Universidad Rey Juan Carlos
Máster Oficial en Informática Gráfica,
Juegos y Realidad Virtual

Proyecto Fin de Máster

Simulación de Fractura de Objetos Elásticos

Autor: Daniel Goldáraz Lanas

Tutor: Miguel Ángel Otaduy

Fecha

Agradecimientos

En primer lugar, agradecer a mis padres Javier y Alicia y a mi hermana Irene todo el apoyo que me han dado durante el desarrollo de este proyecto.

A mi tutor, M.A. Otaduy por el interés mostrado y por estar disponible siempre que ha sido necesario. También a Sara Carolina Swartchman por el tiempo dedicado y por las mejoras ofrecidas sobre el proyecto.

Por último, a todos mis amigos y compañeros, tanto los de toda la vida como los nuevos, que me ha ofrecido diferentes puntos de vista y me ha animado a terminar.

Gracias a todos

Resumen

Este proyecto se enmarca dentro de la simulación de efectos físicos sobre objetos, más concretamente en la fractura físicamente realista. La fractura se produce cuando un objeto pierde su conectividad interna debido a una deformación, por lo que termina partiéndose en varios pedazos. Este tipo de simulación es bastante costosa, ya que implica cambios importantes en la topología de los objetos y son necesarios cálculos avanzados para mantener el realismo.

Con este tipo de simulaciones realistas se aporta una gran riqueza a las animaciones por ordenador, aplicando dinamismo y permitiendo crear efectos avanzados. Las técnicas relacionadas con la fractura han ido avanzando en el tiempo, de manera que su implementación es lo suficientemente rápida como para aplicarla en videojuegos o aplicaciones en tiempo real.

Para el cálculo de dicha simulación, el algoritmo implementado se ha basado en el Método de Elementos Finitos (*Finite Element Method, FEM*) que permite saber cómo afectan las fuerzas al objeto cuando existe una colisión. Junto con este método, que va a permitir un cálculo físicamente realista de la deformación producida en los objetos, se define un método geométrico (*diagramas de Voronoi*) para el cálculo de las nuevas piezas.

Por lo tanto, durante la simulación van a calcularse las fuerzas internas que sufren los objetos ante una determinada colisión. Tales fuerzas van a estar determinadas por parámetros externos, especialmente por la composición del material que conforma el objeto.

Una vez definidas dichas fuerzas, se calculan las nuevas piezas que se van a crear a causa de la fractura. Dichas piezas se van a definir mediante una variante del método de Voronoi y hasta el momento de la colisión no se va a conocer su geometría final. Además, al incluir una generación pseudo-aleatoria de piezas, la fractura no será la misma ante las mismas condiciones de inicio. Con esta combinación entre métodos físicamente realistas y métodos geométricos se consigue que la fractura no sea conocida a priori, con lo que el realismo aumenta dentro de la simulación.

Para obtener una aplicación más manejable y amigable, se ha integrado también el motor de gráficos OGRE, que va a permitir observar la fractura de una forma más realista (con la aplicación de texturas y una pequeña interfaz de usuario).

Abstract

This project is about the simulation of physical effects on objects, more specifically about realistic fracture. Fracture is created when an object loses its internal connectivity due to deformation and finally breaks into several pieces. This type of simulation is usually expensive because it involves changes in the topology of objects and advanced calculations are necessary to maintain realism.

This type of realistic simulations provides a lot of details to animations, applying dynamism and allowing the creation of advanced effects. The techniques related to fracture have improved over time, so their implementations are fast enough to be included in videogames or real-time applications.

To calculate the simulation, the implemented algorithm is based on the Finite Element Method (*FEM*), which allows to know the forces that affect the object when a collision exists. With that method the internal forces are calculated and depending on external parameters (that define the hardness of the object) some pieces will be created.

This process is iterative so the number of pieces and theirs topology are unknow a priori. When the object collides and the force applied is strong enough, the algorithm begins to calculate some random pieces (the pieces are not completely random, because they depend on the point of collision and other values). When the algorithm ends, the topology of the object changes and new pieces are created.

This algorithm provides realistic simulation of fracture without specifying the number of pieces or the distribution of each piece. The only parameter necessary to get a realistic fracture is the hardness of the object. Neither is it necessary to specify the starting point of fracture, because the physics engine will take care of it (in this case, we use *Bullet*).

This project also integrates a graphics engine (OGRE) which makes the application more aesthetic and useful (using textures, a Graphical User Interface and other effects).

Contenido

1. Introducción

1.1. Conceptos Básicos

En este apartado se van a explicar algunos conceptos esenciales para poder entender cómo es el algoritmo implementado en este proyecto. Todo lo que aquí se expone es introductorio, y será desarrollado en mayor medida durante los siguientes puntos.

1.1.1. Fractura

La fractura de un objeto (fractura mecánica) es aquélla que se produce cuando un objeto separa en dos o más objetos diferentes e independientes. Toda fractura depende de dos valores internos propios: la deformación (*strain*) y la tensión (*stress*). En función de estos dos valores, el objeto puede tener diferentes comportamientos: fracturarse en nuevas piezas o sufrir una deformación que va a cambiar su forma, pero sin llegar a partirse.

Por lo tanto, según la composición interna de los objetos, pueden definirse dos tipos de fractura: *Brittle* ("quebradiza") o *Ductile* ("dúctil"). En la primera de ellas no existirá deformación plástica (como en la fractura de un jarrón de arcilla) y en la segunda sí (como podría darse en algún tubo de un metal débil).

1.1.2. Deformación y Tensión (*Strain, Stress)*

La relación entre estos dos conceptos va a definir cómo va a producirse la fractura. Dependiendo de la composición del material, la relación que va a existir entre estas dos medidas definirá cuándo y cómo se produce la fractura y si va a existir una deformación constante o el objeto volverá a recuperar su forma inicial.

La deformación o *strain* (representado por **ε**) define el cambio de forma y tamaño que sufre el objeto al aplicar una fuerza. El esfuerzo o *stress* (representado por **σ**) define la fuerza por unidad de área en el entorno de un punto material sobre una superficie. Esta definición se aplica tanto a las fuerzas localizadas como fuerzas distribuidas que actúan sobre la superficie.

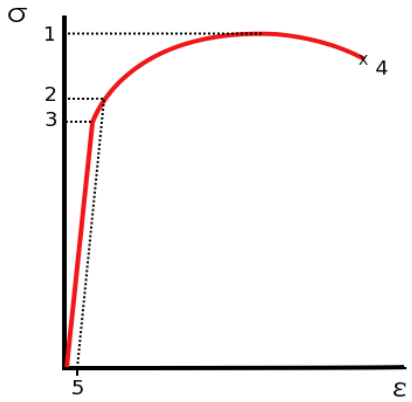

Ilustración 1. Gráfica de relación entre el *strain* y el *stress* en el Aluminio. Los diferentes puntos definen la deformación que se produce hasta llegar al punto de fractura

En la gráfica anterior se pueden comprobar las dependencias de la fractura en función de estos dos parámetros, y cómo va a comportarse un material. Primero doblándose hasta un punto máximo en el cual se obtiene una "fuerza de fractura" (*fracture strength)* para que después se produzca la fractura del objeto en función de dicha fuerza.

Todos los objetos tienen un comportamiento similar, al comienzo van a soportar la fuerza que se está aplicando sobre ellos, hasta llegar a un punto máximo de tensión (*ultimate tensile strength,* punto 3 en la gráfica*)* en el cual aparece una deformación que va a ser irreversible. A partir de ese momento el objeto sigue deformándose hasta llegar a la máxima fuerza soportada que si se sobrepasa producirá la fractura.

A diferencia de los objetos dúctiles, los objetos quebradizos o *brittle* suelen tener el punto máximo de tensión igual que el punto de fractura, ya que no sufren deformaciones antes de partirse.

1.1.3. Método de Elementos Finitos

El método de elementos finitos (*Finite Element Method*, *FEM*) es un método numérico general para la aproximación de soluciones a ecuaciones en derivadas parciales. Este método es perfecto para la solución de problemas físicos sobre geometrías complicadas.

Para el algoritmo que se ha implementado este método es esencial, ya que permite calcular la deformación cuando se produce una colisión, lo que va a permitir conocer el estado de la geometría tras la colisión y calcular las fuerzas internas (*strain* y *stress*) para así poder definir la fractura correctamente.

Con el método FEM, se convierte un problema definido en términos de ecuaciones diferenciales en un problema de forma matricial (mejor para solucionar en un

ordenador). De esta manera se consigue un resultado aproximado en un número finito de puntos o nodos que se van a obtener a partir de la geometría. Este conjunto de puntos se denomina red y forma una malla de retículos (en este caso tetraedros). Cada uno de estos elementos es un "elemento finito" y va a calcularse una solución para cada uno de ellos generalizándola para el conjunto final.

Una propiedad importante del método es la convergencia. Al considerar particiones de elementos finitos cada vez más finas (teniendo por lo tanto más puntos) la solución numérica calculada converge rápidamente hacia la solución exacta de la ecuación diferencial.

La forma general de crear un algoritmo usando este método sigue tres pasos: 1) Un pre-proceso donde se genera la malla y se discretizan los diferentes nodos. 2) El cálculo en sí, donde se utilizan los elementos finitos de la malla para obtener un conjunto de *N* ecuaciones con *N* incógnitas y que puede ser resuelto con cualquier *solver* de ecuaciones lineales. 3) Por último, es necesario un post-proceso donde se actualizan los nodos a los resultados obtenidos en el cálculo anterior y se obtiene una nueva malla y otras características derivadas (como las fuerzas internas).

1.2. Motivación

En este proyecto se trata de conseguir y mejorar una fractura de objetos físicamente correcta y por lo tanto lo más realista posible. Generalmente toda fractura conlleva un alto coste computacional a no ser que se utilicen "trucos" para simular la fractura, evitando con ello la mayoría de los cálculos pesados. El problema de este tipo de simulaciones radica en la necesidad de un artista que defina cómo se va a producir la fractura e intente crearla todo lo realista posible. Aún así en la mayoría de los casos siempre se nota la mano del hombre al visualizar una fractura, especialmente cuando ocurren grandes fracturas sobre un solo objeto. En cambio si se producen multitud de fracturas como puede ocurrir en la simulación de un derrumbamiento el ojo no puede captar todos los detalles. Por ello en este proyecto se trata de conseguir una fractura relativamente rápida y que no dependa en absoluto del artista, sino del material del que están formados los objetos. Para conseguirlo ha de simularse cómo reaccionan los objetos al romperse y evitar la pre-definición de los trozos que van a formarse.

El objetivo del algoritmo implementado permitir definir varios objetos con diferentes materiales en los que van a aparecer fracturas muy diferentes ante las mismas condiciones externas, algo muy interesante de cara a pequeñas simulaciones e incluso videojuegos.

1.3. Objetivos

En este apartado van a explicarse los diferentes objetivos marcados a la hora de comenzar el proyecto. Se trata de una serie de requisitos a cumplir y una forma de orientar el proyecto durante sus fases de desarrollo.

- ***Problema Estático***

Cuando existe una colisión y una fuerza se aplica sobre el objeto que estamos examinando, se va a calcular la deformación como la solución a un problema estático. La idea es obtener un sistema de elementos finitos y ante las fuerzas externas, calcular la deformación total sobre la geometría. Este cálculo no varía con el tiempo porque se calcula la deformación final y requiere para su resolución de un sistema de ecuaciones no-lineales y un método específico como el Método de Newton para su resolución. Para obtener dicha resolución se ha seguido especialmente el trabajo publicado por Müller et al. (1)

- ***Definición de la fractura***

En este punto y tras el cálculo de la deformación final, se calculan las fuerzas internas que van a definir la fractura. Usando los valores internos propios del objeto, y un valor externo que va a definir la dureza del material, se va a especificar el número de piezas nuevas que se van a crear ante la colisión que se ha producido y conociendo la deformación final calculada en el punto anterior. En este punto solo se va a definir qué está ocurriendo sobre el objeto una vez ha colisionado, y es en el siguiente punto donde se va a definir la geometría final de las distintas piezas.

- ***Definir las piezas creadas mediantes puntos pseudo-aleatorios***

Durante la fractura de los objetos se van a ir creando distintas piezas conforme la fractura avanza. Para ello, se utiliza un método de creación de puntos pseudo-aleatorios que van a diferenciar las piezas (creando un mapa de *Voronoi*). De esta manera, se van a diferenciar los tetraedros según pertenezcan a un punto o a otro, y en los límites es donde aparecerá la fractura.

Para obtener estos puntos se ha utilizado un una función de distribución (CDF) que ordena los tetraedros según su deformación ya que es más probable que se rompan cuanto más deformados estén. Esta parte del algoritmo utiliza la energía de fractura para decidir el número de piezas nuevas que se van a producir, siguiendo el trabajo publicado por Zheng (2)

- ***Integrar el motor de físicas Bullet***

Para dar mayor realismo a todo el algoritmo y poder así tener resultados rápidos se ha decidido integrar el motor de físicas *Bullet* junto con el desarrollo anterior. De esta manera se pueden crear simulaciones rápidas porque *Bullet* maneja muy bien los

sólidos rígidos. Además, en este motor no hay nada parecido al algoritmo implementado de fractura.

- ***Integrar el motor de gráficos OGRE***

Para terminar, se ha decidido integrar el motor de gráficos OGRE que dará un mejor aspecto a la aplicación de prueba haciéndola más interactiva y permitiendo la inclusión de elementos gráficos más avanzados de una forma más sencilla.

1.4. Aportaciones

En este punto se van a describir brevemente las aportaciones que promueve este proyecto.

- Crear un algoritmo eficaz y realista de fractura de objetos.
- Evitar la necesidad de pre-definir los trozos o piezas que se van a formar antes de la fractura.
- Permitir cambiar la composición de los materiales para obtener diferentes fracturas en el mismo objeto y con las mismas condiciones iniciales.
- Permitir la fractura de varios objetos sin obtener siempre el mismo resultado.
- Integrar el motor de físicas *Bullet* en todo el proceso
- Integrar OGRE como motor de gráficos.

1.5. Observaciones

Este proyecto se ha integrado en la investigación del grupo GMRV (Grupo de Modelado y Realidad Virtual) por lo que algunas partes del código ya se encontraban desarrolladas con anterioridad. Particularmente, la creación de las mallas de tetraedros e inicialización del sistema de elementos finitos.

También es importante señalar la colaboración de Sara Carolina Schvartzman en varias partes del proceso como la creación de mallas suaves una vez se haya producido la fractura. Se han reutilizado varias partes del código para optimizar este proyecto.

2. Estado del Arte

En este apartado va a hacerse una descripción de los antecedentes a este proyecto. Se explicará brevemente cómo se ha evolucionado la simulación de la fractura en los últimos años y se mostrarán ejemplos concretos de trabajos anteriores. También se expondrán diferentes motores físicos y se explicará cómo desarrollan la fractura físicamente realista.

2.1. Simulación de la Fractura

La fractura en tiempo real es algo relativamente nuevo dentro del mundo de los gráficos, especialmente aquélla que trata de ser todo lo físicamente realista posible. Hace tiempo que se investiga este tipo de simulación aunque los resultados están empezando a verse hoy en día y aún es necesario el uso de "trucos" para acelerar su implementación.

En la simulación, O'Brien (3) (4), ha sido el principal impulsor de algoritmos de fractura realista y éstos empezaron a publicarse hace apenas 10 años (1999-2002). A pesar del gran auge de los gráficos por ordenador en los últimos tiempos, la fractura aún es un tema pendiente ya que tiene una gran complejidad. Esto se debe especialmente a la necesidad de cambiar la topología dentro de los objetos, ya que este proceso requiere redefinir la nueva forma y aplicarla a la simulación lo cual es bastante costoso.

En el mundo de los videojuegos o del cine esta característica está menos desarrollada y normalmente se necesita que algún artista "cree" o defina cómo se va a producir la fractura, la pre-calcule y espere a un momento determinado para lanzar toda la animación.Este proceso hace que la fractura no sea todo lo realista que debería ser y ante ojos atentos siempre se nota que el objeto ha "saltado" o se ha roto de una forma extraña (o al contrario, siempre se rompe igual).

Ilustración 2. Explosión simulada por PhysX (5) en Mafia II

Sin embargo, últimamente los motores de físicas están avanzando enormemente en este campo permitiendo fracturas mucho más realistas y en tiempo real, lo que dota a la experiencia (en el caso de un videojuego) de mucho mayor realismo y permite una mayor inmersión.

Aún así en la mayoría de los casos hoy por hoy es necesario definir cómo son las piezas que forman el objeto final por lo que la partición siempre será la misma (aunque la manera de actuar puede ser muy diferente). Esto es necesario para reducir el tiempo de re-definición de nuevos objetos aunque en un futuro debería cambiar y las piezas se definirán dinámicamente y justo en el momento en el cual el objeto deba fracturarse debido a una fuerza recibida o cualquier otra acción.

En cuanto a las películas, generalmente no se suelen utilizar algoritmos de fractura de cara a partir los objetos ya que en este tipo de animaciones el artista tiene la posibilidad de elegir el comportamiento final de los objetos, sea realista o no. En cambio, los algoritmos utilizados suelen centrarse en controlar cómo se comportan las piezas, debido a su gran número. Las películas no siempre tienen la necesidad de mostrar la realidad físicamente correcta y los artistas pueden tomarse ciertas libertades para crear explosiones mucho más espectaculares o incluso crear efectos de fractura que no pueden existir en la vida real.

Aún así existe una gran dificultad a la hora de animar objetos pasivos. Este tipo de objeto no suele intervenir de gran manera en las escenas. Estos objetos suelen colocarse en la escena y definir sobre ellos unas condiciones iniciales o proponer una ecuación que defina su comportamiento a lo largo de toda la simulación o animación en la escena. En estos casos es donde los algoritmos de fractura (o los de simulación en general) pueden ayudar.

Con este tipo de algoritmos se consigue evitar un gran trabajo a los animadores. Este tipo de prácticas comenzaron a usarse en películas como *Antz* donde se utilizó una simulación del agua (según O'Brien (3)) que no necesitaba una animación propia sino que se trataba de una simulación en el tiempo.

2.2. Trabajos anteriores

En este apartado se van a exponer los trabajos realizados con anterioridad al algoritmo implementado. Al tratarse de un tema bastante novedoso y que está comenzando a incluirse dentro de las simulaciones, no existen muchos algoritmos específicos.

Entre los trabajos más importantes es necesario mencionar a O'Brien (3). Este autor mostró los primeros algoritmos de fractura entre los años 1998 y 2002. El primero de ellos trata sobre la fractura quebradiza mientras que el segundo simplemente mejora el algoritmo anterior para simular la fractura plástica.

Antes de él, Terzopoulus y Fleischer (6) (7) presentaron una técnica para el modelado de deformaciones visco-elásticas y plásticas. En dicho método se utilizan tres tensores métricos que definirán las funciones de energía. Dichas funciones van a medir la deformación sobre curvas, superficies y volúmenes. Además estas funciones de energía proveen de las bases para formular un modelo de deformación continua que permite el uso de diversos métodos de discretización.

En cuanto a la fractura, uno de los métodos basado en una técnica de diferenciación controlada por splines (8) permite demostrar cómo ciertos efectos de la fractura se pueden modelar. Para ello se establecen los coeficientes elásticos entre nodos a cero siempre que la distancia entre ellos sea superior a un límite.

Otro trabajo que ayudó a O'Brien en su algoritmo fue el que presentó Norton et al (9) y que muestra una forma de animar sólidos en 3D que se rompen cuando están sujetos a grandes tensiones. Esta técnica utiliza un sistema masa-muelle para modelar el comportamiento de los objetos. Cuando la distancia entre dos puntos de masa "atados" supera un determinado límite o umbral, la simulación corta los "muelles" (en cuanto a la conexión entre nodos). Para evitar que se formen mallas extrañas (podría darse el caso de la existencia de grandes piezas colgando de un pequeño nodo) se suelen partir todos los muelles que forman un cubo alrededor de los dos puntos mencionados antes.

Las técnicas anteriores ((6), (7), (10) y (9)) tienen ciertas limitaciones importantes. Primero, cuando se produce la fractura no se conoce ni su posición ni su rotación exactas. Esto hace que estas técnicas solo puedan modelar efectos realistas que ocurran en una escala mucho mayor que el espaciado existente entre los nodos.

Otra limitación es que las superficies de fractura están siempre restringidas a los límites definidos en la malla inicial, cuando comienza la simulación. Por ello el patrón de la fractura mostrará ciertos artefactos (*artifacts*). Estos defectos se notan especialmente cuando la discretización de la malla sigue un patrón regular, lo cual es lo más común (3). Esto reduce el realismo de las simulaciones porque la fractura no es del todo libre o no es todo lo precisa que debería.

O'Brien también menciona otros trabajos anteriores en los cuales se explica cómo crear los "cracks" o patrones de fractura dentro de las explosiones, o cómo se crean los patrones en barro seco usando un sistema masa-muelle (11). En otro de los casos se utilizan voxels para definir los objetos y crear la fractura (10) o usar creadores de patrones recursivos para dividir las superficies (12).

Como bien se dice en (3) la fractura ha sido un tema especializado en la ingeniería mecánica por lo que muchas técnicas utilizadas han sido creadas para analizar datos de fractura muy específicos normalmente creados para comprobar durezas de materiales. En el estudio de la fractura se han investigado varios métodos como el FEM, el método de diferencias finitas o simulaciones de partículas. Libros como (13) explican todo este trabajo.

A pesar de que la simulación de fracturas tiene mucho que ver con estos dos campos (gráficos por ordenador e ingeniería) los requerimientos de cada campo son muy diferentes. En la ingeniería se necesitan datos muy precisos y totalmente reales a diferencia de las animaciones por ordenador, donde lo único necesario es que el resultado "parezca" correcto o al menos, que el ojo humano no pueda notar nada raro. Esto hace que tanto en el algoritmo aquí implementado, como en todos los trabajos anteriores, lo único necesario es el resultado final. A pesar de utilizar herramientas propias de la ingeniería no es necesario un grado muy alto de precisión.

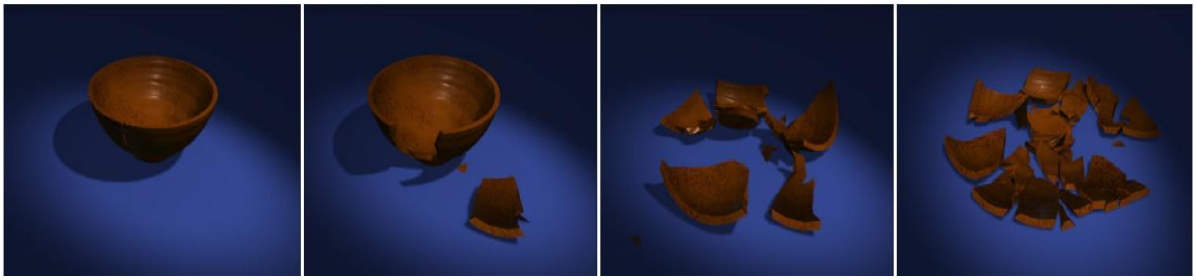

Ilustración 3. Fractura sobre un mismo objeto disminuyendo la dureza. O'Brien et al. (3)

El algoritmo propuesto por O'Brien intenta mejorar todo lo explicado anteriormente implementando las cosas de otro modo. La explicación de este algoritmo se completará en los siguientes apartados (ver sección 2.5).

Otro autor que presentó un artículo sobre como modelar la fractura fue Müller (1). En dicho artículo, Müller además de mostrar un modelo de fractura, presenta una forma de solventar ciertos problemas que aparecen cuando se usa el FEM. Müller explica cómo los modelos de elasticidad linear son comúnmente utilizados en el cálculo de deformaciones gracias a lo estables y "baratos" que son. Sin embargo estos modelos no están preparados para grandes deformaciones rotacionales, ya que van a aparecer grandes distorsiones en la geometría. Una de las soluciones (propuesta por Capell et al. (14)) separa la geometría del objeto en pequeñas divisiones basadas en el esqueleto y cada parte utilizará su rotación local en cada *frame* a la hora de calcular las fuerzas elásticas interiores. Aún así este método produce un problema en los límites entre dos partes de un objeto. Una alternativa para solucionar este problema es el método de "*warped stiffnes*" propuesto por Müller (15) que se utiliza dentro de este proyecto.

Ilustración 4. El perro de la izquierda muestra los errores que se producen al usar un FEM linear bajo grandes rotaciones. Müller et al. (1)

Por último es necesario comentar trabajos más posteriores, como el de Zheng (2) el cual se basa en el trabajo de Bao (16). Este artículo es muy interesante de cara a calcular la fractura si bien su orientación es totalmente distinta. En el trabajo de Zheng se intenta calcular el sonido una vez se ha roto un objeto.

Aun así en el artículo se describe el cálculo de fuerzas internas de cara a definir cómo se va a calcular la fractura. En vez de utilizar un plano (como en el caso anterior) Zheng propone calcular dos energías diferentes que definirán la condición de salida del bucle de fractura, es decir, cuándo termina la fractura del objeto y cuántas piezas nuevas van a aparecer. Estas son la Energía de Tensión (*Strain Energy*) que es constante una vez el objeto ha colisionado y la Energía de fractura, que irá creciendo conforme el objeto se va partiendo (Ver sección 4.1). Esta última energía depende de un parámetro que definirá la composición del material de tal manera que su crecimiento dependerá en gran medida de si el material es duro o frágil.

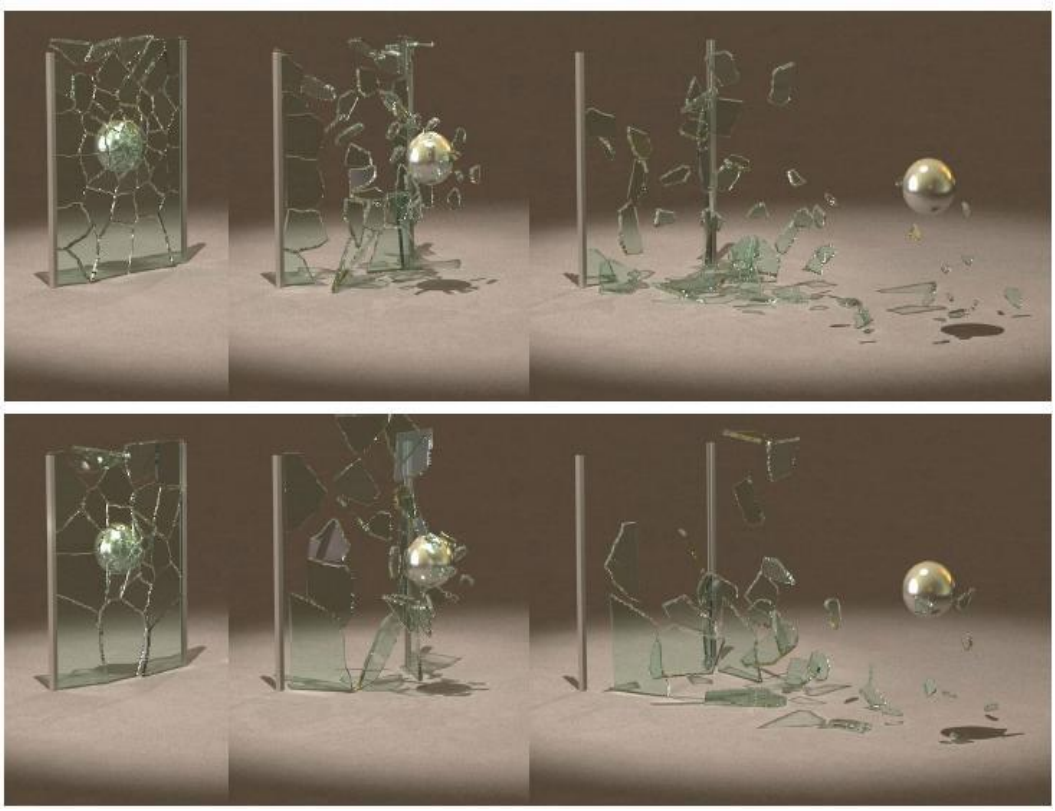

Ilustración 5. La diferencia en la consistencia del material, crea trozos y fracturas diferentes. Zheng et al. (2)

En este artículo también explican una manera de definir la velocidad a posteriori que deben tener los objetos, una vez hayan sido fracturados. Aunque la mayoría de los motores físicos suelen calcular las velocidades finales, en el caso de la fractura, donde se incluyen nuevas piezas, es necesario definirla de alguna manera. Siempre se podría reutilizar la velocidad anterior a la colisión y fractura del padre, aunque en este artículo calculan para cada pieza su velocidad correcta, lo que da mayor realismo a la simulación.

2.3. Tipos de Fractura (Precalculada, no Precalculada)

En este apartado se va a hablar sobre una característica muy importante a la hora de modelar o simular una fractura. Antes de comenzar con la simulación es necesario decidir si la fractura ya estará "creada" o se creará en el momento y dinámicamente. En la mayoría de los casos se suele optar por definir antes de nada cómo son los trozos que se van a formar y por tanto qué comportamiento va a tener la fractura.

Esta forma de actuar puede ser todo lo complicada que se quiera, haciendo los cachos uno a uno para después unirlos (para lo que se necesitaría un artista que decida qué cachos van a aparecer) o utilizando técnicas como los campos de Voronoi, de manera que el creador solo necesite definir algunos puntos y un programa o *script* se encargue de decidir qué cachos van a aparecer. La fractura pre-calculada tiene por

tanto varias ventajas, aunque también tiene sus inconvenientes. Entre las ventajas, se obtiene un control absoluto sobre qué posibles piezas van a aparecer, así como una reducción considerable de cálculo al simular la fractura. Esta reducción viene dada porque no es necesario un remallado dinámico sino que los trozos están pegados y son conocidos (ya están preparados para su simulación).

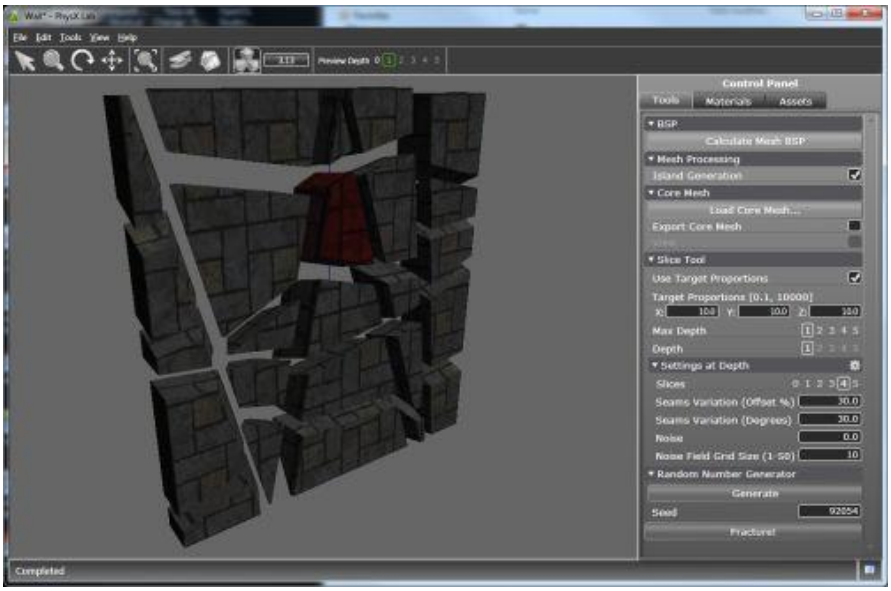

Ilustración 6. Definición de la fractura antes de la colisión en APEX

La desventaja más importante de esta técnica radica en la pérdida de realismo. Una vez un objeto colisiona y se rompe, la fractura va a depender de valores físicos que pueden variar de una colisión a otra. Esto hace que los posibles trozos o piezas que van a aparecer dependerán de las fuerzas internas y de cómo se disperse la fractura. Si los trozos están predefinidos las piezas ya están decididas, por lo que siempre van a ser las mismas sea como sea el impacto que produce la fractura. Esto resta realismo a una simulación ya que si, por ejemplo, la fractura de varias paredes se define de la misma forma, un usuario podría notar que siempre aparecen las mismas piezas, golpee como golpee las paredes.

Para solucionar esto, generalmente los artistas o programadores han de crear diferentes formas de fractura para cada pared, o tratar de hacer “trucos” para engañar al ojo del usuario. Por lo tanto, para conseguir un gran realismo es necesario aportar un mayor esfuerzo a la creación de las piezas antes de la fractura.

El caso inverso se produce si se utiliza una técnica como la de este proyecto en la que la fractura no esté pre-calculada. En este caso los objetos contendrán información sobre el material del que están hechos, y una vez se realice la colisión comenzará un cálculo que va a definir las distintas piezas que se producen. En este caso se van a utilizar parámetros físicos (como el *toughness* o dureza) que van a producir varios cálculos y la necesidad de remallar las distintas piezas para separarlas del objeto

original. Esto tiene un coste computacional mucho mayor ya que el remallado no es algo trivial y suele ocasionar que toda la aplicación se ralentice. Sin embargo el resultado va a obtener un gran realismo porque al estar basado en la física de los materiales los resultados van a ser mucho más diversos (de hecho las piezas que se crean no van a ser conocidas hasta finalizar la simulación).

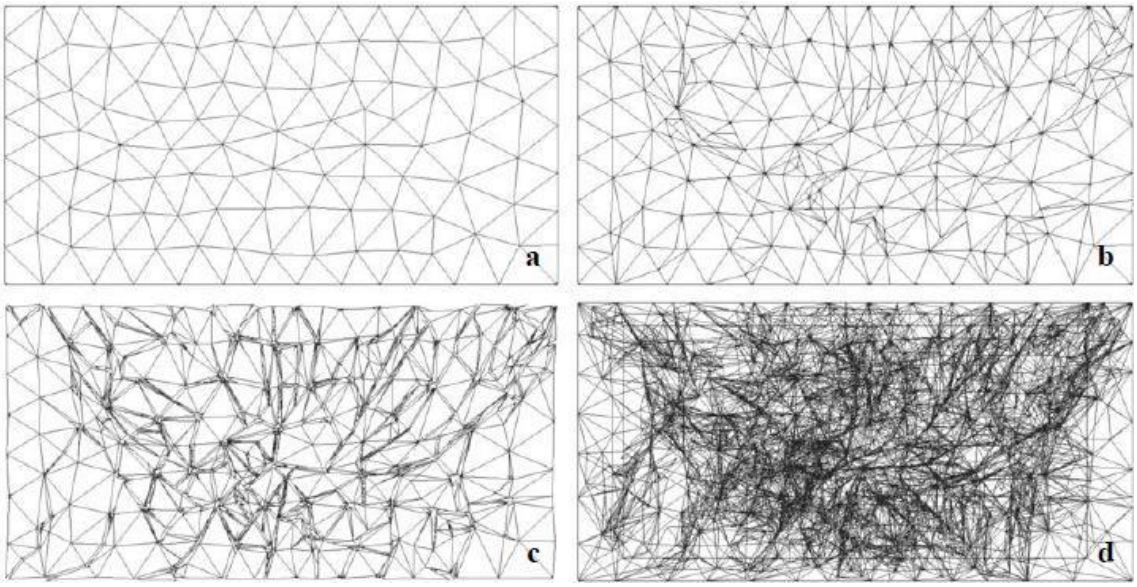

Ilustración 7. Fracturas internas en el remallado al ocasionarse la colisión (sin pre fractura). O'Brien et al. (3)

Con esta técnica la fractura va a depender únicamente de la colisión recibida, por lo que las fracturas no tienen por qué coincidir aunque los objetos estén definidos con los mismos parámetros físicos. De esta manera, para conseguir mayor realismo es necesario un coste bastante alto en cuanto a cálculo se refiere. Hoy en día en la mayoría de las aplicaciones en tiempo real se utilizan piezas ya calculadas e intentan introducir algo de aleatoriedad dentro de la fractura para que no siempre se produzca el mismo resultado. En el caso de aplicaciones relacionadas con la ingeniería suele ser al contrario, ya que el coste computacional no es un problema prioritario (siempre dentro de unos márgenes) y la velocidad no es tan importante como el resultado final.

De todas formas está mucho más generalizada la fractura pre-calculada, hasta el punto de que la mayoría de los motores e incluso algunos programas en gráficos suelen incluir una pre-fractura antes de comenzar a simular. Es de suponer que con el crecimiento de los PCs y el uso de tarjetas gráficas cada vez más potentes en un futuro será mucho más sencillo definir el material de los objetos y que la fractura no depende del creador, sino que surja de una forma correcta y real.

2.4. Motores de física con fractura

Por último, dentro de este apartado se van a mostrar diversos motores o plug-in que permiten la fractura de objetos. El primero de ellos es un plug-in diseñado por la empresa Pixelux y O'Brien, pensada para herramientas de modelado, más concretamente 3ds Max y Maya.

- ***DMM***

Este plug-in permite simular una fractura dentro de las propias herramientas de modelado, de tal manera que siguiendo unos cuantos pasos podemos crear varias simulaciones bastante creíbles. DMM (Digital Molecular Matter (17)) se denomina como una tecnología avanzada de simulación, utilizada por los estudios LucasArts en el videojuego "Star Wars: The Force Unleashed". Con esta tecnología en una escena se pueden definir varios tipos de objetos. Están los objetos fracturables (mediante una malla de tetraedros que se crea automáticamente), los objetos rígidos, que no se rompen, y diversas herramientas para poder unir dos objetos fracturables (una pared y un cristal) aunque tengan diferente dureza.

El uso de este plug-in es bastante sencillo y consigue resultados realmente impresionante en materiales parecidos a la madera (que al romperse producen astillas y desgarros). En cambio, cuando se intenta simular un material más duro (como un metal o una piedra) el resultado no es tan espectacular. Una de las desventajas de esta herramienta radica en el tratamiento de mallas de alta resolución. Esto produce que un objeto con una malla de alta resolución y formada por muchos tetraedros no pueda partirse en trozos grandes y pequeños, como se espera en cualquier fractura. Lo que ocurrirá, es que la malla se partirá en todos los tetraedros que lo forman, por lo que aunque se produzca una colisión pequeña, que solo debería producir algunas piezas pequeñas y otras más grandes, el objeto se partirá en muchos tetraedros, lo que resta realismo a la simulación.

Ilustración 8. Colisión en múltiples trozos con DMM y una malla de alta resolución

Aun así, estos casos se pueden evitar definiendo varias mallas dentro del mismo objeto, de manera que se pre-establece que va a existir una pieza grande y otras más pequeñas. Esto nos lleva a lo comentando en los apartados anteriores, es necesaria la mano de un artista para definir la fractura antes de que esta se produzca. Sin embargo

DMM permite que los objetos se fracturen de distintas maneras en función del material del que están compuestos por lo que da una solución intermedia donde es necesario conocer cómo se va a ocasionar la fractura, pero el programa nos ahorra mucho esfuerzo de cara a definir las piezas nuevas y su comportamiento.

Como ellos mismos definen, esta tecnología permite quitar el efecto "gomaespuma" que existe en las aplicaciones de tiempo real cuando se parte un objeto, permitiendo que este parezca un objeto mucho más frágil (los trozos creados tienen una forma mucho más real). Para conseguir este efecto se está utilizando toda la teoría de los artículos de O'Brien, calculando el *stress* entre puntos ante una colisión y decidiendo planos de fractura a la hora de partir el objeto inicial en diferentes piezas. Esta tecnología permite también cierta deformación en los objetos una vez se ha producido la fractura consiguiendo una fractura plástica, donde el objeto se deforma antes de partirse.

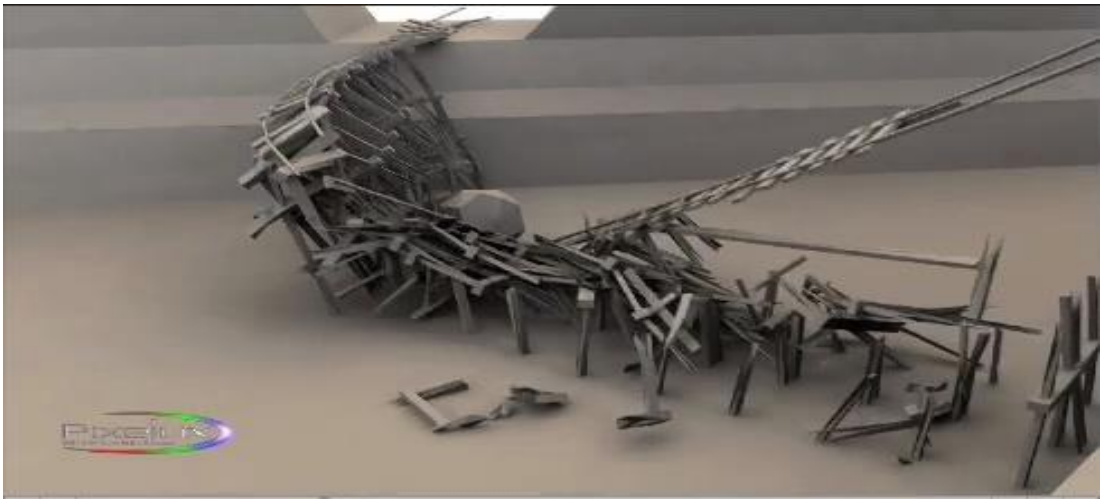

Ilustración 9. Demostración del motor DMM.

Esta tecnología por tanto, nos da una buena aproximación a la fractura en tiempo real, si bien su utilización no es todo lo sencilla que parece. Existen varios problemas relacionados con los materiales y su forma de partirse que empeoran las sensaciones y hacen difícil conseguir el resultado que se quiere. Por ejemplo, hay varios problemas al juntar un cristal dentro de una pared, ya que es necesario ajustar muy bien ciertos valores porque si no el cristal se romperá ante el contacto de la pared, o incluso puede que la pared se destruya y el cristal simplemente se deforme.

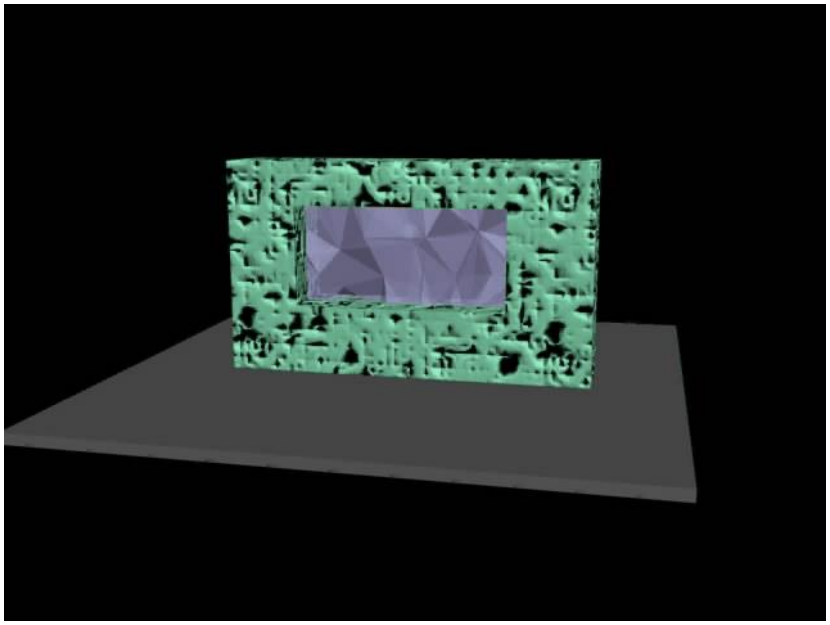

Ilustración 10. El cristal ya está roto aunque aún no se ha producido ninguna colisión. Esto se debe al ajuste de parámetros entre la pared y el cristal.

- ***APEX Destruction (PhysX)***

APEX es la herramienta que nos facilita NVidia de cara a crear fracturas dentro de su motor de físicas PhysX. Se define como un *framework* que da la posibilidad a los artistas de crear sistemas dinámicos complejos sin tener experiencia en programación. Esta herramienta está totalmente enfocada al mundo artístico, si bien contiene una base teórica bastante fuerte, el artista nunca la va a ver y simplemente va a poder decidir cuál va a ser el resultado final. Uno de los motores de videojuegos que hace uso de este *framework*, es el famoso UDK (*Unreal Development Kit*) y se ha utilizado en videojuegos como *Batman Arkaham Asylum* o *Mafia II*.

Dentro de las múltiples herramientas que proporciona APEX, la que interesa de cara a este proyecto se denomina APEX Desctruction (5) y es la que nos va a permitir definir cómo se produce todo lo relacionado con fracturas o explosiones. Esta herramienta permite crear fracturas ante efectos externos (como el disparo de una pistola) y utiliza la tecnología de pre-fractura, ya que el artista define las piezas y qué objetos se van a romper antes de que el usuario interactúe. Entre otras cosas permite definir distintos materiales (como cristal o madera) y ayuda al artista al decidir la forma en que se va a partir, de tal manera que un cristal no va a crear astillas como sí lo hará la madera. De esta forma el propio programa controla el resultado final.

Como se puede ver esta herramienta está totalmente orientada a personas ajenas a la física, por lo que los términos a ajustar no van a tener que ver con tensiones o *stress* interno porque hace uso de otros parámetros para definir estos valores y enmascararlos para que su uso sea sencillo para cualquier persona.

Ilustración 11. Demostración de la fractura en APEX Destruction

- ***Havok y otros motores***

No se puede encontrar mucha información del resto de motores físicos, bien porque son privados y no muestran cómo se puede crear la fractura o porque no tienen una herramienta específica para la fractura de elementos.

En el caso del motor Havok (18) existe una herramienta denominada *Havok Destruction* que permite la creación de objetos fracturables. Por la poca información que se puede encontrar parece una herramienta similar a APEX, ya que está muy orientada al artista que no tiene conocimiento de cómo se produce la fractura sino que solo quiere saber el resultado final. Por lo tanto, esta herramienta permite definir qué partes dentro de un objeto se van a partir, permite crear deformaciones ante estímulos y la creación de estructuras básicas (por ejemplo una pared de ladrillos) en las cuales no es necesario definir la forma de los trozos ya que el programa se encargará de ello por su cuenta.

Dentro de los motores libres, *Bullet* da la posibilidad de crear ”objetos compuestos” de tal manera que mediante código se puede formar un objeto creado a base de múltiples trozos y mediante el uso de restricciones o *constraints* definir en qué momento se va a producir la fractura. Como ya se comentó antes en este tipo de motores no hay una herramienta específica para crear la fractura, por lo que es el usuario el que tiene que implementar cómo va a ocurrir. Para ello, puede crearse objetos compuestos y separarlos cuando ocurre la fractura o crearse su propia fractura. Esta es una de las razones por la que se eligió este motor al crear este proyecto aunque no se hace uso de formas compuestas.

Existen ciertos plug-in que utilizan *Bullet* como motor para simular efectos físicos. Por ejemplo, el plug-in *Dynamica* (19) desarrollado por Disney y pensado para Maya, permite la simulación de efectos físicos y la creación de efectos especiales. Una de las películas en

las que se utilizó dicha herramienta fue *Bolt*. Aún faltan por comentar otros motores de físicas, pero no se ha encontrado muchas referencias a la fractura si bien en casi todos los casos se utilizan técnicas similares a la de *Bullet*. Con ello se pretende crear un objeto ya partido y establecer conexiones entre los trozos para romperlas ante determinados estímulos.

2.5. Definición de la fractura *"quebradiza"* y *"plástica"*.

En este apartado se va a explicar las bases de la fractura según O'Brien. Este autor propone un modelo continuo y un cálculo de la deformación y el esfuerzo diferentes. No se ha creído necesario exponer los cálculos en esta sección, ya que se va el algoritmo implementado no va a hacer uso de ellos, pero se pueden leer en (3) y (4) para más información. En cuanto al modelo de fractura, O'Brien muestra conceptos muy importantes que han sido la base para el algoritmo actual.

Para empezar O'Brien define que su modelo de fractura está basado en la teoría mecánica de fractura lineal. El concepto principal es que la fractura aparece debido al *stress* interno cuando un material se deforma. Para ello es necesario simular o modelar las deformaciones que van a sufrir los objetos y que crearán dicho *stress*. Una de las diferencias más importantes en esta teoría es que no se disipa la energía en regiones muy cercanas a la aparición del "crack", de manera que no aparece plasticidad en los modelos.

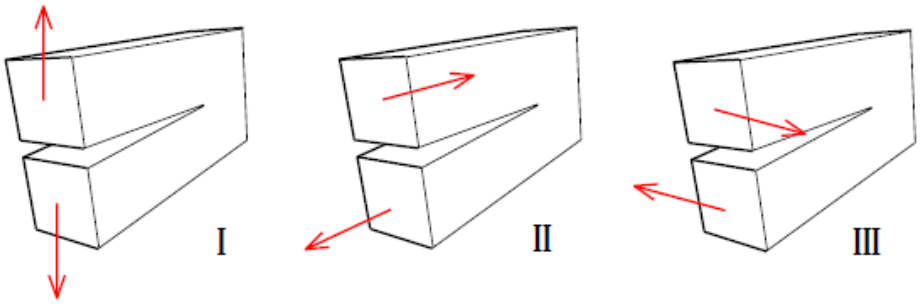

Ilustración 12. Tres fuerzas que pueden crear un crack. O'Brien et al. (3)

Como no se está disipando la energía en la región anterior los modelos serán siempre "quebradizos" (como podría ser el cristal o la arcilla). Esta definición no significa que los materiales sean débiles sino que "quebradizo" (del inglés *brittle*) se refiere al hecho de que una vez el material comienza romperse, la fractura creada tiene una fuerte tendencia a propagarse a través del material como si fuera conducida por la energía elástica almacenada en el interior.

O'Brien define también tres modos de aplicar las fuerzas dentro de un crack. Estos "cracks" se podrían ver como la separación que se produce en un elemento cuando se está fracturando y dependiendo de cómo se apliquen las fuerzas la fractura actuará de

una manera o de otra. Las fuerzas de tracción opuestas harán que el crack continúe en una dirección que es perpendicular a la mayor de estas fuerzas y por lo tanto, las fuerzas de compresión detendrán a los cracks que sean perpendiculares a ellas (como se ve en la Ilustración 12). Para calcular cómo se va a producir la fractura, se comprueban las fuerzas internas entre los nodos de la malla de tetraedros. O'Brien utiliza la forma de los objetos para determinar en qué punto aparece la fractura, basándose en que normalmente la fractura comienza en las superficies cóncavas.

El algoritmo de O'Brien por tanto es el siguiente: Tras cada paso de simulación, el sistema resuelve el cálculo de fuerzas internas que actúan en todos los nodos de la malla (tanto fuerzas de tracción como de compresión) descartando las partes no balanceadas. Para cada nodo, las fuerzas resultantes se utilizan para formar un tensor que va a describir cómo afectan las fuerzas internas a la hora de separar ese nodo. Si esta acción es lo suficientemente grande (superando un límite establecido previamente) el nodo se va a partir en dos y se va a calcular un plano de fractura. Todos los elementos adjuntos al nodo se dividen siguiendo el plano calculado reasignando los tetraedros a uno de los dos puntos recién creados, creando una discontinuidad en el material y definiendo dos "trozos" diferentes (cada uno asignado a los lados creados). Todos los valores asociados a los nodos (como la masa) se recalculan para los elementos afectados (3).

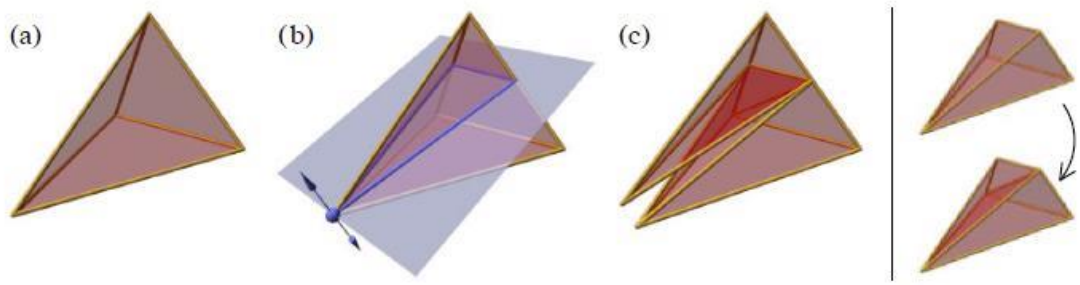

Ilustración 13. Cálculo del plano de fractura y separación de las piezas duplicando los nodos. O'Brien (3)

Una vez la simulación haya determinado la localización y orientación del nuevo plano de fractura, la malla se necesita recalcular para mostrar la discontinuidad que se ha creado. Es importante que la orientación de la fractura se preserve ya que la aproximación con los diferentes elementos del objeto creará efectos negativos no deseados. Para evitar esto, el algoritmo propuesto por O'Brien recalcula la malla solo en el área local que rodea a la fractura que se ha creado, dividiendo los elementos que intersecan con el plano de fractura y modificando los elementos vecinos para asegurar la consistencia. Para ello, se crean dos nodos diferentes en el propio nodo que se ha partido con los mismos valores de velocidad y posición local y global. Las masas se van a calcular más adelante.

Después se examinan los elementos adjuntos al nodo inicial, y se ve si intersecan con el plano de fractura, si no es el caso el elemento se reasigna a uno de los dos nuevos nodos dependiendo en qué cara del plano se encuentre. En el caso de que el elemento interseque con el plano, se necesita partir a lo largo del plano para crear nuevos trozos. Esto no es algo trivial porque es necesario comprobar los tetraedros vecinos para definir cómo se va a formar el corte evitando mallas erróneas o con agujeros. Cuando este proceso acaba, todos los valores anteriores sobre los nodos no son validos y tienen que recalcularse (como por ejemplo la masa asociada a los nodos).

Antes de pasar al siguiente punto se va a explicar brevemente el otro trabajo de O'Brien (4) en el cual adapta el algoritmo anterior para crear una fractura dúctil. En este proyecto no se ha llegado a este punto aunque es una de las posibles líneas futuras a llevar a cabo. Como bien explica O'Brien, la definición de material elástico suele llevar a error porque su significado técnico es el de un material que vuelve a su forma original cuando las fuerzas que lo deformaban cesan. Sin embargo en la realidad no existe ningún material puramente elástico. Un material quebradizo no significa que sea especialmente frágil, sino que se comporta de forma elástica hasta que llega a su punto de fractura. En los materiales como el cristal el punto de fractura está muy cercano conforme empieza la deformación lo que supone que no exista casi deformación sino que aparezca directamente la fractura del objeto.

En contraste con los materiales elásticos, un material plástico no vuelve a su posición inicial cuando se ha deformado y las fuerzas han cesado. Cuando un material como el plomo se deforma no recupera su forma inicial, éste sería un comportamiento plástico. Como se explicó en el punto 1.1.2 (página 1) todo material tiene un comportamiento elástico hasta que llega a su "límite elástico". En este momento comienza con un comportamiento plástico irreversible hasta un punto en el cual cede y se rompe. De este modo si el punto de fractura coincide con el "límite elástico", se hablará de un material quebradizo. En cambio si todavía existe un cierto margen se hablará de un material dúctil. Esta definición es importante ya que la deformación elástica almacena energía, mientras que la deformación plástica la disipa. Por lo tanto en un material elástico, cuando se produce un crack, solo se necesita una pequeña cantidad de energía para que se propague, y en cambio en un material dúctil se necesita bastante más energía ya que la propia deformación plástica está absorbiendo parte de ella.

Se puede ver el algoritmo completo dentro del artículo (4) pero básicamente es necesario cambiar el *strain* para permitir que exista cierta deformación plástica. Al cambiar este valor, el *stress* también variará y será necesario cambiar el límite para que la fractura se produzca en otro momento (tras la deformación plástica). El cambio de la deformación trata de separar la deformación plástica de la elástica

$$\epsilon = \epsilon^p + \epsilon^e$$

Tras esto, solo hay que cambiar el cálculo de los valores anteriores y seguir el algoritmo previamente definido, adaptando algunos puntos para formar la deformación.

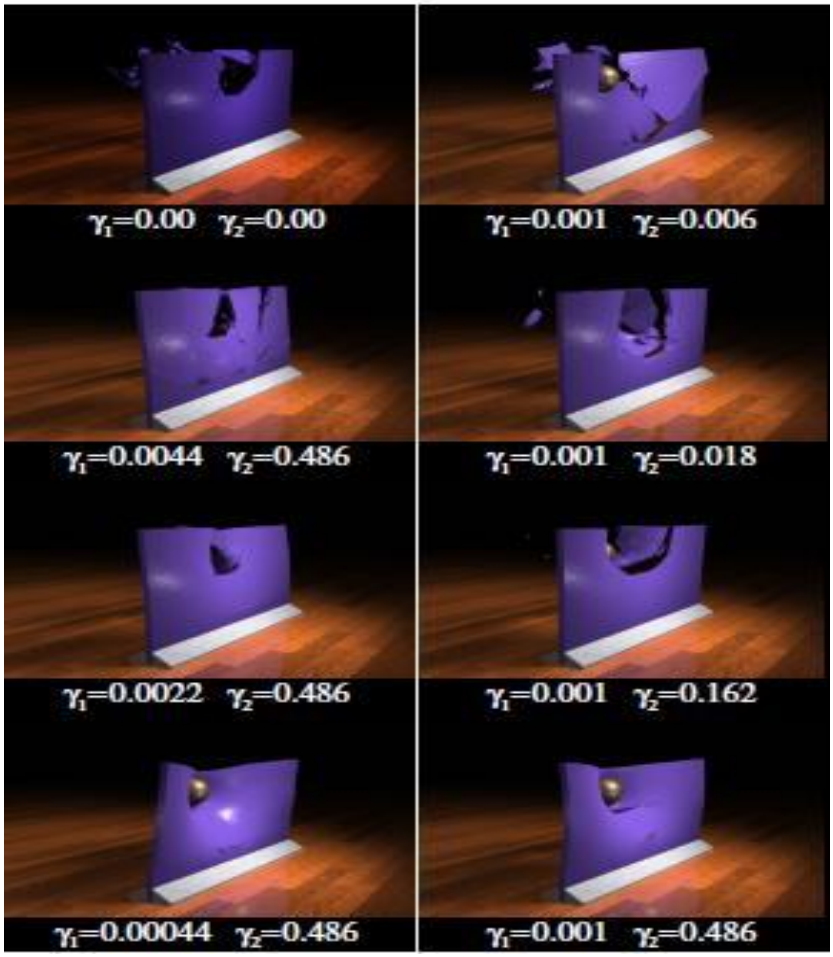

Ilustración 14. Distintas deformaciones plásticas en función del material. O'Brien (4)

2.6. *Bullet* y OGRE.

Antes de terminar es conveniente explicar las características y decisiones por las que se han escogido estos dos motores a la hora de integrarlos en el proyecto. En ambos casos se han escogido motores de código libre, permitiendo por lo tanto que la aplicación se pueda ejecutar en diferentes plataformas y dando mayor libertad al programador para modificar el código interno del motor si así lo cree necesario. Con esto se consigue evitar fallos que pudieran aparecer en casos específicos (como podría ser la fractura).

En el caso de *Bullet*, este motor de físicas está teniendo una gran importancia en estos días, utilizándose en muchos programas (como Houdini11) ya que permite gran flexibilidad al usuario para simular un mundo físico con exactitud. Este motor está muy optimizado y tiene una gran comunidad que aporta soluciones a los problemas más comunes que pueden tener los usuarios noveles. Además, el tener código libre permite

ampliar o modificar lo que ya está hecho, permitiendo una centralización sobre el problema que se quiera resolver. En este proyecto su uso se ha centrado en la simulación de sólidos rígidos y en la captura de colisiones de cara al cálculo de la fractura.

En general es un motor muy útil y en el que se obtienen resultados rápidamente, si bien en ciertas ocasiones han surgido dificultades a la hora de integrarlo dentro de la aplicación (ver punto 5) en general este motor permite simular comportamientos complejos, como la inclusión de muchas piezas nuevas y su simulación, de forma sencilla y rápida.

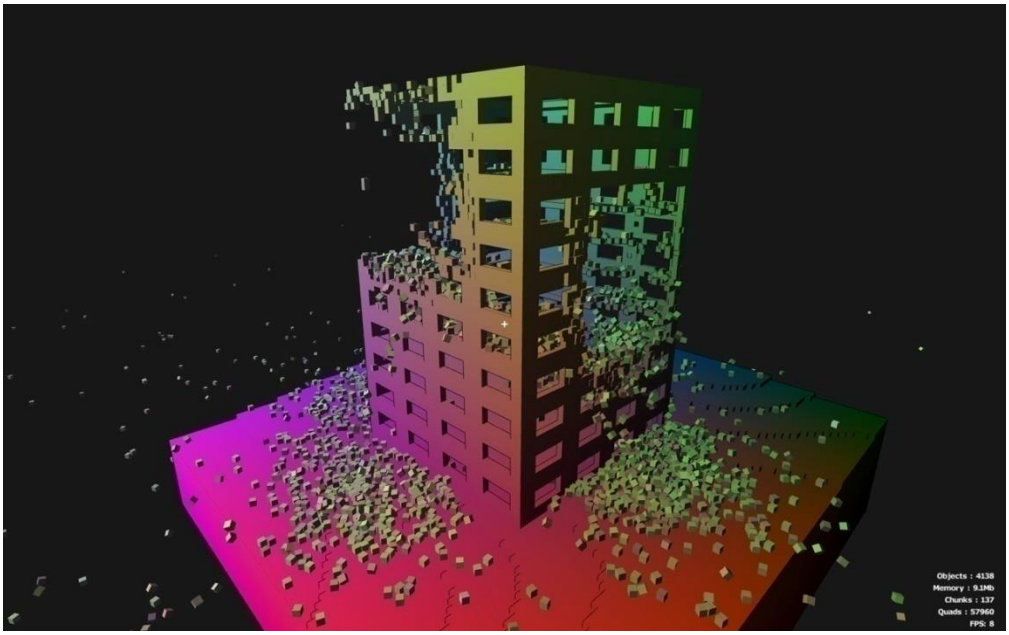

Ilustración 15. Simulación de miles de piezas en Bullet

En cuanto a OGRE, este motor de gráficos proporciona una nueva capa de abstracción sobre lenguajes como OpenGL o DirectX. Con ello se consigue mayor flexibilidad de cara a poder mostrar lo que ocurre dentro de la simulación, facilitando la inclusión de sombras o efectos gráficos de una forma intuitiva. Este motor está ideado para utilizarse en multitud de aplicaciones, por lo que es lo suficientemente flexible para permitir complejidades de escena muy grandes.

En el caso de este proyecto, el motor simplemente se ha usado para permitir una mejoría visual y apreciar mejor el algoritmo de fractura. Por lo tanto, se han incluido sombras y efectos visuales (como la composición de los materiales de los objetos) que facilitan a los usuarios comprender qué se está simulando y a qué se debe su comportamiento. Por ejemplo, la inclusión de sombras facilita mucho comprobar cuándo se produce una colisión con el suelo y entender la posición tridimensional de los objetos. En cuanto a la visualización de varios materiales, permite diferenciar los objetos de cara a comprobar si su fractura es correcta porque un objeto parecido al

cristal debe partirse en muchas más piezas que un objeto con un material que simule un metal. Este motor también da flexibilidad de cara a introducir luces y cámaras en la escena para poder apreciar mejor la simulación final.

Ilustración 16. Diferentes usos del motor OGRE

3. Cálculo de deformaciones

En este apartado se va a explicar el cálculo de las deformaciones y su implementación. Para una mayor comprensión primero se explicarán las bases en el cálculo de deformaciones para más adelante explicar el caso concreto de este proyecto.

3.1. Método de Elementos Finitos

El FEM puede tener múltiples interpretaciones y diferentes modelos de cara a su implementación e integración en herramientas de ingeniería. A pesar de ello el concepto y su definición es siempre la misma. Se trata de un Método numérico que permite obtener una solución numérica aproximada sobre un medio continuo (en este caso un cuerpo) sobre el que hay definidas diferentes ecuaciones diferenciales. Para hallar la solución el cuerpo se divide en un número elevado de subdominios denominados "elementos finitos". Dentro de estos elementos se encuentran los diferentes nodos que forman el objeto. Estos nodos pueden pertenecer a diferentes elementos finitos, formando relaciones que se utilizarán más adelante. El conjunto de relaciones de una determinada variable entre los nodos que forman la malla se puede escribir como un sistema de ecuaciones lineales. Dicha matriz se denomina matriz de rigidez y el número de ecuaciones es proporcional al número de nodos.

Por todo lo anterior este método se suele utilizar a la hora de calcular qué deformaciones puede sufrir un objeto ante unas fuerzas externas. Dichas fuerzas se van a aplicar sobre ciertos nodos que, dependiendo de sus relaciones con el resto de la malla, van a modificar la posición de varios nodos (por ejemplo siguiendo la segunda ley de Newton) para finalmente describir qué deformaciones ha sufrido la malla.

Con este método se pueden solucionar sistemas de ecuaciones diferenciales muy grandes que son complicados de solucionar de forma analítica. Una de sus propiedades más importantes es la convergencia, ya que si se parte la malla en elementos cada vez más finos la solución numérica converge rápidamente hacia la solución exacta del sistema de ecuaciones.

Dentro de este proyecto, se ha utilizado un modelo para calcular las fuerzas elásticas basado en el artículo de Müller (1) que utiliza la técnica del *"warped stiffness"* y evita

que aparezcan fallos cuando se producen deformaciones rotacionales grandes. Este modelo va a utilizar un FEM basado en tetraedros para resolver las ecuaciones diferenciales parciales. Para comenzar, es necesario definir cómo se calcula la matriz de rigidez o *stiffness* denominada 'K'.

En un medio continuo, la teoría de la elasticidad lineal dice que la deformación de un objeto viene definida por el vector

$$u(x) = [u(x), v(x), w(x)]^T$$

lo cual significa que cada punto $x = [x, y, z]^T$ que pertenezca a la malla sin deformar corresponde al punto $x + u(x)$ dentro de la malla deformada. El primer paso es reemplazar el desplazamiento continuo $u(x)$ por un grupo de vectores de desplazamiento definidos solo en los vértices de la malla de tetraedros. Por lo tanto para cada tetraedro *e* el desplazamiento se puede interpolar como

$$u(x) = H_e(x) \cdot \hat{u}$$

donde $H_e(x)$ es una matriz de dimensiones 3×12 que contiene las funciones de forma del tetraedro y $\hat{u} = [u_1, v_1, w_1, \dots, u_4, v_4, w_4]^T$ es la colección de vectores de desplazamiento para los cuatro nodos del tetraedro. Una vez descrita la deformación se van a describir el *strain (ε)* y el *stress (σ)* que serán los parámetros más importantes de cara a la fractura. Como ya se ha comentado con anterioridad el *strain* es una medida normalizada de deformación que representa el desplazamiento entre las partículas en un cuerpo en relación con una longitud de referencia inicial. Existen varias formas de calcular esta medida y para este proyecto se ha optado por utilizar el tensor de Cauchy que va a permitir calcular la deformación ante unas fuerzas externas aplicadas sobre el objeto.

Por tanto, utilizando el tensor de Cauchy que al ser simétrico se puede reescribir como un vector 6×1 y el vector de desplazamientos, obtenemos

$$\epsilon = B_e \cdot \hat{u}$$

donde B_e es una matriz constante en el tiempo de dimensiones 6×12 que puede ser pre-computada para cada tetraedro ya que no varía con el tiempo. Para definir σ necesitamos utilizar la ley de Hooke de tal manera que se pueda tener el resultado

$$\sigma = E \cdot \epsilon = E B_e \hat{u}$$

en este caso, E es una matriz 6×6 que depende de dos parámetros, el módulo de Young y el coeficiente de Poisson (1). El *stress* o esfuerzo elástico nos va a permitir medir las fuerzas internas que actúan dentro de un cuerpo deformable. De ahí su relación directa con el *strain* del cuerpo. Cuantitativamente σ va a medir el promedio de la fuerza por unidad de área en una superficie que pertenece al cuerpo donde actúan las fuerzas

internas. Es necesario remarcar que todos estos cálculos son necesarios para cada uno de los elementos (en este caso tetraedros) que forman la malla y también recalcar que ambos valores (*ε y σ*) son constantes dentro de cada tetraedro, por lo solo se necesitan calcular al comenzar la simulación.

Con todo lo anterior, se asume que las fuerzas elásticas f_e que actúan sobre los nodos de un elemento se derivan de la energía de deformación (*strain energy*) la cual depende del *strain* y el *stress* ya calculados. Utilizando la deformación de Cauchy, las fuerzas son linealmente dependientes del vector de deformaciones û con lo que

$$f_e = K_e \cdot \hat{u}$$

donde

$$K_e = V_e B_e^T E B_e$$

que es la matriz de rigidez del elemento y V_e el volumen. Por último la matriz K total de la malla (de dimensiones $3n \times 3n$) es un conjunto de todas las K_e de los elementos (1).

Una vez obtenida la matriz de rigidez, el siguiente paso es ver cómo ante unas fuerzas externas vamos a calcular la deformación que se produce en el cuerpo. En este proceso es donde entra la importancia del "*warped stiffnes"*, ya que va a evitar los problemas mencionados anteriormente. Generalmente para obtener la resolución de la deformación que va a aparecer ante unas fuerzas externas, es necesario definir un sistema de $3n$ ecuaciones diferenciales lineales para los n vectores de posición (1).

Las ventajas de utilizar fuerzas elásticas linealizadas es que la matriz de rigidez K es constante por lo que se puede pre-calcular antes de que la simulación comience. Sin embargo las fuerzas elásticas linealizadas solo son validas si las deformaciones son semejantes a la condición de equilibrio. Ante grandes deformaciones rotacionales, se crean errores graves como el mostrado en la sección 2.2.

Para solucionar este tipo de errores Müller propone utilizar la técnicas del *"Element-Based Warped Stiffness"*. Esta técnica ya definida en su otro trabajo (15) proponía calcular las fuerzas elásticas para cada vértice en un estado local de coordenadas sin rotación. Sin embargo como se explica en (1) existían ciertos problemas en este proceso (relacionados con las matrices de rotación de los vértices) que se han solucionado extrayendo las matrices de rotación de los elementos en vez de los vértices. Para un único tetraedro con una matriz de rigidez K_e, las fuerzas f_e actuando sobre los cuatro nodos son

$$f_e = K_e \cdot (x - x_0) = K_e \cdot x + f_{0e}$$

x es un vector con las posiciones de los cuatro vértices y f_{0e}contiene fuerzas de compensación (*force offset*). Si se supone que la matriz de rotación de la deformación R_e es conocida podemos usar la técnica del "*warped stiffnes*" para calcular las fuerzas

$$\begin{aligned} f_e &= R_e K_e \cdot (R_e^{-1} x - x_0) \\ &= R_e K_e R_e^{-1} x - R_e K_e x_0 \\ &= K_e' x + f_{0e}' \end{aligned}$$

donde R_e tiene unas dimensiones de 12×12 que contiene cuatro copias del la matriz de rotación 3×3 en su diagonal. Con este cálculo se consiguen las mismas fuerzas que al rotar el objeto. Se asegura entonces que las fuerzas f_e sumen cero. Para calcular por tanto las fuerzas elásticas de toda la malla se obtiene

$$f = K' x + f_0'$$

En este caso K' y f_0' serán la suma de todas las matrices y fuerzas complementarias de los diferentes elementos

$$K_e' = R_e K_e R_e'$$

$$f_{0e}' = R_e f_{0e}$$

Una vez hechos todo estos cálculos hay que resolver el sistema de ecuaciones que se forma para obtener las deformaciones que sufre el objeto. En los siguientes apartados se explican otros problemas derivados (se necesita resolver el problema estático) y algunas ecuaciones varían para adaptarse al algoritmo que se ha creado.

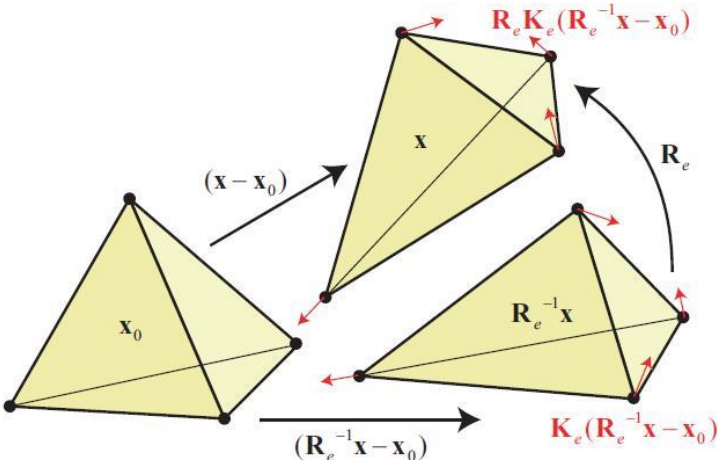

Ilustración 17. Esquema de la técnica "*Element-Based Warped Stiffnes*"

3.2. Resolución del Problema Estático

El primer paso que se tuvo que solucionar en la implementación del algoritmo fue la solución del problema estático que aparece cuando un cuerpo recibe unas fuerzas externas ("condiciones de contorno") y se quiere calcular como se va a deformar. A diferencia del caso dinámico no se va a obtener una solución a cada paso de tiempo, sino que se obtendrá el resultado final obteniendo la deformación total que sufre el objeto ante las fuerzas. El sistema de ecuaciones que se va a obtener es un sistema no lineal por lo que habrá que usar un método concreto para su resolución. En este proyecto se ha implementado el método de Newton-Raphson para obtener la solución al sistema.

La ecuación que se va a solucionar para cada punto de la malla, va a seguir la segunda ley de Newton

$$M\ddot{x} = \sum F_i(x)$$

con la condición de que las fuerzas externas sumen siempre cero (que será en un estado de reposo). Esta ecuación se puede escribir de una forma diferente para poder aplicar el método de Newton-Raphson. Por lo tanto tendremos

$$F_{int} + F_{ext} = 0$$

Este es el caso estático genérico, que generalmente no es lineal por lo que necesita de una linealización para poder resolver el problema que se plantea. Para hacer la linealización, se utiliza un punto conocido (x0, el estado inicial) con lo que la ecuación queda

$$F_{int}(x_0) + {}^{dF_{int}}/_{dx} * (x - x_0) + F_{ext} = 0$$

Aun así, esta es una formulación general del problema. En el caso concreto de este proyecto, se utiliza la ecuación definida por el *stiffness warping* (sección 3.1) que ya se encuentra linealizada, quedando como fuerzas internas

$$F_{int} = K'x + f_0'$$

$$K'x + f_0' + F_{ext} = 0$$

donde K sería la matriz de rigidez total de la malla y x la posición final. Una vez formulado el problema, se aplica el método de Newton-Raphson, creando iteraciones hasta que la diferencia entre dos soluciones sea menor que una variable de error ε. Si sustituimos la ecuación anterior con los valores conocidos de nuestro sistema de elementos finitos, y despejamos la posición obtenemos

$$K \cdot (x_i - x_0) + K \cdot (x_{i+1} - x_i) + F_{ext} = 0$$

$$K \cdot (x_{i+1} - x_0) + F_{ext} = 0$$

$$x_{i+1} = x_0 - K^{-1} \cdot F_{ext}$$

Esta sería una iteración de todo el método, se tendría que repetir este proceso hasta que el siguiente valor no superara el error preestablecido. Esta condición puede tardar mucho en cumplirse o ser inmediata, por lo que este punto suele ralentizar la simulación. Hay que tener en cuenta también que este método no asegura la convergencia global a no ser que se seleccione un valor inicial lo suficientemente cercano a la raíz buscada.

De cara a este proyecto, el siguiente paso es adaptar la ecuación anterior a unos *solvers* ya creados, que van a solucionar todo el sistema de ecuaciones. Estos *solvers* actúan sobre sistemas de ecuaciones de la forma

$$A \cdot x = B$$

donde A y B son matrices. Por lo tanto, hay que adaptar la ecuación de la siguiente iteración a la que es aceptada por el *solver*. Los pasos a seguir son

$$x_{i+1} + K^{-1} \cdot F_{ext} = x_0$$

$$K \cdot x_{i+1} + F_{ext} = K \cdot x_0$$

$$K \cdot x_{i+1} = K \cdot x_0 - F_{ext}$$

con este cambio en la ecuación, podemos diferenciar las dos matrices como

$$A = K$$

$$B = K \cdot x_0 - F_{ext}$$

y obtendremos la solución en x del caso estático. La implementación a partir de este punto es muy sencilla, hay que crear un bucle (con un valor máximo para no entrar en un bucle infinito) y en cada iteración calcular el siguiente valor de la posición. Una vez hecho esto, se compara con el valor anterior para comprobar si el error es menor que Є (que define el límite de error permitido), y si se cumple esta condición parar el bucle.

3.3. Condiciones de Contorno

Una vez completado el cálculo del caso estático y explicadas las bases del proyecto, en los siguientes puntos es necesario explicar cómo disponer las condiciones de contorno para poder calcular la deformación final.

Una vez el objeto ha colisionado, es necesario definir la malla de tetraedros y el sistema de elementos finitos que se va a encargar de dar la solución a la deformación del objeto. Lo primero es crear una malla de tetraedros basada en el objeto elegido y asignar el módulo de Young y el coeficiente de Poisson para poder calcular las matrices internas y conocer la matriz de rigidez total. Una vez hecho esto se obtienen las fuerzas de colisión del objeto y el punto donde se han de aplicar. Si la fuerza es mayor que un límite (para evitar fuerzas muy pequeñas) pasamos a crear los nodos, vértices y aristas de la malla final, se asignan los vecinos y se asigna una masa a cada nodo en función de la masa total y el volumen del objeto.

A la hora de calcular el caso estático es necesario que al menos tres nodos se encuentren fijos porque se necesita esta condición para poder hallar la solución. Para ello se utiliza un método en el que tras la colisión del objeto, se traza un radio a partir del punto de colisión fijando todos los tetraedros que se encuentren fuera de la circunferencia formada por dicho radio. Los tetraedros fijados no sufrirán la deformación causada por el impacto por lo que la elección del radio es importante para permitir que el objeto se deforme todo lo necesario pero siempre existan al menos tres nodos fijos.

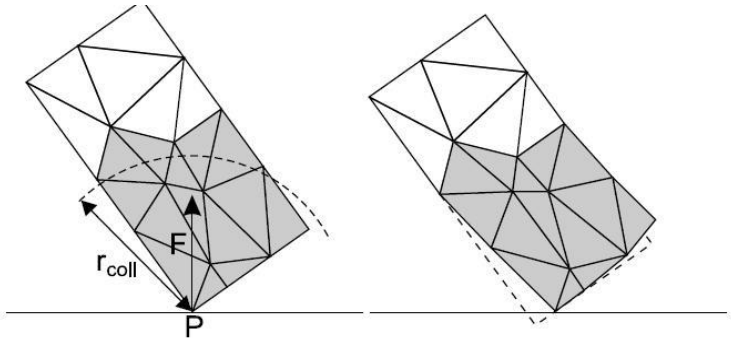

Ilustración 18. Solo los tetraedros dentro del radio rcoll se van a deformar por la colisión en el punto P. El resto de tetraedros se mantendrán fijos. Müller et al. (20)

Una vez hecha esta operación se aplica la fuerza sobre todos los nodos libres para deformarlos, resolviendo así el caso estático y obteniendo la deformación final.

3.4. Cálculo de Energía

El siguiente paso, una vez calculada la deformación total sobre el objeto es el cálculo de las fuerzas internas que van a encaminar la fractura. Todos los valores que se definen en este apartado, como son la energía de deformación y fractura, son necesarios para el algoritmo de fractura definido en el apartado 4.

El algoritmo consistiría en calcular la deformación (ϵ) y el esfuerzo (σ) correspondientes a cada tetraedro dentro del cuerpo tras la deformación. Tras esto se calcula la "densidad de energía de deformación" o *strain energy density* como

$$E_d = \frac{1}{2} \cdot \sigma \cdot \epsilon$$

Una vez calculada la densidad, se calcula la energía de deformación o *Strain Energy* total. Esta energía viene definida por

$$E_S = \sum_t (E_d \cdot V_t)$$

donde E_d y V_t son la densidad y el volumen de cada tetraedro.

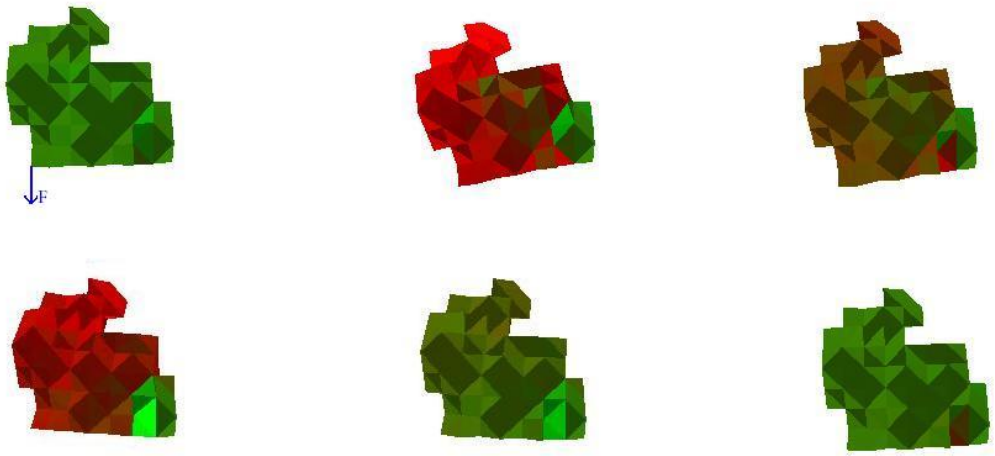

Ilustración 19. En esta figura se refleja la *strain density* sobre los tetraedros al aplicar una fuerza hacia abajo. Cuanto mayor predomina el color rojo, mayor densidad, y mayor posibilidad de que aparezcan puntos de fractura. También se pueden observar cómo los tetraedros más a la derecha están fijados y nunca se deforman.

4. Algoritmo de Fractura

4.1. Motivación

Este punto es el más importante dentro de todo el proyecto, ya que se explica cómo se va a formar y desarrollar el algoritmo de fractura que va a producir la partición de los objetos en diferentes piezas para simularlas y obtener el resultado final.

Tras una explicación de las bases y artículos que han dado base al algoritmo, se va a hacer una exposición general de todo el proceso de fractura de los objetos. Una vez explicada esta parte se expondrán con mayor amplitud los puntos más importantes dentro del algoritmo, como el cálculo de los diagramas de Voronoi que van a definir la fractura o la partición de los tetraedros. Para terminar, es necesario introducir las nuevas piezas dentro de los motores de físicas y de gráficos, por lo que se comentarán los aspectos más importantes.

4.2. Bases de la Fractura avanzada

Tras la explicación de las bases del proyecto respecto a los algoritmos de fractura, en este apartado se va a hablar brevemente del algoritmo propuesto por Zheng (2) para el cálculo de las nuevas piezas creadas por dicha fractura. Este proyecto se ha basado en gran medida en esta parte y en el apartado de creación de piezas se explicarán mejor algunas cuestiones. Sin embargo una pequeña aproximación al trabajo inicial que se expone en el artículo es conveniente.

El método que propone Zheng se basa en el uso de la energía interna que se genera al producirse la deformación (sección 3.4.). Haciendo uso de esta energía se va a calcular un número de trozos que formarán la fractura final. Otra particularidad de este algoritmo es que aunque puede basarse en un esquema de Voronoi, las piezas se van a crear en el momento y no va a existir ningún esquema anterior a la fractura (como sí se utiliza en el trabajo anterior de Bao (16)). En este caso se va a ir poco a poco incrementando el número de trozos hasta que se consiga superar una condición preestablecida.

Antes de nada, se necesita estimar la energía requerida para generar la fractura sobre una superficie con una determinada área.

$$E_F = G_C A_F$$

G_C define la dureza del material ante la fractura, que representa la habilidad del material para resistir la fractura, y A_F es el área de la fractura que está siendo creada. $G_C$es un valor externo que se puede modificar para comprobar diferentes materiales y cómo es su fractura. Para evitar una fractura muy grande la condición que se debe cumplir hace que la E_F consumida tenga que ser menor que la energía de deformación (del inglés *quasistatic strain energy*) multiplicada por un parámetro de disminución $\eta\ \epsilon(0,1)$. La condición que se debe cumplir para terminar la creación de piezas es

$$E_F \leq \eta\, E_S$$

En este proyecto el valor por defecto de η será 0.8, el mismo que utiliza Zheng en sus ejemplos (2). Una vez decidida esta condición, se procede a la generación de puntos sobre la malla de tetraedros. Estos puntos van a definir las nuevas piezas que se forman, por lo que se pueden llamar ”puntos de fractura”. Mediante el uso de diagramas de *Voronoi* se van a dividir los nodos de la malla inicial haciendo uso de los puntos calculados, partiendo la malla y creando nuevas piezas. Por cada nuevo punto de fractura colocado en la malla, la energía de fractura va a aumentar hasta el momento en el que se cumpla la condición anterior. Por lo tanto, la fractura final del objeto va a depender del número y la posición de los distintos puntos de fractura.

Para el cálculo de la colocación de los puntos sobre la malla, el algoritmo de *Zheng* utiliza patrones de fractura basados en diagramas de *Voronoi*. Una vez colocados, se usan distancias ponderadas para asignar los nodos a uno de estos puntos. Estas distancias ponderadas se calculan en función de la distancia del punto al nodo y de la energía de fractura propia del tetraedro que contenga dicho nodo. Con ello se consiguen crear pequeñas piezas con un alto grado de energía de deformación (2).

En cambio, en este proyecto la creación de los puntos y la asignación de los nodos se calcula de otra manera. La posición de los puntos va a depender de la energía de deformación de los tetraedros pero introduciendo una componente aleatoria. El resultado será que los tetraedros con mayor energía de deformación van a tener mayor cantidad de “puntos de fractura”, aunque estos se van a expandir por toda la malla gracias a la componente aleatoria. Una vez colocados los puntos, los nodos se asignarán en función de la distancia.

4.3. Pre-Proceso

Antes de comenzar la simulación de los objetos en el mundo virtual, es necesario hacer una fase de pre-proceso en la que se va a crear la malla de tetraedros y el sistema de elementos finitos. También se van a posicionar los diferentes objetos según los parámetros que establezca el usuario, en cuanto a posición de inicio y rotación. Este paso es necesario para establecer la dureza del material y conocer cuáles son los tetraedros existentes ante una malla de puntos que define el objeto inicial.

El mallado y creación de un sistema de elementos finitos suelen ser procesos costosos ya que van a depender del número de puntos inicial que tenga la malla. Además existe un parámetro que va a definir cómo de fina va a ser esta malla de tetraedros. Cuanto mayor sea dicho parámetro, más fina será la malla, con un mayor número de tetraedros y con un mayor coste de tiempo para la aplicación. Cada objeto del mundo virtual es independiente a la hora del cálculo de fuerzas internas, por lo tanto cada objeto tendrá su propia malla de tetraedros y un sistema de elementos finitos asociado, que se va a encargar del cálculo de la deformación y fuerzas internas de dicho objeto.

4.4. Cálculo de la fractura

Tras la fase anterior, el siguiente paso es comenzar con la simulación de los objetos en el mundo real. En este proyecto se está simulando el lanzamiento al vacio, por lo que la única fuerza que va a afectar a los objetos es la fuerza de la gravedad en el mundo físico virtual con lo que las características más importantes de cara a la fractura son la altura a la que se encuentre y el material del objeto.

La simulación continúa mientras no se produzcan colisiones. Existen dos tipos de objetos dentro de la simulación: los objetos fracturables y los objetos estáticos. Estos últimos nunca se van a romper pero van a ocasionar la fractura del resto de los objetos, como podría ser el suelo. Los objetos fracturables son aquellos que se van a romper cuando colisionen contra un objeto estático, u otro objeto fracturable.

Una vez se produzca una colisión, se va a comenzar con el cálculo de la deformación de los objetos fracturables. Para ello, primero se aplica la fuerza resultante de la colisión a los nodos de la malla que hayan chocado. Estos nodos sufrirán una deformación que se extiende por la malla del objeto. Como se ha explicado anteriormente (sección 3.3.) no todos los nodos van a sufrir la deformación, ya que se necesitan al menos tres nodos fijos. Una vez se haya cambiado la posición de los nodos en la malla, se calcula el caso estático (sección 3.2.) para conocer las fuerzas internas que se han creado dentro del objeto.

Una vez se conocen estas fuerzas, se procede al cálculo de las diferentes piezas que se van a crear. En la sección 4.2. ya se ha explicado cómo el cálculo de piezas termina siendo un bucle en el que, conforme se crean más piezas la energía de fractura va a incrementarse hasta el momento en el que se cumpla la condición de salida.

4.5. Fase de cálculo de nuevas piezas

Una vez conocidas las fuerzas internas que aparecen en el objeto tras la deformación, se pasa al cálculo de las piezas que van a aparecer al producirse la fractura. Para ello, se van a crear diferentes “puntos de fractura” sobre la malla de tetraedros. Estos puntos van a terminar definiendo las piezas que se forman mediante

un diagrama de Voronoi. Para ello, los nodos dentro de la malla van a ser asignados a los puntos de fractura según su proximidad.

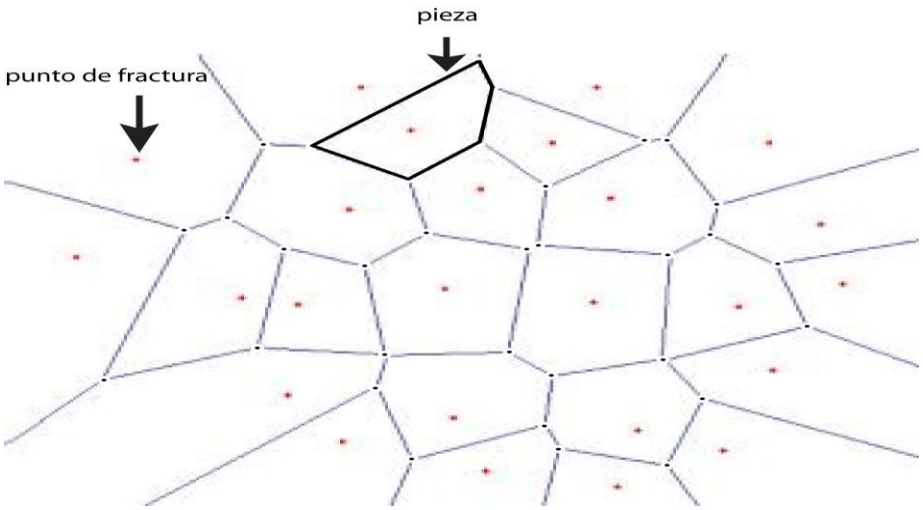

Ilustración 20. Diagrama de Voronoi y su utilidad de cara a la creación de las piezas.

Al tratarse de un bucle, el número de puntos va a crecer hasta el momento en el que se cumpla la condición. Por lo tanto, primero se colocan dos puntos y se establece cuales son las piezas, con ello se actualiza la energía de fractura y se comprueba si se cumple la condición de salida. Si no se cumple, se introduce un nuevo punto, se rehace el cálculo de piezas y se vuelve a comprobar la energía de fractura. Este proceso continúa hasta que se cumpla la condición, por lo que conforme crezca el número de puntos, aumentará el número de piezas nuevas y cada vez serán más pequeñas.

La colocación de los puntos sobre la malla es un proceso pseudo-aleatorio, ya que se va a intentar que la fractura no sea siempre igual, pero se va a seguir un criterio a la hora de colocar los diferentes puntos. Este criterio es la energía de deformación propia de los tetraedros de la malla. Por lo tanto, los tetraedros que estén sufriendo mayor deformación tendrán más posibilidad de tener puntos de fractura y acumular diferentes piezas. Con ello se consigue que la fractura tenga un comportamiento real, ya que las piezas se van a amontonar en las partes que mayor deformación estén sufriendo. En cambio, las partes de la malla menos deformadas van a tener un número menor de puntos de fractura, y las piezas serán más grandes.

De cara a obtener una fractura distinta en cada caso, se introduce una componente aleatoria para que las partes menos deformadas también tengan puntos de fractura, obteniendo así una fractura algo más caótica e imprevisible. En la sección 4.7. se explica con mayor profundidad cómo se va a crear esta aleatoriedad y cómo se van a calcular las nuevas piezas en función de los puntos de fracturas afectando a la energía de fractura del objeto.

4.6. Desarrollo y profundización del algoritmo: creación de "puntos de fractura"

En este apartado se desarrolla el proceso de creación y colocación de "puntos de fractura". Como ya se ha explicado, cada punto de fractura colocado en la malla va a definir una nueva pieza. Estos puntos van a ir aumentando de cara a que la energía de fractura también aumente. Con cada nuevo punto introducido, se van a crear nuevos triángulos que definen la nueva pieza.

Al tratarse de un proceso en el que se van introduciendo nuevos puntos con cada iteración del algoritmo, las piezas finales solo estarán totalmente definidas cuando se cumpla la condición de salida. En los casos anteriores, la introducción de un nuevo punto de fractura puede alterar la forma y el número de triángulos que forman las piezas cercanas al nuevo punto introducido.

Para calcular los triángulos que van a formar las nuevas piezas, es necesario dividir los tetraedros de la malla inicial en función de los puntos que se hayan introducido, como se explica en la sección 4.8. Conforme el número de nuevos triángulos va aumentando, lo hace el área de fractura, por lo que también lo hará la energía de fractura interna del objeto. Por lo tanto, el proceso a seguir es: calcular un nuevo punto de fractura, colocarlo en la malla, recalcular las nuevas piezas mediante la partición de tetraedros y creación de nuevos triángulos, calcular la nueva energía de fractura (según la fórmula $E_F = G_C A_F$) y comprobar si esta energía es lo suficientemente grande. Si no lo es, se vuelve a comenzar este proceso.

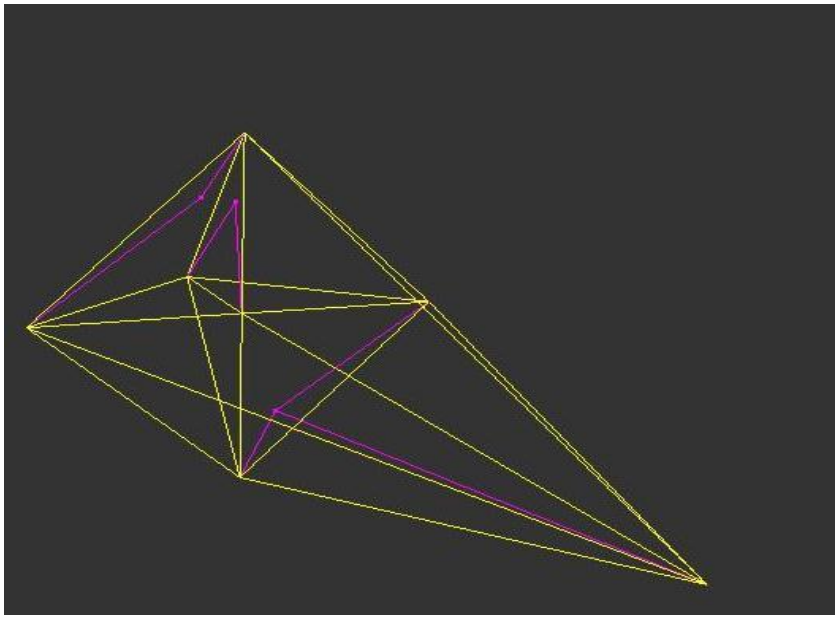

Ilustración 21. Cálculo de los puntos de fractura en un objeto al comienzo del proyecto.

Para conseguir que la fractura sea algo caótica, y cambie entre una simulación y otra, es necesario crear una aleatoriedad en la creación de los puntos de fractura. Para ello,

se va a hacer uso de la Energía de deformación de cada tetraedro y de una función de distribución (CDF) que permita elegir un tetraedro de forma pseudo-aleatoria.

Siempre va a tener un componente aleatorio pero se da más prioridad a los tetraedros con mayor E_d. El proceso que se sigue es el siguiente, primero se obtiene la suma de todas las E_d de los tetraedros. A partir de entonces se comienza desde cero a sumar la división entre la E_d propia de un tetraedro y la suma total. Este valor se suma a todos los anteriores formando una sucesión de valores en orden creciente entre cero y uno. Si un tetraedro tiene una E_d grande el intervalo entre la suma anterior y la posterior será mayor.

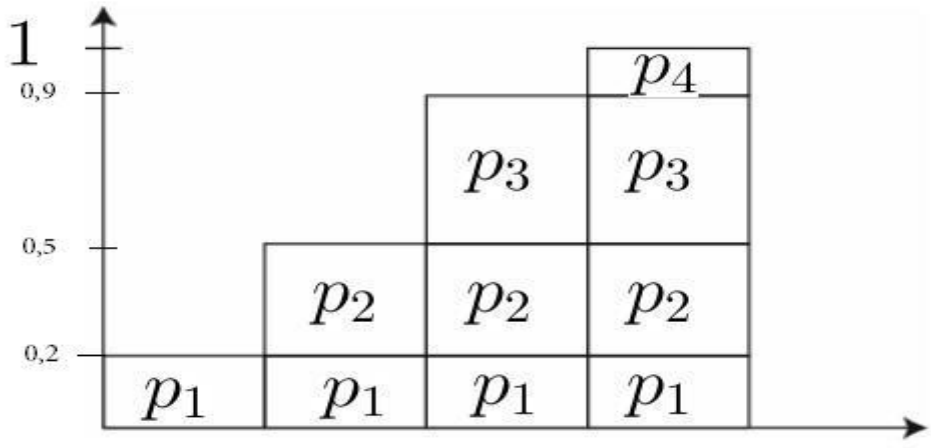

Ilustración 22. Tabla de valores de probabilidad según el CDF

Una vez calculada esta sucesión el proceso para seleccionar un tetraedro donde colocar un "punto de fractura" depende de un valor aleatorio entre cero y uno. Con este valor se comprueba en la sucesión en qué intervalo cae y ese será el tetraedro elegido. En la implementación solamente hay que comparar el valor aleatorio con los diferentes números de la sucesión y en cuanto el valor aleatorio sea menor, se escogerá el tetraedro anterior. Una vez elegido el tetraedro es necesario comprobar que no haya más de cuatro "puntos de fractura" asignados dentro de él, si este fuera el caso se volvería a elegir un tetraedro aleatorio. Con esta condición se evita crear piezas muy pequeñas en una parte del objeto y muy grandes en otro lado. Una vez seleccionado y comprobado que el "punto de fractura" se puede asignar, se calcula un punto aleatorio dentro del tetraedro escogido utilizando las coordenadas baricéntricas.

Por ejemplo, si se tienen una malla de cuatro tetraedros que ha sufrido una deformación. El tercer tetraedro sufre mayor deformación, tras el está el segundo, el primero y por último el cuatro tetraedro, que apenas ha sentido la colisión. De cara a colocar un punto en estos cuatro tetraedros, la energía de deformación sigue el mismo orden anterior. Por lo tanto, la gráfica anterior nos muestra cómo se van a acumular

las diferentes probabilidades en función de los tetraedros y normalizando todo el proceso. Una vez terminado este proceso, se coge un número aleatorio entre cero y uno. Dependiendo cuál sea el intervalo en el que se encuentre el número se escogerá un tetraedro u otro. Como se puede observar, el tetraedro con mayor posibilidad (por tener un intervalo de 0.4) es el tercer tetraedro, el que tenía mayor energía de deformación.

4.7. Partición de tetraedros

Una vez calculados los puntos de fractura y colocados dentro de la malla, se van a calcular los diferentes triángulos que van a formar las piezas en función de dichos puntos. Para ello, se necesita partir los tetraedros existentes, de tal manera que dependiendo de a qué punto pertenezcan los nodos de la malla, el tetraedro se podrá partir de cuatro formas diferentes, creando diferentes triángulos.

Tras colocar los diferentes puntos de fractura dentro del objeto inicial, se pasa a ver qué nodos van a pertenecer a cada uno de los puntos. Si en un tetraedro todos sus nodos pertenecen al mismo punto de fractura, no le ocurrirá nada. Sin embargo se pueden dar 4 situaciones diferentes donde uno, dos, tres o incluso los cuatro nodos pertenezcan a diferentes puntos de fractura. Para cada una de estas situaciones aparecerán diferentes decisiones que se toman para partir dicho tetraedro. Estas particiones definen nuevos triángulos y nodos por lo que es necesario tetraedralizar las piezas resultantes, ya que al crearse la fractura la topología va a cambiar.

El caso más sencillo es cuando sólo uno de los puntos se ha de separar en el tetraedro. Para este caso se crean dos nuevos triángulos formados por el punto medio de las tres aristas que unen el punto a separar de sus vecinos. Al hacer esta separación se van a introducir 6 nuevos nodos (tres por cada pieza separada) y dos nuevos triángulos.

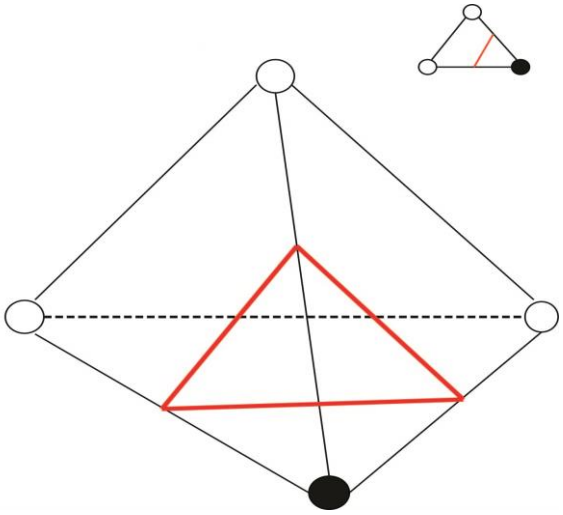

Ilustración 23. Separación de un nodo introduciendo dos nuevos triángulos.

El siguiente caso corresponde a la separación de dos nodos. En este caso los dos nodos que se van a separar pertenecen a un punto de fractura diferente a los restantes. La solución escogida se basa en partir el tetraedro por su mitad, utilizando los puntos medios de las diferentes aristas que unen los nodos a separar.

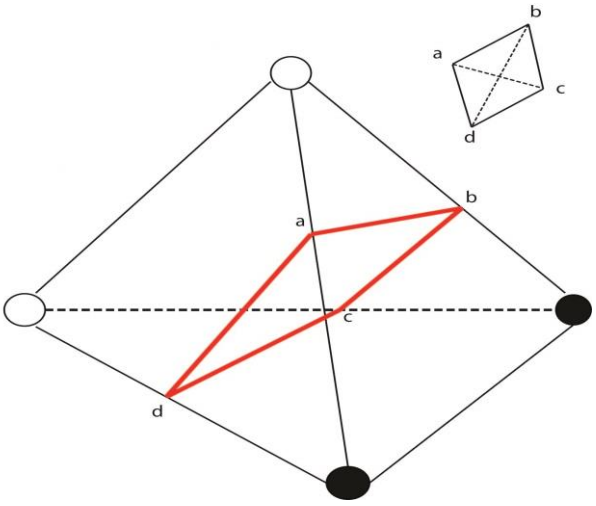

Ilustración 24. Separación de dos nodos partiendo por la mitad.

Con esta partición se crean dos nuevos triángulos por cada pieza (cuatro en total) y se introducen ocho nodos nuevos. En la imagen se muestra una de las posibles particiones (dos nodos a la derecha) pero la partición sería similar en cualquiera de los casos.

En el tercer caso se va a seguir un razonamiento similar a los casos anteriores y se puede ver la partición como el caso anterior pero dividiendo una de las mitades en dos nuevos trozos. De esta manera, por una parte quedaría la mitad del tetraedro formado por los dos nodos pertenecientes al mismo punto de fractura y se introducen nuevos puntos y nuevos triángulos para partir en dos el nuevo trozo, dejando el tetraedro partido en tres piezas

diferenciadas. En este caso el número de nuevos nodos sería de diez y seis nuevos triángulos.

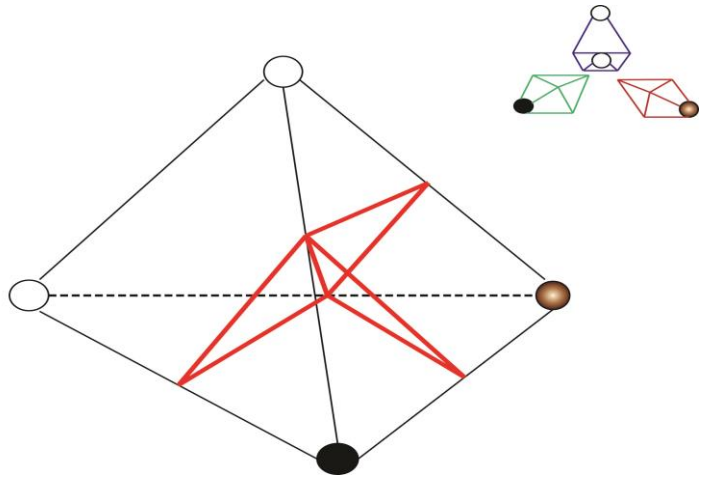

Ilustración 25. Separación en tres piezas diferentes

Para el último caso se crean cuatro nuevas piezas debido a que cada nodo está asignado a un punto de fractura diferente. Este caso es el más complicado de todos porque hay que calcular un *centroide* o punto medio dentro del tetraedro, que va a definir el punto de unión entre las cuatro nuevas piezas.

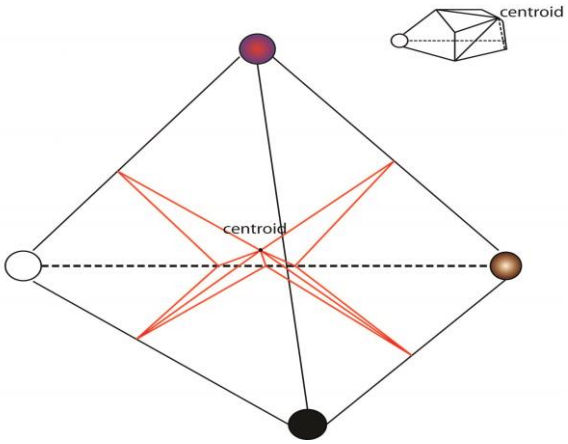

Ilustración 26. Separación en cuatro piezas diferentes.

De esta manera, a la hora de calcular las nuevas piezas que van a aparecer se van a formar hasta seis triángulos nuevos por cada nueva pieza. Estos triángulos se forman conectando el *centroide* con los puntos medios de las aristas que conectan al nodo con sus vecinos, así como el centro de cada uno de los triángulos que forma el nodo con sus dos vecinos. Con estos puntos se va formando una nueva pieza que será la que se introduzca como resultado final.

4.8. Integración con *Bullet* y OGRE

En este punto se explica cómo es el proceso mediante el que se integra el motor de físicas *Bullet* con todo lo anteriormente expuesto. Este motor es importante porque va a simular el movimiento de los objetos y va a proporcionar información sobre las colisiones que se producen.

Se ha hecho uso también del motor de gráficos OGRE, el cual va a permitir crear una pequeña interfaz gráfica así como dar mayor realismo a la simulación. Aun así, este motor también es totalmente independiente y se podría utilizar directamente OpenGL o incluso obtener datos en un fichero de texto sobre cómo se está produciendo la simulación.

4.8.1. Fase de introducción de pieza

Una vez las piezas han sido creadas y se ha cumplido la condición entre energías internas, se procede a introducir las nuevas piezas dentro del mundo físico virtual y simular su movimiento. Para ello, primero es necesario eliminar el objeto inicial, y colocar en su lugar las diferentes piezas que se han creado como si fueran sólidos rígidos independientes. Como ya se comentó anteriormente, en este proyecto se ha simulado solo la primera fractura, por lo que las nuevas piezas creadas no se van a volver a partir.

Antes de introducir las piezas en el mundo físico es necesario conocer su centro de masas para que la simulación sea correcta, ya que éste es diferente para cada pieza. Este proceso ocasionó algunas dificultades en el proyecto porque el centro de masas calculado por *Bullet* no era del todo correcto, y era necesario recalcularlo utilizando la caja contenedora (*Bounding Box*) propia de cada pieza. Una vez hecha esta corrección, simplemente hay que decidir la velocidad que tendrán las piezas tras la colisión para obtener resultados realistas.

En toda fractura, las diversas piezas que se forman tienden a dispersarse en función de las fuerzas que la han ocasionado. En la sección 4.8. se explica en mayor profundidad cómo se ha creado el cálculo de velocidades, pero el resultado final es recrear cómo las piezas se dispersan una vez ha existido una colisión y el objeto final se ha dividido.

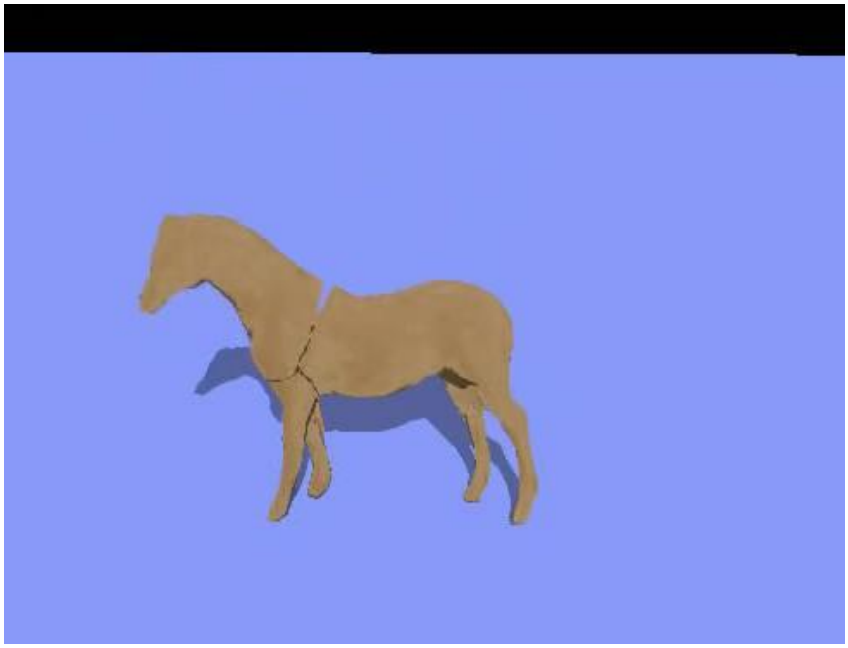

Ilustración 27. Fractura de un objeto.

Una vez se han calculado las piezas, se ensamblan todos los triángulos propios de cada pieza y se forma un sólido rígido. Tras esto, se calcula un nuevo tensor de inercia y una nueva masa para la pieza. La masa se calcula en función del volumen final de la pieza y la masa inicial de todo el objeto. Una vez establecidos los parámetros del nuevo sólido rígido, se vuelve a calcular su nuevo centro de masas (como ya se ha dicho en la sección 4.6.) y la velocidad inicial de cada pieza.

4.8.2. Velocidades de nuevas piezas

De cara a crear una simulación realista es importante que las piezas nuevas tengan una velocidad coherente con la colisión. En cuanto a este punto, se han barajado varias maneras de calcular esta velocidad sobre las piezas. Las primeras ideas se basaban en utilizar las propias velocidades calculadas por *Bullet* antes y después de producirse la colisión. De esta manera, en el primer caso se aplicaría una velocidad contraria a la que llevaba el objeto antes de colisionar o se podría aplicar la velocidad que tenía el objeto justo después de la colisión.

Estas dos velocidades daban buenos resultados aunque se ha querido probar una forma más avanzada que explica Zheng (2). Según este artículo las velocidades lineales y angulares post-fractura se definirían como

$$v_d^+ = v_d^- + \frac{\tau}{m_d} \sum_{triangle\ i} s_i$$

$$\omega_d^+ = \omega_d^- + \tau I_d^{-1} \sum_{triangle\ i} j_i$$

donde v_d y ω_d son la velocidad lineal y la angular respectivamente, m_d es la masa e I_d es una matriz de inercia de la pieza "d", τ define la duración efectiva que han ejercido las fuerzas de fractura. El siguiente paso es definir cómo afecta el estrés interno a cada nuevo triangulo calculando la fuerza (s_i) y el toque (j_i) que se ejerce

$$s_i = a_i P_i n_i;\ j_i = r_i \times a_i P_i n_i$$

Para este cálculo se necesita el área del triangulo (a_i), la normal (n_i) y el tensor que se esté utilizando dentro del método de elementos finitos (en el caso de este proyecto el tensor de Cauchy). El último paso es definir la ecuación que junte las distintas velocidades para despejar τ y poder así calcular las velocidades a posteriori. Según Zheng se utiliza la ecuación de Energía cinética, definida como

$$\Delta E_K = E_S - E_F$$

$$\Delta E_k = \frac{1}{2} \cdot \sum_{debris\ d} (m_d \cdot \|v_d^+\|^2 - m_d \cdot \|v_d^-\|^2 + (\omega_d^+)^T \cdot I_d \cdot \omega_d^+ - (\omega_d^-)^T \cdot I_d \cdot \omega_d^-)$$

Esta ecuación es cuadrática en τ por lo que sólo tendremos que coger la solución menor positiva. Una vez sustituidas las velocidades y simplificada la ecuación anterior, podemos obtener τ y con ella las velocidades a posteriori que se van a aplicar a las diferentes piezas. A pesar de que este proceso es mucho más complejo que utilizar las velocidades propias de *Bullet*, el resultado tiende a ser más realista porque la velocidad depende de las energías internas y no simplemente de la colisión.

5. Resultados

En este punto se van a exponer los resultados obtenidos una vez se ha conseguido implementar y ejecutar todo el algoritmo. También se explicarán varios problemas relacionados con el resultado final, así como una breve descripción de la creación de la interfaz de usuario que permite probar diferentes escenarios de una forma sencilla y rápida.

En este proyecto, se permite la fractura de cualquier tipo de objeto que se incluya dentro del mundo simulado. Como demostración, la fractura se va a producir siempre en función de la altura que se tenga el objeto y la fuerza que aplique la colisión. Por lo tanto, no se van a poder introducir fuerzas externas (exceptuando la gravedad) a los objetos sino que siempre se van a dejar caer al vacío los objetos. Sobre estos objetos se puede aplicar una rotación que hará que la fractura difiera ya que los puntos de contacto serán otros diferentes, y por lo tanto la deformación cambiará.

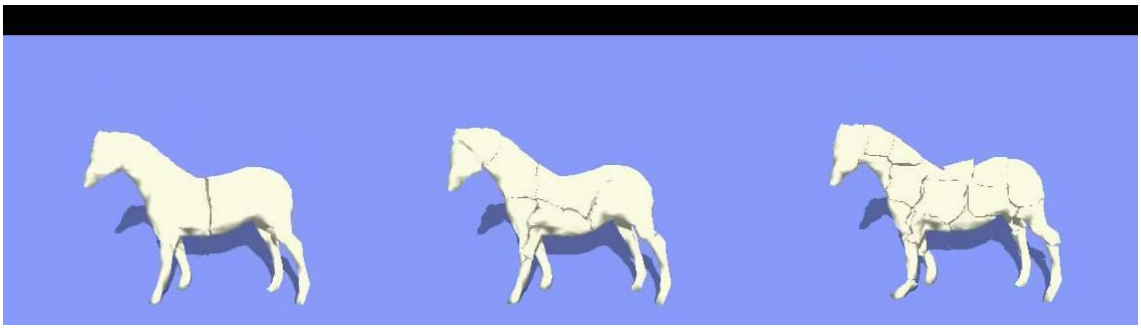

Ilustración 28. Diferentes fracturas del mismo objeto ante diferentes alturas y con el mismo material.

Además de la altura, los objetos cambiarán su fractura dependiendo del material del que estén compuestos. Más concretamente, la variación del parámetro $G_C$hará que la fractura aumente o disminuya, cambiando el comportamiento del objeto ante la colisión.

Como demostración en el contexto del proyecto, se ha decidido simular solo la primera colisión. Esto quiere decir que una vez un objeto se haya partido, los trozos que van a aparecer no van a volver a fracturarse. El proceso para calcular esta nueva fractura

sería análogo al algoritmo original. Con las simulaciones obtenidas se satisface el objetivo principal del proyecto.

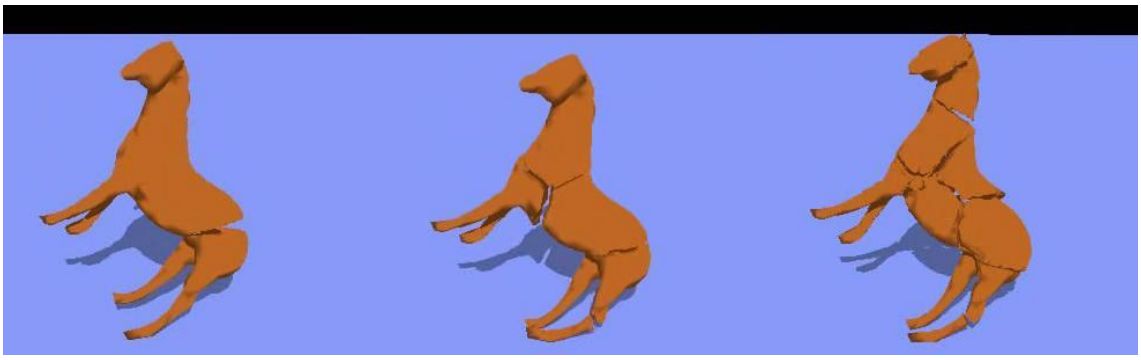

Ilustración 29. Diferentes fracturas ante misma altura con diferentes materiales

Otra prueba creada, es lanzar varios objetos y ver cómo colisionan entre sí, de cara a comprobar si las colisiones se calculan de forma correcta y los objetos que aún no hayan sufrido una fractura se parten al colisionar con cualquier otro objeto. Cómo se comprueba en la siguiente ilustración, este comportamiento está correctamente implementado.

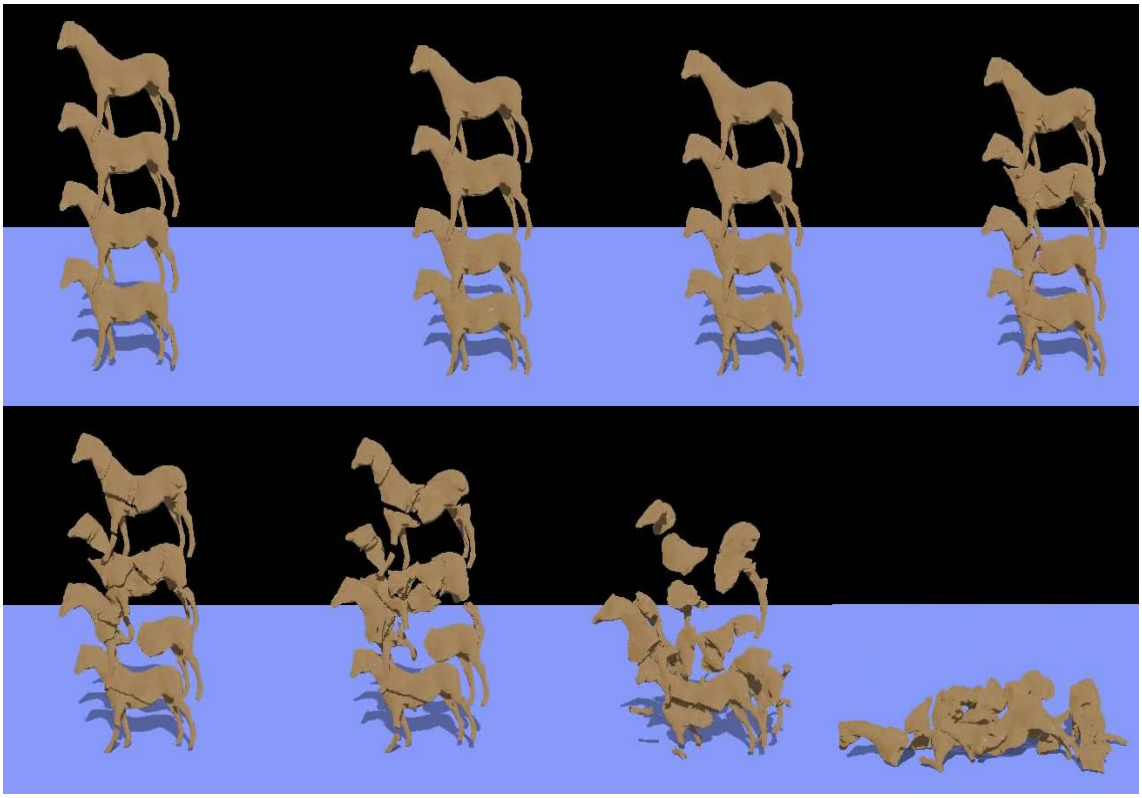

Ilustración 30. Lanzamiento de múltiples objetos para comprobar colisiones.

Para poder medir y comprobar el comportamiento del algoritmo implementado, se han hecho una batería de pruebas en las que se espera ver si los resultados obtenidos son los esperados y, por lo tanto, el algoritmo se comporta de una manera

correcta.Estas pruebas consisten en varios lanzamientos consecutivos desde diferentes alturas y con diferentes materiales, para comprobar cuál es el número de piezas obtenido y cuanto tiempo se tarda en calcular la fractura.

En las primeras pruebas, se ha lanzado un objeto desde alturas cada vez mayores (desde 1 hasta 10 metros) utilizando la misma dureza. Con esto se puede comprobar cómo afecta la altura al número de piezas y al tiempo de fractura. Antes de analizar las gráficas, es necesario remarcar que el algoritmo implementando siempre va a partir el objeto. Esto es importante ya que el número de piezas mínimo siempre va a ser dos, y no se contempla que un objeto choque y no se parta.

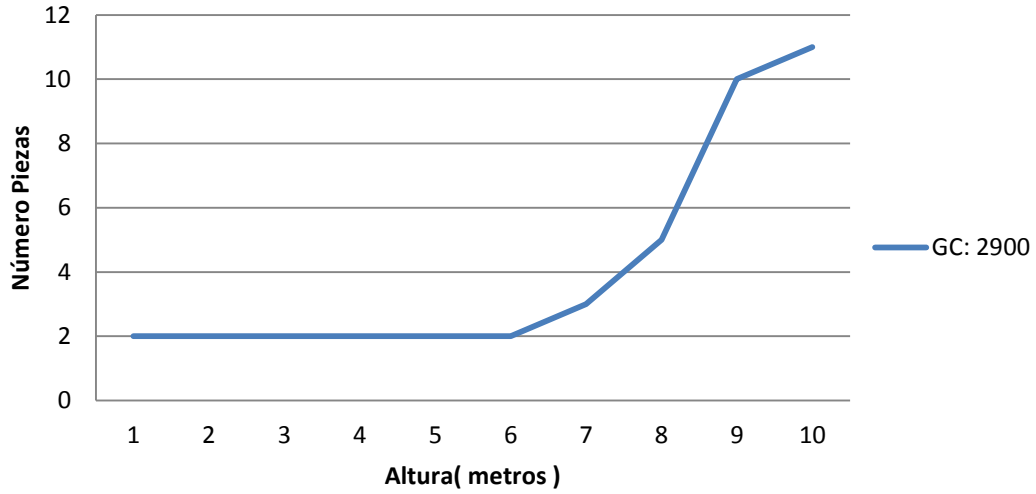

Ilustración 31. Gráfica que muestra cómo crece el número de piezas en función de la altura desde la que se arroja el objeto.

Como se puede ver en la gráfica, al principio el objeto es muy duro, por lo que se parte solo en dos piezas. A partir de los 7 metros es cuando se empieza a incrementar el número de piezas que aparecen. Desde ese momento cuanto más alto se tire el objeto, en más trozos se va a partir. Se puede observar también como, en función de la dureza del material, el número de piezas dependiente de la altura va a crecer bastante, así conforme más alto se suelte un objeto débil mayor será el número de piezas que se va a formar, obteniendo un comportamiento esperado.

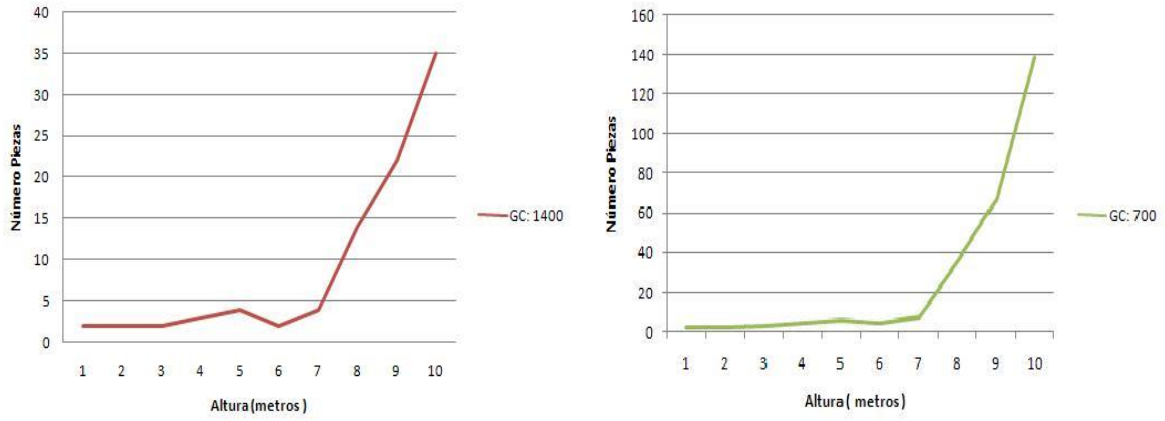

Ilustración 32. Diferentes gráficas sobre la relación Nº de piezas/ Altura con diferentes durezas.

Para obtener más datos sobre cómo afecta la dureza del material a la fractura, se ha creado otra batería de pruebas en las cuales los objetos siempre se lanzan desde una misma altura. Estas alturas van a ir creciendo cada vez más, en este caso se han escogido tres valores cinco, ocho y doce metros. Lo que variará por tanto es la dureza del material, empezando por un material duro como el hormigón (con un Gc de 2000), para terminan con materiales muy frágiles como el cristal. El resultado esperado es que conforme disminuya la dureza sea cual sea la altura el objeto tendría que incrementar el número de piezas en las que el objeto se fractura. Otro comportamiento esperado es comprobar cómo, cuanto mayor es la altura más piezas aparecen.

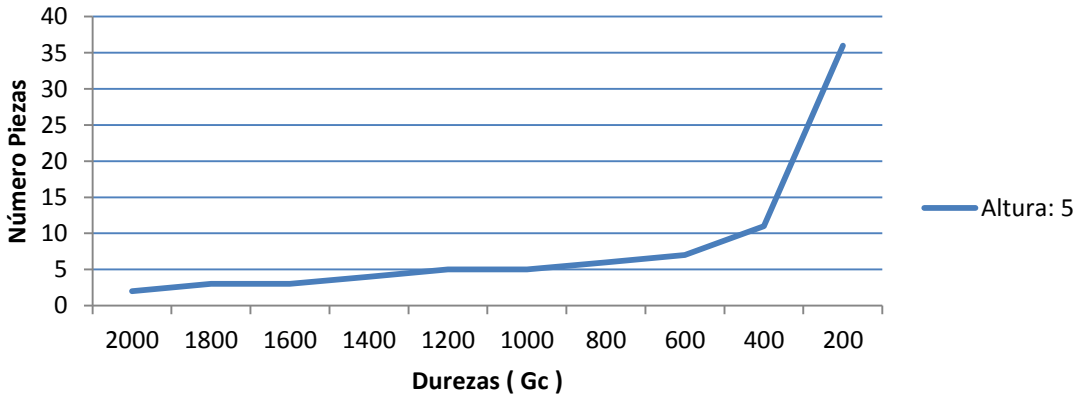

Ilustración 33. Gráfica relación Número de piezas /Dureza con una altura de lanzamiento de 5 metros

En el primer caso, con una altura relativamente baja de cinco metros, se comprueba un crecimiento progresivo del número de piezas conforme la dureza va disminuyendo, hasta llegar un punto en el que disminuir la dureza incrementa mucho la fractura del objeto (al pasar de un Gc de 400) cuando se pasa de obtener unas 10 piezas a pasar a casi 40 piezas.

Este comportamiento se da con las otras dos alturas, aunque el incremento de la altura hace que el número de piezas crezca muchísimo más. Por ejemplo, desde una altura de 8 metros se puede observar el mismo comportamiento. Se comienza con un crecimiento regular, aunque se están creando más piezas que en el caso anterior, para en un punto crecer en gran medida, hasta pasar de unas 5 piezas, hasta casi 400.

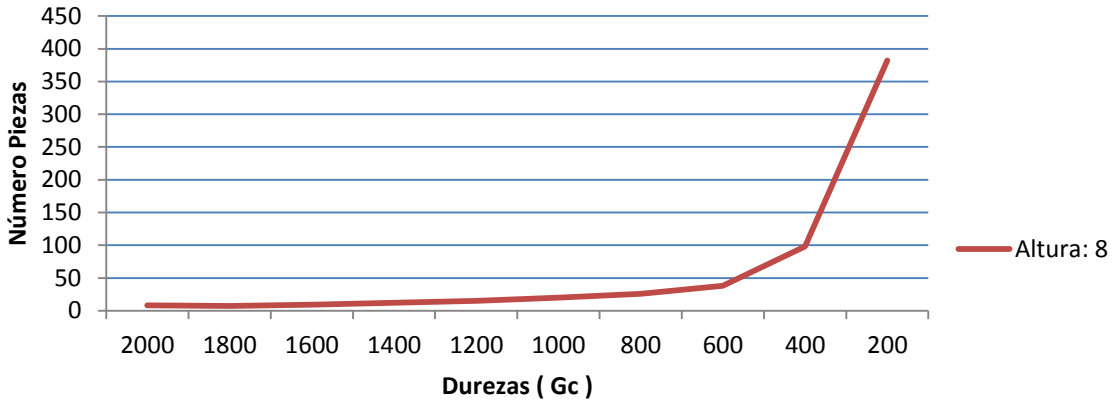

Ilustración 34. Gráfica relación Número de piezas /Dureza con una altura de 8 metros.

Este comportamiento es el mismo lanzando los objetos desde una gran altura como es doce metros, aunque en este punto los tiempos y números de piezas se disparan, hasta el punto de crearse más de mil piezas con un Gc de 600. Por lo tanto, el comportamiento de los objetos en función de los parámetros básicos de altura y dureza en cuanto al número de piezas creadas es el esperado.

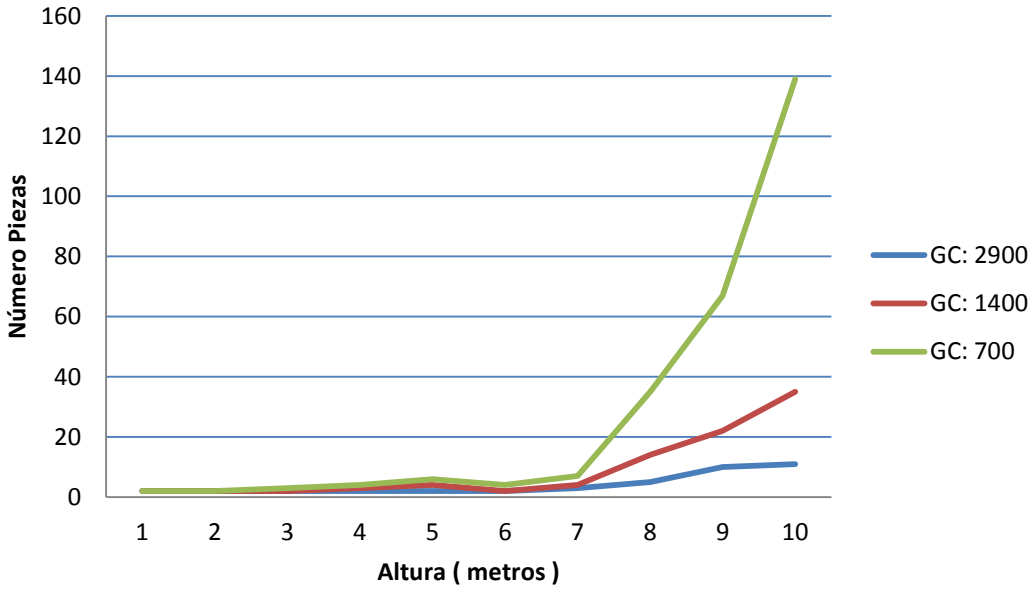

Ilustración 35. Comparación del número de piezas creado en función de la altura para tres durezas diferentes.

Otro punto que se quiere comprobar es cómo afectan la altura y dureza al tiempo que se tarda en calcular la fractura. Estas pruebas son necesarias para comprobar cuánto tarda el algoritmo implementado y comprobar si se tiene algún punto crítico por el que el tiempo se incremente. Las pruebas han sido similares a las anteriores, calculando tiempos en segundos en función de la altura a la que se lanza el objeto y de la dureza del mismo. Según la implementación del algoritmo, el resultado esperado es que conforme aumente el número de piezas creadas se tarde más en calcular la fractura total. Por lo tanto, cuanto mayor sea la altura o menor la dureza del material, mayor será el tiempo de cálculo.

El tiempo es un parámetro crítico, ya que aunque en este proyecto no se espera alcanzar simulaciones en tiempo real siempre es importante obtener unos tiempos buenos y cuanto menores mejor. Una vez ejecutadas las pruebas, se comprueba cómo los tiempos para fracturas con pocas piezas son bastante bajos (se podría decir que más o menos se tarda un segundo por pieza) y que conforme las fracturas son mayores y las piezas aumentan el tiempo de fractura aumenta con ellas.

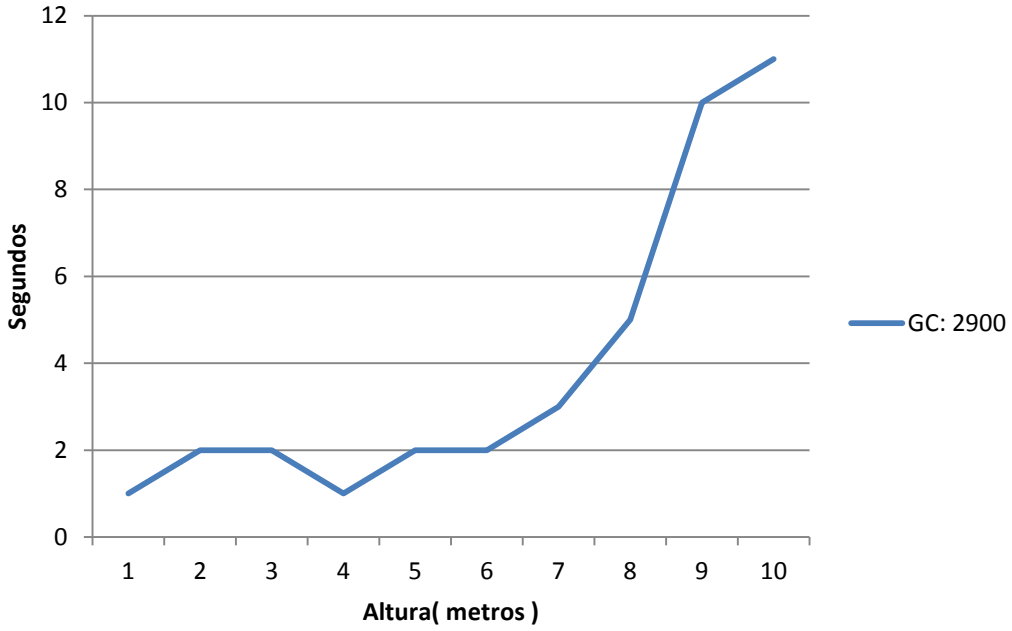

Ilustración 36. Gráfica relación tiempo/altura.

Todas las gráficas terminan siendo similares solo que conforme la dureza va siendo menor, los tiempos crecen con el número de piezas. Aun así, para fracturas básicas en las que el número de piezas no crece mucho las simulaciones obtenidas dan muy buenos resultados, ya que los tiempos son muy bajos y no se notan parones entre la colisión del objeto y su fractura. En casos más extremos, donde los objetos se dividen en muchas piezas diferentes, los tiempos crecen bastante, aunque siempre de un modo casi constante con el número de piezas creadas.

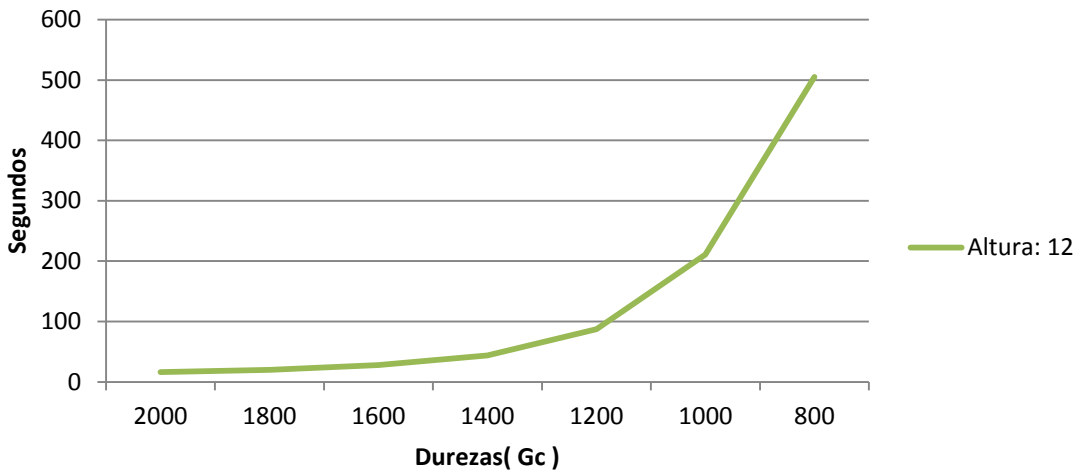

Ilustración 37. Gráfica relación Tiempo/Dureza con una gran altura de lanzamiento.

Por lo tanto, se puede concluir de manera bastante fiable que el comportamiento del algoritmo es predecible, en cuanto a que el cambio de la altura y de la dureza del material va a crear mayor número de piezas y por lo tanto el tiempo de fractura va a aumentar. Estos resultados también son esperados de cara a lo que ocurre en la realidad por que cualquier objeto se partirá en más trozos cuanto mayor sea la altura desde la que se lanza, y el mismo objeto tendrá diferentes comportamientos ante la fractura dependiendo de su dureza.

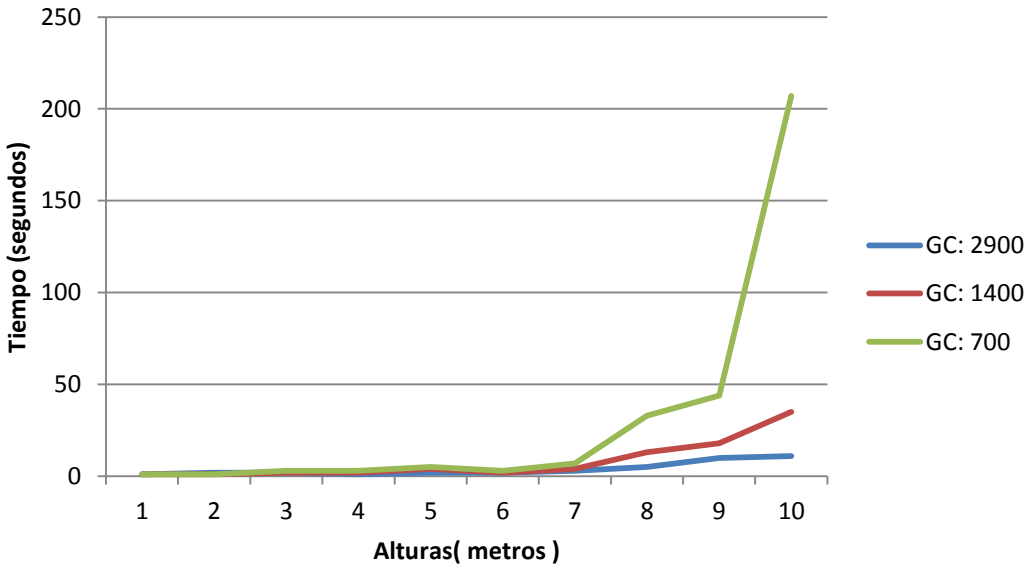

Ilustración 38. Gráfica que compara los tiempos de fractura del mismo objeto con diferentes durezas, lanzado desde varias alturas.

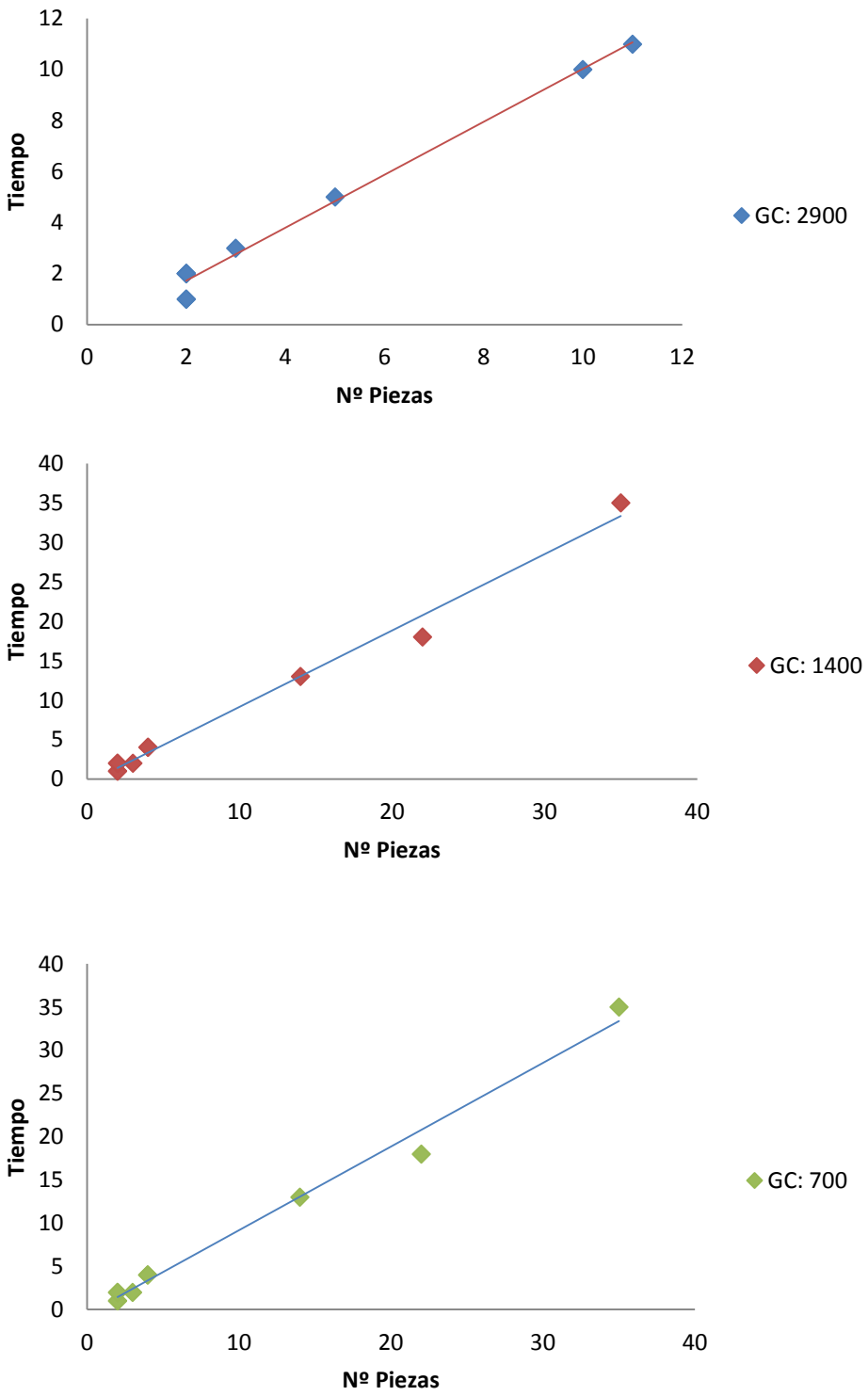

Ilustración 39. En estas tres gráficas se observa la dispersión de los diferentes valores conforme se lanzan los objetos desde alturas mayores.

Por último, en las gráficas anteriores se muestra la dispersión de los diferentes valores y la relación entre el tiempo de fractura y el número de piezas que se crean. Como se puede observar, el crecimiento es estable y cuanto más alto es el número de piezas, mayor es el tiempo requerido para el cálculo de la fractura.

Otra parte a comentar dentro de la aplicación es la Interfaz de usuario. Con ella se permite a cualquier persona comenzar simulaciones sobre diferentes objetos. Esta interfaz permite de una forma sencilla crear multitud de simulaciones, con uno o varios objetos y con diferentes posiciones y tipos de material. A pesar de no ser una parte importante dentro del proyecto ya que su uso se limita a crear nuevas simulaciones sin tener que pasar los diferentes parámetros por línea de comandos, facilita en gran la creación y colocación de objetos a fracturar para usuarios que no conozcan como se utiliza la aplicación.

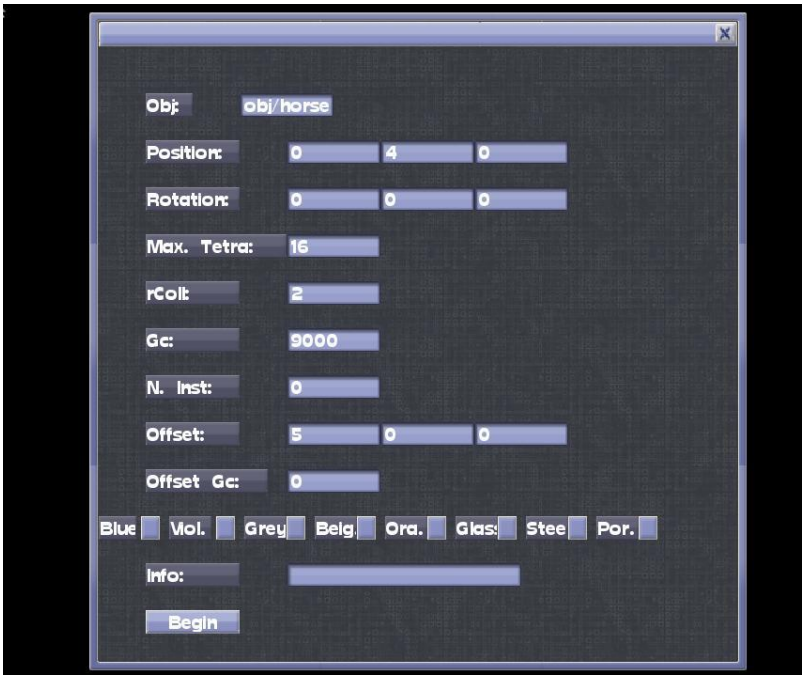

Ilustración 40. Interfaz de usuario con todos los datos necesarios.

6. Conclusión y Líneas futuras

Como último punto, en este apartado se hablará sobre las conclusiones observadas sobre el algoritmo, cuáles son sus puntos fuertes y cuales sus limitaciones, expresadas como líneas futuras de cara a mejorar el proyecto.

Una vez obtenidos los resultados y analizado el algoritmo, se puede concluir que este proyecto cumple con el objetivo principal: simular la fractura físicamente correcta de objetos lanzados al vacio. Esta fractura se puede producir en uno o varios objetos lanzados en un mundo virtual, de tal manera que al ocasionarse una colisión, se calcule cómo se va a producir la fractura, cuáles son las nuevas piezas que van a aparecer y cómo se van a simular estas nuevas piezas.

Uno de los aportes clave en este proyecto es el hecho de que los objetos no necesitan estar pre-fracturados y por lo tanto el cálculo de los trozos se producirá una vez el objeto comience a romperse. Con ello se consigue un mayor realismo a la hora de simular las fracturas, evitando repetición de piezas y permitiendo a los usuarios crear su propia fractura simplemente utilizando dos parámetros: la altura y la dureza del objeto. Hay que tener en cuenta que el algoritmo de fractura utiliza fuerzas internas del objeto, por lo tanto la fractura que se produzca va a ser físicamente realista, sean cuales sean los parámetros introducidos por los usuarios.

La mayor complejidad dentro del algoritmo radica por lo tanto, en el cálculo de las fuerzas internas y las diferentes piezas que se van a crear a la hora de fracturar el objeto. Es en este cálculo donde más optimizaciones se pueden aplicar en el futuro, de cara a conseguir simulaciones más rápidas, especialmente a la hora de simular comportamientos complejos donde la fractura crea un gran número de piezas. Aún así, una vez examinados los resultados del punto anterior, se puede comprobar que para casos normales, donde los objetos se partan en no más de 10 piezas, la simulación es suficientemente rápida como para que los usuarios apenas noten que se está realizando un cálculo intenso a la hora de crear la fractura.

Aún así, de cara al futuro se pueden intentar optimizar algunos aspectos como por ejemplo:

- **Permitir la fractura plástica**. De este tipo de fractura se ha hablado en algunas partes de la memoria y es aquella que además de fracturar el objeto, lo deforma de manera constante. Este comportamiento se aprecia bien en objetos como una viga de metal, la cual antes de partirse se deforma, y una vez fracturada, la deformación se mantiene constante. Para este tipo de simulación, es necesario incluir un grado de plasticidad a los materiales de los objetos. También es necesario modificar el cálculo de fuerzas internas ya que se debe permitir la deformación antes de la fractura.
- **Lanzamiento de objetos con una fuerza inicial.** En el algoritmo desarrollado solo se ha implementado el lanzamiento al vacio de objetos sin aplicación de fuerzas externas. Esto causa que en todas las simulaciones la fuerza de colisión sea contraria a la de la gravedad. Una posible mejora sería permitir lanzamientos de objetos con una determinada dirección al aplicárseles una fuerza inicial. Para ello sería necesario modificar algunos aspectos como las fuerzas que se producen al colisionar y cómo se aplican a los objetos, ya que estas fuerzas podrían tener cualquier dirección.
- **Permitir la fractura de las nuevas piezas formadas en un caso anterior**. Con esto se conseguiría una fractura más realista ya que si un objeto se parte antes de tocar el suelo y sus piezas siguen cayendo, se podría volver a partir dichas piezas y mejorar así el realismo final de la simulación.
- **Incluir la fractura progresiva**. Este punto no se ha desarrollado aunque muchos motores están incluyéndolo últimamente. Se trataría de permitir la aparición de grietas sobre los objetos. La fractura progresiva no se forma en el instante en el que existe una colisión, sino que el usuario puede ver cómo progresa sobre los objetos y apreciará la formación de grietas.
- **Mejora de los tiempos de cálculo**. En el caso de objetos frágiles lanzados desde una gran altura, el cálculo de piezas ralentiza bastante la simulación, por lo que cualquier optimización en este proceso siempre va a beneficiar al proyecto.

Bibliografía

1. *Interactive virtual materials.* **Müller, Matthias and Gross, Markus.** [ed.] Canadian Human-Computer Communications Society. London, Ontario, Canada : s.n., 2004, Proceedings of Graphics Interface 2004, pp. 239,246.

2. *Rigid-body fracture sound with precomputed soundbanks.* **Zheng, Changxi and James, Doug L.** 2010, ACM Trans. Graph., pp. 69:1,13.

3. *Graphical Modeling and Animation of Brittle Fracture.* **Hodgins, James F. O'Brien and Jessica K.** s.l. : ACM Press/Addison-Wesley Publishing, August 1999, Computer Graphics Proceedings, Annual Conference Series, pp. 137–146.

4. *Graphical Modeling and Animation of Ductile Fracture.* **James F. O'Brien, Adam W. Bargteil, and Jessica K. Hodgins.** s.l. : ACM Press, August 2002, Proceedings of ACM SIGGRAPH 2002, pp. 291–294.

5. **NVidia.** APEX Destruction. [Online] http://developer.nvidia.com/apex-destruction.

6. *Deformable models.* **Fleische, D. Terzopoulos and K.** 1988, The Visual Computer, pp. 4:306–331.

7. *Modeling inelastic deformation: Viscoelasticity, plasticity, fracture.* **Fleischer, D. Terzopoulos and K.** 1988, Computer Graphics (SIGGRAPH '88 Proceedings),, pp. 269,278.

8. *Regularization of inverse visual problems involving.* **Terzopoulos, D.** July 1986., IEEE Transactions on Pattern Analysis, pp. 8(4):413–424.

9. *Animation of fracture by physical modeling.* **A. Norton, G. Turk, B. Bacon, J. Gerth, and P. Sweeney.** 1991, The Visual Computer, pp. 7:210–217.

10. *Animating exploding objects.* **O. Mazarak, C. Martins, and J. Amanatides.** June 1999, Graphics Interface '99.

11. *Generation of crack patterns with a physical model.* **K. Hirota, Y. Tanoue, and T. Kaneko.** 1998, The Visual Computer, pp. 14:126,137.

12. *A visual model for blast waves and fracture.* **Fiume, M. Neff and E.** 1999, Graphics Interface '99, p. June.

13. **Anderson, T. L.** *Fracture Mechanics: Fundamentals and Applications.* s.l. : CRC Press, Boca Raton, second edition, 1995.

14. *Interactive skeleton-driven dynamic deformations.* **S. Capell, S. Green, B. Curless, T. Duchamp, and Z. Popovic.** 2002, Proceedings of ACM SIGGRAPH, pp. 586,593.

15. *Stable real-time deformations.* **M. Müller, J. Dorsey, L. McMillan, R. Jagnow, and B. Cutler.** 2002, Proceedings of ACM SIGGRAPH Symposium on Computer Animation, pp. 49,54.

16. *Fracturing Rigid Materials.* **Bao, Zhaosheng and Hong, Jeong-Mo and Teran, Joseph and Fedkiw, Ronald.** 2007, IEEE Transactions on Visualization and Computer Graphics, pp. 370,378.

17. **Pixelux.** DMM Engine and DMM Plugin. [Online] http://www.pixelux.com/.

18. **Havok.** Havok Destruction. [Online] http://www.havok.com/products/destruction.

19. **Bullet.** Maya Dynamica Plugin. [Online] http://bulletphysics.org/mediawiki-1.5.8/index.php/Maya_Dynamica_Plugin.

20. *Real-Time Simulation of Deformation and Fracture of Stiff Materials.* **Matthias Müller, Leonard McMillan,Julie Dorsey and Robert Jagnow.** s.l. : EUROGRAPHICS 2001 Computer Animation and Simulation Workshop, 2001, EUROGRAPHICS 2001 Computer Animation and Simulation Workshop, pp. 99,111.