Tesis de Licenciatura

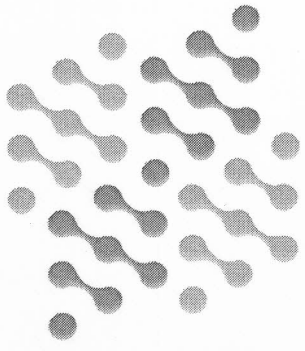
Sistema integral de simulación de prendas en tiempo real

Natalia Davidovich
natty@fibertel.com.ar

Martín Massera
mmassera@dc.uba.ar

Director
Dr. Claudio Delrieux
usdelrie@criba.edu.ar

Año 2006



DEPARTAMENTO DE **COMPUTACIÓN**

Facultad de Ciencias Exactas y Naturales - UBA

Universidad de Buenos Aires – Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Resumen

La simulación de prendas de vestir se ha convertido durante la última década en un tema de gran relevancia para la computación gráfica. Tiene aplicación directa en diversos contextos como el diseño de indumentaria, la industria cinematográfica y desarrollo de videojuegos.

Una prenda de vestir está compuesta principalmente por tela. El modelo más utilizado para simular tela es el denominado *sistema masa-resorte*. Las tradicionales soluciones numéricas propuestas para resolver los sistemas masa-resorte han sido, por distintas razones, poco eficientes.

Muchos métodos alternativos fueron ideados con el objetivo de atacar la falta de eficiencia de las soluciones numéricas tradicionales. La mayoría de ellos lograron un aumento considerable de la eficiencia introduciendo diversas simplificaciones al sistema masa-resorte o utilizando aproximaciones en los resultados numéricos. En términos generales, se logra aumentar la eficiencia a costa de disminuir la calidad de simulación. Esta es la razón por la que ninguno de los métodos propuestos constituye la solución final al desafío de lograr que la simulación pueda ejecutarse en tiempo real.

El presente trabajo estudia el desarrollo de un sistema capaz de simular en tiempo real prendas vestidas por personajes en movimiento. El principal objetivo es conseguir que se aproveche al máximo la capacidad computacional disponible para poder simular la mayor cantidad de tela con el mejor aspecto posible. Se propone un sistema de niveles de detalle (LOD) que dinámicamente evalúa y determina de forma heurística, para cada porción de tela, las características de simulación más adecuadas. De esta manera, es posible distribuir la capacidad computacional disponible para así alcanzar un equilibrio entre calidad de la simulación de cada porción de tela y la cantidad de tela a simular. Como resultado se obtuvieron incrementos de performance de entre 250% y 500%, sin mayores sacrificios de calidad.

Abstract

The simulation of garments has become a subject of great relevance into the computer graphics community during the last decade. Garment simulation is extensively used in several different contexts such as garment design, film industry and videogames.

A garment is essentially made of cloth, and the mostly used model for cloth simulation is the mass-spring model. However, the traditional computational solutions for mass-spring systems have always been not efficient enough for real time applications.

Thus, many alternative methods have been devised in order to solve de lack of efficiency in the traditional numeric methods in mass-spring models. Most of these methods achieve a better performance by introducing several simplifications to the original unrestricted model, or by approximating the computational steps of the traditional methods. As can be expected, the simulation performance is increased at the expense of the quality of the results. For this reason, no single method can be regarded as the definite solution to cloth simulation in real time.

This work presents the development of a system able to simulate moving characters dressed with several garments in real time. The main purpose is to take the most advantage of the available computing power in order to simulate the largest amount of cloth with the best possible quality. We propose an adaptive system of levels of detail (LOD) that dynamically evaluates and heuristically determines, for each piece of cloth, the most adequate simulation method. In this way, it is possible to intelligently distribute the computing power available to achieve the best compromise between quality of simulation for each piece of cloth and amount of cloth to simulate. As a result, the system's performance with LOD increases twofold to fivefold with equivalent quality.

1 Introducción 6
1.1 Propuesta de este trabajo 7
1.2 Motivos para encarar la tesis 8
1.3 Plan de trabajo 9
1.4 Estructura de este documento 9
1.5 Elementos de una simulación de prendas 10
1.6 El modelo de la simulación de prendas 11
1.7 El personaje 11
1.8 Las prendas 12
1.9 El escenario 15
2 Modelo de la tela 16
2.1 Los modelos considerados 16
2.2 Modelo sistema masa-resorte 17
2.2.1 Debilidades 18
2.3 Distintas estructuras de mallas 18
3 Estado del arte 22
4 Evolución del sistema 23
4.1 Integración 23
4.1.1 Breve introducción a la mecánica del modelo. 23
4.1.2 Objetivos de la integración numérica 25
4.2 Métodos considerados 25
4.2.1 Métodos Explícitos 25
4.2.2 Métodos Implícitos 26
4.2.3 Jacobiano constante 26
4.2.4 Thin Objects 27
4.2.5 Evitando la superelasticidad 27
4.2.6 Modelo de Verlet 28
4.2.7 Algoritmo de restricciones 28
4.2.8 Algoritmos no físicos: algoritmo de piel 29
4.3 Colisión 29
4.3.1 Colisión contra volúmenes de frontera 30
4.3.2 Colisión Piel 34
4.3.3 Auto Colisión 34
5 Desarrollo de un sistema integral de simulación de prendas 36
5.1 Características deseadas del sistema 36
5.2 Consideraciones sobre la plataforma 37
5.2.1 Motor gráfico 37
5.3 Pasos en el desarrollo 38
5.4 Objetos base 39
5.4.1 Molde 39
5.4.2 Malla-molde 39
5.4.3 Malla-tela 40
5.4.4 Prenda 41
5.4.5 Tela-simulada 41
5.5 Algoritmos para manipular objetos base 41
5.5.1 Cómo utilizar un molde 41
5.5.2 Cómo crear una malla molde 41
5.5.3 Cómo poner la ropa a un personaje 43
5.6 Arquitectura 46
5.6.1 Requerimientos de la arquitectura 46
5.6.2 Descripción de la arquitectura 46
5.6.3 Manejando los algoritmos 48
5.7 Algoritmos que implementamos 49
5.7.1 Implementación de algoritmos de integración 49
5.7.2 Implementación de algoritmos de colisión 53

5.7.3 Implementación de la auto colisión 57
5.8 Mediciones 58
5.8.1 Plataforma 58
5.8.2 Mediciones simples 59
5.8.3 Mediciones compuestas 61
6 Alcanzando el tiempo real: LOD 63
6.1 Introducción a la animación por computadora. 63
6.1.1 Concepto de tiempo real 63
6.1.2 Frecuencia de refresco 63
6.1.3 La cámara y el observador 64
6.2 LOD: una forma de disminuir el costo de generación de cuadros 64
6.3 Alcances y límites de nuestro LOD 65
6.4 Nuestra propuesta de LOD 66
6.4.1 Unidades de Cómputo (UC) 67
6.4.2 Determinando las UC disponibles 67
6.4.3 Repartiendo el poder computacional 68
6.4.4 Usando el poder computacional 71
6.4.5 Las combinaciones de algoritmos 73
6.5 Resultados 74
7 Conclusiones y trabajo futuro 78
8 Bibliografía 80

1 Introducción

La simulación por computadora de prendas de vestir ha adquirido en la última década gran relevancia en diversas áreas como la computación gráfica y la industria textil. La simulación de prendas por computadora nos permite visualizar y manipular prendas virtuales resultantes de la representación de modelos de prendas reales. El objetivo es obtener la simulación del comportamiento de una prenda de vestir puesta sobre un personaje en movimiento.

Las computadoras se han vuelto una herramienta indispensable para modelar y simular. Como el poder computacional aumenta, los usuarios y las aplicaciones demandan cada vez más nivel de realismo en este tipo de dominios. Esto es particularmente evidente en computación gráfica, donde formas geométricas más sofisticadas y los objetos físicos comienzan a ser modelados en el contexto de complejos ambientes físicos.

La simulación de prendas resulta de gran utilidad para las decisiones de manufactura en la industria de la indumentaria, dado que proporciona una poderosa herramienta al diseñador de indumentaria. Le permite visualizar sobre un modelo virtual con las características corporales más adecuadas las prendas que está diseñando. Esto permite pasar rápidamente del diseño a un prototipo virtual (ver la prenda *en acción*), evitando las dificultades asociadas a crear el prototipo real. También permite cambiar el personaje sobre la que se prueban las prendas, crear distintos talles, y hacer pequeñas variaciones sobre las prendas, ya sea de color, texturas, etc. En definitiva permite agilizar el proceso de diseño, dotándolo de flexibilidad, parametricidad y portabilidad.

Otra aplicación importante es en computación gráfica: para realizar películas de animación, videojuegos o todo tipo de animaciones. En estas áreas se dedica mucho esfuerzo a lograr realismo en las animaciones. Para obtenerlo, es necesario que el movimiento de la ropa se vea real. Se le debe dar un trato cuidadoso a este aspecto dado que, al estar el ojo humano constantemente expuesto en la realidad cotidiana ante la ropa interactuando con el cuerpo que la viste, ha desarrollado gran sensibilidad para distinguir un comportamiento correcto de uno incorrecto.

Un punto crítico de la simulación de prendas es la simulación de las telas asociadas a dichas prendas. Existen muchos tipos de simulación de telas y podemos diferenciar dos grandes ramas: las que tienen por objetivo obtener resultados precisos y las que tienen por objetivo lograr una estética verosímil. Las primeras suelen ser utilizadas para aplicaciones de ingeniería textil, estudios sobre las fibras de una tela, etc. Las segundas están orientadas a producir resultados que ante los ojos de un humano se vean reales.

Las simulaciones que tienen por objetivo lograr una estética verosímil, pueden dividirse en dos subconjuntos: las simulaciones orientadas a animación, que si bien no buscan resultados de precisión absoluta dedican mucho tiempo de procesamiento

a alcanzar la más alta calidad visual, y las orientadas a interactividad, que tratan de simular las prendas lo mejor posible en tiempo real. Nuestro trabajo está centrado en este último tipo de simulaciones, orientándonos a aplicaciones graficas en tiempo real (Ej.: videojuegos) y al diseño de indumentaria asistido por computadora, donde el diseñador necesita no solo ver sino manipular las prendas que creó.

A partir de principios de los 90 comenzaron a realizarse modelos de implementaciones relacionadas con simulación de tela. En un primer momento, el tiempo real era algo lejano dado que los modelos utilizados tenían grandes problemas de inestabilidad y el poder computacional disponible en ese momento no permitía lograr buenos resultados dado que no era posible que la simulación evolucione con pasos lo suficientemente grandes para lograr el tiempo real. A finales de los 90, se idearon métodos de resolución que atacan el problema de la inestabilidad, abriendo el camino hacia la simulación en tiempo real. Sin embargo estos métodos, aunque constituyen un avance significativo, logran simulaciones muy limitadas en tamaño. Los trabajos enfocados en el tiempo real tuvieron como objetivo mejorar la performance, muchas veces con un compromiso en la calidad de simulación. Esto se logra realizando optimizaciones, suposiciones de contexto y simplificaciones de los modelos. Ninguna de estas propuestas constituye la solución final al problema de simulación en tiempo real, teniendo cada una distintos contextos donde son más adecuados.

Nuestra contribución radica en dos factores:

- realizamos un análisis integral de un sistema de simulación de prendas de vestir.
- proponemos un sistema de niveles de detalle para lograr aprovechar al máximo el poder computacional disponible.

1.1 Propuesta de este trabajo

Abordamos en este estudio la simulación de prendas de vestir desde una perspectiva integral y no como una mera simulación de tela. Con este objetivo, recorreremos de principio a fin el camino que nos conduce a obtener en pantalla un modelo virtual vistiendo prendas de vestir simuladas en tiempo real.

Entonces, intentaremos cubrir todos los aspectos involucrados. Obviamente, daremos mucha importancia a la simulación de tela ya que es crucial, pero también estudiaremos el modelo de una prenda desde el punto de vista del diseñador de indumentaria, la representación de la tela para el sistema de simulación, las formas de traducir una en la otra, la representación del personaje virtual que la usaría, etc.

Dentro del sistema que planteamos, la tela no cumple ninguna labor funcional, sino que es exclusivamente estética. Es decir, si bien la tela será afectada por el entorno, el personaje sobre el que está puesta y otros factores, el entorno no será afectado por la tela. El objetivo es, en definitiva, poder visualizar prendas sobre personajes en movimiento. Este el paso anterior a realizar una simulación completa

de prendas (y tela), donde la prenda puede afectar los movimientos del personaje que la usa para, por ej., modelar la distinta libertad de movimientos entre una pollera suelta y un par de jeans ajustados. Esto está fuera del alcance de este estudio.

Un sistema con las características que describimos aquí sólo sería útil si logra simular todo el sistema en tiempo real. Analizaremos y utilizaremos las técnicas de simulación que nos permitan alcanzarlo, tratando de usar el 100% de poder computacional de la manera que nos sea más provechosa.

Nuestro objetivo a nivel visual es alcanzar la mayor calidad posible, que en este contexto significa que la simulación sea lo más realista posible, o en su defecto lo más verosímil posible, dentro de los límites que nos impone la meta del tiempo real.

Evidentemente el objetivo visual conspira contra el objetivo de performance, siendo que mayor calidad implica un mayor costo a nivel de los algoritmos de simulación. Para lograr que confluyan estas dos aspiraciones proponemos un sistema de niveles de detalle, una técnica ampliamente usada en otras áreas de la computación gráfica, más conocido en inglés como *level of detail* (LOD).

La idea de un sistema de LOD es sencillamente: "Distribuir el trabajo computacional disponible según el nivel de detalle que se desee en cada porción de tela". Los algoritmos computacionalmente más pesados, es decir más complejos y con entradas más grandes, dan como resultado simulaciones más verosímiles. Nosotros buscamos un equilibrio entre calidad de la simulación de cada porción de tela y la cantidad de tela a simular.

Proponemos distintos algoritmos heurísticos de LOD para alcanzar el tiempo real en una escena de complejidad arbitraria, maximizando la calidad visual final. Estamos interesados en resolver manifestaciones de falta de realismo en los modelos dados por la falta de capacidad computacional para generar simulaciones real time.

1.2 Motivos para encarar la tesis

Muchos estudios y desarrollos relacionados con la simulación de telas en real time han sido realizados durante la última década. Pocos de ellos ha sido encarado con un enfoque lo suficientemente integral como para que sea posible incorporarle un sistema de LOD. Proponemos un sistema modular para hacer posible la integración de un sistema de LOD que enriquezca la simulación tomando decisiones en tiempo real para aplicar el nivel de detalle más adecuado.

1.3 Plan de trabajo

Para comenzar, primera tarea que realizamos fue recolectar información sobre el estado en el que se encontraba la investigación en temas de simulación de prendas. Una vez distinguida la *cresta de la ola,* establecimos los objetivos a cumplir y las metas a alcanzar. Entendimos que si bien había mucha información dispersa sobre cómo simular telas y cómo vestir personajes, no había información integrada sobre cómo desarrollar un sistema de simulación de prendas vestidas por personajes humanos al cual pueda integrársele un sistema de LOD. Por lo tanto, decidimos desarrollar nuestro propio sistema con la modularidad adecuada.

En primer lugar construimos un sistema para simular telas. Esto implicó elegir un modelo, armar el framework asociado al modelo seleccionado e implementar los algoritmos necesarios y hacer mediciones para establecer la factibilidad de la utilización de los algoritmos.

Una vez desarrollado el sistema capaz de simular una pieza de tela en tiempo real, lo extendimos para que las telas tuvieran forma de prendas y le agregamos la los personajes que las visten. Fue necesario desarrollar procesos que pusieran la ropa al personaje y que detectaran colisión e hicieran reaccionar a la tela ante la colisión. Finalmente, desarrollamos el módulo que define el nivel de detalle más adecuado para la simulación.

1.4 Estructura de este documento

El cuerpo principal de este trabajo está estructurado en cuatro partes. La primer parte se corresponde con una introducción a los modelos y conceptos de las simulaciones de prendas y de tela. En la segunda se analizan todas las cuestiones asociadas con la implementación de una aplicación que cumpla con nuestros objetivos funcionales y visuales. La tercera parte trata sobre el sistema de LOD y las formas de alcanzar el tiempo real. Finalmente, se elaboran conclusiones y se discuten pasos a seguir.

La primera parte comprende los primeros cuatro capítulos del presente trabajo. En el capítulo inicial presentamos una introducción a la simulación de prendas, analizando los elementos que la conforman. El foco de los capítulos 2 y 3 está puesto sobre la simulación de telas, analizando el capítulo 2 el modelo físico con el que se representa la tela de una prenda y el 3 cómo se lleva a cabo la simulación sobre dicho modelo físico. El capítulo 4 incluye una breve reseña del estado del arte.

La segunda parte está compuesta por de capítulo 5 en el cual analizaremos el desarrollo de un sistema integral de simulación de prendas, proponiendo y examinando distintas soluciones. La tercera parte está compuesta por el capítulo 6, en el cual presentamos y analizamos el sistema de Niveles de detalle (LOD). En este capítulo, además, presentamos los resultados visuales y de performance obtenidos

por el sistema en su totalidad. Sobre el final del trabajo presentamos, en el capítulo 7, las conclusiones y los distintos caminos que se pueden seguir para profundizar y extender este trabajo.

1.5 Elementos de una simulación de prendas

Una simulación[1] es una imitación de alguna porción de la realidad que trata de representar ciertos aspectos del comportamiento real de la misma. La representación se logra a través de un modelo. Para hacer un modelo es necesario plantear una serie de hipótesis, de las cuales se desprende qué características de la realidad representada son considerados en el modelo y cuales no.

Los resultados de la simulación se visualizan a través de una animación. El sistema de simulación que presentamos está compuesto principalmente por tres elementos: Personajes, Prendas que visten a los personajes y el escenario en el cual tiene lugar la acción. El foco del presente trabajo está puesto sobre las prendas, dado que es el único elemento que se maneja mediante simulación. El personaje y el escenario son solo animaciones que, si bien influencian el comportamiento simulado de las prendas, son principalmente animaciones que acompañan la animación de las prendas.

En términos generales, en el proceso de modelado de una simulación pueden distinguirse dos subprocesos bien definidos. El primero de ellos, tiene por objetivo modelar de los elementos que componen la realidad. Se basa en un conjunto de variables u objetos que dan como resultado la estructura estática de la simulación. El segundo de los subprocesos, es el que tiene la responsabilidad de hacer evolucionar al sistema.

Claramente el hilo conductor necesario para el subproceso que hace evolucionar al sistema, es el modelado del paso del tiempo. En primer lugar es necesario, para una correcta evolución del sistema, sincronizar el paso del tiempo de forma tal que el tiempo trascurra sobre todos los objetos del sistema por igual. En nuestro caso, por tratarse de una simulación en tipo real, el paso del tiempo debe ser idéntico al paso del tiempo en la realidad del espectador observando la simulación.

En los modelos matemáticos, por ejemplo, el comportamiento generalmente puede ser descripto por una función dependiente del tiempo. A partir de esta función, y de acuerdo con las condiciones iniciales, es posible determinar el estado de todo el sistema para un instante de tiempo arbitrario. Para la mayoría de las simulaciones por computadora, al no ser posible utilizar una función continua, se utilizan como sustituto de este esquema cuando es muy difícil determinar el estado total de un sistema conociendo las reglas que rigen el comportamiento de los elementos atómicos de la simulación. En una simulación por computadora, representamos esta realidad mediante estados que evolucionan discretamente con

[1] http://www.lib.utexas.edu/engin/guides/anthropometry.html

el paso del tiempo. Al tener incrementos de tiempo lo suficientemente pequeños, tenemos la ilusión de ver un modelo evolucionando de manera continua.

1.6 El modelo de la simulación de prendas

Distinguimos los tres principales elementos que componen la simulación; los personajes, las prendas y el escenario. En la presente sección analizamos las distintas características de cada uno de ellos y establecemos su inclusión o exclusión en nuestro modelo. Para cada uno de estos tres elementos, describimos la forma en que los modelamos, construimos y como concebimos que su estado vaya evolucionando según el transcurso del tiempo.

1.7 El personaje

Cuando hablamos del **personaje**, nos referimos al individuo en la simulación que vestirá las prendas virtuales. En principio el objetivo es representar mediante un modelo virtual las características corporales de un ser humano. Según el contexto, ya sea en una película de animación o videojuego los personajes pueden tener proporciones irreales o no ser humanos (como por ejemplo el Ogro o la dragona de la película Shrek).

Un modelo 3D puede ser representado de muchas maneras, siendo al día de hoy la más utilizada la malla de triángulos, es decir, un conjunto de triángulos que forman la superficie de un objeto. Esta malla representa la superficie: lo que vemos del objeto. Esto representa de forma estática la fisonomía del personaje.

Para la generación de un cuerpo en tres dimensiones existen dos caminos posibles. El primero es a través de un programa de modelado en 3D como ser el 3D Studio, en el cual un artista *esculpe* (define) los triángulos para obtener la figura deseada. Mediante este proceso es posible definir una figura arbitraria. La calidad del producto final depende, en este tipo de modelado, de la habilidad del artista, su imaginación y el tiempo invertido necesario para modelar. El otro medio posible para obtener un modelo es utilizando técnicas de antropometría. La antropometría[2] es el estudio comparativo de las medidas del cuerpo humano y sus propiedades. Permite obtener medidas de cuerpos con características determinadas basándose en datos estadísticos. Este método permite generar personajes con fisonomía humana.

Para modelar los movimientos del personaje es necesario algún mecanismo que permita dirigir el cambio de posición de estos triángulos. El más aceptado, aunque no el único, es el mecanismo que utiliza el concepto de *esqueleto*. El movimiento de un personaje vertebrado es una consecuencia directa del movimiento de los huesos de su estructura. El mecanismo utiliza la información sobre el movimiento de los huesos para dirigir el cambio de posiciones de los

[2] http://www.lib.utexas.edu/engin/guides/anthropometry.html

triángulos. Un esqueleto es un conjunto de "huesos" asociados de manera jerárquica. Cada vértice de cada triángulo está asociado a uno o más huesos de manera tal que si el hueso se mueve el vértice seguirá el movimiento. Por lo tanto, para conseguir movimientos de nuestro modelo hay que animar el esqueleto, lo cual reduce el problema de animar miles de vértices a animar unas decenas de huesos. A esto se lo llama animación esqueletal. Estos huesos pueden considerarse cuerpos rígidos y ser animados mediante física de cuerpos rígidos o pueden moverse artificialmente mediante una animación definida por un artista. El origen del movimiento de los huesos excede el alcance del presente trabajo. Simplemente asumimos un movimiento arbitrario del esqueleto, el cual puede no ser influenciado por el entorno.

1.8 Las prendas

Las prendas, como ya mencionamos, se encuentran en el foco de nuestro estudio. Esta es la razón por la que la descripción de la forma en que son modeladas se extiende a lo largo de todo el cuerpo principal del presente trabajo. Esta sección tiene por objetivo describir la forma en que fue concebido, desde una perspectiva general, el modelo.

En primer lugar es necesario modelar la materia prima de la prenda, la tela. En una animación guiada por una simulación, además debe modelarse la estética de los elementos deben modelarse también sus características físicas. La animación de las prendas está guiada por una simulación, por lo que debemos establecer, para el elemento básico de la prenda, la tela, un modelo físico. Luego, la tela tiene doble modelado: el físico y el estético.

Una porción de tela puede verse como un lámina 2D moviéndose en un espacio 3D. En el modelo, la tela se discretiza con *puntos de control*, los cuales pasan a ser la unidad básica de la tela.

El Modelo estético

Debe ser posible representarlo en el espacio 3D en el que se mueve, de la misma manera que cualquier objeto que se desea visualizar en una realidad de tres dimensiones de manera estática, como un conjunto de triángulos 2D dispersos en una realidad 3D. De esta manera, es posible dibujar en la pantalla la tela en un espacio tridimensional. En las figuras 1 y 2 puede visualizarse la imagen de una pieza de tela representada con triángulos. En la figura 1 los triángulos están claramente delineados mientras que en la figura 2 están pintados según el modelo de iluminación. La posición que van tomando los puntos de control con el paso del tiempo está guiada por la simulación.

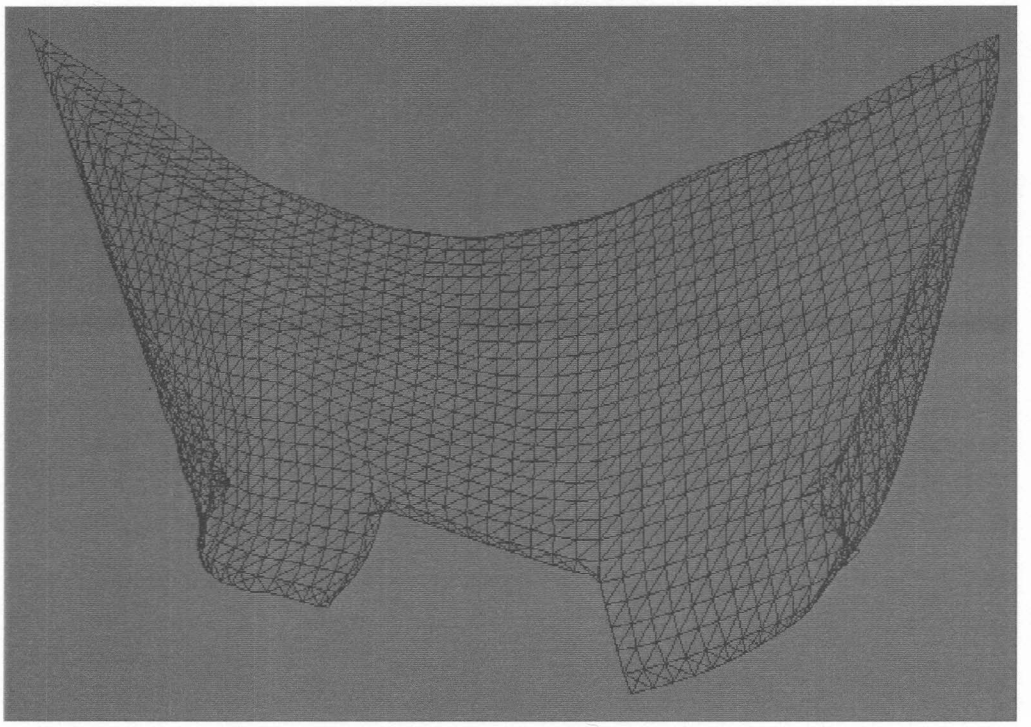


Figura 1: Modelo estético de la tela: pieza de tela representada como un conjunto de triángulos claramente delineados

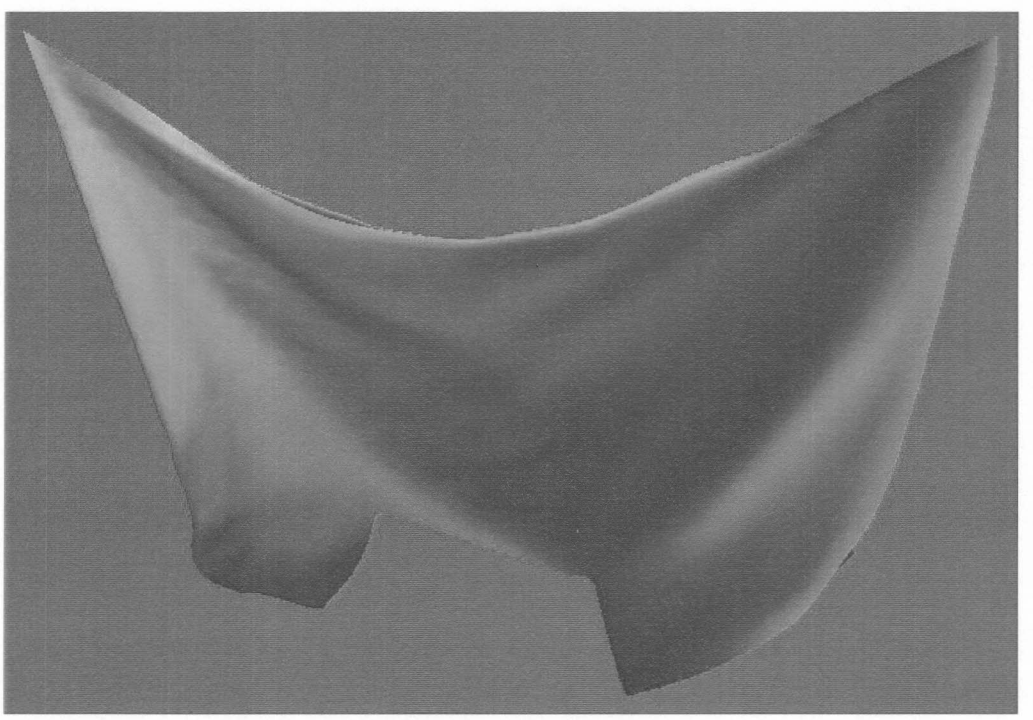


Figura 2: Modelo estético de la tela: pieza de tela representada como un conjunto de triángulos pintados según el modelo de iluminación

Modelo físico

El modelo físico tiene el objetivo de modelar las características físicas de la tela. Los nodos que en el modelo estético denominamos puntos de control, en el modelo físico son tratados como partículas que se enlazan a través de resortes (figura 3). Las partículas modelan la masa de la tela, mientas que los resortes su flexibilidad. Este tema es tratado en detalle en el capítulo 2.

Resumiendo, las partículas se mueven según las reglas que rigen la simulación de la tela, y son movidas según los resultados de la simulación.

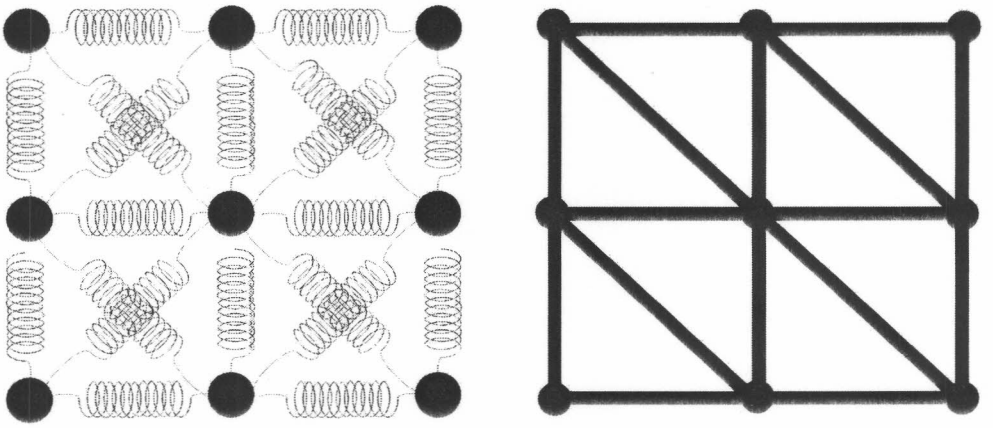


Figura 3: El modelo físico a la izquierda vs. el modelo estático a la derecha

Tenemos una estructura física de una tela. Este modelo físico, debe evolucionar con el paso del tiempo según lo rigen las reglas de simulación física. Según leyes de la mecánica clásica con la información sobre las condiciones actuales es posible predecir la posición en que van a estar.

Un punto imprescindible para el correcto comportamiento del sistema, es el modelado de la interacción entre los distintos objetos de la simulación. Tratamos este tema en esta sección dado que la tela es el único elemento de la simulación que reacciona ante el contacto con otro objeto. Luego, Se debe detectar si dos objetos de la simulación poseen alguna superficie de contacto (ver detección de colisión). La detección de colisión es, básicamente, buscar para un instante dado qué elementos ocupan el mismo lugar físico en el espacio 3D de la simulación. Y de ahí tomar alguna medida al respecto (reacción ante a la colisión).

Es importante destacar que las prendas son reactivas, es decir, no son capaces de moverse por sí mismas. Su movimiento, en todos los casos, es causado por factores externos. Las telas es principio, permanecen inmóviles mientras no sean influenciadas por el entorno.

Para nuestro modelo, detectamos la colisión de la tela con sí misma, con el personaje y con el escenario. Solo consideramos la respuesta de colisión en la tela, es decir, que ni al personaje ni a los objetos de la escena les importa estar

chocándose con la tela. Profundizaremos sobre la evolución del sistema en el capítulo3.

1.9 El escenario

El escenario está formado por varias entidades disímiles entre sí, por lo que amalgama una colección de elementos. El escenario es el complemento en escena al personaje y las prendas, es decir, está compuesto por todos los objetos que rodean al personaje y las prendas, que afectan o son afectados por ellos. Entre las entidades que componen el escenario, pueden distinguirse dos grupos: los objetos físicos y las entidades correspondientes al ambiente.

Los objetos que pertenecen al conjunto de los objetos físicos (una silla, una mesa) generalmente están representados, al igual que el personaje, por mallas de triángulos y consecuentemente se relacionan con las prendas de un modo muy similar al que lo hace el personaje.

El ambiente es un objeto conceptual, dado que no es un objeto en sí mismo, sino que sólo puede percibirse observando los objetos sobre los que tiene influencia. Está compuesto por las entidades *intangibles* que afectan a nuestro personaje y sus prendas: la fuerza de la gravedad y el viento. Por viento nos referimos al efecto del aire en movimiento sobre las prendas o del aire quieto sobre las prendas en movimiento. El viento puede representarse mediante una velocidad constante en una dirección o mediante las fórmulas de Navier-Stokes para modelar fluidos. La gravedad se puede representar como una fuerza que afecta nuestras prendas y los objetos de la escena por igual. Luego, es posible simular, por ejemplo un ambiente lunar disminuyendo la intensidad de la fuerza de gravedad y anulando el viento.

2 Modelo de la tela

La tela es una estructura laminar resultante de una disposición de fibras. Esta estructura laminar puede idealizarse como una superficie de dos dimensiones moviéndose en un espacio tridimensional. Puede ser descripta tanto por sus propiedades físicas como por sus propiedades geométricas.

Desde el punto de vista geométrico, una pieza de tela puede sufrir deformaciones. Estas deformaciones no son relativas a algún punto exterior, sino que son relativas a su propia geometría local. Las deformaciones se caracterizan según la dirección de la deformación con respecto al plano tangente a la tela en el punto donde la misma se deforma. Pueden ser estiramientos (stretch), deformaciones tangenciales a la superficie a la que pertenece el punto o pueden ser doblados (bend), es decir, deformaciones que tienen una componente normal o perpendicular a la superficie

Desde un punto de vista físico, queremos cerciorarnos de que los fenómenos que estamos intentando simular sean físicamente correctos, es decir, que las deformaciones resistidas por una pieza de tela están gobernadas por las leyes de la física que limitan las formas de los objetos según sus características.

Durante años, las características físicas de las telas han sido estudiadas en distintos contextos. Se han desarrollado métricas que establecen la resistencia de los distintos materiales textiles a deformarse, dependiendo del tipo de material, su densidad, la intensidad de la tracción y en que dirección es aplicada la tracción.

El resultado más relevante para el presente trabajo es el relacionado con el aumento de la longitud según la intensidad de la tracción aplicada. Retomamos este tema en el capítulo 5 cuando plantemos evitar la *superelasticidad.*

Desafortunadamente, al igual que en muchos de los fenómenos de la realidad, es muy difícil modelar con exactitud una tela. El objetivo aquí es hacer un modelo de la tela, para aproximar su movimiento. Existen muchos modelos para representar telas. En esta sección, presentamos la principal taxonomía sobre los modelos considerados.

2.1 Los modelos considerados

Muchos modelos fueron desarrollados con el objetivo de simular el comportamiento de según lo describen Gibson y Mirtich en [GM]. La tela es un caso particular de objeto deformable. La principal taxonomía de estos modelos se corresponde con los dos principales enfoques comentados en la anterior sección: físicos y geométricos.

Los **modelos físicos** corresponden a modelos mecánicos para simular de forma realista la tela. Se utilizan métodos basados en principios físicos para los cuales es necesaria gran potencia computacional. Estos intentan ser simulaciones

realistas de complejos procesos físicos que pueden ser difíciles o imposibles de modelar con modelos no físicos.

Los modelos físicos se dividen en dos grandes grupos: Continuos y Discretos. El primero de los grupos está compuesto por modelos físicos más precisos, los cuales tratan a los objetos deformables como serie continua: cuerpos sólidos con la masa y energías distribuidas en todas partes. Los modelos en el segundo de los grupos, modelan los objetos como un conjunto de puntos de masa discretos. Mediante la descripción del comportamiento independiente o en pequeños grupos de las distintas partículas se obtiene la descripción del comportamiento global de todo el sistema.

Los **modelos geométricos**, son aproximaciones que no utilizan dinámica, sino heurísticas basadas en el criterio de los desarrolladores de alguna aplicación en particular. Tienen por objetivo dar como resultado una deformación verosímil de los objetos que representan. Muchas aplicaciones, particularmente en diseño, emplean técnicas puramente geométricas. Generalmente estas técnicas son computacionalmente mucho más eficientes que las simulaciones físicas. Su mayor debilidad radica en el hecho de que las deformaciones deben ser explícitamente especificadas dado que el sistema desconoce acerca de la naturaleza de los objetos que están siendo simulados.

El modelo que utilizamos como base teórica para el desarrollo de nuestro sistema integral de simulación de prendas de vestir es el denominado Sistema Masa-resorte. De acuerdo a la anterior clasificación, este modelo corresponde a los modelos físicos y discretos.

2.2 Modelo sistema masa-resorte

El Sistema masa-resorte es un modelo sencillo e intuitivo. Se lo utiliza para modelar todo tipo de objetos flexibles como la tela. Es el candidato ideal para simulaciones en tiempo real, dado que es fácil de implementar y entre los modelos físicos es uno de más simples en términos de complejidad computacional.

En el sistema masa-resorte, la tela se modela mediante un conjunto de partículas y un conjunto de resortes. Las partículas (puntos discretos con masa finita) representan la masa de tela y los resortes modelan las propiedades elásticas de la tela.

Las partículas se conectan a través de los resortes formando una malla. La forma en que se conectan las partículas definen la estructura de la malla (este tema se encuentra desarrollado en la sección 2.3: Distintas estructuras de Mallas. Las fuerzas que ejercen los resortes sobre las partículas definen la resistencia que presenta la tela a deformarse. Claramente esta resistencia es relativa a la estructura de la malla definida. Así, según se defina la topología de la malla, la masa asociada a las partículas y la fuerza asociada resortes es posible definir

distintos tipos de tela. Por ejemplo, si se desea una tela pesada y poco flexible (por ejemplo tela para sillón), se deben asociar partículas con más masa y resortes más duros que para definir una tela elástica y liviana como la Lycra.

Si asumimos que las telas poseen una densidad homogénea, entonces el modelo masa-resorte debe estar compuesto por partículas de idéntica masa. En caso contrario, es decir, bajo el supuesto de que una porción de tela no posee la misma densidad en todos lados, cada una de las masas del modelo debe tener un peso asignado coherente con la distribución de masa de la tela. Análogo es el caso del modelado de las fuerzas de doblado y estiramiento. Si se asume que la tela posee una disposición regular de las fuerzas, luego la mejor opción es armar una topología de conexiones entre masas con resortes lo más regular posible.

El cálculo de las fuerzas ejercidas por los resortes sobre las partículas y la consecuente reacción de las partículas antes estas fuerzas se realiza a través de métodos de integración numérica (este tema es tratado en detalle en la sección 3.1: Breve introducción a la mecánica del modelo).

2.2.1 Debilidades

Los sistemas masa-resorte tienen algunas desventajas. El modelo discreto es sólo una aproximación a la física verdadera que ocurre en un cuerpo continuo. El enrejado se modela con sus constantes del resorte, y los valores apropiados para estas constantes no son siempre fáciles de derivar de características materiales medidas.

Finalmente, los sistemas de masa-resorte exhiben un problema de dureza si las constantes del resorte son grandes. Estas constantes se utilizan para modelar los objetos que son casi rígidos, o para modelar objetos rígidos deformables. Los sistemas de masa-resorte son problemáticos porque tienen estabilidad pobre, requiriendo el integrador numérico tomar pequeños intervalos de tiempo. El resultado es una simulación más lenta.

2.3 Distintas estructuras de mallas

Tal como un trozo de tela real se crea entrelazando de diferentes modos numerosas fibras de hilo, conectar las partículas de un modelo de sistemas de partículas puede ser hecho también de muchas formas diferentes. La confección de las distintas topologías de conexión entre las partículas puede dividirse en tres grandes grupos:

- **Mallas Regulares:** Son las más simples y la solución más restrictiva. Se trata de una malla de partículas dispuestas en una cuadrícula según puede visualizarse en la figura 4. La bondad de este acercamiento es que la disposición de los resortes y partículas es igual en toda la malla, logrando una representación de la tela uniforme. La desventaja se presenta a la hora de modelar telas que no presenten resistencia uniforme a las deformaciones.

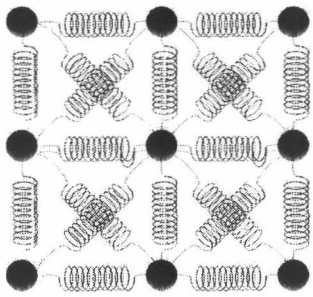


Figura 4: Malla Regular

- **Mallas Adaptativas Regulares:** Es una colección de mallas regulares de distintas granularidades unidas entre sí. Un ejemplo de este tipo de mallas puede visualizarse en la figura 5. Las restricciones topológicas producen fracturas en la interfase entre las distintas mallas de distintas granularidades.

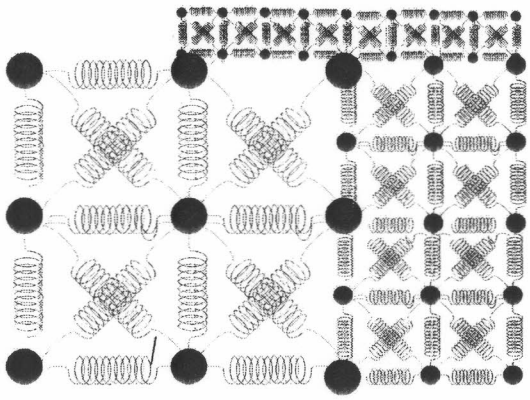


Figura 5: Malla Adaptativa Regular

- **Mallas Adaptativas Irregulares:** Es el método menos restrictivo, produce mallas continuas, pero con distintas granularidades y uniones entre las partículas. Generalmente se suele realizar una malla triangular. Un ejemplo de malla adaptativa irregular visualizarse en la figura 6.

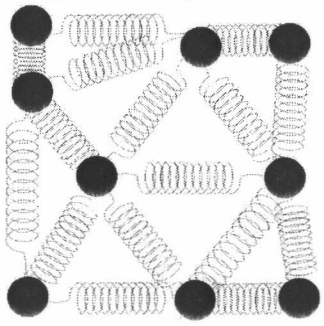


Figura 6: Malla Adaptativa Irregular

Nuestro modelo elástico es una malla en principio de n x m masas virtuales, cada una de las cuales está enlazada con sus vecinos por resortes de longitud distinta a cero. El enlazado se lleva a cabo entre los nodos vecinos de dos formas diferentes, cada una representando un tipo de resorte: resortes estructurales y de corte. Asumiendo que las masas están alineadas horizontal y verticalmente, podemos referenciarlas de la misma forma que se referencian posiciones en una matriz. Luego, tenemos que se unen para formar una malla regular de la siguiente forma:

Resortes estructurales

Los resortes estructurales manejan la tracción y compresión de la tela, sirven para sostener el paño en su estado natural: establecen la estructura básica de la tela. Se conectan horizontalmente y verticalmente con sus cuatro vecinos más cercanos del siguiente modo: Un resorte une la masa la masa en la posición [i, j] y la masa en la posición [i+1, j] y otro resorte une la masa en la posición [i, j] también se une con la que está en la posición [i, j+1]. La topología de los resortes estructurales se encuentra ilustrada en la Figura 7.

Los resortes estructurales sirven para sostener el paño en su estado natural cuadrado, pero fuerzas sobre el paño no son transmitidas adecuadamente por un modelo que divide las fuerzas oblicuas en componentes ortogonales a lo largo de un pequeño número de nodos. Naturalmente, un modelo unido (conectado) sólo con resortes estructurales pero con un número grande de partículas modelaría las correctamente las fuerzas, pero ya que el costo computacional restringe el número de nodos podemos usar, los resortes de corte son necesarios para contribuir con lograr un aspecto realista de propagación de las fuerzas.

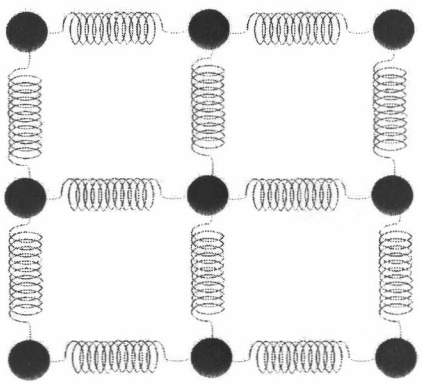


Figura 7: Resortes Estructurales

Resortes de Corte

Los resortes de corte previenen la distorsión de la malla. Se conectan con sus vecinos en posiciones diagonales del siguiente modo: un resorte une la masa la masa en la posición [i, j] y la masa en la posición [i+1, j+1] y otro resorte une la

masa en la posición [i+1, j] también se une con la que está en la posición [i, j+1]. La topología de los resortes de corte se encuentra ilustrada en la Figura 8.

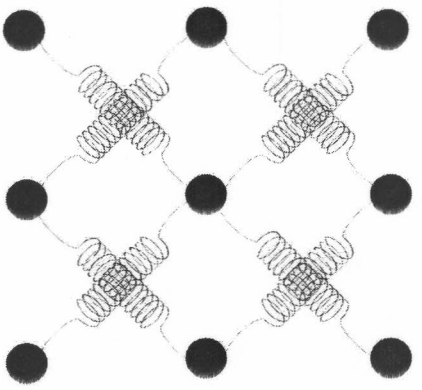


Figura 8: Resortes de Corte

El modelo de malla que utilizamos, es el resultado de utilizar resortes de corte y estructurales en una malla regular. En la figura 9 puede visualizarse el modelo de malla utilizado en el presente trabajo para modelar las telas.

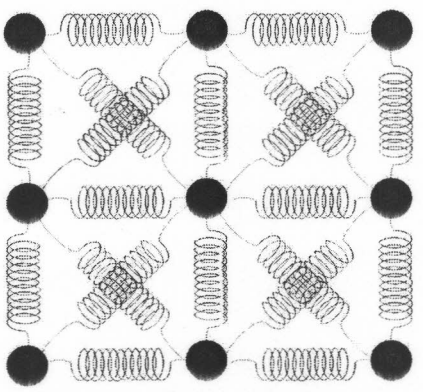


Figura 9: Modelo de Malla utilizado en el presente trabajo.

3 Estado del arte

Esta sección tiene por objetivo presentar los trabajos de investigación y publicaciones relacionados con nuestro trabajo. Principalmente Incluimos aquellos que de alguna manera sirvieron como base a nuestro estudio.

En 1992 Baraff y de Witkin publican uno de los primeros trabajos en el campo de la simulación de telas [BW92] el cual describe las bases del tratamiento de cuerpos flexibles deformables. En los años siguientes Breen, House y Wozny publican un paper que presenta algoritmos para predecir la caída de la tela [BHM]. Volino y Thalmann publican [VCT95], una aproximación para modelar la tela a partir del modelo de sistema masa-resorte y [VM95], publicación en el cual proponen el algoritmo de auto colisión más ampliamente utilizado en la simulación de prendas. En 1998 Baraff y Witkin [BW98] dan el gran paso hacia la estabilidad de las simulaciones: presentan métodos que permiten intervalos de tiempo más largos que los que podían utilizarse con los algoritmos conocidos hasta el momento, sin perder estabilidad. Su mayor debilidad radica en que es necesario resolver un sistema de n x n ecuaciones.

En el año 2000 se publica [DMB] con una solución razonable al problema de resolver un gran sistema de ecuaciones según proponían, Baraff y Witkin. El método consiste en reemplazar la solución de un sistema lineal grande por una aproximación. Luego, corregir las desviaciones resultantes de la aproximación utilizando variables invariantes en la física. Ese mismo año se publica [MHG] proponiendo un método que aproxima la integración implícita. A partir de la las ecuaciones de la integración implícita y utilizando la aproximación propuesta en [DMB] e introduciendo una nueva aproximación, presenta una solución lineal, la cual abre el paso hacia las simulaciones en tiempo real. En el 2002 se publica [BFA] donde Bridson propone un algoritmo robusto y eficiente (aunque no alcanza el tiempo real) para el manejo de la colisión, las fuerzas de contacto y la fricción en la simulación de telas. Ese mismo año, la gente de Miralab publica [CMT02] proponiendo un modelo de simulación eficiente pero muy restringido, el cual sólo puede utilizarse para simular prendas usadas por personajes de morfología humana. Más tarde Fuhrmannm, Gross and Luckas [FGL03] presentan un método de integración eficiente para simular telas. El modelo que utilizan es una simplificación del modelo masa-resorte gracias a la cual les es posible alcanzar el tiempo real. Reemplaza el concepto de resorte por el de restricciones. Esta simplificación permite utilizar integración explícita con grandes intervalos de tiempo sin perder estabilidad. Es una solución simple y eficiente que tiene por debilidad la pérdida de elasticidad del modelo. Métodos para hacer adaptaciones de las granularidades de las mallas asociadas a las simulaciones de telas fueron presentados en [VL03] y [VL05].

4 Evolución del sistema

Una vez modelados los elementos que componen la realidad, es necesario establecer las reglas y procedimientos que rigen el comportamiento de estos objetos. Estas reglas y procedimientos rigen la evolución del sistema y pueden ser divididos en 3 categorías: integración, colisión y auto-colisión. Trataremos en este capítulo la problemática de cada una de esas y métodos para aplicarlas.

4.1 Integración

4.1.1 Breve introducción a la mecánica del modelo.

El fenómeno más importante que observamos a nuestro alrededor es el de movimiento. El movimiento de un cuerpo es influenciado por los cuerpos y fuerzas que lo rodean, debido principalmente al conjunto de sus interacciones.

Se denomina Mecánica Clásica, a la parte de la física que analiza las fuerzas que actúan sobre un objeto. La mecánica contribuye principalmente a analizar y predecir la naturaleza de los movimientos que resultan de las diferentes clases de interacciones. Está dividida principalmente en tres piezas: la Cinemática, que estudia el movimiento de los cuerpos sin tener en cuenta sus causas, la dinámica, que estudia los objetos sometidos a fuerzas que no están en equilibrio, y la estática, que trata con objetos en equilibrio. Para nuestro análisis utilizaremos las dos primeras.

Para poder hacer avanzar la simulación es necesario calcular, dada la configuración del sistema en un instante de tiempo dado, la configuración que tendrá el sistema un instante de tiempo posterior. En la realidad continua que queremos modelar, el tamaño del intervalo de tiempo entre un instante y el siguiente tiende a cero. Dado que en una simulación por computadora es impracticable que este intervalo tengo un tamaño infinitesimal, es necesario discretizar el tiempo y que este intervalo tenga un tamaño arbitrario. A este intervalo de tiempo llamamos *paso* en la simulación. Así, con la información de la configuración inicial del sistema, se hace avanzar inductivamente a la simulación, es decir, paso a paso, cubriendo intervalos arbitrariamente largos en el tiempo.

Para obtener la configuración del sistema en un instante dado, contamos con la información que nos brinda el modelo masa-resorte. Este modelo lleva asociada la información sobre: las masas (cuerpos puntuales con masa), los resortes (cuerpos ideales sin masa) y la interacción entre ellos, es decir, la información necesaria para caracterizar la dinámica del sistema. En la figura 10 se ilustra el modelo de fuerzas del sistema.

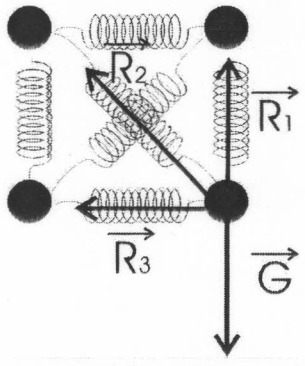


Figura 10: Modelo de las fuerzas ejercidas sobre una partícula: R1, R2 y R3 son las fuerzas internas ejercidas por los resortes, y G es la fuerza de gravedad

La información concreta con la que contamos es:

- la masa y posición de cada una de las partículas
- las características elásticas de los resortes
- la topología de enlace entre partículas y resortes
- la interacción entre el sistema y el medio.

Con esta información y utilizando resultados de la dinámica y la cinemática, podemos obtener, para un instante arbitrario (t_0), la velocidad y aceleración de cada una de las partículas. Dado que la aceleración se define como la tasa de variación de la velocidad, es posible calcular a partir de la aceleración y la velocidad en t_0, la velocidad en $t_1 = t_0+\Delta t$. Con el mismo razonamiento, es posible calcular a partir de la velocidad (la tasa de variación de la posición) y la posición de una partícula en t_0 su correspondiente posición en $t_1 = t_0+\Delta t$.

Para cada masa partícula i del modelo, sabemos que:

1) $F_i = m_i\, a_i$

Además, de las leyes de la mecánica sabemos que

2) $F_i = \Sigma\, f_i + f_e$

Donde las f_i son las fuerzas internas a nuestro sistema de partículas y las f_e son las fuerzas externas al sistema. Dado que el único tipo de interacción entre partículas que consideramos en el sistema es la de enlace a través de resortes, las fuerzas internas del sistema están compuestas exclusivamente por las fuerzas ejercidas por los resortes sobre las partículas. Las fuerzas externas se componen principalmente por las fuerza de gravedad, rozamiento con el aire y con las superficies con las cuales esté en contacto.

Considerando a la ecuación 1) para un instante t_0, es posible conocer la aceleración en t_0 de la masa i despejando la variable a_i. El método utilizado para

obtener la velocidad a partir de la aceleración y la posición a partir de la velocidad se basa en resolver un sistema de ecuaciones diferenciales. La resolución analítica de este tipo de sistemas es compleja de llevar a cabo por un computador, por lo que a lo largo de la historia se han desarrollado distintas técnicas y métodos numéricos para resolverlas. Existen muchos métodos, cada uno de ellos con sus virtudes y defectos. Veremos más en detalle estos métodos en las próximas secciones.

4.1.2 Objetivos de la integración numérica

dónde hace integración numérica

La integración numérica es la aproximación computacional de una integral, que utiliza técnicas de métodos numéricos. Utilizamos la integración numérica como un medio para resolver los las ecuaciones diferenciales que describen la evolución de nuestro sistema.

Entre los criterios para elegir el algoritmo de integración podríamos contar:

- **Convergencia:** Cuando el paso tiende a cero, las soluciones numéricas coinciden con las analíticas. Todos los métodos para ser útiles deben ser convergentes.
- **Precisión:** También llamado orden de convergencia. Es una medida de cuan rápido converge el método para h → 0, o en otras palabras, cuan precisa es la solución para un h dado.
- **Estabilidad:** Describe cómo los errores en los datos de entrada se propagan a través del algoritmo. En un método estable, los errores debidos a las aproximaciones se van atenuando a medida que a computación procede. En un método inestable, cualquier error en el procesamiento se magnifica a medida que el cálculo procede.
- **Eficiencia:** Nos dice cuanto procesamiento implica ejecutar el algoritmo. Asociado a esto está el orden de complejidad, que nos dice como se incrementa este procesamiento al incrementar el tamaño de la simulación (cantidad de partículas, cantidad de resortes, etc.).

Han sido desarrollados muchos métodos para resolver este tipo de ecuaciones, algunos más precisos que otros, los cuales comentamos en las próximas secciones

4.2 Métodos considerados

4.2.1 Métodos Explícitos

El más antiguo y simple método de integración es el llamado el método explicito de Euler. El algoritmo discretiza el tiempo en pasos de longitud h y se dice que es una formula explícita de único paso, dado que utiliza sólo un intervalo de tiempo. Utiliza la información en el principio del paso del tiempo para obtener la información en el

final del paso del tiempo. Es decir, utiliza la tasa de cambio de la aceleración y la velocidad en el paso actual para calcular el valor en el próximo paso de la velocidad y la posición.

La estabilidad del método requiere intervalos de tamaño pequeño, por lo que no pude conseguirse con una eficiencia suficiente para ser utilizado en una simulación en tiempo real. Una perspectiva clara y moderna sobre este tema puede encontrarse en [MH].

4.2.2 Métodos Implícitos

Una alternativa al método explícito de Euler es el método implícito de Euler [BW98]. Este método, mejora la escasa estabilidad del método de integración explícita a costa de plantear las ecuaciones de forma inversa. En lugar de usar la tasa de cambio de la velocidad en el punto actual para calcular el valor en el próximo paso, utiliza la tasa de cambio y el siguiente punto. Pero esto, claramente no es conocido hasta que el valor del punto actual es conocido, por lo tanto el proceso se convierte en iterativo (o implícito). Justamente esta es la razón de la dificultad de la implementación del algoritmo, dado que el valor de las variables en el paso siguiente es dependiente del valor de la función en el paso siguiente. Luego, para encontrar el valor de las variables en el siguiente paso, es necesario resolver un sistema de ecuaciones. lineales?

Las ventajas asociadas a este método residen en el hecho de que el método implícito de Euler, a diferencia del explícito, puede usar intervalos de tiempo de mucho mayor tamaño. Esto requiere un precio más alto en términos de tiempo y espacio computacional para calcular cada intervalo, además de su mayor dificultad.

La experiencia muestra que, para un intervalo de tiempo o paso de integración dado, hay un valor crítico de dureza sobre el cual la resolución numérica del sistema es divergente. Resultados matemáticos acerca de ecuaciones lineales muestran que las soluciones numéricas están mal condicionadas si es más grande que el período natural del sistema. Es decir, el cuadrado del paso de integración debe ser inversamente proporcional a la dureza. Si bien éste no es un problema para cómputos off line, impide ser usada en aplicaciones que requieran tiempo real, dado que deben usarse pasos muy cortos para poder asegurar la estabilidad.

¿Qué? ¿más grande?

4.2.3 Jacobiano constante

Este método, presentado en [DSB] modifica el método de integración implícita de Euler con el objetivo de aliviar el costo computacional de resolver un sistema de ecuaciones lineales por cada intervalo de tiempo. Asumiendo intervalos de tiempo constantes, masas de las partículas y dureza de los resortes invariantes en el tiempo realiza pre-cómputos que luego son utilizados durante la simulación. El método introduce una aproximación a la integración implícita para predecir la próxima posición (la aproximación evita resolver el sistema de ecuaciones lineales

utilizando los valores precalculados), y luego corrige los errores que pudieran haber surgido como resultado de la aproximación haciendo valer ciertos invariantes de la física, como el momento angular y lineal. Finalmente utiliza el método para evitar *superelasticidad* descripto un par de secciones más adelante. Su principal debilidad radica en el hecho de que no es posible aplicar intervalos de tiempo adaptativos ni hacer cambios de mallas sin perder eficiencia al tener que repetir los cálculos en el pre cómputo.

4.2.4 Thin Objects

El algoritmo *Thin Objects*, propuesto por [MHG], es una aproximación al método de integración implícita de Euler que logra que la complejidad en el cálculo de la próxima posición sea lineal. El método reformula las ecuaciones planteadas por la integración implícita y utiliza una versión mejorada de la aproximación planteada en el método Jacobiano Constante. Este algoritmo, a diferencia de Jacobiano Constante, sólo asume para la aproximación que la dureza de los resortes es uniforme. Luego, puede ser utilizado con intervalos de tiempo adaptativos y permite modificar las mallas durante la ejecución.

4.2.5 Evitando la superelasticidad

Las reglas que rigen el movimiento de las telas en la realidad no son ciertamente idénticas a las propiedades elásticas ideales. Por esta razón, bajo ciertas condiciones de stress estos modelos elásticos pueden dar como resultado una implausible deformación de la tela, dado que su elongación es proporcional a la fuerza aplicada. Estas características ocurren especialmente cuando el modelo elástico es sujeto a grandes fuerzas y así a altos grados de deformación.

Las telas no poseen propiedades de superelasticidad. Su elasticidad es no lineal, y su resistencia a estirarse aumenta rápidamente cuando el grado de deformación se incrementa. La máxima deformación de la mayoría de las telas está alrededor del 10%.

Se han desarrollado métodos para evitar el fenómeno de superelasticidad mediante correcciones: luego de aplicar la integración con alguno de los métodos anteriores, se aplica un post-procesamiento donde se van corrigiendo uno por uno los resortes que quedaron demasiado estirados (o comprimidos) acercando (o alejando) el par de masas unidas hasta que el resorte que las une tenga una longitud permitida. Este proceso se realiza iterativamente, ya que al corregir un resorte puede estar llevando otro a un tamaño incorrecto. Aunque teóricamente este proceso puede tomar un número arbitrario de iteraciones para converger, la experiencia indica que con pocas (del orden de las 5) se evita la superelasticidad. Una revisión más profunda sobre este tema puede encontrarse en [XP]

4.2.6 Modelo de Verlet

Del mismo modo que en el método que evita la superelasticidad, en muchos casos existe la necesidad de corregir la posición de una partícula. Por ejemplo, al realizar la corrección de colisión o incluso al permitir la interacción directa del usuario con la malla de tela. Estos cambios rápidos de posición discrepan con la información que tiene el sistema con respecto a las velocidades, lo cual puede traer inestabilidades. Para que el sistema sea estable, tenemos que mantener esta información de manera consistente. [DMB] propone actualizar la velocidad, luego de las correcciones, tomando en cuenta el cambio de posición. Para la partícula i en el instante n esto se computa de la siguiente manera:

$$V_i^n = (X_i^n - X_i^{n-1}) / dt$$

El efecto de esta ecuación es que las velocidades son dadas únicamente por la posición de la partícula, por lo que los cambios de posición afectan directamente la velocidad en la integración del próximo paso de tiempo. Notemos que, si la posición no se vio afectada por algún elemento externo, la velocidad se mantiene intacta ya que

$$X_i^n = X_i^{n-1} + V_i^n \, . \, dt$$

Este esquema de integración equivale al de Verlet, donde no se mantiene información de las velocidades sino que se guarda la posición actual y la posición anterior. Si uno desea saber cómo era la velocidad del cuadro anterior debe utilizar la primera ecuación y ni siquiera nos preocupamos por mantener las velocidades actualizadas, ya que nunca las guardamos. Cualquier cambio de posición de las partículas *es* un cambio en la velocidad.

Este cambio de posiciones se puede realizar en muchas partes distintas del proceso de simulación, por lo que decidimos estandarizar el uso de este método de integración para todos los algoritmos.

4.2.7 Algoritmo de restricciones

Una alternativa a la integración implícita es el modelo propuesto por [FGL03]. Este proceso se basa en el proceso de evitar la superelasticidad que vimos anteriormente.

Las partículas de tela son influenciadas por fuerzas tanto externas (gravedad, viento, etc) como internas (atracción / repulsión de los resortes). Las fuerzas internas se modelan por restricciones en la distancia entre las partículas. Como en el proceso correctivo para evitar la superelasticidad, evitan que la unión de dos partículas se contraiga o estire más allá de cierto rango. Luego, si el resorte tiene un tamaño permitido, este no ejerce ninguna fuerza (ni atracción, ni repulsión) entre ambas partículas. Esta es una simplificación muy fuerte de la realidad, pero aun así permite representar las telas de una manera muy realista.

En estos sistemas las causantes de la inestabilidad son las fuerzas internas, que este algoritmo elimina. Por lo tanto, al tener sólo fuerzas externas el sistema se puede integrar explícitamente. Esto da como resultado un sistema que puede ser integrado explícitamente en grandes pasos de tiempo y que sin embargo es estable.

4.2.8 Algoritmos no físicos: algoritmo de piel

Si bien no entran dentro de la categoría de integración, existen varios métodos para mover las partículas de tela que no siguen ningún modelo físico sino que se basan en conceptos geométricos. Estos algoritmos dan resultados de alguna manera plausibles y parecidos al de los modelos físicos, aunque para la mayoría es obvio que no es real. Existen muchos modelos geométricos [GM] que describen el comportamiento de objetos flexibles como la tela. Nosotros elegimos implementar el método propuesto en [CMT02], al cual llamamos "algoritmo de piel".

Este algoritmo sigue lo que se hace normalmente en juegos y aplicaciones gráficas en la actualidad: hacer que la tela siga indefectiblemente el movimiento de la piel del personaje sobre el que está puesta. La diferencia es que en estas aplicaciones se modela *directamente* la tela como piel del personaje.

Nosotros, una vez puestas las prendas sobre el personaje, ubicamos para cada partícula cuál es el punto más cercano en la piel del personaje. Luego ubicamos qué huesos de la animación del personaje mueven este punto de la piel, y asignamos los mismos huesos a la partícula de tela. De esta manera nos aseguramos que la partícula siga los mismos movimientos que la piel, pero siempre conservando la distancia entre la piel y la tela.

Este algoritmo es evidentemente poco verosímil, pero la ventaja que tiene es su gran performance y además, es la base de una solución intermedia: la colisión por piel.

4.3 Colisión

Un aspecto muy importante en la animación por computadora es la interacción entre objetos en el mundo virtual, siendo la colisión la principal de estas interacciones. El problema de hacer interactuar consistentemente los objetos mediante la colisión es posible dividirlo en dos procesos disjuntos: la detección de la colisión y la reacción ante la colisión.

La detección consiste en examinar si durante la ejecución de la simulación ha ocurrido colisión entre la ropa y el personaje, la ropa y algún otro objeto en la escena o si la tela ha colisionado consigo misma. La reacción, asumiendo que la colisión ha ocurrido y se conoce la posición exacta donde se produjo la colisión, consiste en modificar la siguiente posición y velocidad de los vértices que han colisionado aplicándole nuevas fuerzas.

Dada la exigencia computacional que este proceso requiere, el manejo de colisión puede volverse el principal cuello de botella y ser la causa de dificultades para mantener la animación en tiempo real. Dado que para lograr una animación plausible, tanto la piel como los modelos de las telas son representados por mallas de numerosos polígonos, es muy importante desarrollar algoritmos robustos y eficientes que permitan alcanzar de manera viable simulaciones en tiempo real.

Los materiales sumamente deformables como las telas, dado que sistemáticamente se doblan, retuercen y caen están sujetos a un gran número de puntos de contacto cuando se encuentran próximos a otros objetos de movimiento. Más aún, las telas están muy expuestas al contacto consigo mismas generando una gran cantidad de áreas de auto-colisión.

Todos los puntos que están sobre la superficie de la tela pueden, potencialmente, chocar con un objeto externo o consigo mismas en un instante arbitrario en el tiempo. Dado que las telas son objetos flexibles muy delgados, hasta pequeñas interpenetraciones pueden conducir a que el paño sobresalga del lado incorrecto. Esta situación es visualmente inquietante y puede ser difícil de corregir después de que suceda.

Una vez detectada la colisión se requiere computar la reacción ante la misma, cuyo objetivo es corregir la situación para que no se dé la interpenetración. Gracias a la integración de Verlet, podemos corregir directamente las posiciones de las partículas de tela: movemos las partículas hasta que estén en contacto con el objeto que chocaron (sin llegar a penetrarlo). También, modificamos la velocidad (a la posición anterior) para reflejar la pérdida de energía y el rozamiento.

Veremos aquí los algoritmos que implementamos de colisión de la tela contra el cuerpo.

4.3.1 Colisión contra volúmenes de frontera

Una de las tácticas más utilizadas para detectar la colisión contra un cuerpo es, en lugar de utilizar un modelo que defina al cuerpo en detalle (polígonos), usar una aproximación del mismo. El cuerpo es modelado entonces con figuras geométricas muy básicas, las cuales evitan tener que chequear la colisión sobre todos los polígonos del cuerpo. A estas formas se las llama volúmenes de frontera (bounding volumes).

En nuestra implementación particular, basándonos en [RUD02] usamos elipses para modelar el cuerpo ya que permite una buena aproximación al cuerpo sin caras planas. El problema de las caras planas (y por lo tanto de usar cajas de colisión) es que es muy notorio al ver una tela apoyada en ellas. En las figuras 11, 12 y 13 vemos un ejemplo de cómo se modela un cuerpo con elipses.

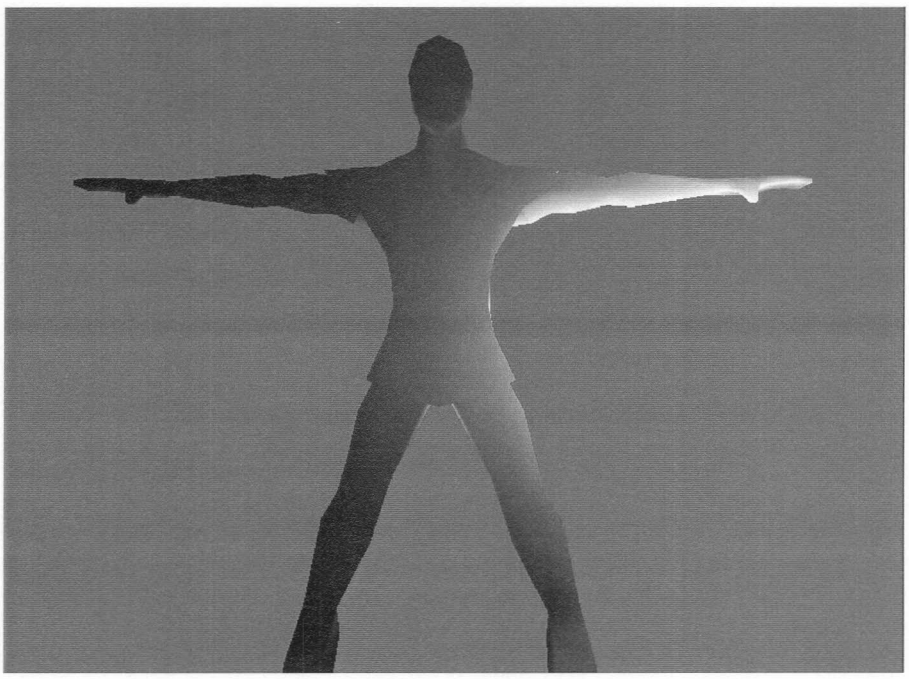


Figura 11: Cuerpo modelado con triángulos utilizado para la representación visual

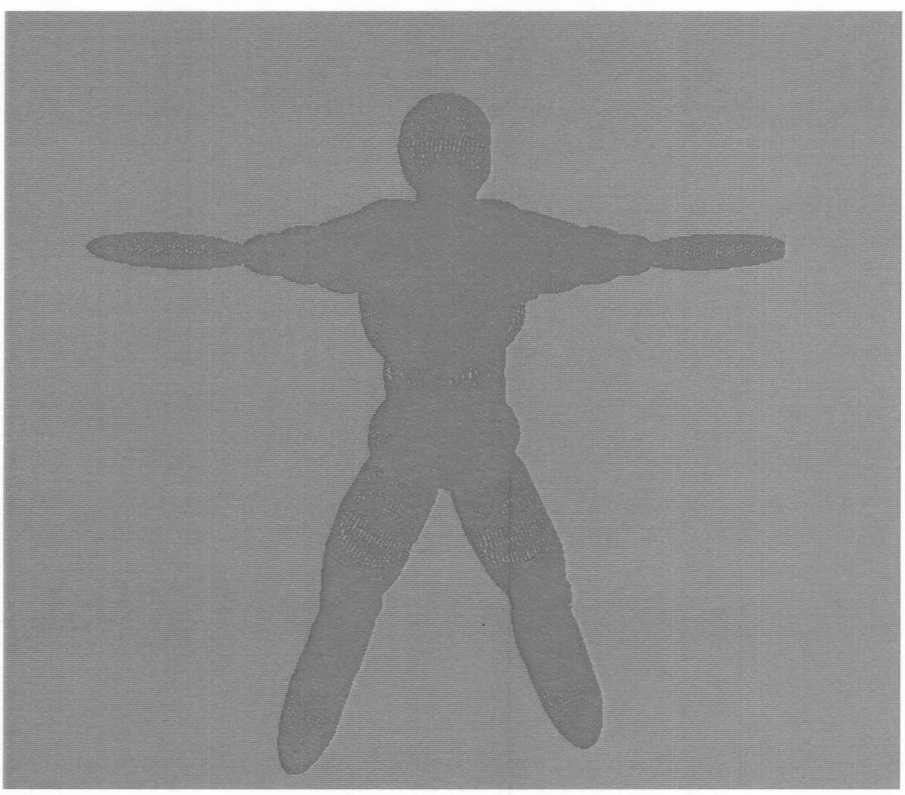


Figura 12: Modelo simplificado del cuerpo, construido con elipses, utilizado para colisión

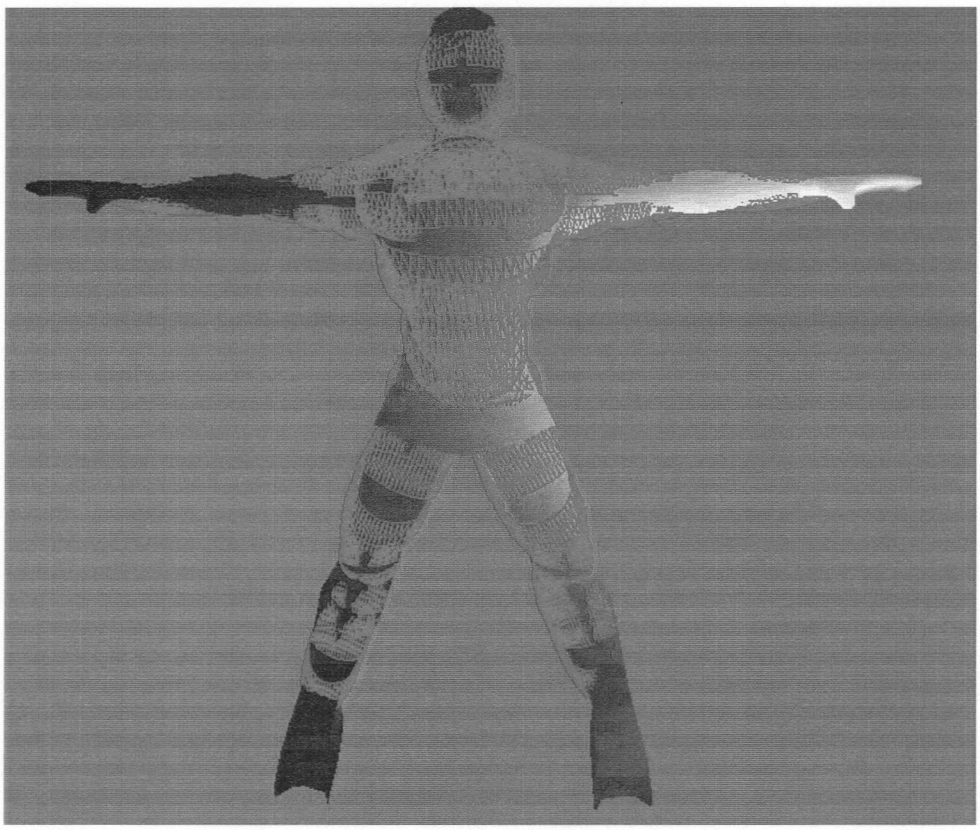


Figura 13: Superposición de ambas representaciones: gráfica y de colisión

Además las elipses proveen una manera muy económica de verificar si un punto está dentro o fuera de él: con un cambio de espacio se puede pasar a tener una esfera y ahí es sólo cuestión de ver la distancia .

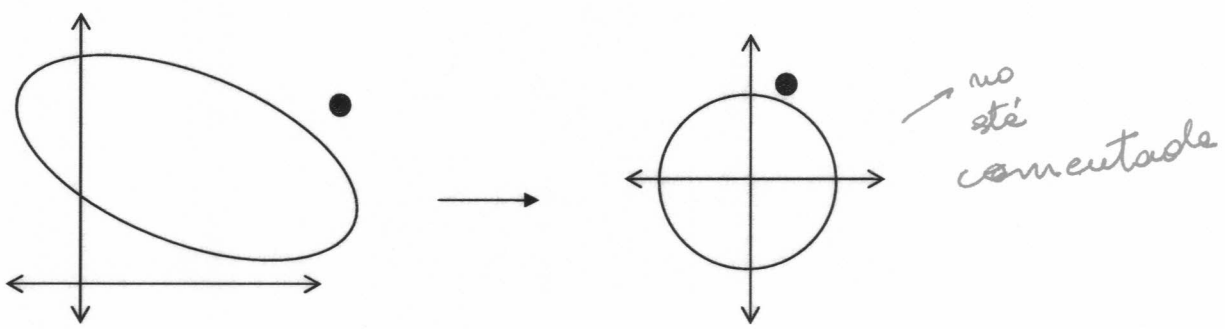


Figura 14: Pasaje de elipse descentrado a esfera centrada

Los pasos de este algoritmo son:

Preproceso:

Se define la frontera utilizando elipses. La precisión con que esté definida la frontera del personaje resultante depende casi exclusivamente de la habilidad de quien defina los tamaños y orientaciones de las elipses. Las elipses se modelan como esferas deformadas por una matriz: los escalados representan los tamaños en distintos ejes, la rotación y la traslación se usan para que siga el cuerpo.

Detección:

Para cada partícula se verifica si está penetrando alguna de las elipses que definen al cuerpo del personaje. Para esto se toma la elipse y se le aplica la transformación inversa a la de modelado para volver a la esfera. Allí vemos si la distancia de la partícula hasta el centro es menor que el radio. En caso afirmativo, está colisionando.

Reacción ante la colisión:

Consiste en tomar el segmento desde el centro de la esfera y extenderlo hasta la superficie de la esfera. Luego, aplicar la transformación y quedara el punto sobre la superficie de la elipse.

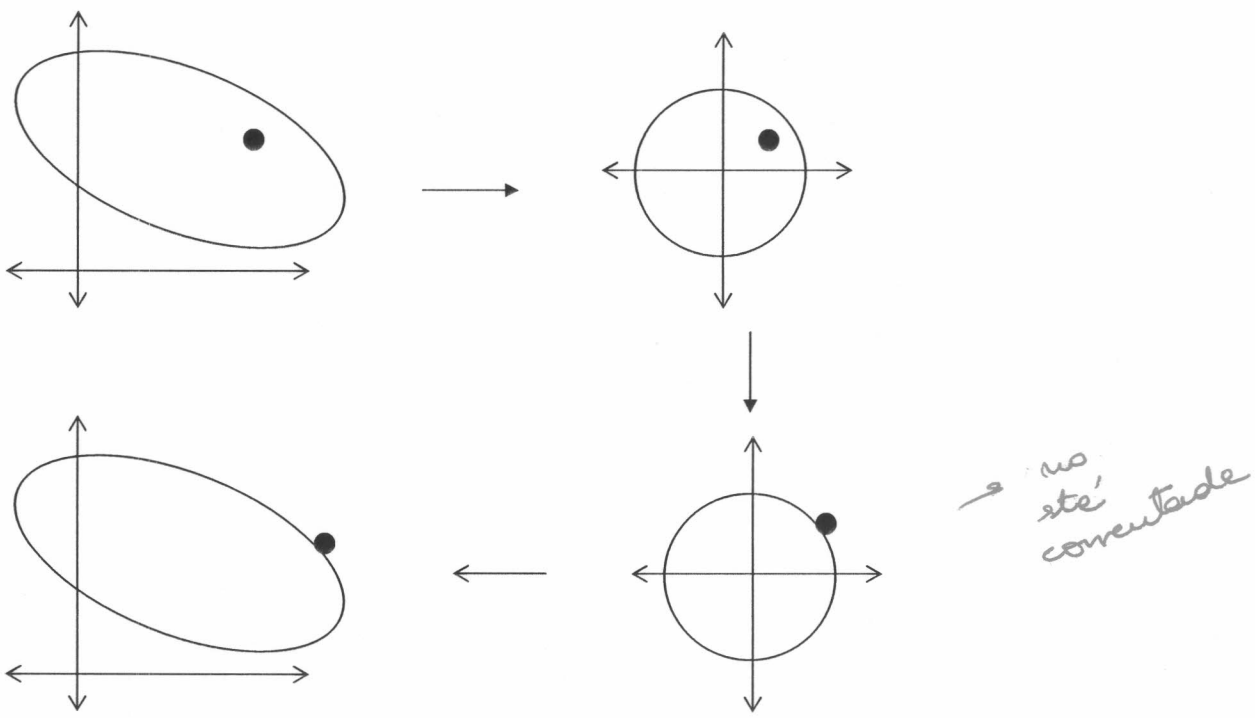


Figura 15: Detección y resolución de penetración sobre las elipses mediante cambio de espacio

4.3.2 Colisión Piel

Si bien el anterior algoritmo es de muy buena performance buscamos una alternativa de menor costo computacional. Buscamos encontrar un algoritmo que permita a distancia intermedia tener tela con vuelo con algún método de integración física, pero no necesariamente 100% adaptada al cuerpo. Es decir, buscamos un algoritmo que permita un nivel intermedio entre la integración y colisión correctas y los algoritmos no físicos que exhiben un movimiento inverosímil de las prendas.

Para esto nos inspiramos en el algoritmo de piel: Usar la piel para colisionar, no para mover la tela. Para este algoritmo asumimos que la tela siempre está más o menos cerca de la misma porción de piel. La idea del algoritmo es: para cada partícula chequear que no se esté introduciendo dentro de la porción de piel a la que está asignada en el algoritmo de integración de piel.

Detección de colisión:

Esto se logra asignando a cada partícula en la prenda puesta sobre el personaje un triángulo correspondiente a la piel del personaje. Hay un lado del triángulo de cual es correcto que la partícula se encuentre (el lado del triángulo correspondiente a la zona exterior del cuerpo) y otro que es incorrecto (el lado del triángulo correspondiente a la zona interior del cuerpo). Es decir, que de encontrarse de este lado, la tela está penetrando el cuerpo del personaje.

Entonces establecemos como única restricción que todas las partículas tienen que estar del lado correcto del plano del triangulo, y solamente hacemos esta verificación. Este algoritmo se basa en la premisa de que el cuerpo está compuesto generalmente por formas convexas, cilíndricas (brazos, torsos, piernas).

Reacción ante la colisión:

Consiste en llevar la partícula a la posición de intersección entre el segmento definido por su trayectoria y el plano del triángulo que tiene asociado. Luego proyectamos la trayectoria que fue cortada sobre el plano, aplicamos el efecto de rozamiento y movemos la partícula esa distancia.

Las fortaleza de este algoritmo es sin duda su performance, mientras que la desventaja es que se basa en una premisa muy fuerte sobre la forma del cuerpo que generalmente no se cumple por completo.

4.3.3 Auto Colisión

En las secciones anteriores vimos el proceso de la colisión entre una porción de tela y un cuerpo rígido. Otro aspecto muy importante en el proceso de simulación de telas es la auto colisión. Este problema no puede ser resuelto con las aproximaciones anteriores por diversas razones:

- En la auto colisión, las dos partes que intervienen el la colisión reaccionan ante el choque (en la colisión con cuerpos rígidos, sólo la tela reacciona ante la colisión).

- La tela no tiene asociado permanentemente un lado interior y otro exterior, sino que queda determinado por la posición relativa de las partes que colisionan (en la colisión con cuerpos rígidos, el lado interior y exterior del cuerpo rígido no varía).

En la figura 16 se visualiza el corte longitudinal de la simulación de una pieza de tela sobre la cual no se realiza auto colisión.

Entre los métodos para resolver el problema de auto colisión, además de utilizarse la aproximación de detección-reacción suelen utilizarse métodos preventivos. Estos métodos existen principalmente por la complicación que requiere "desenredar" la tela una vez que fue autopenetrada dado que, como ya comentamos, la tela no tiene un lado interior y otro exterior. Luego, evitan que la colisión ocurra exigiendo que siempre un punto en la tela permanezca a una distancia mayor que la distancia mínima establecida.

Los métodos que nosotros implementamos usan el esquema de detección: verifican si hay triángulos de tela que se interpenetraron. Teóricamente, es simple: ver que ningún triangulo se haya interpenetrado con ningún triángulo. Pero así expuesto, se torna un algoritmo cuadrático de chequear todos contra todos. En la sección 5.7.3 veremos de qué maneras implementamos este chequeo.

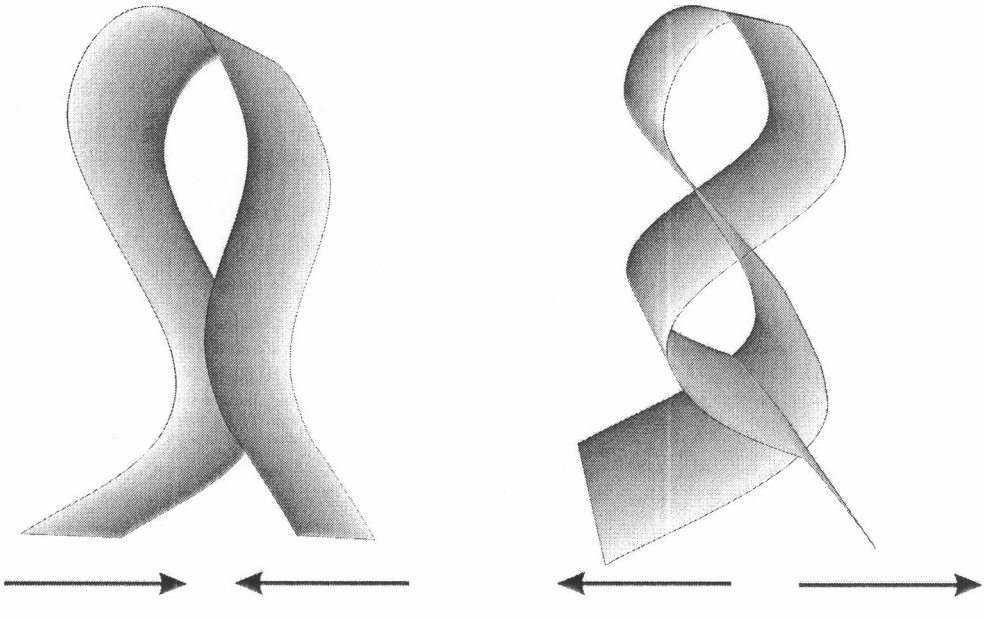


Figura 16: Corte longitudinal de la simulación de una pieza de tela sobre la cual no se realiza auto colisión.

5 Desarrollo de un sistema integral de simulación de prendas

En este capítulo analizaremos el desarrollo de un sistema integral de simulación de prendas, examinando las soluciones aquí propuestas. Se desarrolló como prueba concepto de todo este análisis un sistema integral de simulación, dentro del cual se implementó cada una de las soluciones propuestas. Esto asegura que la solución y el análisis del problema son correctos no sólo desde el punto de vista teórico sino del práctico también.

5.1 Características deseadas del sistema

El sistema a desarrollar está pensado para soportar aplicaciones que necesiten realizar simulación de prendas en tiempo real, por lo tanto la eficiencia y rapidez del sistema son condiciones sine qua non. Como ya discutimos en capítulos anteriores, un rasgo característico de las simulaciones de tela en tiempo real es su tendencia a la inestabilidad numérica. Es imprescindible contar con mecanismos que permitan asegurar la estabilidad de la simulación.

El sistema debe permitir simular la mayor variedad de prendas posible, sobre personajes tanto de morfología humana como no humana. También debe posibilitar la simulación en todo el espectro posible de calidad permitido por el hardware, desde simular una prenda con máxima calidad o muchas con poca calidad, debe ser lo suficientemente flexible como para utilizar distintos algoritmos sobre las mismas prendas. Además la arquitectura debe ser lo suficientemente abierta como para permitir ingresar nuevos algoritmos.

El sistema así es de interés en muchos tipos de aplicaciones, por lo tanto es indispensable que corra en la mayor cantidad de dispositivos posibles. Es deseable aprovechar al máximo el potencial de cada plataforma, pero más importante es que reconozca cuándo la capacidad no es suficiente y escale hacia abajo.

Por lo tanto diremos que las características deseadas son:

- Flexibilidad
- Extensibilidad
- Portabilidad
- Rapidez
- Estabilidad

Veremos en este capítulo los temas relacionados con la construcción de un sistema de simulación de estas características, pero varios aspectos relacionados con la rapidez y estabilidad serán desarrollados en el capítulo siguiente.

5.2 Consideraciones sobre la plataforma

El sistema a desarrollar es muy exigente a nivel de performance ya que para lograr la simulación entran en juego diversos algoritmos muy complejos. Estos algoritmos consumen mucho CPU y el tamaño de los datos de entrada para éstos es muy grande. Esto último no sólo impone un requerimiento de velocidad sobre el código sino también de memoria. Sería deseable tener un programa que sea portable entre diversas plataformas y sistemas operativos, que no sólo permita llegar a las PCs de escritorio sino también a sistemas de entretenimiento, consolas de videojuegos, etc.

Por estas razones elegimos el lenguaje C++, que resulta óptimo: La orientación a objetos de C++ junto con la producción de código optimizado para la plataforma destino y la disposición de compiladores optimizadotes en casi todas las plataformas y sistemas operativos lo convierten en la herramienta de elección, siendo el estándar de facto para las industrias de videojuegos y de aplicaciones gráficas.

Además del enorme poder computacional requerido para la simulación, un sistema así se ve caracterizado por el requerimiento gráfico para el rendering 3D de la simulación, que es bastante alto. Esto impone la necesidad de usar placas aceleradoras 3D para nuestro sistema, lo cual es casi un estándar en las PC de escritorio.

5.2.1 Motor gráfico

Sin embargo, la utilización de las capacidades de estas placas no es consecuencia directa de poseer una, sino que se requiere contar con un motor 3D, que maneje la interacción CPU - GPU[3] y los datos necesarios para la representación gráfica. Un motor 3D se puede realizar desde cero con conocimientos de alguna API gráfica como OpenGL o Direct3D, o puede usarse algún motor ya construido. Existen motores de todo tipo y cubren todo el rango desde tecnología propietaria hasta de código abierto.

Para nuestra implementación, consideramos que desarrollar un motor es una tarea demasiado grande que nos habría distraído de nuestro objetivo principal, por lo que decidimos utilizar un motor de código abierto.

Criterios para evaluar y decidir qué motor utilizar hay varios, dentro los cuales están la facilidad de uso de la API, la documentación existente, la curva de aprendizaje, las funcionalidades que provee, su velocidad, los tipos de archivos que soporta para datos, etc.

En nuestro caso, nosotros precisábamos que el motor tuviera animación esqueletal, soporte para modelos 3D del 3D Studio max, que fuera simple de usar y

[3] Graphics Processing Unit: La placa gráfica

fácil de aprender (para la implementación de un sistema de simulación de prendas dentro de una aplicación de mayor tamaño los criterios pueden ser otros).

Se evaluaron distintas alternativas dentro de los motores de código abierto: OGRE, Cristal Space e Irrlicht, todos los cuales ofrecen las funcionalidades necesarias. Finalmente se optó por Irrlicht que, aunque sea el menos avanzado de los tres en cuanto a funcionalidad, resultó ser el más simple y fácil de aprender.

La desventaja de utilizar un motor es que no es posible predecir cómo será exactamente la funcionalidad que dice tener hasta que esta funcionalidad es necesitada. Esto puede requerir cierto desarrollo sobre el código del motor (si es posible) para ajustar a nuestras necesidades. En nuestra experiencia el motor proveía animación esqueletal pero la API ocultaba al usuario los detalles de la misma, cosa que necesitábamos para el algoritmo de piel. Esto implicó realizar cambios en la API del motor para poder hacer públicos estos detalles. De todos modos, este esfuerzo fue sin duda menor que el de desarrollar un motor desde cero.

5.3 Pasos en el desarrollo

Para la construcción de nuestro sistema de simulación de prendas utilizamos una aproximación bottom-up. Es posible dividir los pasos en el desarrollo de la siguiente forma.

1. **Armado del escenario y métodos que hacen avanzar la simulación:** Consiste en desarrollar los objetos que componen la realidad en la que evoluciona la simulación más la implementación de los algoritmos que hacen avanzar al sistema.

2. **Realización las mediciones:** Diseño, implementación, ejecución y selección de las mediciones más representativas que sirvan de referencia para la toma de decisiones sobre performance y calidad.

3. **Implementación de sistema de LOD:** Desarrollo de los algoritmos y estructuras de datos que den soporte al sistema de LOD

En términos generales, el proceso de desarrollo consistió, en primer lugar, en diseñar e implementar estructuras de datos de y algoritmos sobre los objetos que modelan a la tela. Luego, implementamos el conjunto de algoritmos que permitían simular el movimiento de las telas. Una vez que logramos simular el movimiento de una porción de tela suelta, tomamos mediciones que nos dieron una visión amplia sobre la factibilidad de utilización de cada uno de los algoritmos.

Luego desarrollamos los algoritmos de colisión de la ropa sobre los personajes y de la ropa consigo misma y tomamos nuevamente mediciones. Finalmente desarrollamos el algoritmo que vista a los personajes con las prendas diseñadas y

tomamos las últimas mediciones, las cuales sirvieron al desarrollo del algoritmo de LOD.

En las sucesivas secciones revisaremos con mayor profundidad los puntos más significativos en el desarrollo de la aplicación. El detalle del sistema de LOD lo exponemos en siguiente capítulo.

5.4 Objetos base

En esta sección describimos en detalle la forma en que desarrollamos los objetos que componen la realidad de nuestra simulación.

5.4.1 Molde

El molde es el objeto que modela al croquis utilizado por los diseñadores de indumentaria para representar la forma en que debe cortarse la tela. En nuestra aplicación, el molde puede potencialmente representar una figura geométrica (polígono cerrado) con forma arbitraria. No existen limitaciones para la geometría del molde. Sirve principalmente al proceso de construcción de la malla-molde. En la figura 17 se visualiza el ejemplo del molde de una remera.

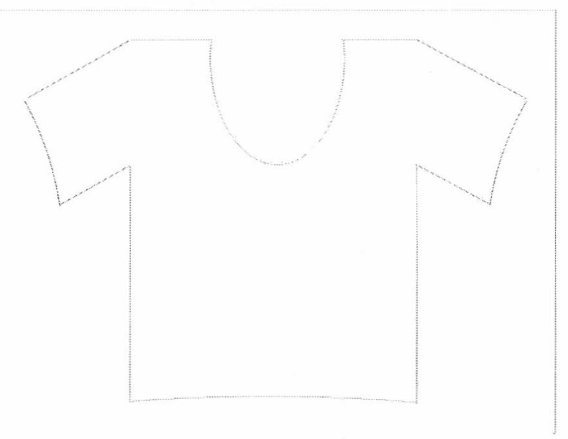


Figura 17: Molde de una remera

La estructura de datos utilizada para almacenar la información asociada a los moldes, consta de un conjunto de polígonos convexos combinados de modo tal que puedan formar figuras geométricas arbitrarias. Es prácticamente en su totalidad inventiva del diseñador.

5.4.2 Malla-molde

Una malla molde es la representación geométrica de una malla, con granularidad y estructura definidas según la densidad y topología que se le asigne, delimitada por la forma del molde y dispuesta sobre un plano. La malla molde se construye a partir

de un molde más densidad y topología adecuadas. El proceso de construcción de una malla molde convierte a la información continua de un molde en una malla con características discretas: nodos asociados por uniones. El proceso completo para crear una Malla Molde se describe en la sección 5.5.2: Cómo crear una malla molde.

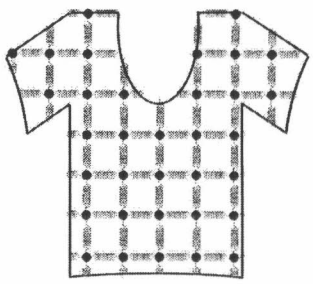


Figura 18: Malla-molde de una remera

La principal estructura de datos es una cuadrícula. La cuadrícula divide al espacio del plano del molde en rectángulos. Cada rectángulo puede tener o no asociada una partícula. Su principal función es proveer de información sobre la posición relativa en un espacio de dos dimensiones de las partículas. En la figura 18 muestra como ejemplo una Malla-molde para una remera.

5.4.3 Malla-tela

Una prenda es el resultado de unir de forma arbitraria varias Mallas Molde, donde además de la topología de la malla se consideran las características físicas de las telas. malla-tela es la representación física de la tela discretizada en masas y resortes. La malla-tela se construye a partir de una malla-molde. Una malla-tela a su vez se puede generar de la composición de varias malla-tela (donde cada una proviene de un malla-molde). Esto representa el proceso de coser juntas dos piezas cortadas en forma de moldes. La costura se implementa agregando nuevos resortes, muy poco compresibles en el lugar donde debieran ir las costuras.

Potencialmente puede ser necesario hacer dos discretizaciones con distintas topologías. Una de ellas necesaria para establecer las partículas y los resortes, y otra de ellas para triangular la figura para que pueda ser mostrada.

Es posible seleccionar de forma arbitraria la tela asociada a la malla-tela dado que la malla-tela tiene asociada una estructura de datos que representa las propiedades físicas de la tela la cual permite asociar una tela con características arbitrarias a cualquier prenda. Es la información que se utiliza al momento de asignar valores a las variables necesarias para la simulación.

La malla-tela conoce las características físicas de la tela necesarias para la simulación, como por ejemplo: el peso especifico, los factores de rozamiento; la flexibilidad (dureza de los resortes), rozamiento con el aire (dos constantes: k_d y k_l) las cuales serán utilizadas por los distintos algoritmos de simulación. De esta

manera, pueden con suficiente información formarse telas arbitrarias, como jean, jersey... etc.

5.4.4 Prenda

Una prenda es un conjunto de mallas tela. Es posible seleccionar un conjunto arbitrario de mallas molde y unirlos también de forma arbitraria para formar una prenda. Dos mallas-tela se pueden pegar para componer una sola, por lo que permiten armar prendas de manera recursiva.

5.4.5 Tela-simulada

A partir de una malla-tela se construye una Tela-simulada, la cual permite representar una o varias piezas de tela juntas. Una Tela-simulada almacena, además de la malla y la representación visual de la misma, datos necesarios para llevar adelante la simulación del movimiento de la tela. Incluye información acerca del cuerpo sobre el que está puesta la Tela-simulada y los algoritmos de simulación y colisión con los que va a simularse.

5.5 Algoritmos para manipular objetos base

Presentamos en esta sección tres algoritmos significativos. Los dos primeros, utilizados para la construcción de los objetos presentados en la sección anterior, y último de ellos utilizado para poner sobre un personaje determinada prenda, es decir, viste al personaje.

5.5.1 Cómo utilizar un molde

La principal función del molde es determinar si un punto está dentro o fuera de la poligonal que lo define. Se lo utiliza en el proceso de creación de la malla-molde para determinar si un punto en el espacio bidimensional es interno o externo al molde de la malla-molde en proceso de creación.

5.5.2 Cómo crear una malla molde

Una Malla Molde se crea discretizando un molde según la densidad y la topología que se desee asociar a la malla. El proceso de creación se divide en dos partes:

- **Puntos internos en la malla**

A partir de un conjunto infinito de puntos alineados vertical y horizontalmente a la distancia definida por la densidad, se descartan todos aquellos que exceden los límites del molde. Dependiendo de la topología definida para la malla, se utiliza la

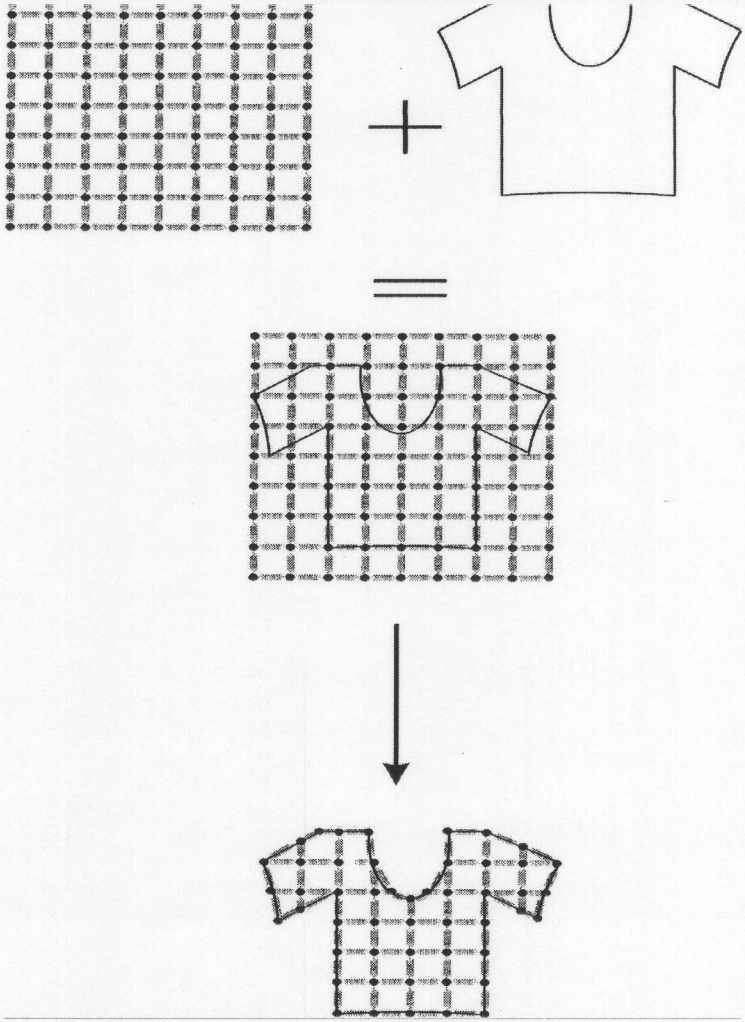


Figura 19: Proceso de creación de una Malla Molde

5.5.3 Cómo poner la ropa a un personaje

El presente algoritmo emula las acciones que debiera realizar una persona para ponerse una prenda. Para un personaje y una prenda determinados, ubica sobre el personaje la prenda, es decir, viste al personaje con la prenda.

Generalmente una prenda es una malla-tela que ha sido compuesta por varias mallas-tela, cada una de estas proveniente de un molde (del mismo modo que sucede en la realidad). Nos basamos en esto para poner la prenda sobre el personaje.

Nuestro acercamiento consiste en ubicar las mallas-tela sin componer alrededor del personaje en reposo, cerca del lugar donde quedarían ubicadas una vez compuestas. Esto se realiza mediante roto-traslaciones de las mallas-tela hasta dicha posición. Luego componemos la malla-tela final añadiendo resortes entre las costuras de las distintas mallas-tela. Finalmente vamos iterativamente llevando los resortes hasta su posición de reposo (aplicando el algoritmo de restricciones) al mismo tiempo que hacemos colisionar la tela contra el cuerpo (aplicando el algoritmo de elipses) para evitar que la tela lo penetre.

En las siguientes figuras vemos una secuencia que ejemplifica este proceso:

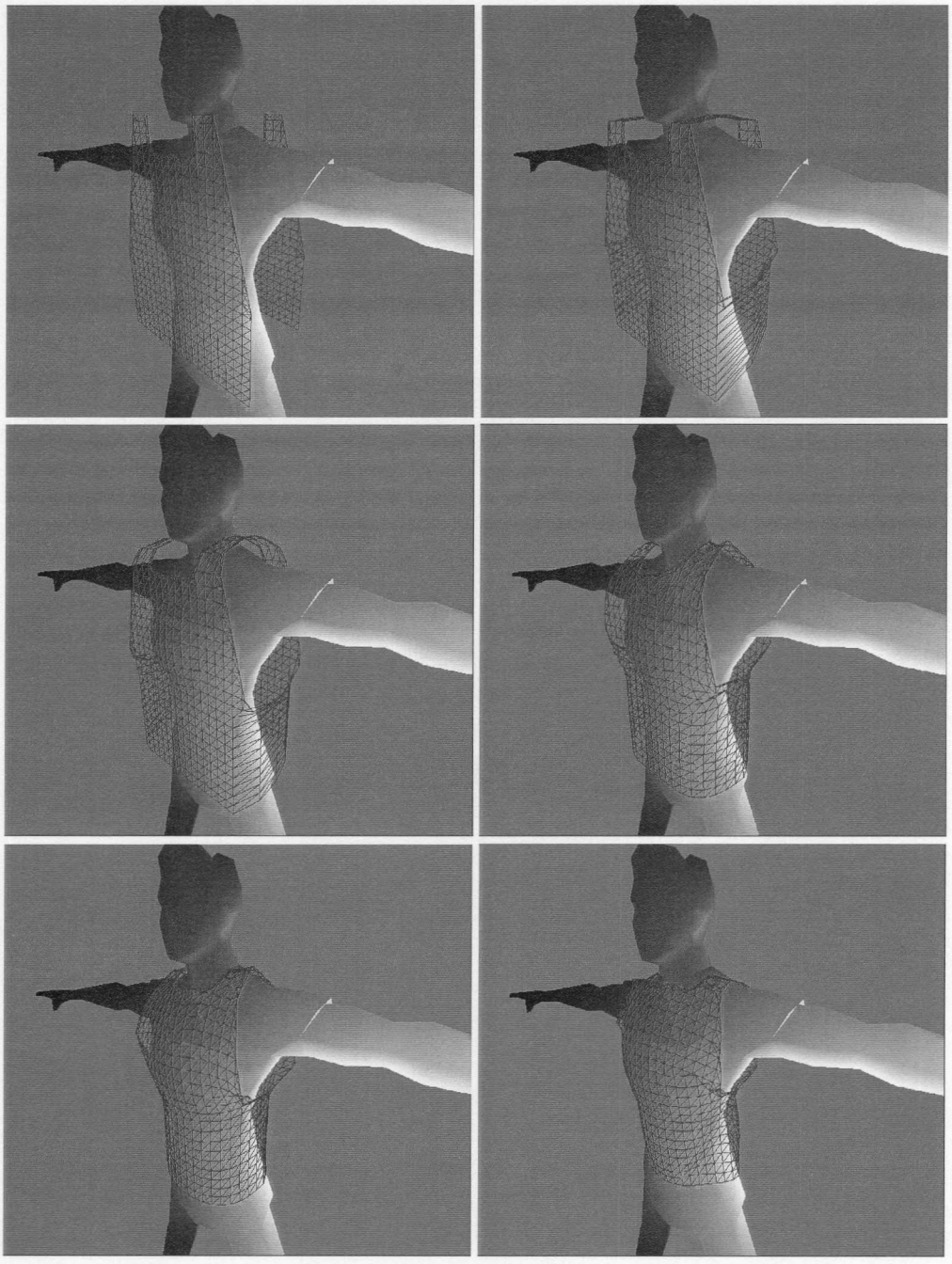


Figura 20: Secuencia de vestido del personaje con una musculosa

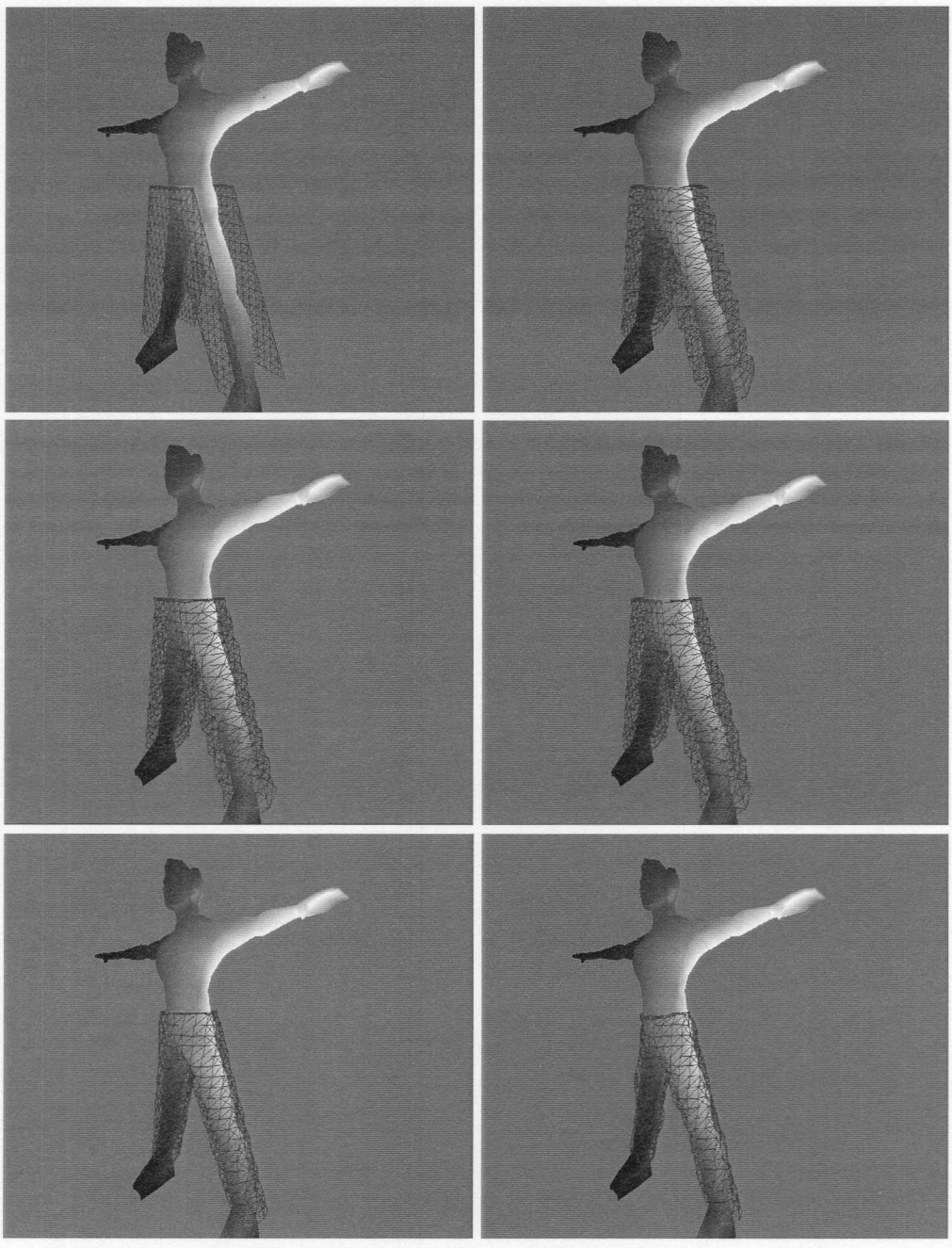


Figura 21: Secuencia de vestido del personaje con un pantalón

5.6 Arquitectura

Los objetos base modelan la realidad, el proceso de diseño y sirven para pasar de diseño a una tela simulada. Pero para lograr la simulación hace falta un sistema que integre todos estos objetos, les de una coherencia y haga manejable una creciente variedad y cantidad de objetos. Son varios los puntos de vista sobre la arquitectura y cada uno de estos impone ciertas restricciones o necesidades a la misma.

5.6.1 Requerimientos de la arquitectura

Desde el punto de vista de datos, nuestro objetivo es el de poder tener un entorno complejo con varios personajes vestidos, que van entrando y saliendo de escena. Necesitamos una arquitectura que pueda manejar esta situación así como mantener acoplada la información de las asociaciones entre las distintas entidades. De lo contrario esta información resultaría abrumadora para el programador.

Desde el punto de vista de simulación, queremos que sea fácil cambiar de algoritmos de simulación para cada prenda. Queremos que esto no dependa de algo prefijado sino que en run-time se pueda alternar entre algoritmos. Esto, por ejemplo, sirve para nuestro sistema de LOD que requiere constantes cambios de algoritmos.

La performance está muy ligada a la facilidad del sistema de cambiar de algoritmos. Queremos que el sistema pueda correr lo más rápido posible pero además que demore lo menos posible en comenzar a funcionar. Muchos de nuestros algoritmos requieren un elevado tiempo de preproceso y en una corrida lo más probable es que no se ejecuten todos estos algoritmos. Entonces, tanto en performance como en simulación, necesitamos que el sistema sea adaptable a las demandas de la corrida en curso sin perjuicio para otras corridas.

La arquitectura, además, debe facilitar la realización de mediciones automatizadas en una gran variedad de escenarios. De esta manera sin intervención humana se pueden correr pruebas extensas en varios escenarios y además pueden guardarse las configuraciones de escenario para correr las pruebas nuevamente si es preciso.

5.6.2 Descripción de la arquitectura

La estructura de la aplicación se basa en unos pocos objetos que logran darle un sentido global a los pequeños objetos que representan datos, de tal manera que constituyan un todo integral y se den relaciones significativas entre los datos.

Estos objetos son el centro de la aplicación y cumplen el patrón singleton: son únicos en el sistema[4]. Estos objetos centrales son 4:

- Escena
- Manejador de cuerpos vestidos
- LOD
- Ejecutante de Casos

La *escena* constituye la metáfora de escena y es el objeto unificante en el sistema. Representa el concepto de escena de computación gráfica: el lugar donde está todo lo que se va a representar en pantalla. En la escena se insertan personajes, personajes vestidos, telas sueltas, otros objetos, luces, la cámara, fuentes de sonido, etc. Está muy ligado al árbol de escena. Es también el objeto que recibe y ejecuta los cambios indicados por el input del usuario. En el objeto escena se unifica el procesamiento de todo lo que se ve en pantalla dando un orden coherente y orquestando los distintos *sectores* de procesamiento de la aplicación.

Uno de estos sectores es el de procesamiento de personajes vestidos. Para aislar el procesamiento de personajes vestidos de todo el resto del procesamiento creamos el objeto *manejador de cuerpos vestidos*. En este objeto se registran específicamente tanto los personajes vestidos como las telas sueltas (un telón por ejemplo). Si bien sólo se encarga de realizar el avance de las simulaciones de tela, lo llamamos manejador de cuerpos vestidos porque es el encargado de mantener el vínculo entre las telas y el cuerpo que las viste.

El *LOD* es el objeto encargado del sistema de niveles de detalle (ver capítulo 6). Trabaja con las telas del manejador de cuerpos vestidos y su accionar es transparente al del manejador de cuerpos vestidos.

Cuando le llegue el turno (turno indicado por la escena) el manejador de cuerpos vestidos avanzará las simulaciones de todas las prendas. Pero antes de avanzarlas llama al sistema de LOD para que se ponga en marcha. Este analizará la escena y como resultado dispondrá las opciones de simulación para todas las prendas. Luego el manejador procederá a simular todas las prendas. Nótese que si el LOD está desactivado, no se ejecuta nada y el manejador de telas directamente pasa a avanzar las simulaciones de las telas.

Finalmente tenemos el *ejecutante de casos*. Nosotros llamamos caso a una configuración de la escena, lo que podría ser por ejemplo un nivel de un video juego: una escena con 3 personas vestidas, una mesa con un mantel, etc. Si queremos pasar a otra escena simplemente cambiamos el caso: una multitud vistiendo camisetas de un equipo. Básicamente se encarga de monitorear si es necesario cambiar de caso y, de serlo, quitar todo lo necesario de la escena e ingresar los nuevos objetos. Este objeto fue muy útil para realizar las mediciones, ya que no era necesaria la intervención humana para cambiar el escenario a medir.

---

[4] En realidad, el sistema que nosotros construimos soporta varias simulaciones en paralelo. En este caso existiría una copia de estos objetos por cada simulación.

En la figura 22 se visualiza la máquina de estados para el procesamiento de un cuadro.

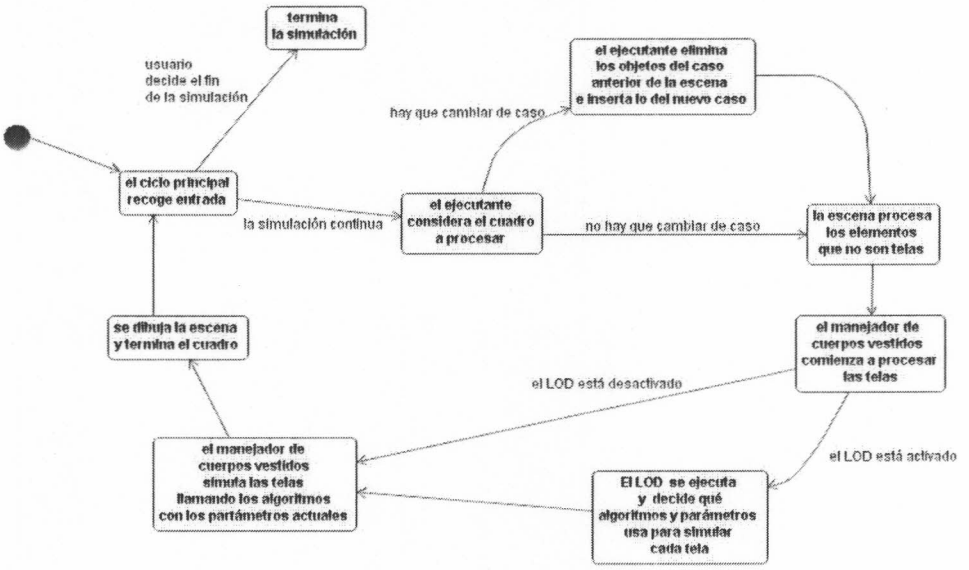


Figura 22: Procesamiento de un cuadro

5.6.3 Manejando los algoritmos

La pregunta que falta responder es cómo hacemos para simular las telas de distinto modo. Como vimos anteriormente, existe un objeto que agrega todo lo necesario para representar uno o varios trozos de tela juntas: tela simulada. En este objeto guardamos no sólo la malla y la información gráfica, sino también qué algoritmos se están usando para simular esta tela, qué parámetros tienen estos algoritmos y todos los datos de estado que necesitan todos los algoritmos.

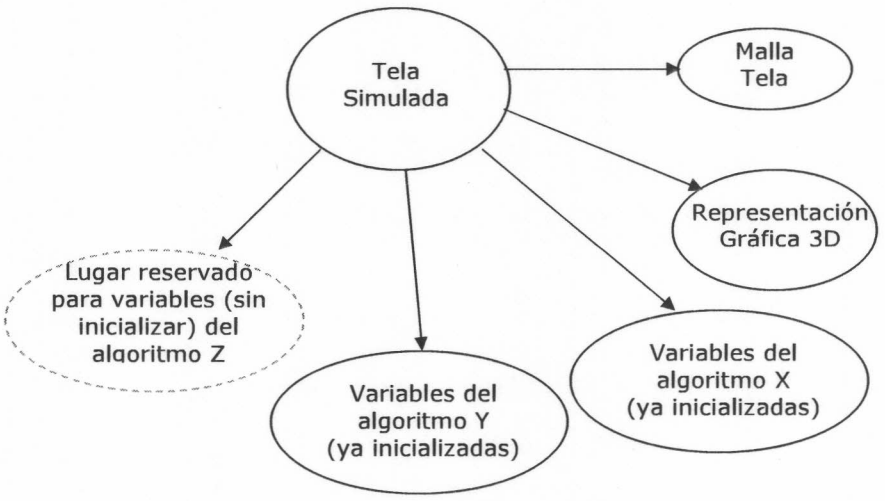


Figura 23: Composición de la tela simulada

Pero no podemos tener todos los datos de todos los algoritmos cargados, ya que eso es un desperdicio de memoria y además habría que inicializar todos los algoritmos. Para controlar esto creamos un objeto base para todos los estados de algoritmos. Este objeto *Algoritmo Base* tiene un método (abstracto) para llamar al algoritmo cuya subclase representa y, además, cuenta con lógica para inicializar el estado del algoritmo recién cuando se quiere ejecutar el algoritmo.

Esto garantiza un código sin inicializaciones que de todas maneras inicializa sólo lo justo. Esta inicialización incluso se puede realizar en un thread aparte para no trabar la simulación. De esta manera se logra un consumo óptimo de memoria y la mejor experiencia de usuario, al reducir comportamientos anómalos y tiempos de espera.

5.7 Algoritmos que implementamos

Al mismo tiempo que construíamos la arquitectura, íbamos construyendo los distintos algoritmos que enumeramos en la secciones 4.2 métodos considerados y 4.3 colisión. Estos algoritmos proporcionaban distintos desempeños, tanto en performance como en verosimilitud, como veremos en la próxima sección.

La mayoría de los algoritmos que implementamos o bien están basados directamente en los textos de los papers en los que fueron publicados o bien fueron explicados en las secciones 3.2 y 3.3. Los algoritmos que implementamos son:

- Integración
  - ✓ Algoritmo de Thin Objects
  - ✓ Algoritmo de restricciones
  - ✓ Algoritmo de piel
- Colisión
  - ✓ Algoritmo de colisión con elipses
  - ✓ Algoritmo de colisión con piel
- Auto Colisión
  - ✓ Chequeo de interpenetración

5.7.1 Implementación de algoritmos de integración

Algoritmo de Thin Objects

Tanto el algoritmo de piel como el de restricciones fueron implementados sin problemas, mientras que el algoritmo de thin objects sólo funcionaba bien al trabajar *solo*. Al hacer interactuar una tela animada por este algoritmo con el cuerpo mediante la colisión o con si misma mediante auto-colisión, la tela instantáneamente tomaba un comportamiento errático y al cabo de unos instantes divergía hacia una situación irrecuperable.

Conjeturamos que este comportamiento se debe a que el algoritmo es demasiado sensible a los cambios instantáneos de posiciones de las partículas (generados por los algoritmos de colisión). En un intento por contrarrestar el desequilibrio causado por estos movimientos, ajustamos las velocidades de las partículas según su cambio de posición para mantener la energía del sistema. Desafortunadamente, las pruebas continuaron mostrando comportamientos anómalos. Luego, al no poder interactuar con otros objetos, este algoritmo de simulación no es útil a nuestro propósito. Nuestro objetivo no es simplemente simular tela sino de tener un personaje vestido.

Algoritmo de Piel

En el algoritmo de piel, queda a libre albedrío del implementador la forma en que se asocia la tela a la piel. Nosotros vimos dos caminos: asociar a mano de la misma manera que se asocia la piel a los huesos en un modelo 3D (proceso comúnmente llamado *skinning*) o asociarla programáticamente. El primer caso requeriría un desarrollo entero de una aplicación (o plug-in para alguna aplicación existente), por lo tanto la descartamos. Entonces realizamos la asociación automática de la tela en la posición inicial.

Luego de haberla puesto sobre el personaje, como vimos en la sección 5.5.3: Cómo poner la ropa a un personaje, asociamos cada partícula al triangulo de piel más cercano. Si bien este acercamiento da buenos resultados, puede ser que se confunda cerca de pliegues muy abruptos de la piel del personaje. El problema de los vértices mal asociados es aún peor con prendas de mucho vuelo, es decir, muy separadas de la piel del personaje, como podría ser un poncho o un vestido con cola. Se podrían conseguir muy buenos resultados combinando el método automático con una posterior refinación artesanal.

En la figura 24 vemos una prenda simulada mediante el algoritmo de piel, mientras que en la 25 vemos un vértice que quedó confundido en el proceso de asociación con la piel.

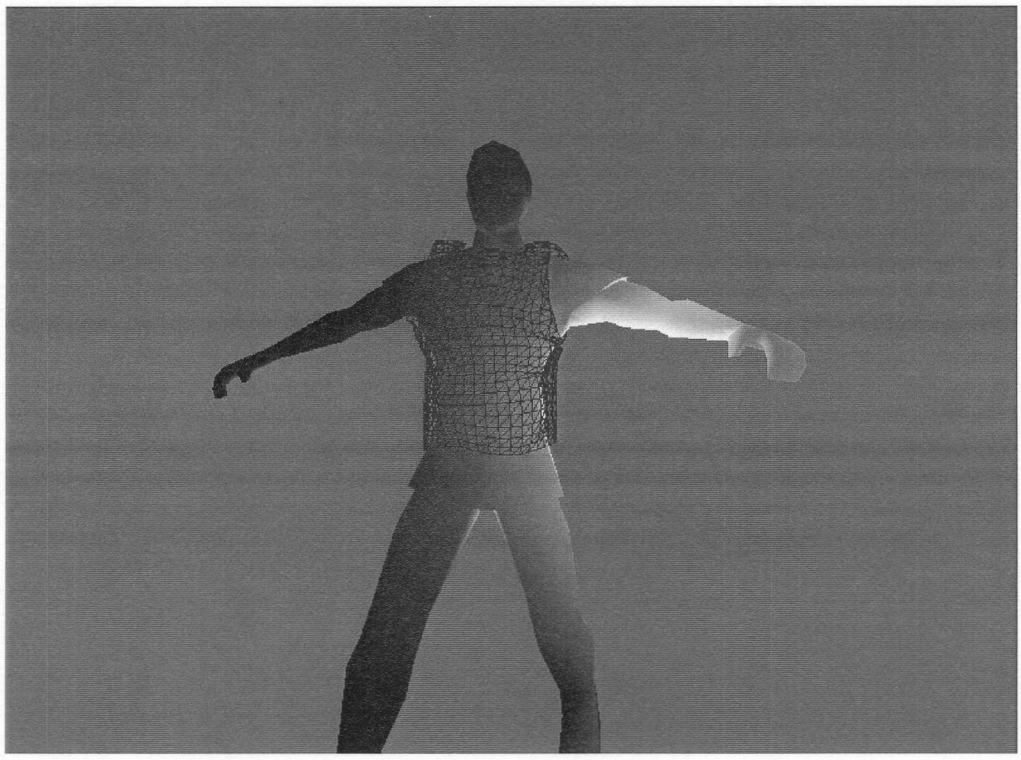


Figura 24: prenda simulada por el algoritmo de piel

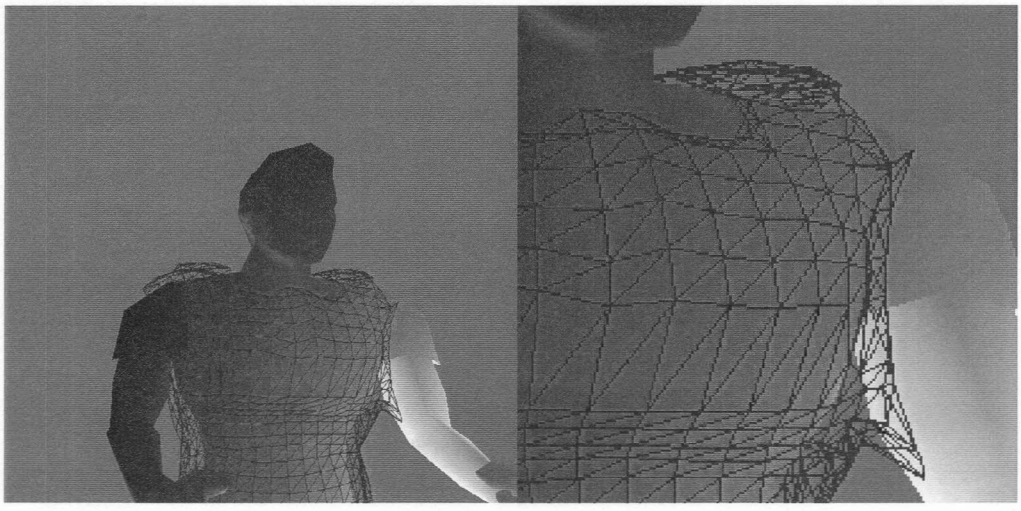


Figura 25: vértice con asociación a la piel errada (izquierda original, derecha acercamiento)

Algoritmo de restricciones

La implementación del algoritmo de restricciones fue bastante directa partiendo del paper que lo propuso originariamente ([FGL03]). La única complicación asociada a este algoritmo fue la presencia del fenómeno de *superelasticidad.* Este fenómeno se percibe si el algoritmo es ejecutado en condiciones de poca performance (estando debajo de los 30 cuadros por segundo). Bajo estas condiciones la tela parece tornarse muy pesada, estirando los resortes y aún superando las correcciones de las restricciones.

Para solucionar este inconveniente incluimos un mecanismo de control de velocidades máximas y de estiramientos. Estos mecanismos fueron parcialmente exitosos, dado que el fenómeno continúa manifestándose. Esto obliga a no ejecutar este algoritmo cuando el rendimiento es escaso. Nuestra propuesta de LOD del capítulo 6 es una forma de prevenir esto. Vemos en la figura 26 una imagen de una tela simulada con el algoritmo de restricciones y en la figura 27 una imagen del fenómeno de superelasticidad.

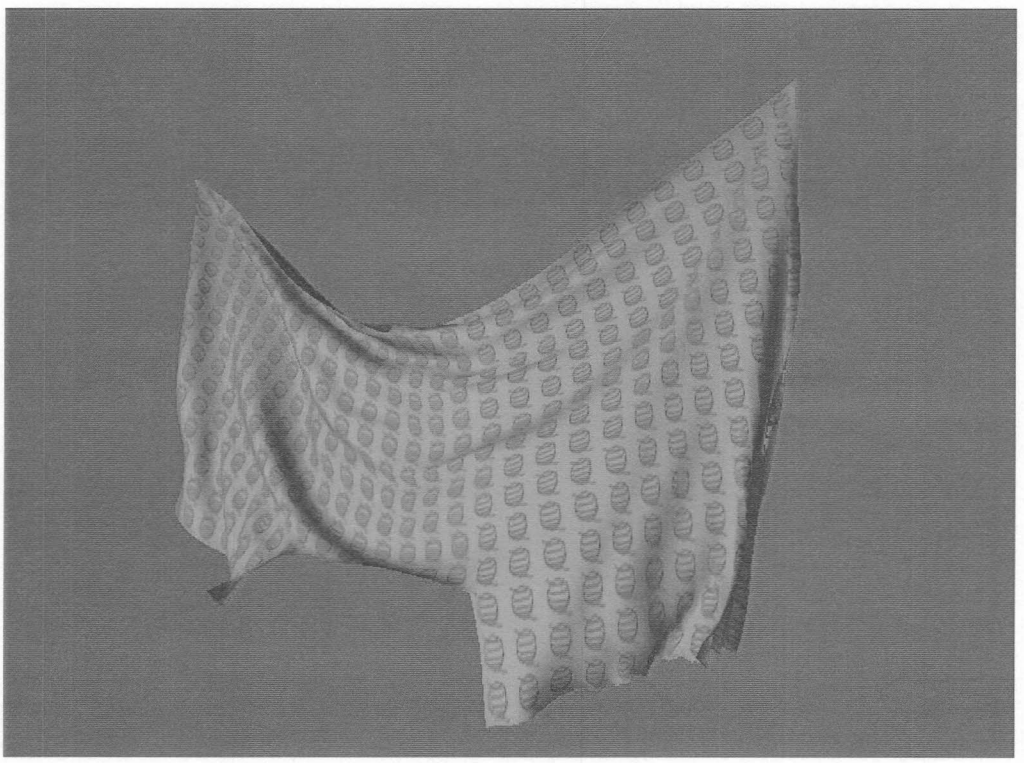


Figura 26: prenda simulada por el algoritmo de restricciones

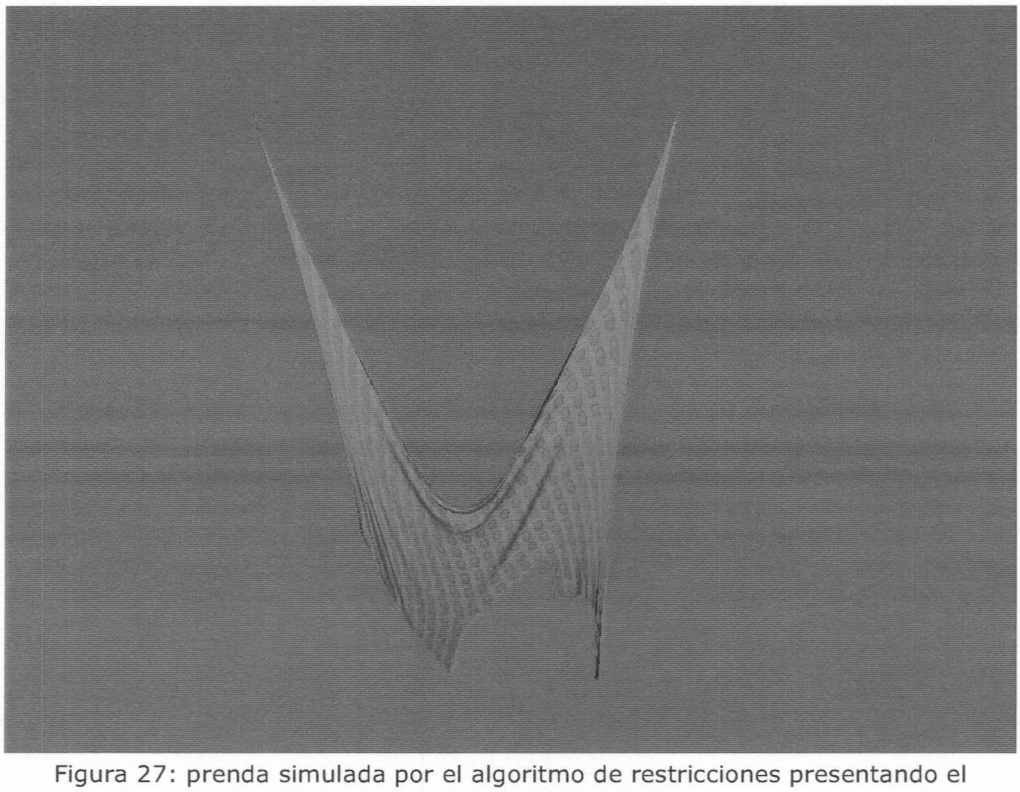


Figura 27: prenda simulada por el algoritmo de restricciones presentando el fenómeno de *superelasticidad*

5.7.2 Implementación de algoritmos de colisión

Tanto el algoritmo de colisión de piel como el de colisión de elipses proveen resultados aceptables, siendo la calidad del resultado obtenido consecuencia casi directa de la la calidad de los datos con que se alimentan los algortmos.

En el caso del algoritmo de elipses, el factor a considerar es cuán bien representado está el modelo visual por su contrapartida de volúmenes de colisión. Si el modelo no está bien representado puede ser que en algún punto la piel sobresalga a través de la tela. También se puede dar el caso opuesto, en el que el personaje parece más gordo de lo que es al tener elipses excesivamente grandes.

En el caso del algoritmo de piel, la bondad de los resultados depende de cuán bien asociadas estén las partículas de tela a la piel. Así mismo, al estar basado en el algoritmo de integración de piel, sufre las mismas desventajas que este y no es tan apto para tratar telas de mucho vuelo como sí lo está el algoritmo de elipses.

Vemos en las figuras 28 una tela con colisión de elipses y luego en la figura 29 la misma tela con colisión de piel. En ambas imágenes el algoritmo de integración es el de restricciones.

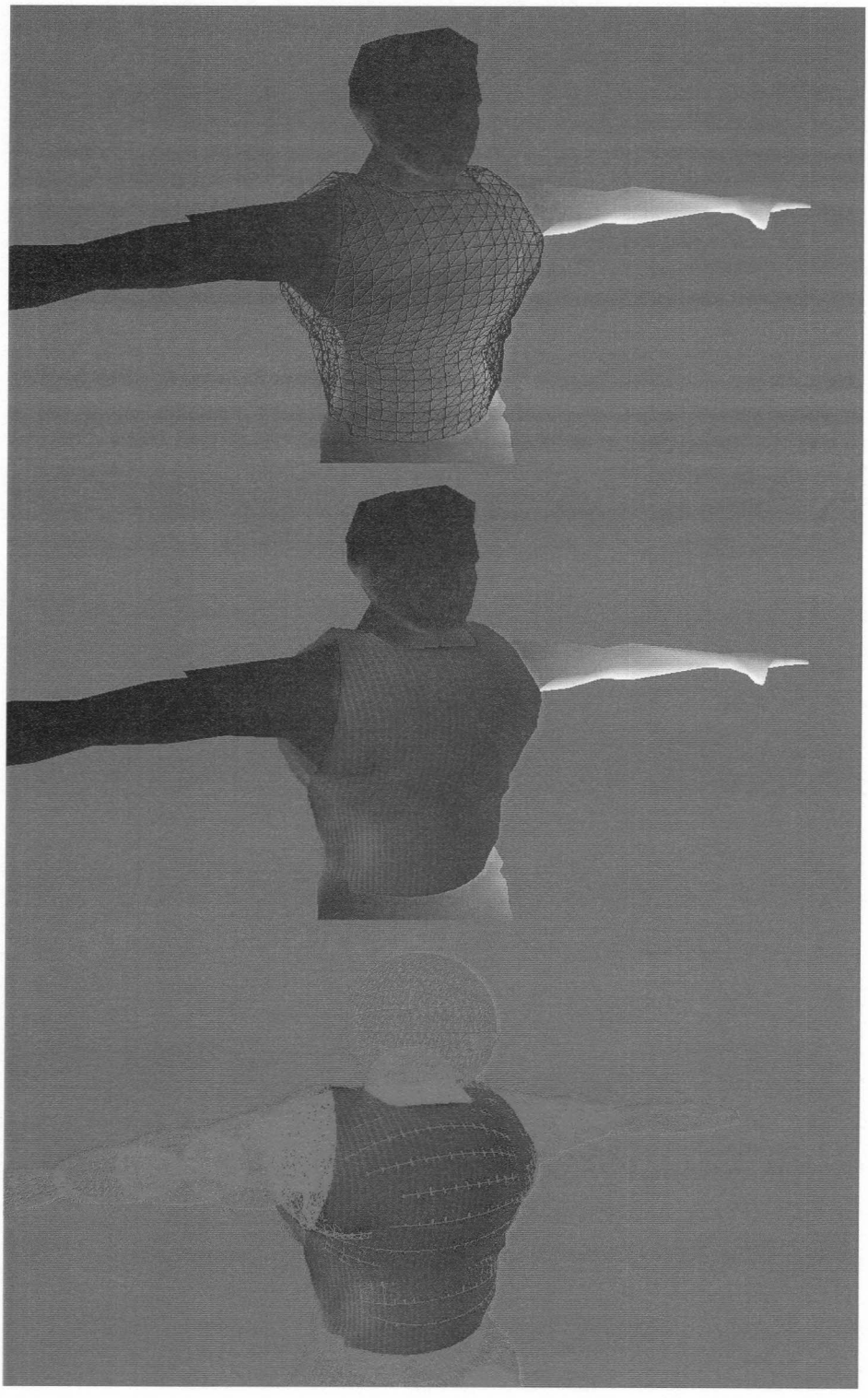


Figura 28: prenda con colisión de elipses. Desde arriba hacia abajo tenemos: a) la malla colisionando en triángulos, b) la malla colisionando renderizada, c) los volúmenes de colisión reales

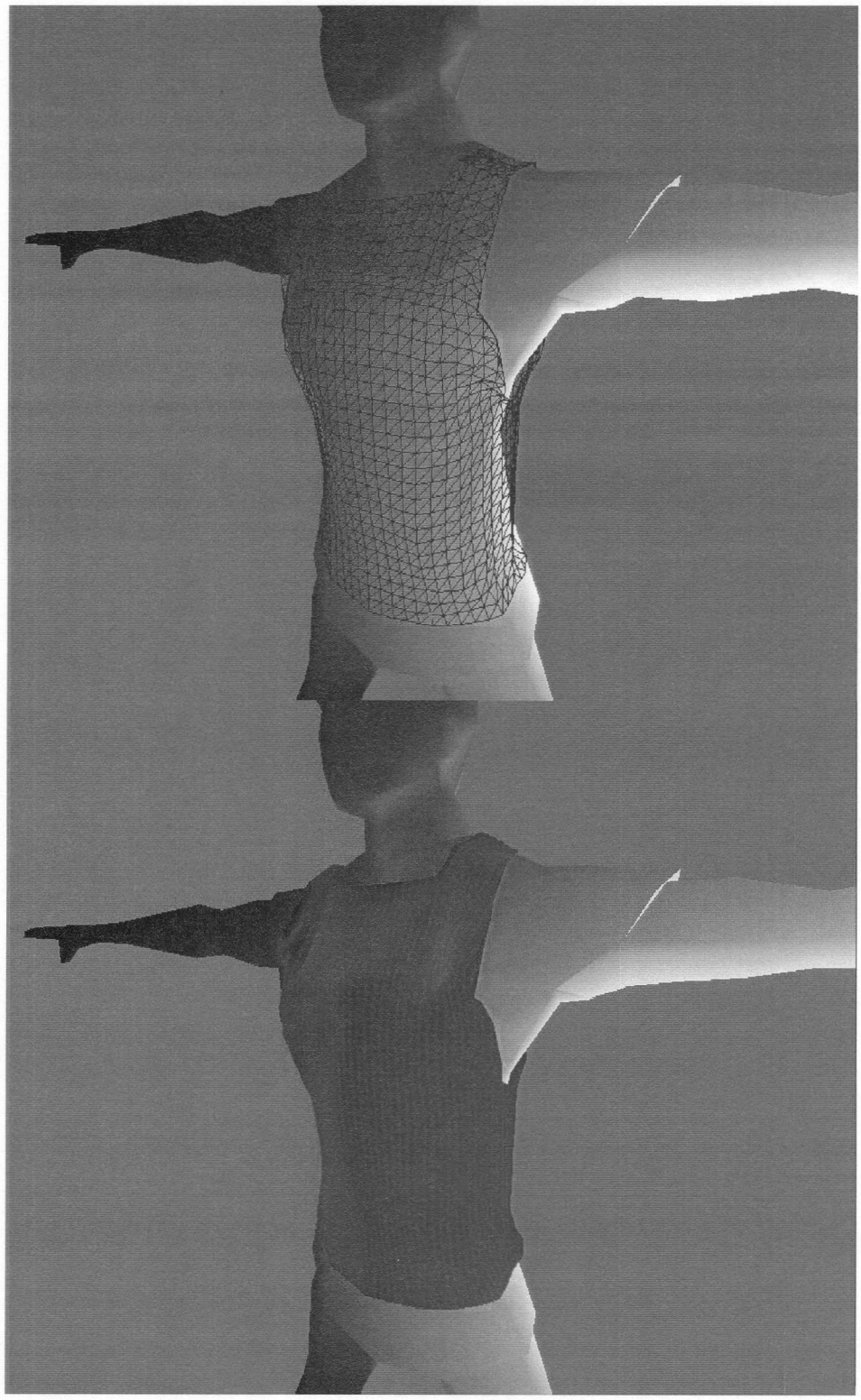


Figura 29: prenda con colisión de piel. Desde arriba hacia abajo tenemos: a) la malla colisionando en triángulos, b) la malla colisionando renderizada

Un problema que puede afectar nuestros resultados es que a veces, por súbitos bajas de performance o movimientos bruscos del personaje, la tela no colisione como debería. Esto puede resultar en la caída de la tela o una manga que escapa del brazo.

Para evitar este tipo de problemas establecimos ciertas partículas *ancla* cuya posición sigue a la piel sin importar qué algoritmos se estén usando para integrar o colisionar. Estas partículas las tomamos de lugares que generalmente siguen a la piel en una prenda real: los hombros de una remera, el cinturón de un pantalón. Es una técnica de la cual no se puede abusar, pues al aumentar la cantidad de partículas ancla la coexistencia de algoritmos de integración se torna notoria. Vemos en la figura 30 una anomalía que es resuelta por partículas ancla en la figura 31.

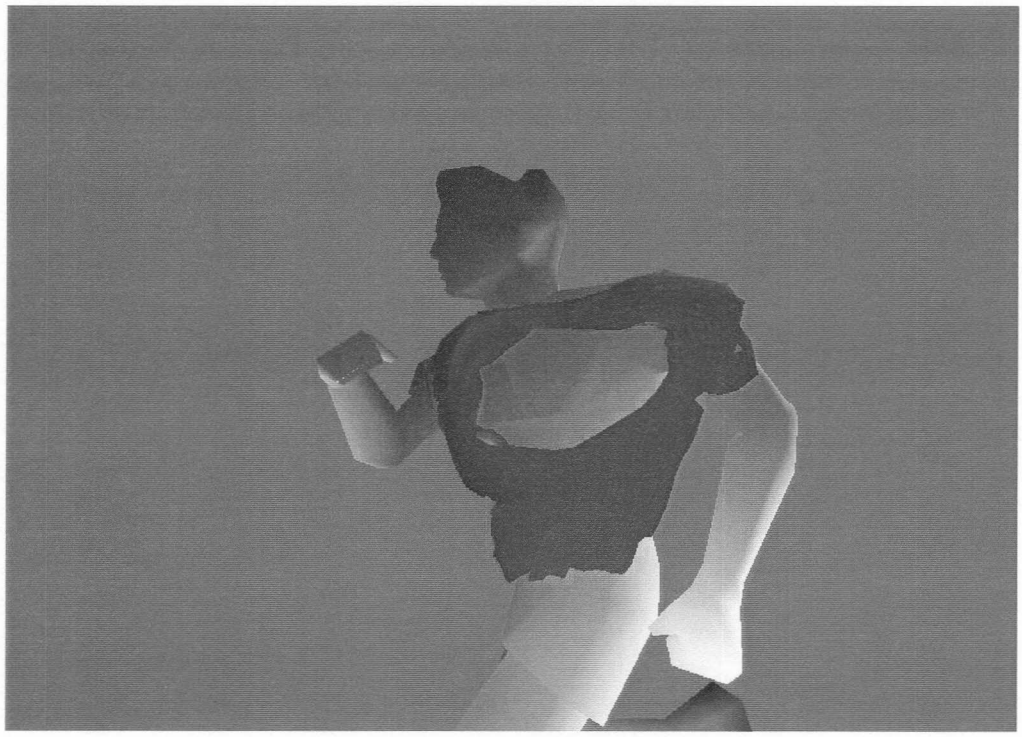


Figura 30: anomalía en la colisión

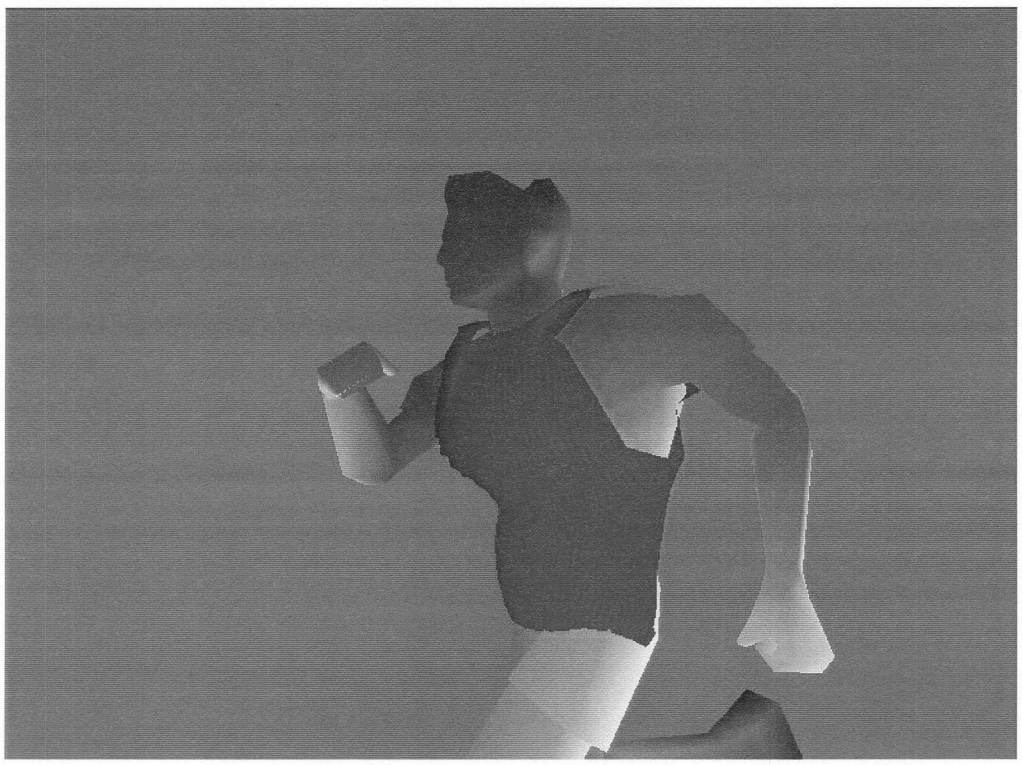


Figura 31: anomalía resuelta por partículas ancla en el hombro del personaje

5.7.3 Implementación de la auto colisión

La detección de auto colisión requiere que verifiquemos para cada triángulo de tela si colisiona con todos los demás triángulos. Intentamos de 3 maneras sucesivas satisfacer esta condición:

- Chequear todos con todos
- Utilizar una biblioteca de detección de colisión
- Realizar un algoritmo de división del espacio

El primero de los algoritmos tiene orden cuadrático y da resultados correctos para entradas muy pequeñas, de hasta 100 partículas. Luego dejaba de tener tiempos de respuesta aceptables.

Como biblioteca de detección de colisión utilizamos la biblioteca PQP[5]. Esta biblioteca permite verificar, dados dos conjuntos de triángulos, qué triángulos están en colisión. Entonces alimentamos el algoritmo con la tela dos veces (como los dos conjuntos de triángulos) y pedimos la detección de triángulos que se están interpenetrando. Obviamente, va a detectar muchas colisiones superfluas de triángulos con si mismos o con los vecinos, que hay que filtrar. Esta manera de

[5] Hecha por el UNC Research Group on Modeling, Physically-Based Simulation and Applications. Disponible en http://www.cs.unc.edu/~geom/SSV/

realizar la auto colisión tiene más performance, pero también deja de ser viable exhibiendo un comportamiento cuadrático, en este caso a las 250 partículas.

Al ver esto pensamos que evidentemente la biblioteca no está preparada para manejar este tipo de situaciones, por lo que decidimos crear nosotros un algoritmo de división de espacio que permita realizar la auto colisión.

El algoritmo es el siguiente:

1. dividimos el espacio, de manera adaptable a los tamaños de la tela, en un cubo de celdas.
2. Luego ponemos cada triangulo en la celda / las celdas en las que está contenido
3. Al finalizar, para cada celda en la que haya más de un triangulo, realizamos un chequeo de todos con todos.

En nuestra implementación, dio mejores resultados que utilizar la biblioteca, pero no una mejora significativa. Maneja situaciones donde haya hasta 500 partículas, pero si la tela se agrupa cerca de un punto, la performance cae drásticamente. Y este comportamiento no es nada compatible con el concepto de real time.

5.8 Mediciones

Las mediciones realizadas han sido pruebas empíricas con el objetivo de medir bajo distintas circunstancias el comportamiento de los algoritmos presentes en nuestro sistema integral de simulación de prendas y a partir de los resultados obtenidos poder establecer parámetros óptimos para la simulación.

En primer lugar, las pruebas más simples de simulación, las cuales tenían como objetivo proporcionarnos un conocimiento temprano sobre la factibilidad de utilizar los distintos algoritmos implementados. Una vez avanzados en el desarrollo, realizamos pruebas un poco más completas, las cuales incluyeron combinaciones de algoritmos de simulación y colisión con el personaje sobre prendas completas. Finalmente realizamos pruebas compuestas, que en vez de probar algoritmos sueltos prueban toda una combinación de algoritmos.

5.8.1 Plataforma

Tanto las mediciones de esta sección como las del capítulo de LOD fueron realizadas a partir de nuestra implementación sobre una máquina con las siguientes características:

- AMD Athlon XP 2600
- 1 gigabyte de memoria RAM
- Placa de video nVidia fx5700 con 128 megabytes de memoria
- Resolución de pantalla 1024 x 768

La implementación se compiló con el Visual C++ 7.1 y se usó la versión 0.12 del Irrlicht Engine. Así mismo, todas las capturas de pantalla corresponden a nuestra implementación sobre dicha máquina.

5.8.2 Mediciones simples

Las primeras pruebas, las mediciones simples, miden la relación existente entre la cantidad de tela, la densidad con que se modela, y el tiempo consumido en generar un cuadro, solo con un algoritmo por separado. La idea de las mediciones es que puedan darnos una idea sobre la relación sobre cómo se comportan los algoritmos aislados, sin interactuar entre sí, cuán eficientes son.

Para realizarlas tomamos un personaje animado y le colocamos una remera. Dicha remera iba variando en granularidad, para obtener simulaciones con cantidades crecientes de partículas. Luego tomamos estas remeras y las simulamos con el algoritmo en cuestión. Para los algoritmos de colisión con el cuerpo, tomamos la tela sin simular pero calculando la colisión contra el cuerpo del personaje en movimiento. En cada caso el objetivo es tomar el promedio de tiempo que tarda en producir un cuadro.

Evidentemente este tiempo incluye también el procesamiento del motor gráfico y la animación del motor, por eso también incluimos el caso en que no se aplica ningún proceso sobre la tela. Este caso es el de "tela sin algoritmos".

Al ver los resultados se puede apreciar claramente la linealidad de los algoritmos y la diferencia de performance que presenta cada uno de ellos. Confirmando las mediciones lo que esperábamos al plantear los algoritmos: el algoritmo de piel es claramente el más rápido y el de thin objects el más lento, siendo el algoritmo de restricciones el paso intermedio. En cuanto a la colisión también se comprueba que el algoritmo de elipses es casi el doble de costoso que el algoritmo de colisión por piel. Los gráficos de resultado son los siguientes:

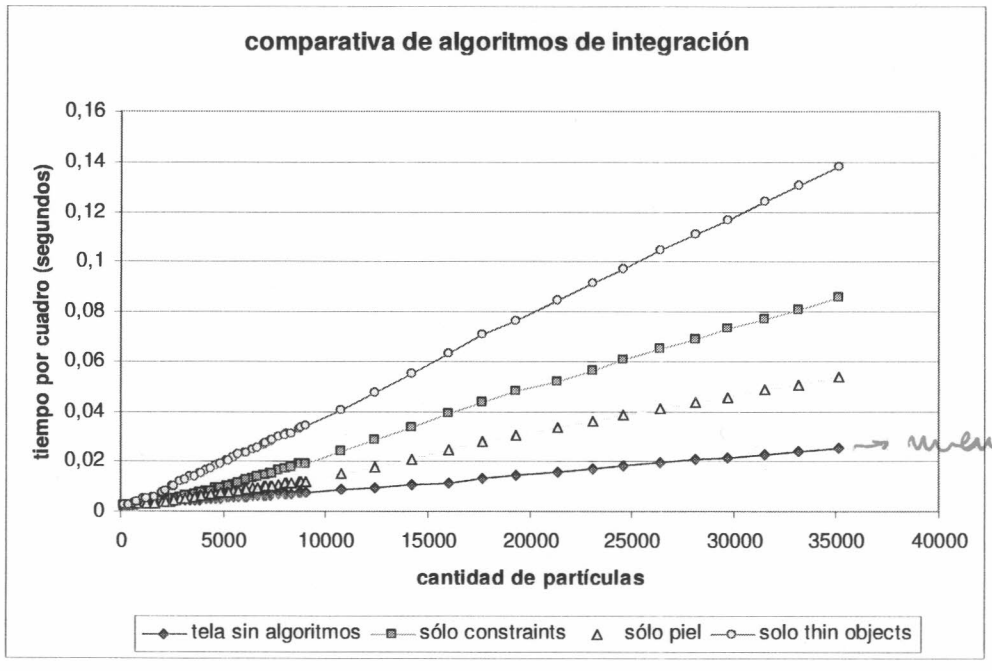


Figura 32: comparación de tiempos para algoritmos de integración

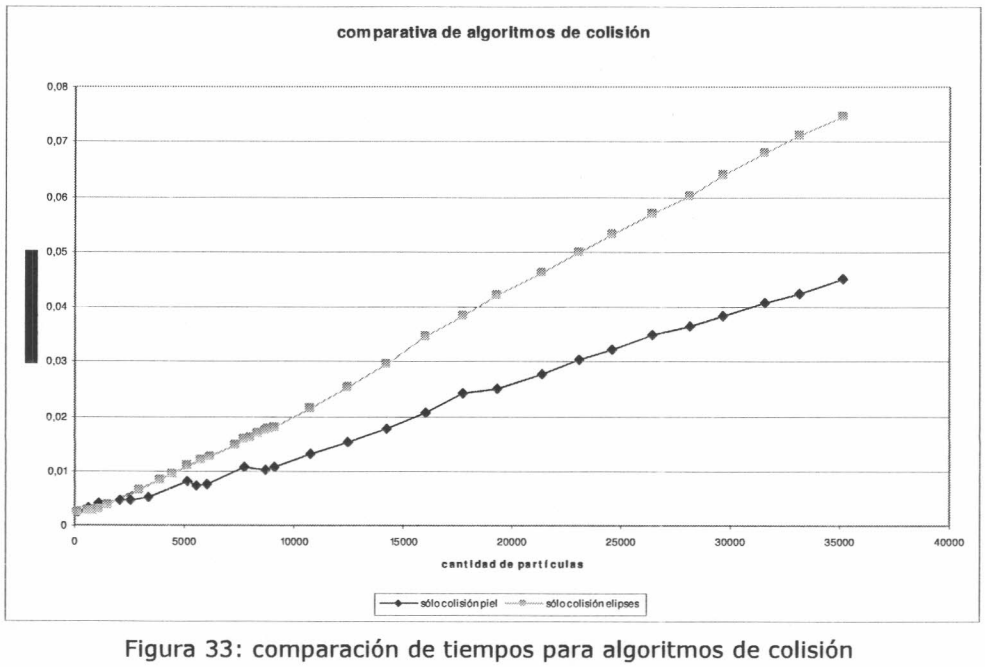


Figura 33: comparación de tiempos para algoritmos de colisión

5.8.3 Mediciones compuestas

Realizar sólo integración o sólo colisión generalmente no tiene sentido, por lo cual realizamos mediciones compuestas. Estas mediciones se basan en el mismo principio que las anteriores pero en vez de aplicar un solo algoritmo aplicamos varios: uno de integración, uno de colisión y opcionalmente auto-colisión (a excepción del algoritmo de piel, que por su naturaleza no necesita otros algoritmos). Como nuestra implementación de auto-colisión no fue del todo satisfactoria con respecto a la performance, analizamos los casos sin auto-colisión. En cuanto a thin objects, a pesar de que el algoritmo tiene comportamiento errático al interactuar con el algoritmo de colisión, incluimos las mediciones.

Más allá de su utilidad para analizar la performance de las implementaciones y ver los límites del sistema, estas mediciones son particularmente útiles para construir el sistema de LOD, como veremos en el próximo capítulo.

Las combinaciones que medimos son:

- Sólo el algoritmo de piel
- Algoritmo de restricciones con colisión de elipses
- Algoritmo de restricciones con colisión de piel
- Algoritmo de thin objects con colisión de elipses
- Algoritmo de thin objects con colisión de piel

Como era de esperar, la secuencia de algoritmos lineales es también lineal. Esto nos permite obtener los parámetros de estas rectas experimentales y estimar cuanto tardará en ejecutar una combinación de algoritmos para una cantidad dada de partículas.

Hemos incluido un gráfico comparativo y otro que representa los mismos datos pero haciendo foco en lo que pasa cerca de los 0,04 segundos. Esta línea es importante ya que marca el límite del tiempo real: 25 FPS significa cuadros de 0,04 segundos (1 / 25 segundos). Si una combinación supera este punto, no nos interesa más ya que no es posible tener una simulación en tiempo real de esta manera.

En dicho gráfico se observa la gran diferencia entre el algoritmo de piel y las demás combinaciones, constituyendo al algoritmo de piel como una muy interesante forma de ahorrar en performance. También podemos apreciar como se ahorra bastante con la colisión de piel en vez de la colisión de elipses.

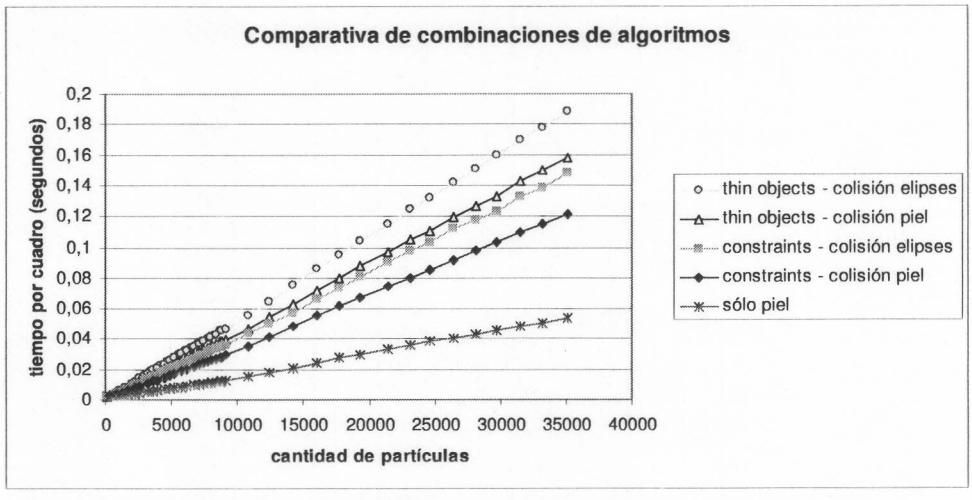


Figura 34: comparación de tiempos para combinaciones de algoritmos

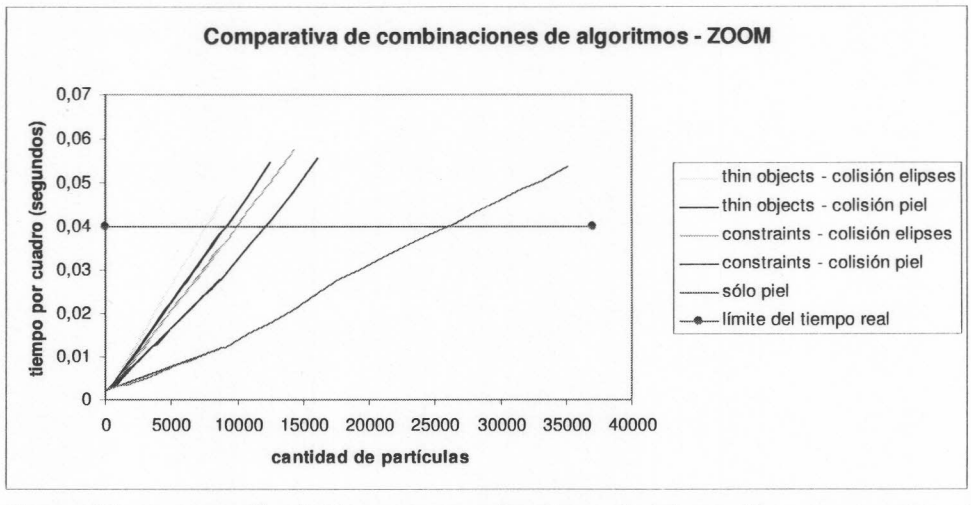


Figura 35: comparación de tiempos para algoritmos de integración – zoom sobre el límite del tiempo real

Combinación	ms / partícula
Sólo el algoritmo de piel	0,0015
Algoritmo de restricciones con colisión de piel	0,0034
Algoritmo de restricciones con colisión de elipses	0,0042
Algoritmo de thin objects con colisión de piel	0,0045
Algoritmo de thin objects con colisión de elipses	0,0054

Tabla 1: comparación de tiempos por partícula

6 Alcanzando el tiempo real: LOD

6.1 Introducción a la animación por computadora.

Los temas de animación pueden ser considerados una extensión lógica del tema de imágenes estáticas, tal que ahora consideraremos un cierto número de imágenes entre las que existe una relación temporal.

Una secuencia de imágenes está caracterizada por el número de imágenes que se muestra por unidad de tiempo (cantidad de cuadros por segundo - FPS), de forma que la impresión obtenida sea la de que existe movimiento. Este último valor debe estar alrededor de las 25 imágenes por segundo para dar la sensación de movimiento continuo.

El término animación viene del griego "anemos" = viento, aliento y del latín "animus" = dar vida. El concepto de animación se asocia habitualmente con el de movimiento. Podemos definir la animación por ordenador como la "generación, almacenamiento y presentación de imágenes que en sucesión rápida producen sensación de movimiento." El fenómeno de la persistencia de la visión (de 100 a 20 ms) permite que se fundan las imágenes y que se pueda generar la ilusión de movimiento. Los distintos fotogramas no deben ser muy diferentes entre sí, y la sensación de movimiento es más fluida a partir de una frecuencia de presentación de unos 24 Hz.

La animación en tiempo real va generando los fotogramas a medida que son necesarios. La animación cuadro por cuadro calcula los cuadros uno a uno y luego los muestra. A medida que los ordenadores se hacen más rápidos, los algoritmos de generación de imágenes se hacen más complejos, por lo que probablemente siempre existan las dos.

6.1.1 Concepto de tiempo real

Para poder considerar una simulación real time, además de la frecuencia de presentación o frecuencia de refresco, existen otras restricciones temporales independientes: la correspondencia entre el tiempo aparente de la simulación y el tiempo físico del observador (los objetos deben moverse en la simulación con la misma velocidad que lo harían en el mundo real, el tiempo interno utilizado por el modelo de la simulación debe tener la misma escala - o un factor constante y conocido - que el tiempo físico).

6.1.2 Frecuencia de refresco

Hemos comentado que una de las características de los sistemas gráficos en tiempo real es la restricción de la frecuencia de refresco, que debe intentar mantenerse en

unos límites adecuados. Así, podemos ver el proceso de generación de imágenes como un ejemplo de problema de control, donde queremos obtener una salida con una cierta característica deseada, en nuestro caso el número de imágenes por segundo. Las posibilidades de control aparecen desde el momento en que podemos variar, para la misma escena, la cantidad de datos a procesar, y por tanto ajustar el tiempo que se tardará en procesarlos. Usualmente esta modificación se efectúa haciendo variar el nivel de detalle de los objetos.

6.1.3 La cámara y el observador

Un paisaje no existe sin alguien que lo mire. Del mismo modo, para que una escena tenga sentido hay que mostrarla en pantalla, la tiene que percibir el observador. En computación gráfica así como tenemos la metáfora de la escena, tenemos la metáfora de la cámara. Esta cámara representa el punto desde el cual estamos mirando la escena y la forma en que la miramos.

El realismo y dedicación con que representamos a cada objeto de la escena muchas veces tienen que ver con la cercanía de este objeto a la cámara. El observador, que mira a través de la cámara, está cerca de este objeto y su atención va a ser captada por él. Cuanto más lejos el objeto, menos influye en la percepción total de la escena y más chico se verá.

Por lo tanto, La relación espacial entre la cámara y cada objeto da una pauta de cuánto deberíamos dedicarnos a mostrar ese objeto. Sin embargo, la falta de dedicación a un objeto distante no debería resultar en una pobre representación del mismo al acercarnos. Es decir, la forma con que muestro un objeto cuando está lejos no debe afectar cuando me acerco y tengo que mostrarlo de una manera mejor.

6.2 LOD: una forma de disminuir el costo de generación de cuadros

Cuando el costo total de la generación de un cuadro es excesivo, la única alternativa para alcanzar el tiempo real es simplificar las tareas asociadas para ganar performance. Para esto existe el concepto de *nivel de detalle*, o más comúnmente llamado LOD (level of detail). La idea es, básicamente, dar varios niveles de detalle a ciertas porciones de la simulación, de tal manera que se pueda utilizar detalles menores para ganar performance o detalles mayores para obtener mayor calidad. Esta asignación de detalles se puede hacer de una manera inteligente, resultando en un menor tiempo de procesamiento para un cuadro muy similar al que hubiera resultado sin aplicar el LOD.

En una aplicación de simulación gráfica existen dos procesos: la simulación (generar / modificar los datos del modelo) y la visualización (mostrar el modelo simulado). Generalmente se asocia LOD con LOD visual, es decir, los métodos para

mostrar los objetos simulados de una manera más eficiente. Para esto se pueden simplificar los modelos 3D, usar formas de renderizado más económicas, etc.

Algo menos común es el LOD de simulación: tener niveles de detalle para la simulación, es decir, para la forma en que se avanzan los estados de los datos del modelo. Esto significa que no vamos a interesarnos en cómo mostrar de una manera más económica las telas simuladas, sino que vamos a utilizar formas más económicas de mover las telas. Esto se puede hacer cambiando los algoritmos de integración y de colisión, así como modificando el tamaño de la entrada de estos algoritmos (cambiando el modelado de la tela en partículas, agregando o quitando partículas de la malla de tela).

Combinando ambas se puede lograr una significativa reducción de tiempos de generación de cuadros, aunque también es cierto que se reduce la calidad final de los mismos. Esta disminución de calidad puede ser notable o no, dependiendo de cómo se apliquen los distintos algoritmos de LOD. Aquí entonces entra en juego una clasificación *subjetiva* de los objetos en escena, indicando qué objetos son más importantes para el espectador, es decir, en qué objetos se centrará su atención. Luego, con esta clasificación, se puede distribuir el procesamiento de manera tal que el impacto en calidad no se vea en los objetos importantes.

Resumiendo, podemos decir que:

El objetivo del LOD es lograr hacer una distribución prioritaria de la capacidad computacional del sistema; poder explotar al máximo el procesamiento, o sea, lograr la "mejor imagen real time posible" dados los recursos disponibles.

6.3 Alcances y límites de nuestro LOD

Antes de saltar a la descripción de nuestra propuesta de LOD, explicaremos aquí los alcances y límites que consideramos para la elaboración de la misma:

Nuestra propuesta estará mayoritariamente enfocada en LOD de simulación, es decir, sobre los algoritmos de simulación, tanto de integración como de colisión. Aunque también consideremos LOD visual, no es allí donde dedicaremos nuestro esfuerzo.

Un sistema que aplique LOD debe poder simular con algoritmos más precisos y con mayor nivel de detalle aquellas prendas que en un instante dado se consideren más importantes para la percepción de la escena. Es claro que a mayor precisión y nivel de detalle, mayor es la capacidad computacional necesaria.

El LOD debe estar actualizado constantemente con la situación actual de la escena y la cámara, dado que cuando cambian las condiciones hay que redistribuir el poder de procesamiento.

El LOD debe establecer, según prioridades preestablecidas, y *prioridades ad hoc* evaluadas por él en run time, la asignación de detalle asociado a cada porción de prenda:

- La prioridad preestablecida la establece el usuario del sistema arbitrariamente (Ej.: le interesa ver el vuelo de una pollera, pero no le importa una simulación precisa de la remera del conjunto, o quiere ver el vuelo en movimiento del oxford de una botamanga, o hay cierto personaje en escena que es más importante por motivos arbitrarios).
- Las prioridades ad hoc las determina el LOD debido a las características de la escena, los objetos y la cámara (Ej.: la cercanía o lejanía a la cámara de cada objeto, el tamaño de los objetos, cuanto se mueve cada objeto, etc.)

La precisión de la simulación se asocia a varios factores:

- La calidad de los algoritmos: cuán verosímil resulta la simulación utilizando ciertos algoritmos es una medida de calidad para los algoritmos. Generalmente, cuanto más directamente el algoritmo aplique las ecuaciones físicas más verosímil es la simulación. Por el contrario, cuanto más se valga el algoritmo de simplificación o suposiciones, menos verosímil tiende a ser. Esto generalmente implica que los algoritmos más verosímiles son los más costosos, y se aplica tanto a algoritmos de integración como de colisión.
- Precisión de la discretización de la tela: cuando discretizamos cada porción de tela podemos asignar distintas cantidades de partículas, resultando en detalles más o menos finos en el movimiento.
- Aplicación de fuerzas físicas y/o del entorno: hay ciertas fuerzas que modifican las telas, como el viento o el rozamiento. Las ecuaciones que modelan estas fuerzas se pueden simplificar o dejar de aplicar en su totalidad.

6.4 Nuestra propuesta de LOD

La idea de nuestra propuesta de LOD es la siguiente: *repartir el poder computacional disponible en las distintas porciones de tela a simular, de tal manera que podamos dedicar mayor o menor energía a simular cada una de ellas.*

De esta descripción se infiere que existen 2 procesos: la distribución de poder computacional y el procesamiento de las porciones de telas. Además, estos procesos no son simultáneos. Primero se reparte el poder de cómputo y luego se simulan las porciones de tela. La distribución, casi por definición de la palabra distribución, implica un proceso global, donde son tenidas en cuenta las características y estados de todas las telas a la vez. De otro modo, la simulación de telas una vez distribuidas la capacidad computacional, se da de a una tela por vez sin interferencia entre telas.

Tanto aquí como a través de la descripción del concepto de LOD y de los alcances y límites se repite una y otra vez las palabras *poder computacional*. También hablamos sucesivas veces de *repartir* este poder, de tal manera que uno puede hacer tal o cual cosa dependiendo del poder disponible. Es imprescindible entonces esbozar una interpretación de este concepto para luego ver cómo podemos usar este poder computacional.

6.4.1 Unidades de Cómputo (UC)

El concepto de poder computacional se puede explicar intuitivamente como la cantidad de procesamiento que disponemos para procesar algo, una porción de tela en este caso particular. En un sentido menos abstracto puede ser visto como la cantidad de tiempo de procesador que dedicamos a cada porción.

El problema de este acercamiento es que no todos los procesadores pueden procesar lo mismo en el mismo intervalo de tiempo. A igual tamaño de entrada un procesador más rápido logra procesar en menor tiempo. Del mismo modo, en el mismo intervalo de tiempo un procesador más rápido procesa una entrada más grande.

Por esta razón necesitamos una manera de cuantificar el poder de cómputo que sea independiente del procesador con que cuente la máquina de turno. Aquí nace el concepto de *unidades de cómputo*. Esta es una unidad de esfuerzo de procesador con respecto al tiempo, lo que podemos procesar en un cierto intervalo de tiempo. Para hacer esta unidad absoluta necesitamos una referencia, por lo que establecemos una computadora referencia y hacemos que lo que podemos procesar sea en realidad lo que podemos procesar en esta computadora referencia. Decimos entonces que una unidad de cómputo es la cantidad de cómputo que se obtiene al procesar durante un intervalo X en la computadora referencia.

6.4.2 Determinando las UC disponibles

Basados en la premisa de que más unidades de cómputo (sin importar cómo están repartidas) implican más tiempo de procesador (sin importar qué procesador) podemos establecer una forma de determinar el total de unidades de cómputo que disponemos en un instante. Cuanto más tiempo dediquemos a la simulación de telas mayor será el tiempo total del cuadro y por lo tanto una menor cantidad de cuadros por segundo (FPS). Por lo tanto podemos establecer una relación inversa entre FPS y UC. Establecemos la cantidad de UC disponibles como la cantidad que si es usada por completo llevan al sistema a 30 FPS y no menos. Vamos ajustando iterativamente las UC de tal forma que nunca pasemos el umbral de 30 fps si estamos debajo de los 30 FPS. Si en algún momento pasamos este límite, inmediatamente se quitan UC para llevarlo nuevamente a 30 FPS.

no se entiende nada

Este proceso permite determinar la cantidad de unidades de cómputo de una manera dinámica, tal que si cambian las condiciones las unidades de cómputo se

o sea fue hicieron un sistema que se adapte a la capacidad de la máquina de turno

adecuarán. Esto es muy útil en aplicaciones como los videojuegos donde entran y salen personajes de la escena, pasando de ambientes tranquilos a ambientes llenos de eventos y por lo tanto más intensivos en procesamiento.

6.4.3 Repartiendo el poder computacional

Una vez determinado el total de las unidades de cómputo disponibles en el sistema, llega el momento de repartirlas entre todas las porciones de tela del sistema. Aquí es donde juega el concepto de prioridad entre las telas. Como establecimos en los alcances y límites estas prioridades tienen que poder ser establecidas por el sistema sin intervención del usuario, pero además debe permitir al usuario modificar estas prioridades.

Antes que nada, debemos decir que cada porción de tela tiene un máximo y un mínimo de UC que puede utilizar. Esto depende de las características de la tela y significa que la forma *más barata* de simular esta tela consume el mínimo de UC mientras que la *más cara* consume el máximo de UC. Partiendo de esta definición, otorgar más unidades que el máximo sería desaprovecharlas en otro lado y otorgar menos del mínimo significaría estar *inventando* UC, ya que en realidad va a consumir el mínimo de todos modos.

Como primera medida entonces el sistema se asegura que todas las telas reciban el mínimo de UC que necesitan. Luego ve cómo repartir las unidades que sobran entre todas. Se ve entonces la primera forma de modificar la distribución de unidades de cómputo: alterar los valores de máximo y mínimo para que una tela importante tenga un mínimo más alto que el resto o una tela poco importante reciba un máximo menor. Luego establecimos una forma de determinar qué telas reciben su máximo completo de unidades de cómputo sin alterar los máximos y mínimos: con un flag que establezca que la tela es de prioridad. Otra forma de alterar la distribución es directamente saltar este paso y decir "para esta tela necesitamos esta forma de simularla sí o sí", cosa que determina un número de UC. Naturalmente pueden existir infinidad de formas de asignar prioridades y cuánto signifique dar prioridades, lo importante es el orden en que se realiza el reparto:

1. se reparten los mínimos
2. se reparten según prioridades establecidas por el usuario
3. se reparten las restantes según las prioridades establecidas por el sistema

Supongamos que nos queda una cierta cantidad de UC luego de las asignaciones de los puntos 1 y 2. Se deben distribuir las UC sobrantes respetando algún criterio de prioridad. El criterio de prioridad puede establecerse sobre cualquier variable que defina un orden total sobre los elementos que componen el LOD. En nuestro caso, asociamos el criterio de prioridad a la distancia a la cámara (a menor distancia a la cámara mayor es la prioridad asignada).

Tomamos entonces todas las telas que no hayan alcanzado su máximo y hacemos una sumatoria de prioridades. El total de sumatoria se corresponderá con

el total de UC, entonces asignamos a cada tela la fracción de UC totales que corresponda a la fracción que es su prioridad sobre el total.

$$Total_{prio} = \Sigma\ prioridad_i$$

$$UC_i = Total_{UC} * prioridad_i / Total_{prio}$$

Es sencillo ver que la sumatoria de las UC_i da el total de UC, o sea que se reparten todas las UC. También se puede notar que dependiendo de la dispersión de los valores de prioridad vamos a tener mayor o menor dispersión en los valores de UC_i. Usar 1 / distancia a la cámara probó ser efectivo para tener poca dispersión cuando las telas están cercanas entre sí y alta dispersión cuando hay una cerca de la cámara y otra lejos.

A continuación se muestran varias figuras que ilustran este concepto. Vemos como en distintos escenarios, con distintas configuraciones de las posiciones de los personajes, se reparten las unidades de cómputo.

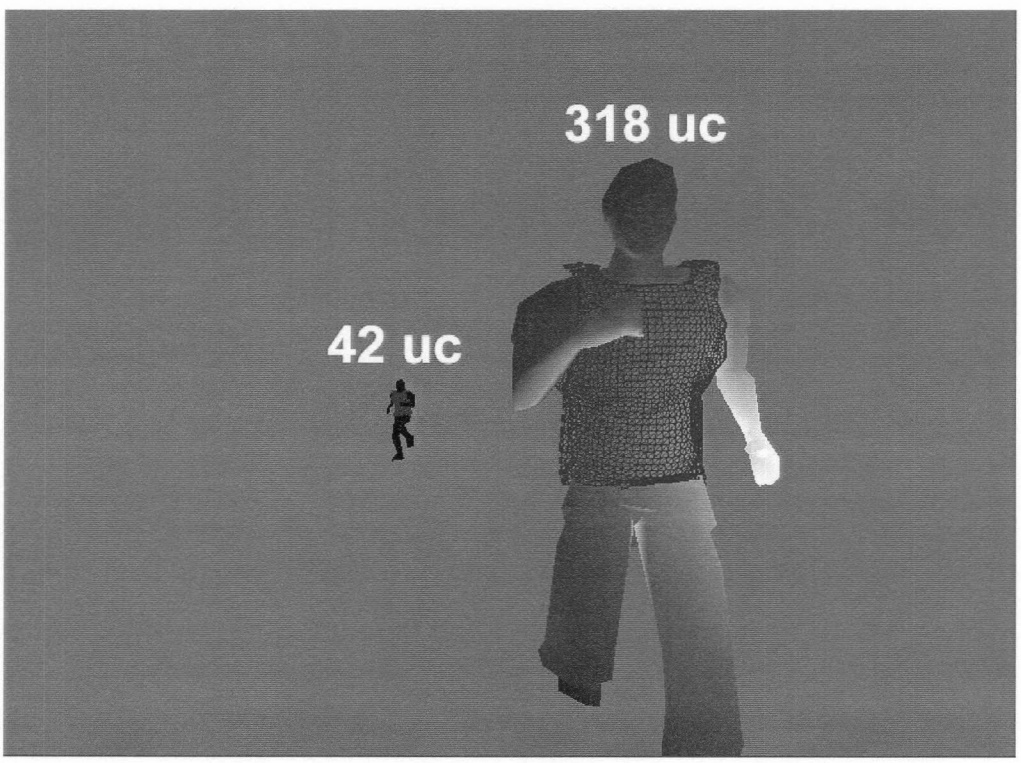


Figura 36: un personaje está cerca de cámara (318 UC) y otro está lejos (42 UC)

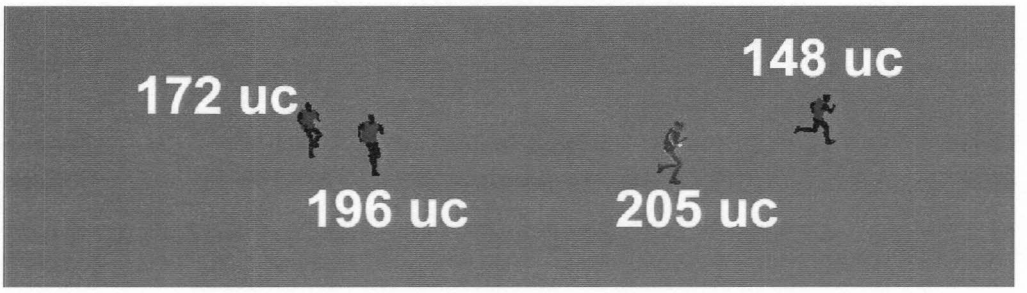


Figura 37: varios personajes lejos de cámara, recibiendo de izquierda a derecha: 172 UC, 196 UC, 205 UC y 148 UC

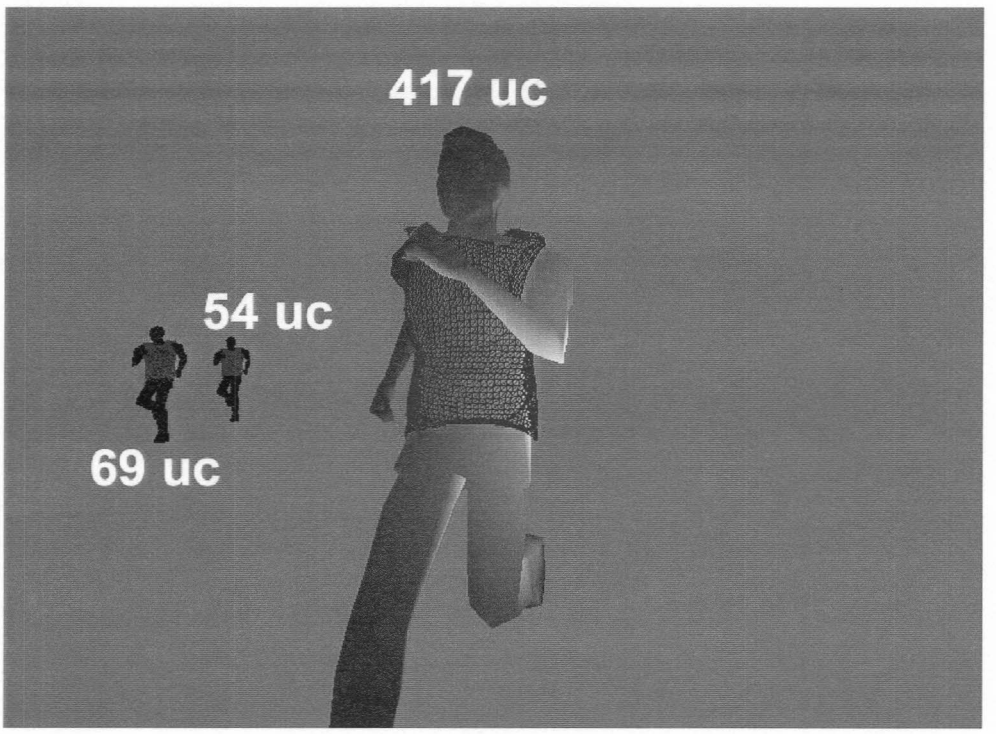


Figura 38: un personaje cercano a la cámara y dos lejos, recibiendo de izquierda a derecha: 69 UC, 54 UC y 417 UC

Repartem las unidades de cómputo disponibles, esto ¿en qué momento se decide? Así mismo se reparten entre puntos de la malla y los algoritmos de la simulación

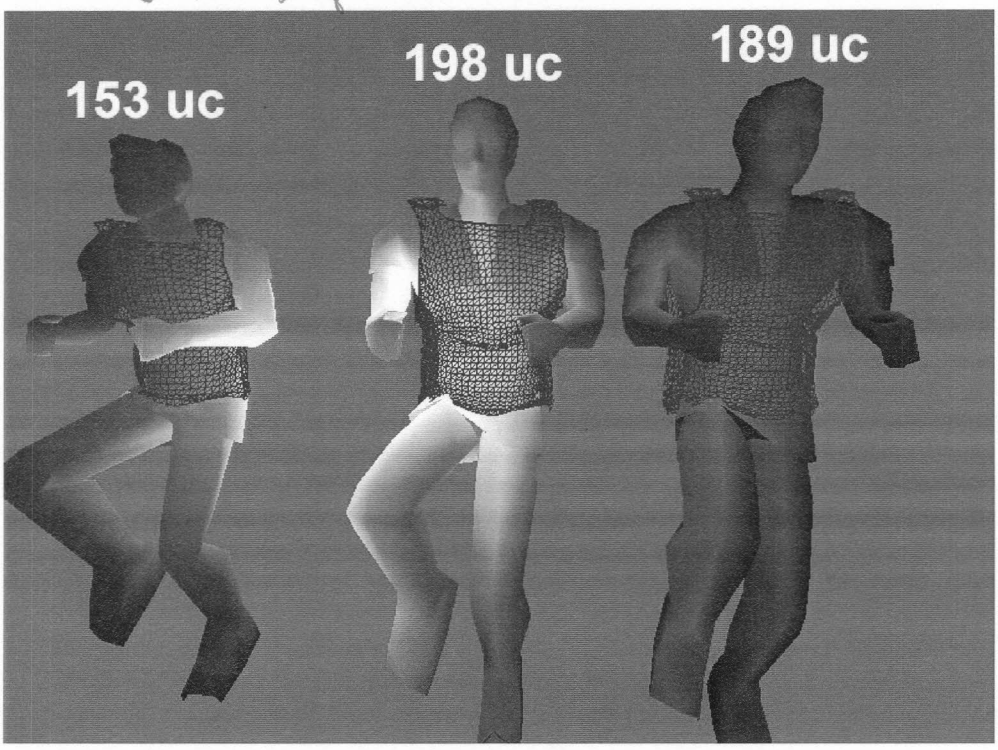


Figura 39: tres personajes cercanos a la cámara, recibiendo de izquierda a derecha: 153 UC, 198UC y 189 UC

6.4.4 Usando el poder computacional

Una vez repartidas las unidades de cómputo, es tiempo de procesar cada porción de tela. A esta altura cada porción tiene una cantidad de UC que determinan qué tipo de simulación es factible sobre ellas. Estamos ante el problema de decidir cual es el mejor forma de simular una porción dada de acuerdo a la cantidad de UC que disponemos.

Cuando hablamos de cómo simular una porción de tela, tenemos 2 variables que resolver:

- cantidad de partículas en la malla de tela
- combinación de algoritmos usar para avanzar la simulación.

En cuanto a la cantidad de partículas en la malla de tela probamos hacerla variar según la cantidad de UC disponibles. El resultado fue poco satisfactorio, dado que al posicionarse la tela cerca de la cámara, los notorios cambios de mallas se hacían con una frecuencia demasiado alta. Esta fue la razón por la que decidimos que la cantidad de partículas debía depender exclusivamente de la distancia a la cámara.

La cantidad de puntos de la malla depende de la distancia del objeto al observador.

¿Esto se decide cuando e cuando?

Una vez asignada la malla, se decide según la cantidad de UC y la cantidad de partículas en la malla, qué algoritmos usar, tanto para integración como para colisión y auto-colisión. El sistema de LOD debe elegir la más adecuada de las *combinaciones de algoritmos disponibles*. Un ejemplo de combinación sería *<algoritmo de restricciones, algoritmo de colisión por elipsoides>*.

Entonces partimos de la siguiente premisa: las combinaciones se pueden ordenar de mejor a peor con un orden total. Mejor o peor es un criterio totalmente subjetivo, asociado al realismo percibido. Se da también que si una combinación es mejor que otra, insume más unidades de cómputo (de otra manera, para qué considerar una combinación menos realista que implique más costo). Por lo tanto el orden de realismo termina siendo igual al orden de costo.

Cada una de estas combinaciones tiene una curva de rendimiento en la que vemos según el tamaño de la entrada (cantidad de partículas) cuánto tiempo toma ejecutar esa combinación en la computadora patrón (UC). Con esta curva, podemos decidir si para una cantidad de partículas y una cantidad de UC para una tela en particular se puede utilizar el algoritmo: viendo si en la curva para esta cantidad de masas insume más UC que las que tenemos disponibles (entonces las que tenemos no alcanzan) o si insume menos. En la figura 40 vemos como se relacionan las unidades de cómputo con la cantidad de partículas a través de la curva de rendimiento del algoritmo, y cómo se determina la cantidad de partículas viable para las unidades de cómputo disponibles:

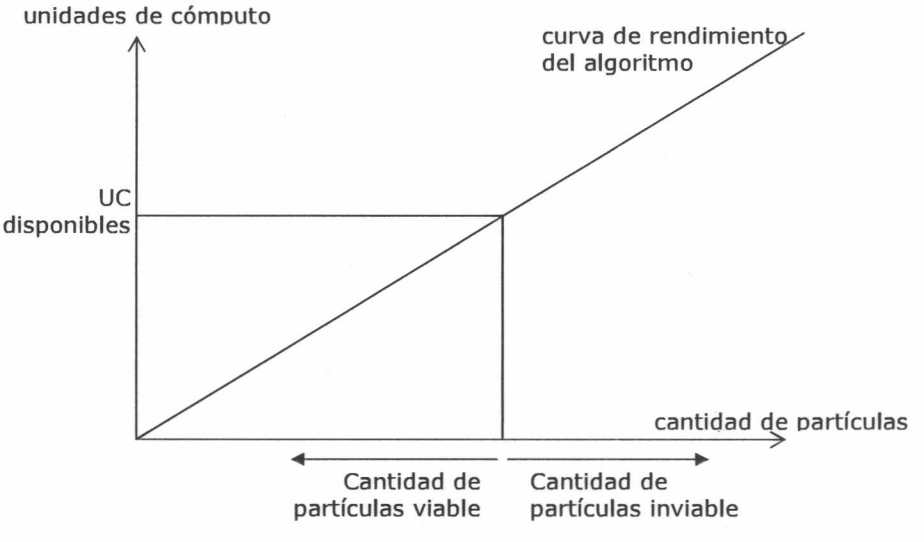


Figura 40: determinando la cantidad de partículas viable

Por lo tanto podemos construir una grilla que en la posición uc_i, cm_j signifique:

La mejor combinación de algoritmos que se puede realizar para cm_j cantidad de partículas contando con uc_i unidades de cómputo.

partic.\ uc	10 uc.	20 uc.	30 uc.	40 uc.	...
50 partic.	Comb. 1	Comb. 2	Comb. 2	Comb. 3	...
100 partic.	Comb. 1	Comb. 1	Comb. 2	Comb. 3	...
150 partic.	Comb. 1	Comb. 1	Comb. 2	Comb. 3	...
...	...	...	...	...	...

Tabla 2: ejemplo de grilla de combinaciones de algoritmos según uc y cantidad de partículas

Una vez que tenemos la cantidad de UC y la malla seleccionada (es decir la cantidad de partículas de la tela) decidir qué combinación usar es O(1), ya que se trata de buscar en esta grilla y ésta se puede precomputar fácilmente.

6.4.5 Las combinaciones de algoritmos

Para la elección de las combinaciones de algoritmos de nuestra implementación nos basamos en los algoritmos que implementamos en la sección 5.7. De estos algoritmos, algunos tuvimos que dejar de lado como es el caso de thin objects y la auto-colisión. Thin objects, porque no tolera las modificaciones por otros algoritmos y auto-colisión, porque ninguna de nuestras distintas implementaciones logró estabilidad más allá de las 200 partículas de tela.

Sin embargo, esta técnica es independiente de las combinaciones de algoritmos que se usen, sólo exige una curva de performance para cada combinación y un ordenamiento total de las combinaciones basado en la calidad de simulación. Entonces, de encontrarse una solución completa para auto-colisión, será cuestión de agregar combinaciones de algoritmos que la incluyan.

Las combinaciones usadas son:

1. Integración Piel
2. Integración Restricciones – colisión de piel
3. Integración Restricciones – colisión de elipses

Los resultados de la sección siguiente se tomaron de nuestro sistema de LOD funcionando con estas combinaciones. Aquí presentamos una versión reducida de la tabla con la que opera el sistema:

partic. \ UC	0	25	50	75	100	125	150	175	200	225	250
250	1	3	3	3	3	3	3	3	3	3	3
500	1	2	3	3	3	3	3	3	3	3	3
750	1	2	2	3	3	3	3	3	3	3	3
1000	1	1	2	3	3	3	3	3	3	3	3
1250	1	1	2	2	3	3	3	3	3	3	3
1500	1	1	2	2	3	3	3	3	3	3	3
1750	1	1	1	2	2	3	3	3	3	3	3
2000	1	1	1	2	2	3	3	3	3	3	3
2250	1	1	1	2	2	2	3	3	3	3	3
2500	1	1	1	1	2	2	3	3	3	3	3
2750	1	1	1	1	2	2	2	3	3	3	3
3000	1	1	1	1	2	2	2	3	3	3	3
3250	1	1	1	1	1	2	2	2	3	3	3
3500	1	1	1	1	1	2	2	2	3	3	3
3750	1	1	1	1	1	2	2	2	2	3	3
4000	1	1	1	1	1	1	2	2	2	3	3
4250	1	1	1	1	1	1	2	2	2	2	3
4500	1	1	1	1	1	1	2	2	2	2	2
4750	1	1	1	1	1	1	1	2	2	2	2

Tabla 3: tabla que usa nuestra implementación del sistema de LOD

6.5 Resultados

En esta sección analizamos el comportamiento del sistema funcionando con LOD comparado con el sistema original. Los resultados se contrastan tanto visualmente como con respecto a la performance.

Existen varias formas de comparar el sistema con LOD y sin LOD, ya que el LOD va variando los detalles (variando los algoritmos y mallas de tela) mientras que sin LOD el detalle es fijo (procesando siempre con el mismo algoritmo y la misma malla de tela).

Una comparación posible es pensar que en ambos casos queremos tener la misma calidad máxima para cada tela. Entonces, el sistema sin LOD tendría todas las mallas en la calidad máxima todo el tiempo y el sistema con LOD llegaría en algunas mallas a la calidad máxima en algunos momentos. La figura 41 muestra ambos casos, logrando 34 fps con LOD y 13 fps sin LOD.

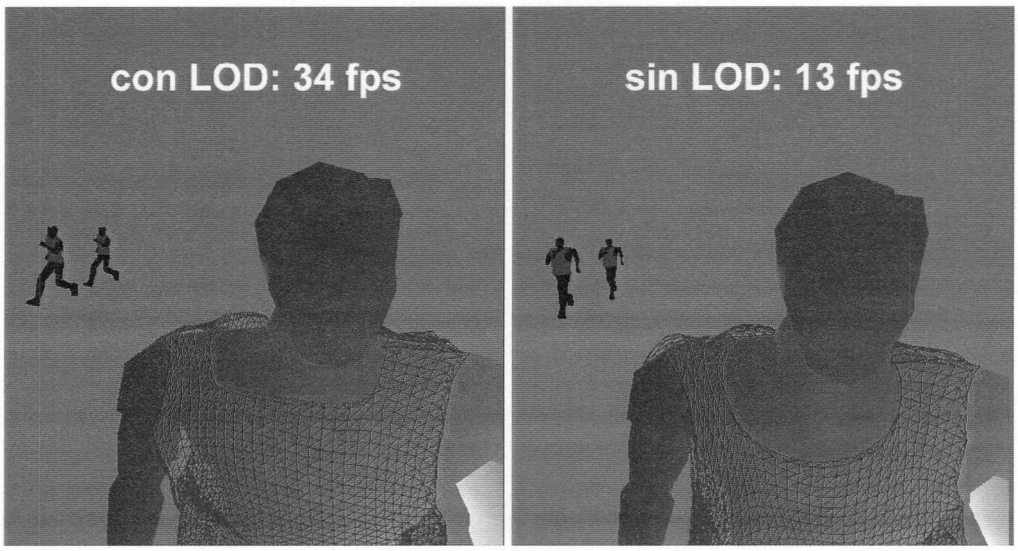


Figura 41: sistema con LOD a la izquierda (34 fps) y sin LOD a la derecha (13 fps), a la misma máxima calidad

Otra forma de comparar es ver, para una cierta cantidad de piezas de tela, cuál es la máximo calidad en tiempo real con y sin LOD. Como resultado obtuvimos un beneficio de cantidad de partículas de alrededor de 300%:

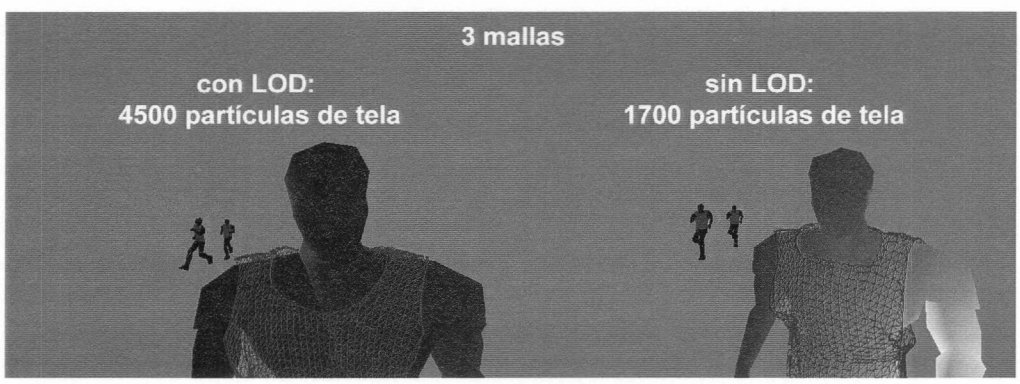


Figura 42: con 3 mallas el sistema con LOD llega a 4500 partículas en calidad máxima mientras que el sistema sin LOD llega a 1700

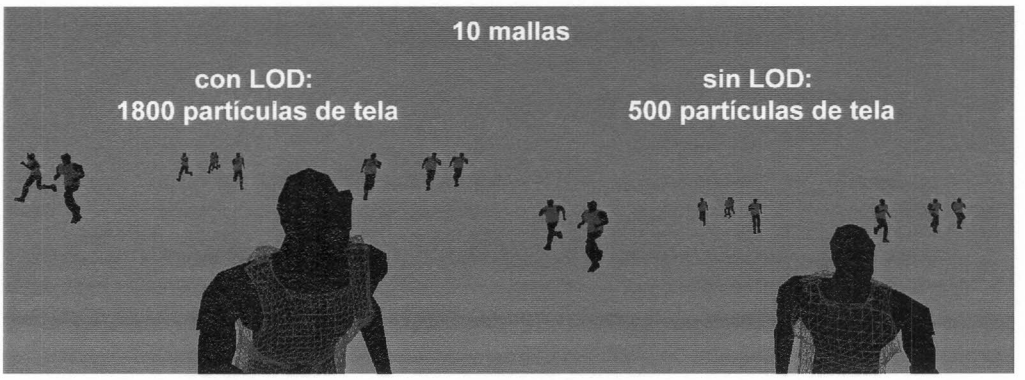


Figura 43: con 10 mallas el sistema con LOD llega a 1800 partículas en calidad máxima mientras que el sistema sin LOD llega a 500

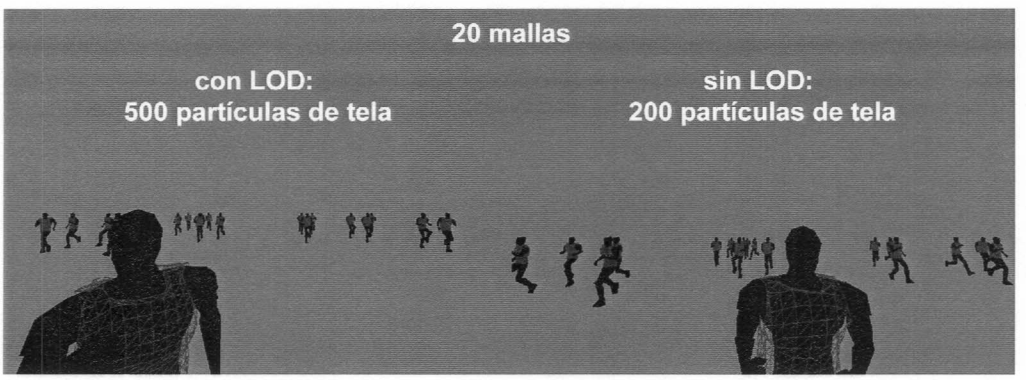


Figura 44: con 20 mallas el sistema con LOD llega a 500 partículas en calidad máxima mientras que el sistema sin LOD llega a 200

Cantidad de mallas	partículas con LOD	partículas sin LOD	relación
3	4500	1700	264%
10	1800	500	360%
20	500	200	250%

Tabla 3: resultados de comparación de calidades a igual cantidad de mallas

Otra posible evaluación es ver dónde cada una de las dos maneras tiene su límite en distintas circunstancias. El límite es la barrera de los 30 FPS y la idea es ver cuántos personajes vestidos logramos introducir en la escena. Realizamos estas mediciones con dos tipos de malla: una de máxima calidad (4000 partículas) y otra de buena calidad (1000 partículas). Los resultados quedan expuestos en la tabla 4.

Forma de simular	4000 partic.	1000 partic.
Sin LOD con Restricciones - elipses	1 prendas	6 prendas
Sin LOD con Restricciones - piel	2 prendas	8 prendas
Sin LOD con piel	5 prendas	20 prendas
Con LOD – calidad máxima: Restricciones - elipses	5 prendas	15 prendas

Tabla 4: Comparativa de límite según forma de simular y calidad máxima de malla

Nótese cómo el sistema con LOD tiene rendimiento similar al de un sistema basado exclusivamente en el algoritmo de piel, pero teniendo prendas en pantalla simuladas con la mejor calidad.

Con respecto a un sistema sin LOD basado en la mejor calidad de tela, tenemos una proporción de 5 a 1 con máxima calidad y de 2.5 a 1 con calidad de tela normal.

7 Conclusiones y trabajo futuro

En este trabajo hemos presentado un estudio amplio sobre un sistema de simulación de prendas en tiempo real. Nuestro trabajo consistió en realizar un análisis de todas las cuestiones involucradas en una simulación de este estilo y la propuesta de integración de todas en un sistema. Resaltamos este hecho ya que en casi toda la literatura sobre este tema los trabajos se concentran en partes muy específicas, dejando baches conceptuales sobre como integrar sus propuestas con las demás existentes.

Una vez realizado el análisis y la revisión del estado del arte proponemos una solución que integre todos estos conceptos. Exhibimos los problemas que hemos enfrentado y nuestra solución a cada uno. En este sentido, nuestro trabajo también sirve como comparador de los distintos trabajos puntuales anteriores: al integrar todos dentro de una misma solución podemos comparar tiempos y ver cuán viables son en una aplicación real.

Dentro de los problemas que hemos enfrentado, está el de la implementación de la auto colisión. Si bien tuvimos solución para algunos casos faltó encontrar una solución que terminara con el problema completamente, sobre todo para simulaciones con prendas de gran detalle. Sin dudas la auto colisión es el talón de Aquiles de la simulación de prendas y su resolución completa queda para un estudio específico del tema de la auto colisión.

Como punto final de nuestro trabajo, presentamos un sistema de niveles de detalle para aumentar la capacidad de simulación de nuestro sistema. Mediante nuestro sistema de LOD obtuvimos simulaciones con mayor cantidad de personajes en pantalla y con más prendas que con el sistema original. Según nuestras mediciones, obtuvimos un incremento de performance de entre 250% y 500% según el caso. Destacamos también el hecho de que su modularidad permite integrarlo o no con el sistema a gusto y en run time, sin modificar el resto del funcionamiento del resto de los componentes.

Uno de los temas que quedó fuera del alcance de nuestro trabajo es el análisis e implementación de distintos tipos de tela. Si bien con los algoritmos de simulación que proponemos se pueden lograr muchos tipos distintos de tela, no realizamos un estudio profundo sobre este tema. Del mismo modo, quedó para futuras investigaciones la forma de visualizar las telas, analizando los modelos de iluminación para cada tipo de tela. Nuevamente, esto requeriría un estudio sobre este tema específicamente.

En cuanto a la performance, otro camino para continuar la investigación es la implementación de los algoritmos de este sistema en GPU, así como las adaptaciones necesarias al motor para realizar el procesamiento en la placa gráfica.

Finalmente, en un área que bordea el diseño de indumentaria, una forma de continuar con este trabajo es la investigación en interfaces gráficas para pasar del diseño de moldes a la ropa en el personaje y, más interesante, realizar el camino contrario: como modelar una prenda en 3D y obtener un molde que la construya.

8 Bibliografía

- [BW92]: David Baraff and Andrew Witkin. Dynamic simulation of nonpenetrating
- flexible bodies. Computer Graphics, 26(2):303–308, 1992.

- [BHM]: Breen D, House D,WoznyM(1994) Predicting the drape of woven cloth using interacting particles. (SIGGRAPH '94) Comput Graph CEd :365–372

- [VCT95] Pascal Volino, Martin Courchesne, and Nadia Magnenat Thalmann.
- Versatile and efficient techniques for simulating cloth and other deformable objects. Computer Graphics, 29(Annual Conference Series):137–144, 1995.

- [VM95]: Pascal Volino and Nadia Magnenat Thalmann. Collision and auto-collision detection: Efficient and robust solutions for highly deformable surfaces. In Dimitri Terzopoulos and Daniel Thalmann, editors, Computer Animation and Simulation '95, pages 55–65. Springer-Verlag, 1995.

- [DSB] Interactive animation of structured deformable objects

- [DMB] Mathieu Desbrun, Mark Meyer, and Alan H. Barr. Interactive animation of cloth-like objects for virtual reality, 2000

- [Mey01] Mark Meyer, Gilles Debunne, Mathieu Desbrun,and Alan H. Barr. Interactive animation of cloth-like objects for virtual reality. The Journal of Visualization and Computer Animation, 12:1–12, May 2001

- [MHG] Real - time Animation Technique for Flexible and Thin Objects

- [BFA] Robert Bridson, Ronald Fedkiw, and John Anderson. Robust treatment of collisions, contact and friction for cloth animation

- [CMT02] Frederic Cordier and NadiaMagnenat-Thalmann. Real-time Animation of Dressed Virtual Humans. EUROGRAPHICS 2003, 21(3), 2002.

- [FGL03] Arnulph Fuhrmann, Clemens Gross, and Volker Luckas. Interactive Animation of Cloth including Self Collision Detection.WSCG, 11(1), February 2003.

- [GM] Gibson and Mirtich. A survey of Deformable Modeling in Computer Graphics. TR-97-19, November 1997.

- [MH]Michael Huth. Numérical Techniques for cloth Simulation. , WSI/GRIS, University of Tübingen. Octover 29, 2003

referencia incompleta?