón, y que con el mismo toque de teclado se muevan las dos al unísono. De este modo, al pulsar sobre uno de los dos botones de teclado configurados para el movimiento (apertura o cierre del telón), ambas realizarán el mismo movimiento, pero de forma simétrica, produciendo el efecto deseado.

Para configurar el movimiento, hemos seleccionado dos teclas, de modo que al pulsar sobre la de “apertura” (basándonos en la tela de la izquierda), la primera partícula de la izquierda no se mueva, la segunda adopte una cierta velocidad, la tercera tome como velocidad el doble de la de su izquierda, la cuarta el doble de velocidad de la tercera, y de manera semejante la quinta respecto de la cuarta.

Además, dado que la tela de un telón suele ser una tela robusta y gruesa, con movimiento lento y noble, hemos variado el parámetro de rozamiento de la tela para ofrecer mayor resistencia al medio.



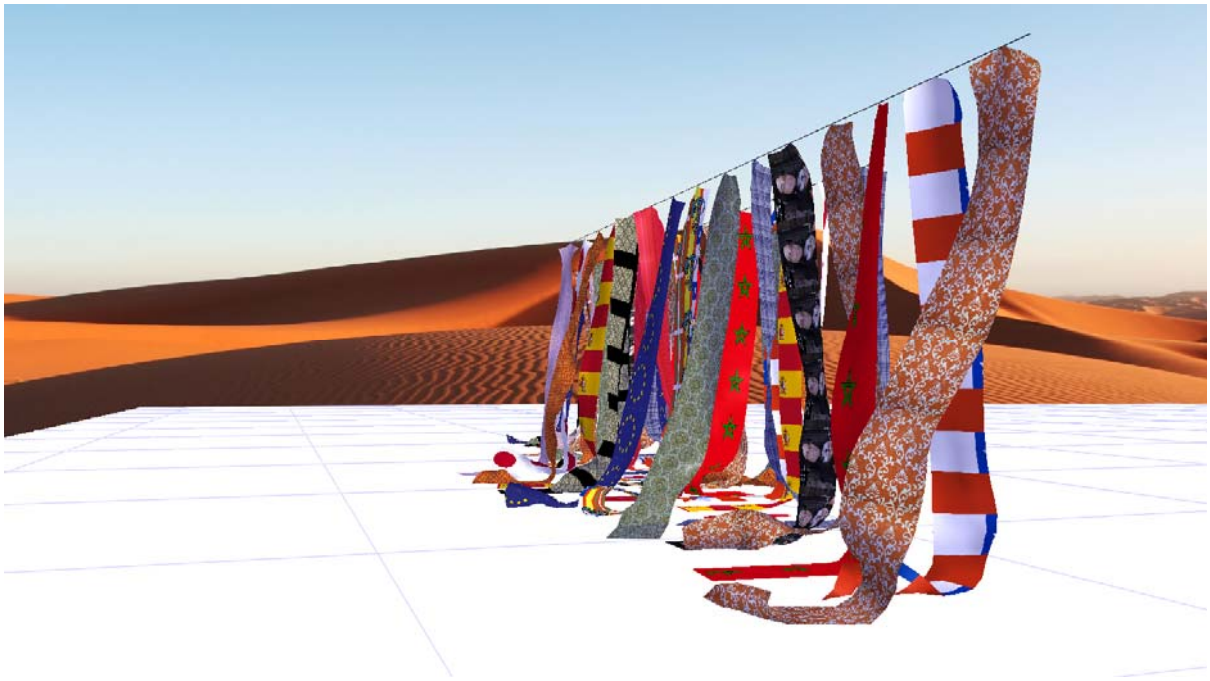




5.5 - Escena artística

Hemos echado a volar nuestra imaginación y hemos creado una escena que consideramos “artística”, por su rareza y su belleza.

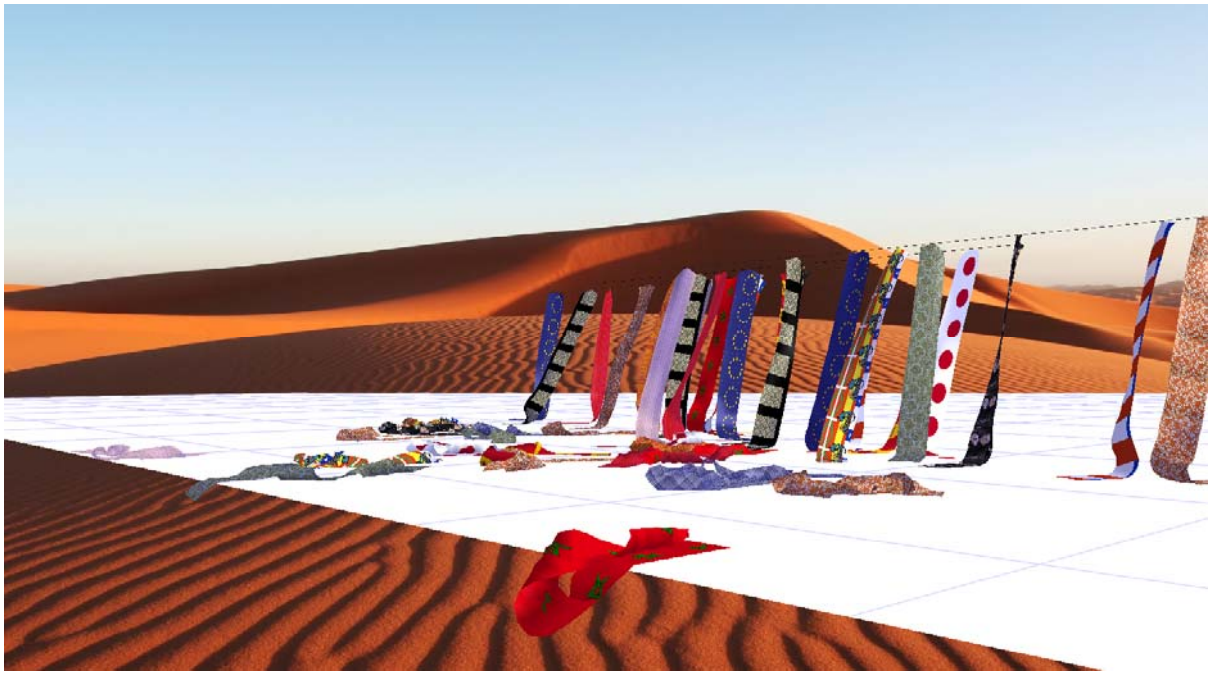
La escena consta de varios mástiles posicionados horizontalmente y en paralelo en el aire, sobre un suelo blanco que podría compararse con un Tatami, bajo el que se encuentra “la nada”. De esos mástiles cuelgan decenas de tiras de tela con diferentes iconos que junto con el fondo contrastan provocando un notado impacto visual muy vistoso.

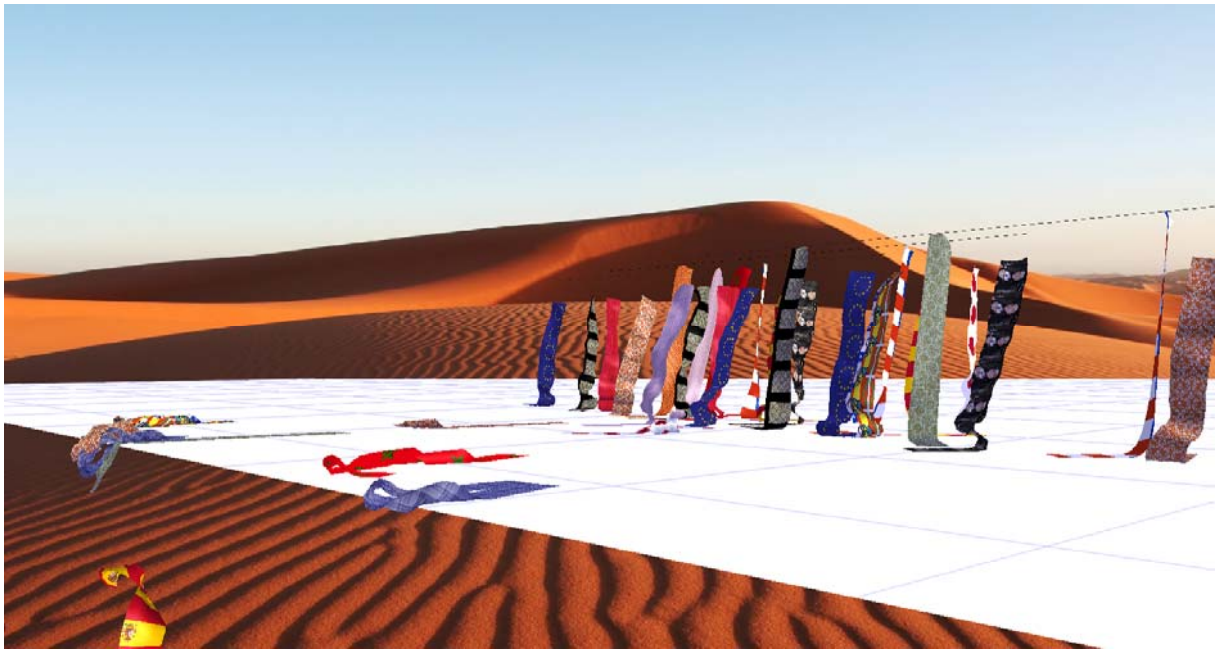


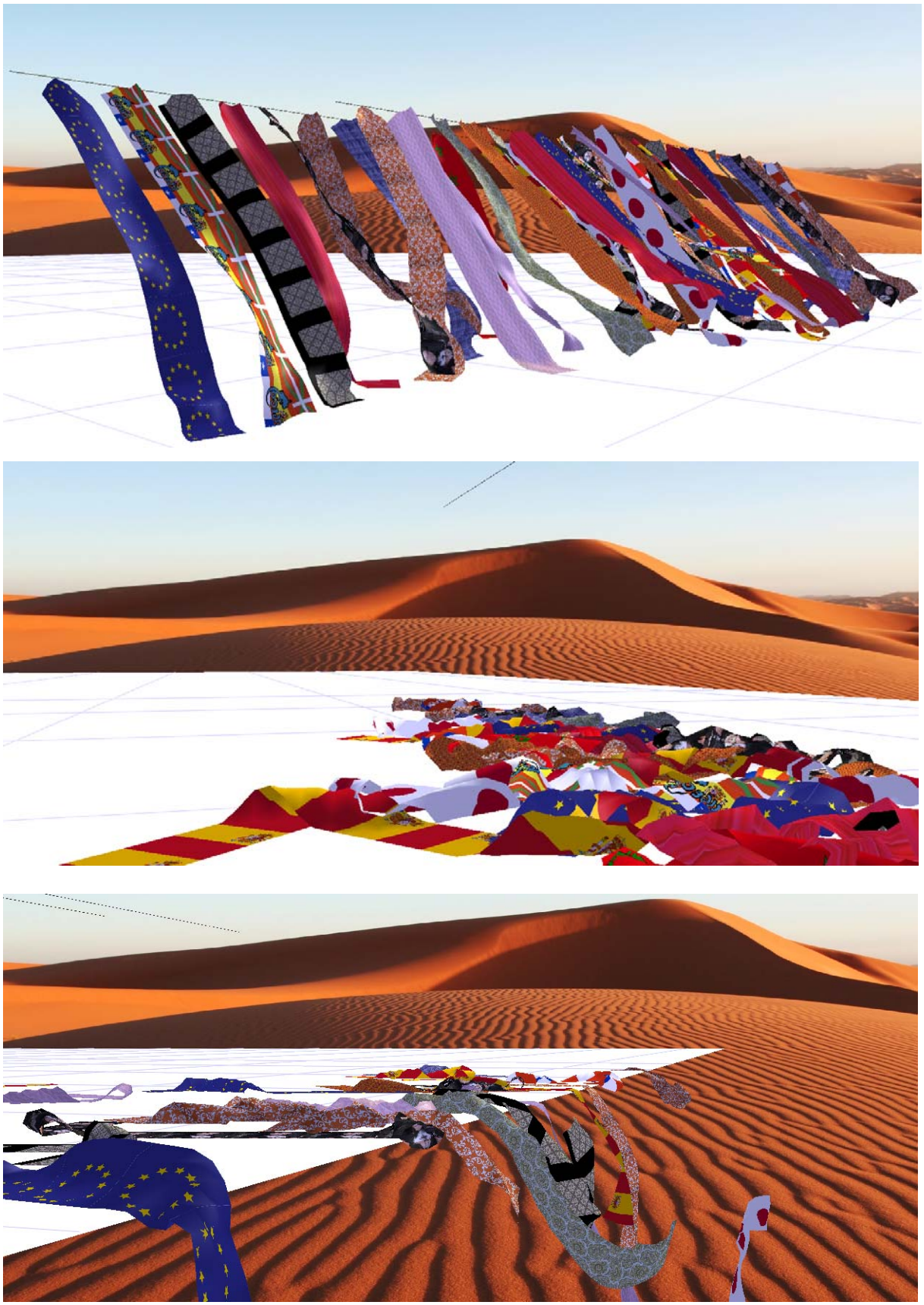
Hemos configurado la escena para que a golpe de teclado, las tiras de tela se vayan desprendiendo de sus respectivos mástiles y se dejen llevar por la fuerza del viento a la que las hemos sometido.

Las telas caen al suelo, donde se posan mostrando la colisión que se produce en ellas con una superficie plana. Dado que la fuerza del viento es superior a la fuerza de rozamiento del suelo, las telas se ven arrastradas por el mismo hasta que salen fuera de sus límites. Al salir de

los límites del suelo, y no encontrar más fuerzas que la gravedad y el viento, continúan con su caída hacia el vacío.







6 - Metodología de trabajo

Para realizar el proyecto se ha decidido optar por una planificación basada en la metodología de programación extrema XP (*Extreme Programming*), ya que por sus características era la que mejor se adaptaba al desarrollo de nuestro proyecto.

Como es evidente no todas las características son aplicables al presente proyecto, por lo que a continuación se irán explicando cómo las hemos ido adaptando a nuestro trabajo "Simulación 3D de Superficies Deformables".

6.1 - Adaptación de *eXtreme Programming* a nuestro proyecto

6.1.1 – ¿Por qué XP?

XP es una metodología ágil basada en la adaptabilidad de cualquier cambio para aumentar las posibilidades de éxito de un proyecto. Esto nos otorgaba flexibilidad en caso de tener que volver atrás ante cualquier problema que nos pudiese ir surgiendo.

Para emplear XP, nuestro proyecto debía cumplir una serie de características:

- Interés sincero por todas las partes en que el proyecto tenga éxito.
- El equipo de trabajo es pequeño (en este caso serían cuatro personas incluyendo a nuestro tutor).

- El equipo dispone de capacidad y ganas de aprender. Imprescindible para llegar a buen puerto al partir de cero en cuanto a conocimientos de la modelación y simulación de sólidos deformables.
- El proyecto tiene un riesgo alto en cuanto a lo innovador de la tecnología. Puesto que era un tema del que apenas teníamos conocimientos era imprescindible disponer de una metodología que nos permitiese volver atrás en un momento dado.

6.1.2 – Desarrollo

En la programación extrema dentro del equipo hay diferentes roles: un equipo de gestión, uno de desarrollo y los clientes finales. Cómo es lógico, en nuestro caso hemos tomado los roles de gestión y desarrollo, siendo nuestro tutor el que ha jugado la parte de cliente.

- **Relación con el cliente (tutor en nuestro caso)**

  En el desarrollo de nuestro proyecto, la labor del cliente (tutor) ha sido fundamental. Ha sido el encargado de darnos al comienzo las pautas a seguir dentro del campo sobre el que decidimos movernos: los sólidos deformables.

  Debido a la falta de experiencia por parte del equipo en el tema de la modelación y simulación de sólidos deformables, ha puesto a nuestra disposición toda la información y documentación necesaria cuando así lo requeríamos, así cómo una disponibilidad absoluta a lo largo del proceso.

  En función de las reuniones mantenidas con el tutor a lo largo del año, hemos ido planificando los diferentes *releases* con la idea de que la duración de cada uno de ellos fuera de

dos a tres semanas, al no tener disponibilidad absoluta en la realización del proyecto. Sin embargo, en ocasiones las dificultades que fueron surgiendo ampliaron la duración de algunos de los *releases.*

Se ha ido informando al tutor de las características de cada uno de los releases, mostrando los resultados obtenidos. El tutor decidía si había que realizar cambios en el diseño y comprobaba que habíamos alcanzo los objetivos propuestos.

Como conclusión a este punto diremos que la relación entre el equipo y el cliente ha sido muy buena y en ello ha contribuido mucho la confianza depositada, y la ayuda recibida desde el primer momento, por parte del cliente.

- **Planificación del proyecto**

Hemos elaborado una planificación basada en seis releases, intentando que la duración de cada una ellos tuviese una duración de un máximo de tres semanas.

Al principio la idea fue llegar al objetivo final a partir de cinco releases, pero al quedar decepcionamos con el resultado final de uno de los releases optamos por añadir otro nuevo para mejorar los resultados.

Los releases finalmente han sido:

1. *Investigación*

   El tutor nos proporcionó documentación acerca de los diferentes modelos de simulación para la

animación de telas. Finalmente optamos por implementar el modelo de Kawabata.

2. *Implementación de Kawabata*

   Desarrollo del modelo de Kawabata. Tras finalizar la implementación quedamos bastante decepcionados con el resultado final, por lo que se decidió añadir un nuevo release.

3. *Implementación mediante muelles*

   Tras estudiar nuevas posibles implementaciones, nos decidimos por una basada en muelles tomando como base las ideas adquiridas del anterior release.

4. *Implementación de escenas*

   Tras quedar satisfechos con los resultados obtenidos el siguiente paso sería diseñar una serie de escenas para mostrar las simulaciones los más vistosas posibles.

5. *Generación de documentación*

   Generación de la documentación a través de una memoria.

6. *Mejoras*

   Tras finalizar todos los releases de manera satisfactoria y habiendo alcanzado el objetivo inicial propuesto por parte del tutor, decidimos ampliar los

objetivos iniciales (Colisiones, optimización de rendimiento, etc.).

Para la realización de cada release ha sido necesaria una reunión para poner los objetivos (tras una reunión previa con el tutor), y una reunión una vez finalizado el objetivo para analizar los resultados obtenidos.

Hemos quedado muy satisfechos con el buen funcionamiento del equipo a lo largo de todo el proyecto. La comunicación ha sido constante con el fin de hablar de los problemas a los que nos enfrentábamos y de ver cómo se iba desarrollando el trabajo.

Se ha utilizado como herramienta de comunicación Google Groups.

- **Diseño, desarrollo y pruebas**

El *desarrollo* ha sido la parte fundamental del proyecto al tratarse de un proyecto de software. Para llevar a cabo el proceso de desarrollo hemos trabajado en equipo la mayoría de las veces.

Sin embargo, al no tener disponibilidad absoluta para trabajar en equipo, hemos dispuesto del correo electrónico y de la herramienta de Google Docs a la hora de compartir documentación como la memoria del proyecto.

Hemos utilizado un repositorio gratuito alojado en www.xp-dev.com.

Para compartir el código hemos utilizado el control de versiones Subversion.

El *diseño* se ha ido revisando y mejorando de manera continuada a lo largo del proyecto según se han ido añadiendo funcionalidades al mismo.

Cada vez que se ha ido añadiendo código al proyecto se ha fijado una *prueba* que debía pasar tras finalizar la implementación. Así hemos conseguido mantener un código estable y no tener que hacer cambios muy bruscos en su diseño.

Como hemos comentado anteriormente, el único cambio reseñable a nivel de código fue durante el release de la implementación del método de Kawabata. Fueron pasando las

pruebas correctamente hasta llegar casi al final con la implementación del trellising, cuya prueba no pasó los resultados esperados y repercutió en toda la implementación anterior.

Las *pruebas* han sido una parte fundamental en el desarrollo del proyecto y en obtener un código simple y funcional de manera bastante rápida.

Aunque XP aconseja trabajar en parejas, al ser un equipo de tres personas nos hemos tenido que adaptar a las circunstancias y el funcionamiento ha sido muy positivo y eficaz.

7 - Aplicación de nuestro estudio y conclusiones

7.1 Aplicaciones

Una vez puesto en escena nuestro modelo de tela y observado los resultados, podemos comprobar las distintas aplicaciones de nuestro proyecto.

La principal aplicación es la representación de diferentes tipos de textiles en videojuegos o cine de animación. Basándonos en las escenas que hemos creado:

- Las banderas son un objeto que aparecen en cualquier escena de videojuego de acción o de aventuras, o de cine de animación que discurra en torno a cualquier horda o grupo de gente organizada. Del mismo modo, cualquier videojuego deportivo que se precie suele estar basado en campeonatos o eventos deportivos reales, por lo que las banderas resultan también indispensables en este tipo de escenas. A pesar de no ser la bandera un objeto relevante para el transcurso del juego, sí se trata de un componente más que, aún pasando desapercibido, puede afectar seriamente en el realismo y calidad del videojuego.

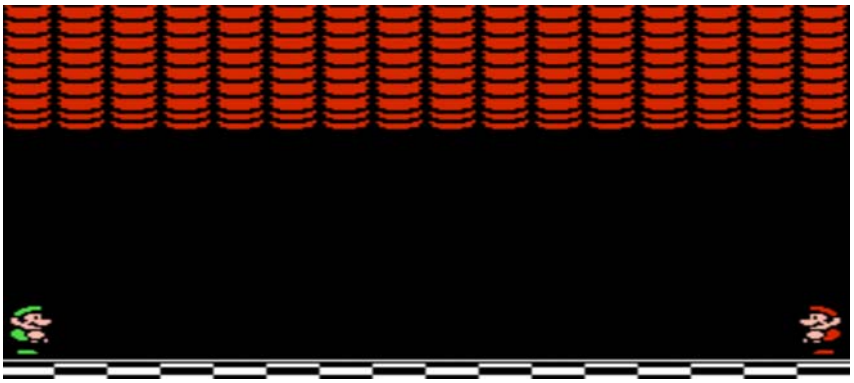
- La caída de telas con las colisiones también forman parte de diferentes escenas de videojuegos o de cine de animación. Un paracaídas cayendo, una alfombra mágica, un turbante que en una película basada en el mundo árabe cayendo por una colina; o llegando un poco más allá del mundo de las telas, aunque con propiedades en cierto modo similares, una pluma que se mueve por efecto del viento; son ejemplos de aplicaciones de nuestro trabajo en el mundo de la animación. Nótese que en los ejemplos dados, los objetos son bastante más relevantes que las banderas del punto anterior, por lo que con más razón se debe cuidar el realismo en el movimiento de la tela para dotar de mayor fiabilidad e impacto visual al espectador.

- Telas sujetas por ciertos puntos que cuelgan y son sometidas a colisiones con objetos se encuentran presentes en multitud de

escenas de animación. Por ejemplo, en el ámbito de los videojuegos deportivos se encuentran en casi cualquier juego: una red de un campo de tenis, vóley ball o vóley playa; una portería de fútbol, balonmano o wáter polo; una red de una canasta de baloncesto… Resulta indispensable para estos casos que el efecto producido sobre las redes sea completamente realista, ya que se encuentran en pantalla en todo momento, y con estos objetos se producen interacciones continuamente.

Del mismo modo es frecuente ver telas colgando en aventuras gráficas que se encuentran en contacto con los personajes y objetos dinámicos de las escenas, por ejemplo las telas que cuelgan en las laberínticas calles de un bazar árabe, un tendedero de ropa sobre el que cuelga el uniforme de un personaje militar, una tela colocada a modo de puerta en una tienda de campaña india…

- Los telones en animación resultan muy frecuentes. Por ejemplo, en videojuegos que permiten al jugador "tocar" un instrumento musical, cantar a modo karaoke, o bailar delante de una cámara que traduce sus movimientos físicos en digitales suelen abrir y cerrar un telón como inicio y fin de la prueba artística del jugador. También se utilizan mucho con imagen de inicio de muchos videojuegos, a modo de entrada a los mismos.



También es frecuente en películas y series de animación ver un telón, por ejemplo en cualquier escena de personajes que tocan instrumentos musicales, realizan juegos con marionetas u obras de teatro…

- Aplicaciones de nuestra "escena artística" caben muchas en nuestra imaginación, aunque no conseguimos especial nitidez en ninguna de nuestras abstractas ocurrencias. No obstante, la

imaginación es un arte y puede dar lugar a escenas dalianas, que brillan por su rareza, imaginación e innovación. La estructura que ofrece nuestra implementación de las telas, permite con gran facilidad llevar a cabo cualquier rareza imaginable.

En cuanto al ámbito de una empresa textil, este simulador de telas podría tener relevancia para comprobar de antemano las propiedades en el movimiento de las prendas y sus respectivas texturas.

Basándonos en las escenas creadas en nuestro simulador, las aplicaciones de nuestro proyecto en este ámbito textil son relativamente escasas, aunque sí que podrían darse uso en el campo textil espectáculo. En una obra de teatro, se busca la veracidad en las escenas representadas. Para ello cualquier tela que aparezca en escena debe cuidarse minuciosamente para su puesta. De este modo, y dado que este tipo de telas tienen un elevado coste, nuestro simulador podría ayudar a definir las propiedades de la tela antes de ser manufacturadas, ahorrando fabricaciones inútiles con su consecuente pérdida de dinero.

7.2 Conclusiones

La realización de este proyecto nos ha permitido conocer un mundo que para nosotros resultaba completamente ajeno, y en el que éramos en cierto modo ignorantes; que realmente nos ha parecido interesante y motivador.

El mundo de la simulación del movimiento de superficies deformables requiere tener conocimientos ya no sólo informáticos, sino físicos y matemáticos. Estos requisitos nos han motivado bastante a la realización del proyecto, ya que se trataba de un reto grande por la cantidad de campos científicos que abarcaba.

Hemos comprobado, por lo abstracto que resultaba para nosotros, que iniciarse en un campo completamente nuevo de la informática requiere muchas horas de búsqueda, documentación, estudio y reflexión hasta que se consigue llegar a resultados visibles. Esta investigación de nuevos campos resulta interesante tanto por la ampliación de conocimientos que se adquiere, como por la escapada de la rutina en la que muchos informáticos se sumen en la vida profesional.

8. Bibliografía

Physical Based Techniques - http://davis.wpi.edu/~matt/courses/cloth/physical.html

Predicting the Drape of Woven Cloth Using Interacting Particles - www.uni-weimar.de/~caw/papers/p365-breen.pdf

Computer Animation. Algorithms and Techniques - Rick Parent

OpenGL® Reference Manual: The Official Reference Document to OpenGL, Version 1.4 (4th Edition) - Dave Shreiner

Resorte. Wikipedia - http://es.wikipedia.org/wiki/Resorte

LWJGL - http://lwjgl.org/wiki/index.php?title=About_LWJGL

Análisis vectorial - http://inicia.es/de/csla/vectores.htm

B. P. Demidowitsch. I. A. Maron, E. S. Schuwalowa. *Métodos numéricos de análisis*. Editorial Paraninfo (1980)

Ecuación diferencial de primer orden - http://www.sc.ehu.es/sbweb/fisica/cursoJava/numerico/eDiferenciales/rungeKutta/rungeKutta.htm

C.K. Chan et al. / journal of Materials Processing Technology 174 (2006) 183-189

A Particle – Based Model for Simulating the Draping Behavior of Woven Cloth – David E. Breent, Donald H. House, Michael J. Wozny