

Universidad Central de las Villas
Facultad de Matemática, Física y Computación
Carrera de ciencias de la computación



Plataforma para métodos de partículas y sin mallas

Autor: Alcides Viamontes Esquivel

Tutores:

Dr. Carlos Recarey Morfa

Dr. Daniel Gálvez Lío

2006

A mis padres

*A todos mis maestros, por
el infinito tesoro de su formación.*

NoMS

Plataforma para métodos de partículas y sin mallas.

Trabajo de Tesis de Pregrado.
Alcides Viamontes Esquivel

Tutores:

Carlos Recarey Morfa
Daniel Gálvez Lío

Resumen:

Se presenta un marco computacional para la implementación de algoritmos de simulación basados en partículas y métodos sin mallas. Se hace fuerte hincapié en los predicados de geometría computacional y las estructuras de consulta espacial necesarias. También se describe el sistema de herramientas de software utilizados, en cuya elección se tomó en cuenta el aspecto de las licencias.

Abstract:

A computational framework for developing simulation algorithms for particle and meshless methods is presented. Strong emphasis is done in computational geometry predicates and spatial query data structures. The set of programming tools is described; in their selection the licenses issue was taken into account.

Tabla de contenidos.

| | |
|---|----|
| Introducción..... | 7 |
| Capítulo 1 Información y estado del arte..... | 9 |
| 1.1 Hacia un nuevo sistema de simulación para la mecánica computacional..... | 9 |
| 1.1.1 Revisión del software existente para el método de partículas..... | 9 |
| 1.1.2 Revisión del software existente sobre métodos sin mallas..... | 11 |
| 1.1.3 Requisitos y Metas..... | 11 |
| 1.2 Cuestiones teóricas..... | 11 |
| 1.2.1 Modelación de medios..... | 11 |
| Modelación continua de materiales..... | 12 |
| Importancia de la modelación discreta de materiales..... | 12 |
| 1.2.2 El problema de la generación inicial del medio..... | 14 |
| Método de partículas..... | 14 |
| Método de puntos..... | 14 |
| 1.2.3 Sobre la naturaleza de los cálculos en la simulación..... | 14 |
| Esquema de cálculo en los métodos sin mallas..... | 14 |
| Esquema de cálculo en los métodos de partículas..... | 15 |
| Integración en el tiempo..... | 16 |
| Método de Euler..... | 16 |
| Método Newmark-Beta..... | 16 |
| Esquema de Verlet (diferencia finita central)..... | 16 |
| Aceleración media constante..... | 17 |
| 1.2.4 Aspectos físicos..... | 17 |
| Ecuaciones del movimiento e integración explícita..... | 17 |
| Lógica de conglomerados..... | 18 |
| Los métodos sin mallas y el modelo físico..... | 19 |
| 1.3 Revisión de algoritmos y bibliotecas existentes..... | 19 |
| 1.3.1 El problema de las licencias..... | 19 |
| 1.3.2 Geometría computacional..... | 20 |
| 1.3.3 Estructuras de consulta espacial..... | 21 |
| 1.3.4 Números aleatorios..... | 22 |
| 1.3.5 Infraestructura. Rutinas numéricas..... | 23 |
| Métodos de optimización..... | 24 |
| Bibliotecas e implementaciones para cuestiones de matemática numérica básica..... | 24 |
| 1.4 Herramientas de construcción de software..... | 25 |
| 1.4.1 Lenguaje de programación..... | 25 |
| 1.4.2 Sistema de control de versiones..... | 26 |
| 1.5 Conclusiones parciales..... | 26 |
| Capítulo 2 Detalles algorítmicos..... | 28 |
| 2.1 Organización..... | 28 |
| 2.2 Estructuras y rutinas geométricas básicas..... | 29 |
| 2.2.1 El punto..... | 29 |
| 2.2.2 El vector..... | 30 |
| 2.2.3 Operaciones entre puntos y vectores..... | 30 |
| 2.2.4 Recta..... | 30 |
| 2.2.5 El rayo..... | 31 |
| 2.2.6 Segmento..... | 31 |

| | |
|--|----|
| 2.2.7 El plano..... | 32 |
| 2.2.8 El triángulo..... | 33 |
| 2.2.9 Paralelepípedo y caja..... | 34 |
| 2.3 Estructuras de consulta espacial..... | 35 |
| 2.3.1 Estructura de consulta por tabla de dispersión..... | 35 |
| 2.3.2 Estructura de consulta de intervalo-intervalo por Kdtree..... | 38 |
| Kdtree genérico..... | 38 |
| Kdtree en NoMS..... | 40 |
| 2.4 Predicado de interioridad para mallas..... | 41 |
| 2.5 Conclusiones parciales..... | 43 |
| Capítulo 3 Notas sobre implementación..... | 44 |
| 3.1 Organización del capítulo..... | 45 |
| 3.2 Software sin ataduras..... | 45 |
| 3.3 Lenguaje de programación, C++..... | 45 |
| 3.4 Bibliotecas empleadas..... | 46 |
| 3.4.1 Boost..... | 46 |
| 3.4.2 Gsl..... | 46 |
| 3.4.3 Base de datos empotradas. Sqlite..... | 46 |
| 3.5 Herramientas de construcción de software..... | 47 |
| 3.5.1 Autotools y Scons..... | 47 |
| 3.5.2 Kdevelop | 49 |
| 3.5.3 Python..... | 49 |
| 3.5.4 Doxygen..... | 49 |
| 3.5.5 Subversion..... | 50 |
| 3.6 Otras herramientas, auxiliares o satélites..... | 50 |
| 3.6.1 Python..... | 50 |
| 3.6.2 Povray..... | 50 |
| 3.7 Decisiones de diseño..... | 50 |
| 3.7.1 Manejo de memoria..... | 50 |
| 3.7.2 Persistencia..... | 51 |
| 3.7.3 Interfaz con lenguaje script..... | 51 |
| 3.8 Estructura física del proyecto..... | 52 |
| 3.8.1 Lista de directorios..... | 52 |
| 3.9 Conclusiones parciales..... | 53 |
| Capítulo 4 Una aplicación: inicialización para el método de partículas y el de puntos..... | 54 |
| 4.1 Nota introductoria..... | 54 |
| 4.2 El problema..... | 54 |
| 4.3 Idea general..... | 56 |
| 4.4 Primera etapa: particionamiento en celdas..... | 56 |
| 4.5 Segunda etapa: lluvia local..... | 58 |
| 4.6 Última etapa: eliminación de las esferas en el exterior del dominio y de los huecos..... | 60 |
| 4.7 Resultados..... | 61 |
| 4.7.1 Tamaño de celda fijo..... | 64 |
| 4.7.2 Tamaño de celda variable..... | 64 |
| 4.7.3 Variando el multiplicador gravitatorio..... | 64 |
| 4.8 Conclusiones parciales..... | 64 |
| Conclusiones..... | 65 |
| Recomendaciones..... | 66 |

| | |
|-------------------|----|
| Bibliografia..... | 67 |
|-------------------|----|

Introducción

La *Ley de Moore* se ha dejado sentir en las últimas décadas, con un potente impacto en el mundo científico y tecnológico. Puede achacársele a la ingeniería moderna el cumplimiento casi cronométrico de esta ley. Pero también ha sido la mayor necesitada de la duplicación regular del poder de cómputo.

Los ingenieros de hoy cuentan con una herramienta colosal que ha puesto más altas sus posibilidades, y también sus aspiraciones. Si al comienzo del siglo XX se formulaban tímidamente métodos que podrían usar el esquema de diferencias finitas a una escala mayor, y se concluía el canal de Panamá, a comienzos del siglo XXI los investigadores son jalonados por las exigencias de problemas industriales y civiles cada vez más complejos, el método de elementos finitos se ha vuelto una herramienta tremendamente habitual, y hay quienes ponen las metas tecnológicas de la humanidad en un puente que une dos continentes sobre el estrecho de *Bering*.

Por estos días con mayor frecuencia el método de elementos finitos se queda estrecho para muchos tipos de problemas. Por ejemplo, aquellos que implican grandes cambios geométricos o deformaciones del modelo de análisis, o intentan predecir el comportamiento de medios con un fuerte grado de discontinuidad inherente. Existen nuevos métodos, pero no se encuentran tan firmemente establecidos. El axioma que sirve de punto de partida a este trabajo es simple: con más esfuerzo de investigación y desarrollo, estos métodos podrían llegar a ser tan ampliamente conocidos y empleados como el método de elementos finitos. En particular, son de interés aquí dos de estos

esquemas: los conocidos como métodos libres de mallas, y los métodos de partículas.

Estos métodos tienen, al menos en cuanto a geometría computacional, fundamentos comunes. De ahí que la **hipótesis** de este trabajo es la siguiente: es posible contribuir al desarrollo de los métodos libres de mallas y de partículas con un software que sirva de plataforma a ambos.

El **objetivo general** es construir un software que sirva de base de partida en la experimentación sobre estos métodos y su aplicación final. Para lograrlo, se plantean los siguientes **objetivos específicos**:

- Identificar las necesidades comunes de ambos métodos, de forma que puedan tener lugar abstracciones de diseño de software. Estas permitirían reusar paradigmas e implementaciones, economizando así esfuerzo de implementación.
- Implementar la base común correspondiente a la matemática computacional necesaria para estos métodos.

De acuerdo a estos objetivos, se plantean las **tareas** a realizar:

- Indagar en la naturaleza física y matemática de estos métodos.
- Establecer mapas que relacionan de un lado conceptos e ideas generales con aspectos y técnicas propias de las ciencias de la computación del otro.

- Revisar y seleccionar el conjunto de bibliotecas y herramientas de software necesarias para emprender este proceso.
- Crear una implementación con los sistemas básicos necesarios para estos métodos.
- Validar la base de software creada con una aplicación de prueba.

Las tareas planteadas deben resultar en un software con los siguientes **requisitos**:

- Debe abarcar los aspectos básicos del trabajo con geometría computacional y la matemática numérica necesaria.
- Las implementaciones deben ser seguras, robustas y usables. Desde el punto de vista de la teoría de la complejidadⁱ y de la ingeniería de software, deben ser correctas.
- El producto final debe servir de apoyo seguro no solo a sistemas de explotación, sino también a la experimentación.

Con el fin de cumplimentar estos objetivos, durante la realización de esta tesis se emprendió una fase de búsqueda de información, la cual permitió constatar que hay considerablemente menos conocimiento en el área de los métodos de partículas y sin mallas, en comparación con el volumen de literatura existente para el método de elementos finitos; y que en efecto la esfera de trabajo tiene pocos caminos establecidos.

Como **novedad científica** en este trabajo pueden señalarse algunos de los algoritmos que se usaron

para implementar predicados de geometría computacional y las variaciones que se hicieron a una estructura de datos conocida, el *kdtree*.

El resultado hasta hoy es una herramienta de investigación, denominada NoMS por las primeras letras de *Non Meshing Simulation* que está siendo usada para diseñar y comprobar nuevos métodos numéricos en el campo de la simulación del comportamiento de materiales continuos y discontinuos. También se espera que se puedan resolver algunos problemas de utilidad práctica directa en el futuro.

Esta tesis está dividida en cuatro capítulos, el primero dedicado a abundar en detalles sobre los métodos a los que servirá de plataforma este trabajo. En ese capítulo también se repasan brevemente los resultados de las investigaciones bibliográficas en distintos aspectos de la geometría computacional, y se revisan las posibles herramientas de software con las que se podría construir un sistema de matemática computacional aplicada. El capítulo 2 está formado por la descripción de los algoritmos implementados en materia de geometría computacional y consultas espaciales; también habla sobre las estructuras de datos y los algoritmos básicos necesarios para manejar mallas de superficie. El tercer capítulo documenta las decisiones que se hicieron en cuestión de ingeniería e infraestructura de software. El capítulo 4 describe una aplicación particular del sistema. El objetivo de incluir dicha aplicación en esta tesis es demostrar la funcionalidad del software.

i Teoría de la complejidad: se trata el término en el sentido de la teoría de sistemas.

Capítulo 1 Información y estado del arte

1.1 *Hacia un nuevo sistema de simulación para la mecánica computacional.*

1.1.1 Revisión del software existente para el método de partículas.

En los métodos de partículas, los resultados y formulaciones teóricas son recientes, y no ha habido tiempo para llevarlos a productos ampliamente adoptados; por otra parte existe la fuerte competencia de los métodos de elementos finitos, bien conocidos y con un sinnúmero de implementaciones computacionales. Otra cosa que pone en desventaja a los métodos de partículas es el hecho de que inherentemente son más costosos computacionalmente. La gran ventaja que tienen es que funcionan con éxito en condiciones donde los métodos clásicos (entiéndase método de elementos finitos, diferencias finitas, elementos de contorno, etc.) son, cuando menos, engorrosos.

Hasta donde conocemos existen al menos dos software que en la actualidad implementan simulación con método de partículas. Uno de ellos es un código comercial, por lo que puede atribuírsele un estatus de completitud bastante avanzado. Se trata de los PFC2D y PFC3D de la compañía HcItasca, radicada en Canadá (ITASCA, 2006)ⁱⁱ. Dicho software tiene una larga lista de rasgos positivos, que se reproduce a continuación por cuanto da una idea de cuales deben ser las características de un software de este tipo:

- Movimiento dinámico e interacción de ensamblajes de partículas de tamaño arbi-

trarios.

- Propiedades a nivel de partícula individual, permitiendo gradaciones continuas de característica.
- Cálculo de doble precisión. No es sin duda la última palabra en cuestión de exactitud, pero es el mejor compromiso para cálculos masivos
- Los siguientes modelos físicos de contacto: muelles lineales, deslizamiento de Coulomb y acotación por contacto o paralelo.
- Lógica de agregación para el trabajo con grupos de partículas, lo que permite simular formas geométricas más generales
- Acepta paredes de compuestos: mallas, polígonos, etc.
- Acepta paredes a base de formas geométricas generales
- Cálculo automático del paso de tiempo.
- La complejidad temporal es lineal con respecto al número de partículas
- Posibilidad de cambiar dinámicamente las condiciones de contorno durante la simulación.
- Amortiguamiento local viscoso y no viscoso.
- Posibilidad de rastrear el comportamiento de la energía del cuerpo, el trabajo reali-

zado, el total de energía de enlace, el trabajo por fricción, la cantidad de energía cinética y la energía por el esfuerzo del cuerpo.

- Medida de algunos de los parámetros anteriores por región.
- Posibilidad de seguir en el tiempo cualquier magnitud, y plotado.
- Modo cuasi-estático de operación, para converger más rápidamente a la solución en condiciones de poco movimiento donde se busca el estado de equilibrio.
- Lenguaje de programación interno y propietario (FISH), que provee poderosas posibilidades de análisis.

El precio de la licencia de usuario es de 5500 dólares, y no corre en paralelo.

El otro software que se conoce es un código interno del Centro Internacional de Métodos Numéricos en Ingeniería (CIMNE), escrito en Fortran por Jurek Rojek. Desgraciadamente no es tan flexible como el PFC2D y el PFC3D, no tenía manual de usuario, y no es nada ameno para ser paralelizado.

El otro problema que aparece de punto común en el uso del método de partículas es la configuración inicial. En la literatura se habla extensamente de este problema (Lohner, 1998) y de distintas soluciones que se han propuesto. Ninguna es suficientemente buena. No fue posible encontrar un algoritmo suficientemente robusto para solucionar el problema de la configuración inicial.

Una de las conclusiones a las que se llega en el estudio del método de partículas es que aún su variante más ligera es inherentemente costosa. El proceso de determinación de contactos (que en el mejor de los casos comprende unas mil instrucciones del procesador), por ejemplo, debe ejecutarse dos de cada tres veces para cada partícula del modelo (un modelo de dimensiones limitadas puede tener cien mil (100000) partículas) en cada iteración, y el número de iteraciones puede ser de varios miles. Por otra parte, los logros que hay hasta el momento en

complejidad computacional no pueden mejorarse: las simulaciones suelen correr en el mejor caso, en una cantidad de operaciones linealmente proporcional a la cantidad de partículas

Ante este escenario, se necesitan enfoques en otras direcciones que permitan expandir la aplicabilidad de los métodos de partículas. Existen dos corrientes. La primera simplemente le da más poder de cómputo al algoritmo mediante el uso de técnicas de programación paralela. La programación paralela implica sus propios retos, y cualquier software que haya de paralelizarse debe ser concebido con esa intención desde el principio. La segunda forma de incrementar la aplicabilidad de las simulaciones basadas en partículas es mezclándola con métodos convencionales y menos costosos. Por ejemplo, se han hecho intentos de usar método de partículas con método de elementos finitosⁱⁱⁱ.

Gethin, Ransing, Lewis, Dutko y Crook consideraron una combinación del método de los elementos finitos con el método de los elementos distintos para simular la compactación de un sistema de partículas. Cada partícula se modela mediante elementos finitos y las interacciones entre partículas se modelaron mediante la técnica de los elementos distintos. Este método es equivalente al modelo de Gurson con análisis en medio continuo ya que los resultados obtenidos a partir de los dos análisis se corroboran. El trabajo demuestra la potencial aplicación de este modelo de elementos distintos deformables al caso de la compactación de materiales granulares en general y de polvo en particular (Recarey, 2003).

Puede conocerse más de este tema en (Mujinza, 1999), pero los resultados no parecen conducir a un fin práctico por el momento. En este frente todavía es necesario trabajo teórico y desarrollar algoritmos de geometría computacional que apoyen el proceso.

En esto de aplicar métodos mixtos, una vía menos explorada consiste en unir los métodos de partículas con los métodos sin mallas. Ambos son distintos, pero existe más afinidad entre método de partículas y método sin mallas en cuanto a infraestructura necesaria (estructuras de consulta de rango), que método de partículas y método de

elementos finitos.

En lo adelante, cualquiera de los tres métodos serán nombrados por sus abreviaturas. El método de elementos finitos será conocido como FEM, el de partículas o método de elementos discretos como DEM, y los métodos sin mallas como MFree.

1.1.2 Revisión del software existente sobre métodos sin mallas

El software más conocido en cuestión de métodos sin mallas es el que aparece referenciado en (Liu, 2002) por el profesor G.R. Liu, Doctor en Ciencias en la Universidad Nacional de Singapur, conocido como Mfree2D.

Otro código (comercial) que incluye MFree es el *Abaqus* 6.5 (ABAQUS, 2006). En este caso se trata de un software apto para fines de producción.

1.1.3 Requisitos y Metas

El software de *Itasca* sirve como referencia en cuanto a lo que puede lograrse en método de partículas. Sin embargo, el alto precio de su licencia lo hacen prohibitivo para fines de producción, no hablar de fines de investigación.

Algunos de los rasgos deseables en un software de este tipo son:

- Hibridizar, con el fin de obtener mayor eficiencia, el método de elementos finitos y el método de partículas. Por ejemplo, en las regiones “suaves” usar un modelo basado en la asunción de continuidad del material, y en las regiones de discontinuidad el método de partículas.
- En el caso del DEM, el uso de un formalismo para definir nuevos modelos constitutivos de contacto por personal sin amplios conocimientos de programación permitiría extender el esfuerzo de investigación entorno a este método. La inclusión de esta característica no solo tiene implicaciones en la ingeniería del software, sino también en aspectos teóricos. En

efecto, sería necesario encontrar vías confiables de ajustar el paso de integración temporal de forma que para un modelo arbitrario los resultados se mantuviesen estables.

- La posibilidad de conocer la incidencia de las estructuras de consulta de rango empleadas en cada caso. Esto tiene que ver con el hecho de que la teoría más elaborada en cuestión de estructuras de consulta de rango tiene que ver con la realización estática de las mismas, donde se asume que la posición del cuerpo, una vez dentro de la estructura, no cambia; se ha trabajado poco en estructuras altamente dinámicas, al menos sobre el campo de las aplicaciones geométricas.
- La instrumentación del paralelismo durante la fase de inicialización y simulación, lo que permitiría escalar el volumen de las simulaciones.

1.2 Cuestiones teóricas

1.2.1 Modelación de medios

Tradicionalmente, la ciencia de los materiales estudia el comportamiento de los mismos a diferentes escalas, con el objetivo de observar y cuantificar los procesos químico-físicos a niveles micro-mecánicos, moleculares y atómicos. La modelación multi-escala del material se encarga de representar estos procesos hasta el nivel microscópico.

La mayoría de las propiedades de los materiales se obtienen a partir de pruebas de laboratorio que se realizan a una escala cómoda, “real”: la macro escala, un nivel en el cual los materiales se pueden considerar homogéneos por haber regularizado el efecto de los elementos micro-estructurales en dichas propiedades “efectivas”.

Por otra parte, procesos de degradación progresiva solo pueden ser explicados considerando características micro-estructurales del material. De ahí la necesidad de caracterizar cada constituyente

y las condiciones de enlace en las interfases.

El proceso de modelación debe alcanzar cierto balance entre profundidad y practicabilidad. Es decir, debe ser lo suficientemente sofisticado para reflejar con exactitud la situación física, de forma que se puedan hacer inferencias válidas. Por otra parte, debe ser lo suficientemente sencillo como para que los cálculos asociados a la solución del modelo sean factibles. Es necesario, en todos los casos, validar el modelo y comprobar que sea adecuado al problema que se quiere resolver.

Modelación continua de materiales

A grandes escalas, el modelo que se realiza de un fenómeno suele descartar la existencia concreta y la singularidad de sus micro-componentes. De hecho, una de las abstracciones que suele introducir el modelo matemático es cierta afirmación implícita sobre la suavidad y la continuidad de los parámetros de interés.

Este enfoque se aplica con éxito en ciertos problemas de mecánica estructural, problemas de fluidos y de conducción del calor.

Importancia de la modelación discreta de materiales

Las técnicas del Método de Elementos Discretos son las formulaciones que mejor describen el comportamiento de los medios discontinuos. La génesis de estas técnicas fue desarrollada por Cundall. La tendencia actual de estas técnicas (Método de los elementos discretos o distintos) esta enfocada a posibilitar el estudio del micro-mundo y la esencia microestructural de los problemas de la mecánica computacional empleando modelos de partículas y micro-estructurales.

Entre las aplicaciones del Método de Elementos Discretos se encuentran:

- Realizar estudios de seguridad con un alto grado de precisión y fiabilidad.
- Prever fallos estructurales severos, al poderse detectar fallos a nivel microestructural (aparición de microfisuras y grietas, su

evolución y concatenación).

- Estudiar el estado tenso-deformacional existente entre partículas y detectar la aparición de microfisuras, las cuales son la génesis de la formación de micro zonas de falla que al final de su evolución y desarrollo culminan en los fallos estructurales o geotécnicos de una estructura u obra de ingeniería.

Estos tres aspectos tienen gran importancia en cualquier trabajo de ingeniería. Las formulaciones con método de partículas actualmente están en boga y se están utilizando con gran efectividad en estudios de problemas de geomecánica, mecánica de suelos y rocas, problemas de transporte, predicción de desgaste de herramientas de corte, estudio de fracturas en elementos estructurales y solución de problemas de fluido-dinámica.

Esta tesis no pretende profundizar en la utilidad de estos métodos, sino en el diseño computacional de un sistema de este tipo. A los interesados en conocer la utilidad del método, los siguientes fragmentos de una investigación previa le serán de utilidad:

“Alonso, y Gilli introducen, en un sistema de partículas distintas, fases líquidas y gaseosas para poder analizar el comportamiento de un suelo limo-arenoso con un contenido bajo de agua. Mediante el modelo, se predicen los cambios en cuanto al contenido de agua y su distribución así como el tamaño de los poros, las fuerzas de contacto y las deformaciones dadas para un estado dado de saturación, de arreglo geométrico y de fuerzas externas aplicadas.

“Las partículas sólidas se modelaron como esferas cuasi-rígidas. También fueron modelados los poros y el menisco de agua así como la transferencia de agua entre estas entidades. Algunos aspectos básicos de la modelación mecánica se tomaron del trabajo de Cundall y Strack.

“Tang-Tat. y Petrakis modelan un suelo granular a partir un arreglo tridimensional aleatorio de esferas de cuarzo mediante el uso del DEM. La simulación tridimensional realizada con una presión baja incluye la determinación del modulo de confinamiento a nivel microscópico para

arreglos de esferas cargadas isotropicamente y anisotropicamente. Los resultados están en adecuación con los ensayos triaxiales llevados a cabo en la Universidad de Texas. Se halló que un cambio de distribución de las fuerzas de contacto juega un papel muy importante en aquel fenómeno.

“Otsu, Mori, y Osakada presentan el uso del método de los elementos distintos para simular el comportamiento microscópico en el proceso de fundición en estado pastoso. En esta simulación, se tiene en cuenta la presión de la fase líquida dividiendo el sistema en varios elementos tetraédricos generados por la conexión de los centroides de los elementos distintos esféricos usando la triangulación Delaunay. Cuando una pieza en estado pastoso se deforma plásticamente, la fase líquida fluye entre las partículas. Esto trae como consecuencia que la fracción volumétrica de fase sólida no sea homogénea. Como que el flujo de la fase líquida es influido por la distribución de la presión en la pieza, se introdujo el efecto de esta en el método de los elementos distintos.

De geomecánica:

“Bagi, Bojtar, y Galos estudiaron la resistencia al fallo a compresión uniaxial de suelos granulares. Los resultados de los ensayos de laboratorio, generalmente, presentan una gran dispersión que suele prestarse a los pequeños errores aleatorios que se producen a la hora de realizar los experimentos. En este trabajo se supervisó este problema con un enfoque micro estructural para demostrar que la aleatoriedad geométrica de la micro estructura tiene un papel decisivo en cuanto a las diferencias entre los resultados de pruebas individuales

“Favier y otros autores presentan un nuevo método de representación de partículas no esféricas de superficie lisa con simetría axial. Las partículas son compuestas de esferas de diferentes tamaños que se solapan y cuyos centros son fijados en posición relativa a lo largo del eje de simetría. La detección de contactos así como la determinación de las fuerzas y deformaciones se lleva a cabo mediante el uso de técnicas estándar de elementos distintos

integrando el comportamiento de cada esfera al de la partícula a la cual pertenece.

“Kuo-Neng, Chang y Meegoda desarrollaron un programa de cómputo denominado ASBAL. Este programa tiene implementado un modelo basado en el DEM y se desarrolló a partir del programa TRUBAL para simular una mezcla asfáltica en caliente. En este caso, el modelo constitutivo emplea varios elementos visco elásticos (elementos de Maxwell, Kelvin-Voigt, y Burger) obteniendo mejores resultados con el elemento lineal visco elástico de Burger. Las simulaciones realizadas con ASBAL se aproximan satisfactoriamente con los resultados experimentales.

“Rojek, y otros autores presentan un modelo numérico para rocas, suelo y medios granulares usando elementos distintos esféricos. La integración explícita en el tiempo presenta una alta eficiencia computacional. El modelo constitutivo empleado en la interfase de contacto entre las esferas toma en cuenta las fuerzas de cohesión que permiten modelar la fractura y la descohesión de los materiales. El permite entonces describir el comportamiento de materiales granulares así como de los materiales sólidos caracterizados por un fallo frágil. Ejemplos verifican la veracidad del algoritmo desarrollado por estos autores pertenecientes al Centro Internacional de Métodos Numéricos en la Ingeniería (CIMNE).

“En publicaciones más recientes, se emplea el DEM para modelar el proceso de fundición mediante el uso de piezas de poliespuma (Lost Foam Casting). La simulación numérica predice los defectos del molde debidos a una compactación insuficiente de la arena alrededor de la pieza de poliespuma. (Recarey, 2003)

Este último es un ejemplo de uso combinado de los métodos:

“El uso combinado del DEM para modelizar las partículas de arena y del MEF para modelar la pieza de poliespuma permite detectar una posible distorsión de la pieza durante el llenado del molde y la compactación. (Recarey, 2003).

1.2.2 El problema de la generación inicial del medio.

Método de partículas

Existen unas cuantas técnicas (Lohner 1998, 2004) para construir la distribución inicial de objetos en los métodos de elementos discretos. Caen fundamentalmente en dos categorías: de expansión y de avance frontal o deposición.

Los métodos de expansión parten de una muestra inicial de puntos o cuerpos, y en pasos sucesivos cada uno de los puntos o cuerpos se expande hasta tocar al vecino. Las principales variaciones se centran en la forma en que las partículas cambian, los grados de libertad del cambio y las condiciones de parada. En forma pura, estos métodos tienen un inconveniente principal, y es su incapacidad para mantener al mismo tiempo la proporción de llenado y la distribución estadística elegida para las partículas. En efecto, dada cierta cantidad de partículas, el volumen que ellas ocupan varía si su tamaño relativo cambia o cambia su número. El cambio relativo de tamaños implica cambio en la distribución de las dimensiones. Por otra parte, variar el número de partículas, a menos que existan espacios vacíos suficientemente grandes, acarrea reacomodamientos con potencial largo alcance. La única forma de mantener el volumen aproximadamente igual sin variar la distribución de las dimensiones es tal vez variando ambos parámetros al mismo tiempo, y no está claro cómo hacerlo.

En el otro flanco, el llenado por avance frontal implica que en cada momento intermedio de funcionamiento del algoritmo, existe una parte del cuerpo con partículas y una vacía. Para diferenciar entre las dos fases, los métodos de avance frontal de una forma o de otra necesitan mantener una representación de la superficie frontera entre las dos fases. En dos dimensiones no suele ser muy complejo, basta con una poligonal. En tres dimensiones, en cambio, el asunto es mucho más difícil. Una superficie compuesta por caras triangulares, por ejemplo, es de por sí bastante compleja y las operaciones de modificación sobre ellas no pueden implementarse trivialmente. Y

por si fuera poco, este tipo de representación para la superficie no suele ser suficiente para reflejar el carácter de la superficie de avance, dada la irregularidad de las formas que se obtienen en dicha frontera.

Método de puntos

En el método de elementos finitos, previo al proceso de cálculo es necesaria la subdivisión del medio en elementos. Los métodos sin mallas son similares al FEM, pero en ellos el conjunto de elementos es reemplazado por un sistema de nodos (puntos) distribuidos sin demasiada regularidad.

Evidentemente, lo que hace tan promisorio a los métodos sin mallas es que la generación de esta nube inicial de puntos es un proceso mucho más barato que la construcción de una malla.

Una forma de generar la nube inicial de puntos consiste en generar una malla, quedarse con los nodos de esta y descartar la información de conectividad (Liu, 2003). Esta variante es poco práctica, porque al fin y al cabo lo que quiere evitarse es la construcción de la malla. También deja la posibilidad de una excesiva regularidad en la distribución de los puntos.

Otro enfoque, seguido por (Lohner, 2004) consiste en ajustar un algoritmo existente para la generación de mallas (Lohner, 1998) para generar puntos. El resultado es un procedimiento que trabaja reduciendo un frente activo de puntos. El frente inicial está compuesto por el contorno completo del cuerpo, del que se van removiendo puntos gradualmente.

1.2.3 Sobre la naturaleza de los cálculos en la simulación

Esquema de cálculo en los métodos sin mallas

El esquema de cálculo en los métodos sin mallas es parecido al de los métodos de elementos finitos, en el sentido de que se ensambla un sistema de ecuaciones global. Este sistema se resuelve y se integra en el tiempo, en caso de problemas di-

námicos.

En más detalles, los pasos son los siguientes (Liu, 2003):

1. Representación del dominio. Se modela el cuerpo sólido usando un conjunto de nodos dispersos por el volumen y la superficie del mismo. La densidad de la distribución de nodos depende el grado de exactitud que se quiera alcanzar, y puede ajustarse adaptativamente durante el propio proceso de cálculo.
2. Interpolación del campo. Puesto que no se usa una malla de elementos en los métodos sin mallas, la variable de campo (por ejemplo, una componente del desplazamiento) u en cualquier punto $\mathbf{x}=(x, y, z)$ se interpola usando el desplazamiento de los nodos dentro del dominio de soporte de \mathbf{x} :

$$u(\mathbf{x}) = \sum_{i=1}^n \phi(\mathbf{x}) u_i = \Phi(\mathbf{x}) \mathbf{U}_s$$

3. Formación del sistema de ecuaciones. Las ecuaciones discretas de un método sin mallas pueden ser formuladas usando las funciones de forma y un sistema de ecuaciones en forma fuerte o débil. Con frecuencia, estas ecuaciones son escritas en forma nodal y luego son ensambladas en un sistema de ecuaciones global. El sistema de ecuaciones global es un sistema de ecuaciones algebraicas para análisis estático, ecuaciones de autovalores para análisis en problemas de vibraciones libres, y ecuaciones diferenciales con respecto al tiempo para problemas generales de dinámica.
4. Resolución del sistema de ecuaciones. Se utiliza un solucionador estándar con este fin.

Esquema de cálculo en los métodos de partículas

El método de partículas usa un esquema de cálculos explícito, puesto que se integran directamente

las ecuaciones de movimiento para cada partícula. Contrasta con los métodos de elementos finitos y métodos sin mallas. Estos pueden usar formas débiles en cuanto a sistema de ecuaciones (probablemente diferenciales en el tiempo) local. Así, mientras los métodos implícitos pueden beneficiarse de la estabilidad numérica de las formas débiles, en el método de partículas no es posible y debe prestarse especial atención al tamaño del paso de integración temporal. Por otra parte los métodos de partículas funcionan perfectamente bien en condiciones de no linealidad (cuando las ecuaciones que se definen localmente no dependen de manera lineal de la función incógnita y sus derivadas, por ejemplo: presencia de discontinuidad geométrica, no linealidad geométrica, no linealidad del material). En el caso de los métodos de elementos finitos y los métodos sin mallas

Los cálculos que se llevan a cabo en los siguientes pasos (Recarey, 2003):

1. Aplicación de una ley de fuerza en función de la posición relativa de la partícula y sus vecinas. En este paso además se calculan fuerzas de fricción y viscosidad a nivel de contacto y globales; también pueden ponerse leyes de interacción con algún campo global. El resultado de este paso es un vector de fuerza resultante y un vector de momento resultante.
2. Uso de la segunda ley de *Newton* para calcular el vector aceleración y la aceleración angular de cada partícula.
3. Se utiliza la información de posición y velocidad de la partícula (en coordenadas y angular) del paso actual y del anterior^{iv} para calcular las nuevas coordenadas y primeras derivadas en el tiempo. En este paso se utiliza un esquema de integración explícito en el tiempo.

Este esquema permite un tratamiento eficiente de los fenómenos presentes en grietas, fisuras y microfisuras, u otros fenómenos que tienen lugar en las juntas, tales como el deslizamiento y la separación. También permite modelar los sistemas sometidos a grandes cambios geométricos.

Integración en el tiempo

En los métodos numéricos, se utiliza un integrador explícito cuando el sistema de ecuaciones o las ecuaciones de gobierno en general modelan un fenómeno dinámico.

Método de Euler

En cuanto a formas de integración, la más simple es la de *Euler*. Dicho esquema se puede obtener del desarrollo en forma de *Taylor* de una función en el tiempo. Por ejemplo, si se conocen las derivadas primera y segunda, y se parte de la forma de *Taylor* de segundo orden

$$f(x_0 + h) = f(x_0) + h f'(x_0) + h^2 \frac{f''(x_0)}{2!} + R_2(x + h) \quad (1)$$

entonces el esquema podría anotarse como sigue:

$$f'_{i+1} = f'_i + h f''_i$$

$$f_{i+1} = f_i + h f'_i + \frac{h^2 f''_i}{2}$$

En la fórmula (1) $R_2(x+h)$ habla sobre el término residual en la fórmula de *Taylor*.

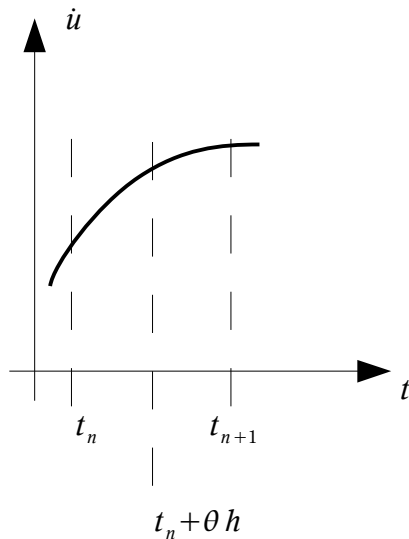


Figura 1: El teorema del valor medio establece que para algún punto

$$\xi = t_i + \theta h, 0 \leq \theta \leq 1 \text{ se cumple}$$

$$f[t_{(n+1)}] = h f'(\xi) + f(t_n)$$

Método Newmark-Beta

Estos métodos se basan en un refinamiento del método de Euler. Dicho refinamiento consiste en la aplicación del teorema extendido del valor medio al término residual en forma integral. Con la aplicación de este teorema es posible reescribir el término residual en la fórmula de *Taylor* como:

$$R_n = (x - x_0)^{n+1} \frac{f^{(n+1)}(x^*)}{(n+1)!} \quad (2)$$

donde $x^* \in [x_0, x]$ depende del orden de la fórmula de Taylor (Weisstein, 2006).

El desarrollo de segundo orden está expresado con claridad en la fórmula (1); el de primer orden se obtiene eliminando los términos relativos a la segunda derivada y escribiendo “ $\dots + R_1(x+h)$ ” en su lugar. Sustituyendo (2) en esas ecuaciones y ajustando ligeramente la notación se obtiene el esquema de integración conocido como *Newmark-Beta*:

$$f'_{i+1} = f'_i + h f''_y$$

donde

$$f''_y = (1 - \gamma) f'' + \gamma f''_{i+1}$$

y

$$f_{i+1} = f_i + h f'_i + \frac{h^2}{2} f''_\beta$$

donde

$$f''_\beta = (1 - 2\beta) f'' + 2\beta f''_{i+1}$$

Esquema de Verlet (diferencia finita central)

El esquema de integración basado en diferencia central, también conocido como método de Verlet, se obtiene del *Newmark-Beta* usando los valores $\beta=0$ y $\gamma=1/2$ (Serón, 1989). A continuación se presenta una deducción independiente que deberá ayudar a entender esta selección de coeficientes. La siguiente identidad elemental del cálculo diferencial sirve de punto de partida:

$$f' \approx \frac{f(x+h) - f(x)}{h}$$

Suponiendo que se conoce el valor de la función en los puntos x_{i-1}, x_i , entonces:

$$f'_{i-1} = \frac{f_i - f_{i-1}}{h} \quad (3)$$

$$f'_i = \frac{f_{i+1} - f_i}{h} \quad (4)$$

De las ecuaciones anteriores puede deducirse que:

$$f''_{i-1} = \frac{f'_i - f'_{i-1}}{h} \quad (5)$$

sustituyendo en la última igualdad las dos anteriores, se obtiene:

$$f''_{i-1} = \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2} \quad (6)$$

lo que conduce directamente a una fórmula para interpolar el valor de f_{i+1} :

$$f_{i+1} = f''_{i-1}h^2 - f_{i-1} + 2f_i \quad (7)$$

Este esquema es más exacto que el de Euler.

Aceleración media constante

Es otro tipo de *Newmark-Beta*, donde se han puesto los coeficientes $\beta = 1/4, \gamma = 1/2$. Su nombre deriva del símil existente con las ecuaciones del movimiento bajo aceleración media cons-

tante.

1.2.4 Aspectos físicos.

Se aborda concretamente del modelo físico para este método por ser el mismo el corazón del método de partículas. Generalmente, de esta parte se ocupa personal con conocimientos de física, sin embargo son necesarias algunas nociones preliminares para construir el software. Durante este epígrafe la referencia principal será (Recarey, 2003).

Ecuaciones del movimiento e integración explícita.

El método de partículas integra directamente las ecuaciones en esta forma.

El estado de movimiento de la partícula puede definirse con las ecuaciones de Newton-Euler:

$$\begin{aligned} m_i \ddot{\mathbf{u}}_i &= \mathbf{F}_i \\ I_i \dot{\boldsymbol{\omega}}_i &= \mathbf{K}_i \end{aligned}$$

donde :

| | |
|-------------------------|--|
| m_i | Masa de la i-ésima partícula |
| \mathbf{u}_i | Ley del movimiento de la i-ésima partícula |
| \mathbf{F}_i | Fuerza externa resultante que actúa sobre la i-ésima partícula |
| I_i | Tensor de momento de inercia que la i-ésima partícula |
| $\boldsymbol{\omega}_i$ | Aceleración angular de la i-ésima partícula |
| \mathbf{K}_i | Momento de fuerza resultante que actúa sobre la partícula |

Todas estas magnitudes entran en el modelo físico, pero desde el punto de vista algorítmico las que juegan un papel más complejo son las que caracterizan la interacción, es decir, el conjunto de

fuerzas $\mathbf{F}_i = \sum_{k=0}^n \mathbf{F}_i^k$ que actúa sobre la partícula.

Si se toman esféricas las partículas, entonces a cada contacto k le corresponde una fuerza \mathbf{F}_i^k que puede descomponerse en tangencial y normal.

La componente tangencial influye en el momento de fuerzas. Adicionalmente, el modelo incluye fuerzas de rozamiento y fricción, que pueden actuar por contacto o de forma única dependiendo del estado de movimiento de la partícula.

Por *modelo físico constitutivo* se entiende las ecuaciones que definen la respuesta de un material frente a un conjunto de acciones formadas por fuerzas externas, suministro de calor u otro agente. Estas ecuaciones han de ser válidas en cualquier circunstancia interna y/o externa; por lo cual deben cumplir una serie de requisitos de valor universal los cuales constituyen los axiomas de consistencia^v. Además en mismas deben intervenir diferentes parámetros^{vi} que definen el estado mecánico en que se encuentra el material. El siguiente fragmento proviene de (Recarey, 1999) y da una idea de la variabilidad y la polémica que existe en torno al tema.

Para predecir el comportamiento del medio se utilizan modelos de comportamiento que solo reflejan la realidad hasta cierto punto. Normalmente las heterogeneidades del material, las limitaciones de los estudios, y el desconocimiento del estado inicial y de las condiciones de contorno impuestas se traducen en ciertas diferencias entre el comportamiento predicho por el modelo y el que se puede medir en la realidad. Sin embargo, en un proceso de diseño de cualquier elemento o estructura en contacto o no con el medio, el ingeniero utiliza constantemente modelos. Estos deben estar debidamente comprobados y calibrados para que se correspondan con la práctica.

El procedimiento de selección o elaboración de un modelo de comportamiento adecuado no es sencillo. En primer lugar se deben conocer los fenómenos importantes que tienen lugar en el medio en esas condiciones, su razón física y su posible formulación abstracta y decidir cuales de ellos son los relevantes para determinado proceso. Con ellos se elaborará el modelo conceptual, o abstracción lógica de la realidad en la cuál vendrán esquemáticamente relacionados, todas las variables y fenómenos relevantes. Este modelo conceptual se concretará en unas ecuaciones o en un esquema numérico que se adapte al tipo de problemas planteado y que, en función de los da-

tos disponibles y de una serie de parámetros, logre predecir el comportamiento del medio ante las solicitaciones.

El tratamiento físico matemático de la interacción a nivel de contacto de un sistema de partículas deslizándose y rotando es complicado. Los diversos modelos simplifican el tratamiento intentando no perder la esencia del fenómeno. En algunos casos se supone que las partículas son indeformables, en otros que no existe rotación relativa, o que todos los deslizamientos ocurren simultáneamente, etc. Por este motivo no se obtienen relaciones tensión-deformación generales, sino que a lo sumo se consiguen relaciones parciales entre algunas variables de deformación y otras geométricas, inspiradas microscópicamente. A menudo se han de establecer hipótesis adicionales, como suponer que al deslizar las partículas debe ser mínima la relación entre el gradiente de disipación de energía por rozamiento interno y el gradiente de energía aportada por la tensión principal mayor (Rowe, 1962, 1971), etc.

Después de esto, queda claro que un software para simulación de método de partículas debe ser, cuando menos, extensible a una gran variedad de modelos constitutivos de contacto.

Lógica de conglomerados

Existen distintas abstracciones de elementos discretos. Por ejemplo, esferas, elipses, poliedros, etc., tratados como elementos individuales o como conglomerados.

Dentro de la parte de conglomerados, los formados por esferas toman en cuenta el conjunto de esferas como un todo. Para esto se usa la llamada “lógica de conglomerados” (en la bibliografía se ve con frecuencia “cluster” por “conglomerado”). A continuación un fragmento de (PFC2D, 2002):

“La lógica del conglomerado provee medios para crear y modificar grupos de partículas esclavas o conglomerados. Un conglomerado se comporta como un cuerpo rígido (las partículas componentes del conglomerado permanecen a una distancia fija de cada una). Los contactos internos al conglomerado son obviados durante el ciclo de cálculo, resultando en un ahorro de tiempo de

cómputo comparado a un cálculo similar en el cual todos los contactos están activos. Sin embargo, los contactos con partículas externas al conglomerado no son afectados (tales contactos se desarrollarán cuando las partículas componentes de la frontera de un conglomerado vienen a hacer contacto con otras partículas). Las partículas dentro de un conglomerado pueden solaparse en cualquier medida; las fuerzas de contacto que existen cuando el conglomerado es creado o cuando una partícula es agregada al conglomerado serán preservadas invariables durante el ciclo. Así, un conglomerado actúa como un cuerpo rígido (con una frontera deformable) que no se romperá en pedazos, independientemente de las fuerzas que actúen sobre él. En este sentido, un conglomerado difiere de un grupo de partículas que están enlazadas mutuamente.”

El párrafo anterior evidencia el trabajo con sistemas de partículas de dos formas: por lógica de conglomerados o por parametrización especial de los contactos.

Los métodos sin mallas y el modelo físico.

Estos métodos pueden atender más o menos la misma variedad de problemas que los métodos de elementos finitos: aquellos que pueden expresarse con ecuaciones diferenciales en derivadas parciales.

El área de aplicación de estos métodos son problemas mecánicos, problemas térmicos y problemas de fluidos. Cada una de estas ramas tiene su propio aparato físico que definen el modelo físico-matemático (ecuaciones de equilibrio, ecuaciones físicas y constitutivas, ecuaciones de compatibilidad, etc.). Ver (Liu, 2003).

1.3 Revisión de algoritmos y bibliotecas existentes.

El objetivo de este epígrafe es ilustrar brevemente los algoritmos y estructuras de soporte imprescindibles para desarrollar un software destinado a la simulación con métodos sin mallas y de partículas. En cada caso, se revisan las categorías, con-

ceptos y algoritmos presentes en la literatura, así como posibles implementaciones existentes.

1.3.1 El problema de las licencias.

El software se distribuye bajo distintos tipos de licencias. Las licencias son, por decirlo de alguna forma, acuerdos legales entre el productor del software y sus usuarios. Una primera clasificación las sitúa en dos tipos: licencias comerciales y licencias de código abierto.

Las licencias comerciales por lo general deniegan la redistribución del software, y están hechas para proteger los intereses comerciales de los productores del software. Sirven para proteger la inversión del desarrollador. No son en sí un fenómeno negativo, aunque según la ley del valor deberían estar sujetas a términos de caducidad más precisos.

Por otra parte, las licencias de código abierto persiguen con frecuencia el fin opuesto: proteger los derechos de modificación y redistribución del software. Dichas licencias estipulan que el software debe redistribuirse en forma de código fuente, al igual que sus modificaciones. El fenómeno de las licencias de código abierto se remonta al pasado de la historia del software (un pasado extremadamente reciente en término de años, pero que en la mente de los programadores más jóvenes se compara con la antigüedad clásica en algunas ocasiones), cuando grandes monopolios de la industria de las telecomunicaciones y el hardware se adueñaron del código que la comunidad de usuarios de Unix había desarrollado.

Es preciso atender al asunto de las licencias desde dos puntos de vista: el de usuario y el de productor.

En el primer caso, el uso de licencias comerciales provee acceso a rutinas y bibliotecas “con soporte garantizado”; pero significa costo adicional del software. Y lo que es peor, trabas legales insuperables en la mayor parte de los casos. Los softwares bajo licencia de código abierto son una opción más atractiva.

En el segundo punto de vista, como productor, el uso de licencias de código abierto en algunos ca-

so implica que el código que se produce también debe distribuirse como código abierto. Si la aplicación que se desarrolla tiene un fin comercial, esto, por supuesto, es un rasgo prohibitivo. Entonces se está frente a un problema muy complejo, que requiere una investigación detallada en cada caso.

En este documento se detallará más sobre las licencias de software en el capítulo 3, cuando se precisen las decisiones que se hicieron en cuanto a productos y bibliotecas de soporte, en los casos en que estas se usaron.

1.3.2 Geometría computacional.

La geometría computacional es la parte de la computación aplicada que se encarga de reflejar mediante estructuras de datos y funciones las entidades del mundo geométrico y sus predicados.

En cuanto a entidades geométricas, muchos recuerdan de sus cursos de geometría básica conceptos como el de punto, recta, plano, etc. Son las entidades o objetos geométricos más simples, pero no son, ni con mucho, los únicos. Existen, por ejemplo, unos cuantos tipos de agregados de entidades, como mallas, polígonos; con una representación en términos de estructura de datos más bien compleja. Otras figuras se definen en base a expresiones analíticas y definiciones conjuntuales. Ejemplo: el conjunto de puntos que equidista de un centro (esfera), o el conjunto de puntos tal que la suma de su distancia a dos puntos fijos es constante (elipse).

Los predicados geométricos son las afirmaciones que pueden hacerse de la relación entre objetos geométricos. Así, los predicados de incidencia tratan la intersección o contención de un objeto en otro. También son predicados aquellos que se encargan de asignarle un valor numérico a cierta métrica definida en el espacio para dos objetos dados (por ejemplo, alguna definición de distancia).

Con la finalidad de evaluar con mayor eficiencia distintos predicados, surgen en algunos casos estructuras de datos especializadas. Por ejemplo, en un polítopo (cuerpo formado por caras de distintas dimensiones conectadas entre sí) es posible

conocer cuales son las caras adyacentes a un elemento dado de antemano aprovechando la conexidad existente en su representación como grafo. Cuando se quieren hallar los dos o tres vecinos más cercanos a un elemento en una nube estática de puntos, puede usarse una triangulación de Delaunay. Existe, para esta cuestión de los vecinos, una amplia gama de estructuras de datos y algoritmos.

Un problema relacionado con la implementación de los predicados de la geometría computacional tiene que ver con la exactitud numérica. Resulta que la solución correcta a muchos problemas pasa por el uso de mayor o menor precisión numérica en el predicado. Como enfoque “torpe”, podría usarse simplemente aritmética real. Una segunda aproximación “menos torpe” sería emplear una biblioteca que brinde una precisión bastante alta, como **gmp**. La tercera aproximación sería usar aritmética exacta. La forma de proceder más simple es conformarse con la precisión brindada en la representación de números flotantes por el sistema. Esto es lo más inseguro, pero también lo más eficiente; y en muchos casos concretos es posible encontrar trucos para verificar y corregir el resultado sin comprometer la eficiencia del programa. Este último proceder se justifica porque algunos predicados y funciones constructivas requieren un gran cuidado con la precisión, mientras que otros pueden considerarse mayormente “seguros”.

En un entorno básico, los predicados geométricos esenciales pudieran ser:

- Predicados de incidencia.
- Predicados de interioridad para polígonos en dos dimensiones y mallas.
- Búsqueda de vecinos.

El libro de cabecera para los predicados de incidencia podría ser (O'Rourke, 2002). Allí se definen los predicados en función de áreas, aprovechando que las áreas pueden definirse como funciones positivas y mediante operaciones de multiplicación, que conservan mejor la exactitud aritmética. Un enfoque alternativo y equivalente es usar álgebra de vectores. El producto escalar de dos vectores se define como

$$\mathbf{v}_1 \cdot \mathbf{v}_2 = \sum_{i=1}^n v_1^i v_2^i ,$$

es decir, en base a multiplicaciones, con lo que también se preserva la exactitud aritmética. Otro detalle está en no usar predicados que requieran una comparación de igualdad. Por ejemplo, tiene sentido preguntar si la proyección de un punto cae sobre un segmento, pero no tiene sentido preguntar si un punto cae dentro de un segmento. Esto se debe al compromiso de usar una precisión aritmética limitada en los cálculos.

De geometría computacional existen múltiples recursos en formas de bibliotecas para distintos lenguajes y propósitos.

La biblioteca de geometría computacional a disposición del público general con mayor popularidad hasta el momento es la CGAL (CGAL, 2006). CGAL está escrita en un dialecto muy moderno de C++, y es, entre otras cosas, parametrizada en el tipo de kernel numérico, por lo que puede operar con flotantes inexactos o con representaciones de números exactos provistas por el usuario (aparte de incluir las propias). Tiene dos desventajas notables. La primera es el tipo de licencia bajo la cual se puede obtener: GPL o bajo licencia comercial, dependiendo de la finalidad que el usuario/desarrollador de la biblioteca tenga. El uso de este producto, si bien aconsejable con fines de investigación, trunca cualquier propósito comercial a largo plazo. La otra desventaja de la CGAL, que es de índole técnica, más concreta y palpable es su volumen y costosa infraestructura; además introduce tiempos de compilación incosteables^{vii} en el resto del código, por lo que no es rentable usar un sistema tan completo para unas pocas operaciones geométricas básicas.

Otra biblioteca revisada fue *FastGEO*, una biblioteca de geometría computacional para Object-Pascal. Dado que la plataforma de trabajo elegida es C++, no fue de interés evaluarla.

El profesor John Burkardt, de la *School of Computational Sciences* en *Florida State University*, tiene en su sitio web un útil compendio de rutinas geométricas básicas implementadas en C y en Fortran. En el mismo no aparece

ninguna proclama de licencia o propiedad, y él en correo directo aseguró que se podía usar las rutinas con cualquier fin. No se hace uso de aritmética exacta, ni siquiera de estructuras de datos propias, lo que hace estas rutinas ideales para su inclusión en cualquier proyecto. La biblioteca es mayormente usable aunque se detectaron algunos errores.

Todo lo visto hasta aquí hace recomendable la implementación de una biblioteca de geometría computacional con rutinas básicas hechas a la medida que:

1. Estén escritas en C++, el lenguaje de implementación escogido.
2. Compartan el mismo código para objetos en dos y tres dimensiones.
3. Se comporten bien ante el uso de la eficiente aritmética nativa (del procesador).
4. Sean suficientemente robustas y eficientes.

1.3.3 Estructuras de consulta espacial.

Una de las clases de predicados con más fuerza en la geometría computacional (y en las bases de datos multidimensionales) por el número de aplicaciones que tiene y por sus derivaciones computacionales son los distintos tipos de consulta espacial. Todas tienen aplicación tanto en geometría computacional como en sistemas de manejo de información de distintos tipos. Por ejemplo, sistemas de información geográficos, gráficos por ordenador, bases de datos espaciales bases de datos de series temporales. En general se les conoce como consultas de rango, aunque el término también denota un tipo de consulta particular. Para evitar ambigüedad en algunos puntos de este trabajo serán llamadas estructuras de consulta espacial.

Lo que sigue es un compendio de los principales tipos de consulta espacial.

- **Consulta de rango:** Sea S un conjunto de n puntos en \mathbb{R}^d , y sea \mathcal{Y} una familia de subconjuntos de \mathbb{R}^d llamados

rangos. Se desea preprocesar S en una estructura de datos de forma tal que para una consulta de *rango* $v \in Y$, los puntos $S \cap v$ puedan ser reportados o contados eficientemente. Típicamente, los rangos son ortohedros con aristas alineadas a los ejes, semiespacios o bolas (Agarwal, 1997).

- **Consultas de punto:** Sea S un conjunto de n cajas en el espacio \mathbb{R}^d , y sea Y un conjunto de puntos. Se desea preprocesar S en una estructura de datos de forma tal que para una consulta de *punto* $v \in Y$, los intervalos $\{\tau \in S | \tau \cap v \neq \emptyset\}$ puedan ser reportados o contados eficientemente.
- **Consultas intervalo-intervalo:** Sea S un conjunto de n cajas en el espacio \mathbb{R}^d , y sea Y un conjunto rangos. Se desea preprocesar S en una estructura de datos de forma tal que para una consulta de *rango* $v \in Y$, los intervalos $\{\tau \in S | \tau \cap v \neq \emptyset\}$ puedan ser reportados o contados eficientemente.
- **Consultas de vecindad:** Sea S un conjunto de n cuerpos en el espacio \mathbb{R}^d ; dado cierto elemento $s \in S$ y un número $r \in \mathbb{R}$, se desean conocer todos los elementos $\{u_i \in S | \phi(u_i, s) \leq r\}$, donde ϕ representa cierta métrica en \mathbb{R}^d ; más comúnmente la euclidiana.

Estos no son los únicos tipos de consulta espacial, pero si los más frecuentes. Existen relaciones entre ellas, que permiten formular unas en función de las otras bajo ciertas condiciones. En particular, existen algoritmos que pueden expresar todas las consultas enumeradas arriba como caso específico de consultas *intervalo-intervalo*.

Sin entrar en detalles, las estructuras más conocidas en la literatura para satisfacer consultas espaciales son (Zheng, 2006; Worm, 2003):

- Árboles de descomposición jerárquica (kd-tree, octree)

- Árboles de cubrimiento (*R-Tree*, *R*-Tree*, Hilbert *R-Tree*).
- Listas de omisión de Intervalos
- Árboles de segmentos
- Árboles de intervalos

En función de las necesidades, existen implementaciones de estructuras de distintos tipos para satisfacer consultas espaciales. El principal problema, desde el punto de vista del desarrollador de programas, es que no existe un enfoque sistemático en su implementación. Es decir, no se conoce una biblioteca que realice los resultados teóricos que existen para este tipo de consultas, ni que implemente las relaciones entre distintos tipos de consultas.

1.3.4 Números aleatorios

Todo método de simulación requiere de una cantidad considerable de números aleatorios o pseudoaleatorios. En especial, el DEM y MFree requieren instancias de números aleatorios; en un caso para la configuración inicial de las partículas y en otro para la construcción de la nube inicial de puntos.

Los números aleatorios son aquellos que se obtienen de forma “impredecible”, por apelar a un concepto intuitivo, que no obedecen a una secuencia fija.

Por otra parte, los números pseudoaleatorios se generan según un algoritmo, y no son realmente aleatorios porque son exactamente predecibles.

En los experimentos de simulación suelen usarse número pseudoaleatorios ya que es más sencillo obtenerlos y porque una secuencia fija, en cuanto tenga propiedades estadísticas “buenas”, es perfectamente aceptable.

La estructura de un generador de números pseudoaleatorios (GNPS) se reduce a cierta cantidad de bits de estado y una rutina que muta estos bits. El diagrama 2 da una idea:

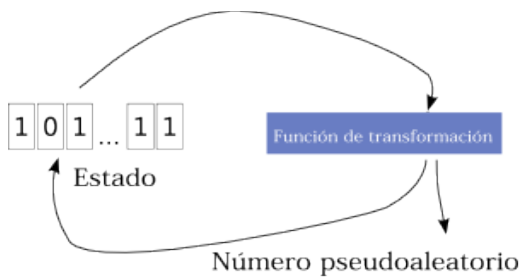


Figura 2: Estructura de un generador de números aleatorios

Si la cantidad de bits presente en el estado del GNPS es B , entonces el número máximo (ideal) de estados posibles es 2^B . La cantidad de veces que puede invocarse la rutina del GNPS sin que se repite la secuencia se conoce como período del generador. Claramente, el período está acotado por la cantidad de estados posibles.

Entre los criterios de calidad de un generador de números entra no solo el deseo de que este tenga un período extremadamente alto. Otras propiedades no estadísticas son (L'Ecuyer, 2006):

- eficiencia
- repetibilidad
- portabilidad

Aparte de esto, un buen GNPS tiene que cumplir propiedades adicionales que garanticen la “aleatoriedad” de la secuencia, cosa que se mide de acuerdo a baterías de pruebas estadísticas (Soto, 1998).

Es sencillo crear un algoritmo básicamente malo para generar números aleatorios. Esa es la razón de que uno de los puntos comunes en la literatura desde los tiempos de la VAX es no fiarse a ciegas del generador de números del sistema. Los presentes en los sistemas operativos actuales suelen ser más confiables, pero sigue siendo mala idea usarlos en experimentos de simulación. Una vez que se decide tomar en las manos el tipo de generador, se encuentran en la literatura dos grandes clases de generadores.

En la primera entran los generadores con recurrencia lineal módulo m . Se basan en la fórmula

$$x_i = (a_1 x_{i-1} + \dots + a_k x_{i-k}) \bmod m.$$

En la segunda gran clase entra el resto de los sistemas de generación de números aleatorios. Se trata de los algoritmos no lineales.

Uno de los más populares en los últimos tiempos es el conocido como *Mersenne Twister* (Matsumoto, 2006), que es un generador basado en recurrencia lineal con excelentes características técnicas; en particular, cumple con todos los requisitos enunciados más arriba sobre rasgos prácticos y estadísticos esperados en un generador de números aleatorios. En el sitio *web*, su autor asegura que dicho generador puede usarse para cualquier fin. Para el propósito actual, esta es la mejor variante encontrada.

1.3.5 Infraestructura. Rutinas numéricas.

Un sistema lo suficientemente general debe incluir soporte para cierto nivel de operaciones numéricas. Por ejemplo, C/C++ no tiene soporte nativo para potenciación. La biblioteca estándar de C contiene muchas de las funciones matemáticas trascendentes: potencias y logaritmos, trigonómicas e hiperbólicas, control de precisión y rangos y algunas constantes. Pero para un proyecto donde la matemática numérica es el alma, probablemente no es suficiente.

En particular, se hacen necesarios algoritmos de manipulación de matrices y álgebra lineal. En una etapa más avanzada del proyecto probablemente se necesitarán solvers para sistemas de ecuaciones lineales con matrices dispersas u otros problemas de mayor grado de complejidad. En algunos casos es necesario utilizar algoritmos de programación convexa.

A la hora de escoger un algoritmo de programación lineal, las decisiones no son demasiado difíciles. Si se trata de un sistema grande, está bien usar métodos de punto interior. Para sistemas de tamaño modesto puede usarse el método *simplex*, cualquiera de sus variantes tiene una complejidad equivalente, por lo que no vale la pena hacer distinciones entre ellos.

Métodos de optimización.

Existe una razón para hacer énfasis en este tipo de algoritmos. Resulta que los problemas de decisión y optimización están presentes de forma natural en casi todas las áreas. Además, muchos problemas pueden expresarse en términos de la optimización de cierto objetivo, y en esos casos pues automáticamente se dispone de un amplio y buen repertorio previo de resultados y métodos.

En la parte de programación convexa, las rutinas y métodos están bien estudiados. Los métodos de gradiente conjugado y cuasi-newtonianos son clásicos en el análisis convexo. En general, estos métodos trabajan construyendo una sucesión de minimizaciones lineales. La secuencia de direcciones de búsqueda se usa para construir una aproximación de la curvatura de la función en la vecindad del mínimo. Con frecuencia, como dirección inicial de búsqueda se escoge el gradiente de la función. La exactitud de la minimización lineal es especificada por el parámetro τ . En el mínimo a lo largo de la línea el gradiente de la función y la dirección de búsqueda son ortogonales, así, la minimización lineal termina cuando $\mathbf{g} \cdot \mathbf{p} \leq \tau$. De los algoritmos de gradiente conjugado, la diferencia entre el denominado de *Fletcher-Reeves* y el de *Polak-Ribiere* está en la fórmula según la cual se actualiza la dirección de búsqueda; en el primer caso es, $\mathbf{p}' = \mathbf{g}' - \beta \mathbf{g}$

con $\beta = -\frac{|\mathbf{g}'|^2}{|\mathbf{g}|^2}$, y en el segundo la única modificación es en este coeficiente β que se escribe como sigue:

$$\beta = \frac{(\mathbf{g}' - \mathbf{g}) \cdot \mathbf{g}'}{|\mathbf{g}|^2}$$

Una parte de la programación convexa menos conocida son los métodos sin derivadas y proximales. Estos métodos se caracterizan porque no usan derivadas, sino que emplean alguna heurística para escoger el próximo punto. Son útiles en condiciones donde la función a optimizar no es continua o no lo es su primera derivada. Estos escenarios aparecen con frecuencia. Quizá los dos métodos más conocidos sean el de *Nelder-Mead* y el de *Hooke-Jeeves*.

Dado un punto inicial (x_0, x_1, \dots, x_n) , el algoritmo de *Nelder-Mead* procede construyendo un *simplex* con los siguientes vértices:

$$(x_0, x_1, \dots, x_n)$$

$$(x_0 + \delta, x_1, \dots, x_n)$$

$$(x_0, x_1 + \delta, \dots, x_n)$$

.....

$$(x_0, x_1, \dots, x_n + \delta)$$

donde δ es el tamaño del paso. En cada iteración el algoritmo trata de mejorar el mejor vértice del *simplex* mediante simples transformaciones geométricas: reflexión, reflexión seguida por expansión, contracción y contracción múltiple. Usando estas transformaciones el *simplex* se mueve por el espacio de búsqueda hacia el mínimo, alrededor del cual se contrae. Para más detalles, ver (GSL, 2004).

Por su parte, la variante de *Hooke-Jeeves* se basa en un *stencil* con forma de estrella. Comenzando por el punto (x_0, x_1, \dots, x_n) , el algoritmo busca el punto con mejor valor entre

$$(x_0 + \delta, x_1, \dots, x_n)$$

$$(x_0, x_1 + \delta, \dots, x_n)$$

.....

$$(x_0, x_1, \dots, x_n + \delta)$$

Si el mejor valor de la función objetivo en esos puntos es menor que en el punto inicial, *Hooke-Jeeves* intenta un paso agresivo y se desplaza duplicando el paso. De fallar escoge el punto original. Si ninguno de los puntos mejora el valor de la función objetivo, entonces trata en las direcciones opuestas, con $-\delta$. Si finalmente no se logra mejorar el valor de la función objetivo, se vuelve a empezar el proceso con $\delta' = \delta/2$.

Bibliotecas e implementaciones para cuestiones de matemática numérica básica.

En materia de bibliotecas e implementaciones existentes, afortunadamente, hay donde escoger.

La cantidad de recursos disponibles en Internet de todos los sabores para matemática numérica básica es colosal. Es imposible mencionarlos a todos. Por su completitud, es destacable la *gsl*, publicada bajo licencia GPL. Por otro lado, con licencia comercial, existe la reconocida **NAG** (NAG, 2006). De punto de partida de la implementación podría servir la *gsl*, aunque el problema de las licencias limita su uso en una versión definitiva.

1.4 Herramientas de construcción de software

A la hora de comenzar la implementación de un software de cualquier tipo, la primera decisión está relacionada con las herramientas básicas que se usarán al programar. Por ejemplo, lenguaje de programación, compilador para dicho lenguaje, editor, IDE, sistema de control de fuentes (si se usa alguno), etc. Estas decisiones moldearán en cierto grado el curso del proceso de desarrollo del software. Luego, se seleccionan “extensiones” que “adelanten lo que ya está hecho”, es decir, bibliotecas o como quiera que se llamen los módulos reutilizables en el lenguaje escogido.

Uno de los criterios para seleccionar la herramienta es su disponibilidad en distintas plataformas. Esto es especialmente justo para el software científico, que necesita ser capaz de correr en una gran variedad de arquitecturas. Es pueril, por ejemplo, asumir que los cálculos científicos se harán siempre sobre clones de PC.

Otro punto que hay que observar es el grado de madurez de las herramientas, es decir si están probadas y si existe personal que las mantiene. Puede conocerse esto visitando el sitio web de los desarrolladores y averiguando cuantos productos exitosos existen sobre ella. También hace falta conocer si se ha usado en proyectos similares al que se quiere desarrollar.

El otro aspecto que debe tomarse en cuenta, y que tiene el mayor peso en el éxito y el desarrollo a largo plazo de una aplicación, es el relacionado con la ingeniería de software. Existen libros y gruesos volúmenes, especificaciones, normas y

metodologías sobre ingeniería de software. Aplican perfectamente a entornos de desarrollo complejos y a la producción de software empresarial. Aquí no se hablará de *esa* ingeniería de software, sino del aspecto más simple de diseñar software mantenible y flexible con los recursos modestos de que podrían disponer uno o dos desarrolladores. A la hora de seleccionar un conjunto de herramientas de construcción de software, se toma en cuenta hasta que punto apoya el proceso de producción de software, al menos en la facilidad de generar código cohesivo y sin acoplamiento.

1.4.1 Lenguaje de programación

Los primeros candidatos son C y FORTRAN. Serán analizados juntos, porque de cierta forma merecen un criterio similar. FORTRAN se diseñó expresamente con la programación numérica en mente, por allá por la década de los 60. De ahí uno de sus rangos principales: los arreglos multidimensionales. Pero en aquella época difícilmente las computadoras hacían otra cosa que ejecutar simples rutinas de cálculo numérico. Esta es la razón por la que FORTRAN es poco capaz de servir como cemento para proyectos de software complejos. En la actualidad, en el índice Tiobe ocupa el lugar 21.

En el mismo índice, en cambio, C aparece en el segundo lugar. C fue diseñado desde el principio como un lenguaje de propósito general. Se dice que es el lenguaje más cercano al ensamblador, sin embargo. Ideal para aplicaciones donde la eficiencia y el tamaño del código sean imprescindibles, no permite implementar con facilidad principios esenciales para el desarrollo de un producto grande y complejo.

Uno de los sucesores de C, tanto en la línea de evolución de los lenguajes de programación como en el índice Tiobe del 2006, es el C++. Como características positivas debe notarse que es orientado a objetos y con un sistema de genericidad que permite incluso metaprogramación. Su compatibilidad con C lo ponen en el centro de convergencia de muchos otros lenguajes y tecnologías. Al igual que su antecesor se trata de un lenguaje enteramente compilado, cosa que acerca los programas redactados en él al máximo de eficiencia. El ma-

yor inconveniente con C++ está en su curva de aprendizaje, y lo costosas que suelen ser las herramientas de gestión de código (e.g. *Refactorización*). Al ser tan grande es posible programar en él dentro de varios dialectos cosa que reduce un poco la interoperabilidad.

El software empresarial tiene como herramienta bandera a Java, un lenguaje parcialmente interpretado y parcialmente compilado en tiempo de corrida, con manejo automático de memoria. El soporte que han dado gigantes de la computación como Sun Microsystems e IBM le permite contar con un conjunto de herramientas satélites más bien vasto y *suites* de desarrollo muy completas. Es un lenguaje simple, y en eso radica su mérito (es más fácil trabajar con él) y su principal inconveniente (es menos potente que, por ejemplo, el C++)

Otra plataforma que no debe escapar a la intención de desarrollo es .NET, de Microsoft. Incorpora todas las cualidades positivas de Java, y supera sus limitaciones con creces. Es además multiplataforma, gracias al esfuerzo de Novell. Su sistema de clases puede considerarse bastante completo.

1.4.2 Sistema de control de versiones

El desarrollo en grupo, o incluso individual de una aplicación, requiere que se tomen cuidados especiales con el código fuente. Primero, si se trata de trabajo en grupo, hace falta asegurar que todos los miembros del grupo tengan acceso permanente a la versión más actualizada del producto, y que puedan trabajar simultáneamente. También surge la necesidad de encontrar qué cambios hizo quién y revertirlos si no son correctos. Básicamente estas son las funciones de un sistema de control de fuentes.

Existen varios, y suelen basarse en dos paradigmas. El primero se conoce como *checkout/lock-edit-checkin/unlock*.

Según este esquema cada desarrollador puede obtener acceso exclusivo a la modificación de un fichero, impidiendo que se hagan al mismo modifi-

caciones concurrentes. El segundo paradigma es *checkout-edit-update/merge-commit*. Con este enfoque puede haber más de un desarrollador editando un mismo fichero, con tal de que los cambios luego sean mezclados.

En el primer grupo entran las herramientas corporativas *Microsoft Visual Source-Safe* y *Team-Source* de *Inprise Corp.*

En el segundo grupo existe una gran variedad. El más conocido es *Concurrent Version System*, evolucionado de *RCS*^{viii}, y es un sistema de control de fuentes que ya está pasando a los oscuros rincones de la memoria. Su rasgo más revolucionario y que luego sería adoptado ampliamente por sus sucesores es el control de versiones y la habilidad de retornar a cualquier punto en la historia de los ficheros bajo su control. La desventaja más criticada de CVS es que no puede seguir la historia del sistema de archivos como un todo.

Dicha desventaja viene a ser superada por *Subversion*, un sistema de control de fuentes que por su filosofía está siendo ampliamente adoptado en todos los confines. Es capaz de realizar entregas atómicas, metadatos asociados a nodos del sistema de control de versiones, un sistema barato de bifurcación y mezcla, y la posibilidad de correr sobre distintos tipos de servidores. De hecho trae un servidor capaz de correr autónomamente; pero puede ponerse también sobre un servidor apache y usar *WebDav* como protocolo de comunicación, con lo que es posible acceder a repositorios remotos sobre distintos soportes y a través de un *proxy*.

Por último, vale la pena mencionar otros sistemas de control de fuentes:

- *BitKeeper*
- *ClearCase*
- *OpenCM*
- *Perforce*

1.5 Conclusiones parciales

La simulación es un campo a la que le han brotado muchísimas nuevas ramas y aplicaciones en

los últimos años, pero la construcción de software para esta ciencia es un proceso que abarca un enorme número de facetas.

Los desarrolladores de aplicaciones de este tipo deben abarcar el conocimiento existente en diversas cuestiones de la matemática computacional, manejo y almacenamiento de datos, y modelos físicos. Solo así podrá enfrentarse a la variedad de desafíos que supone la complejidad de un producto de este tipo.

El conocimiento de las posibilidades y limitaciones técnicas y algorítmicas son los principales rectores del proceso de desarrollo de software.

ii En las referencias donde no sea posible dilucidar el

autor de forma precisa, el campo de autor se substituirá por una llave arbitraria que sirva como memonímico a la referencia.

iii El autor de este trabajo no se responsabiliza con los posibles errores en las citas textuales, las mismas fueron incorporadas por un proceso digital que preserva los detalles del texto original.

iv El uso de integración por diferencias centradas, por ejemplo, requiere del uso de información de al menos dos pasos anteriores

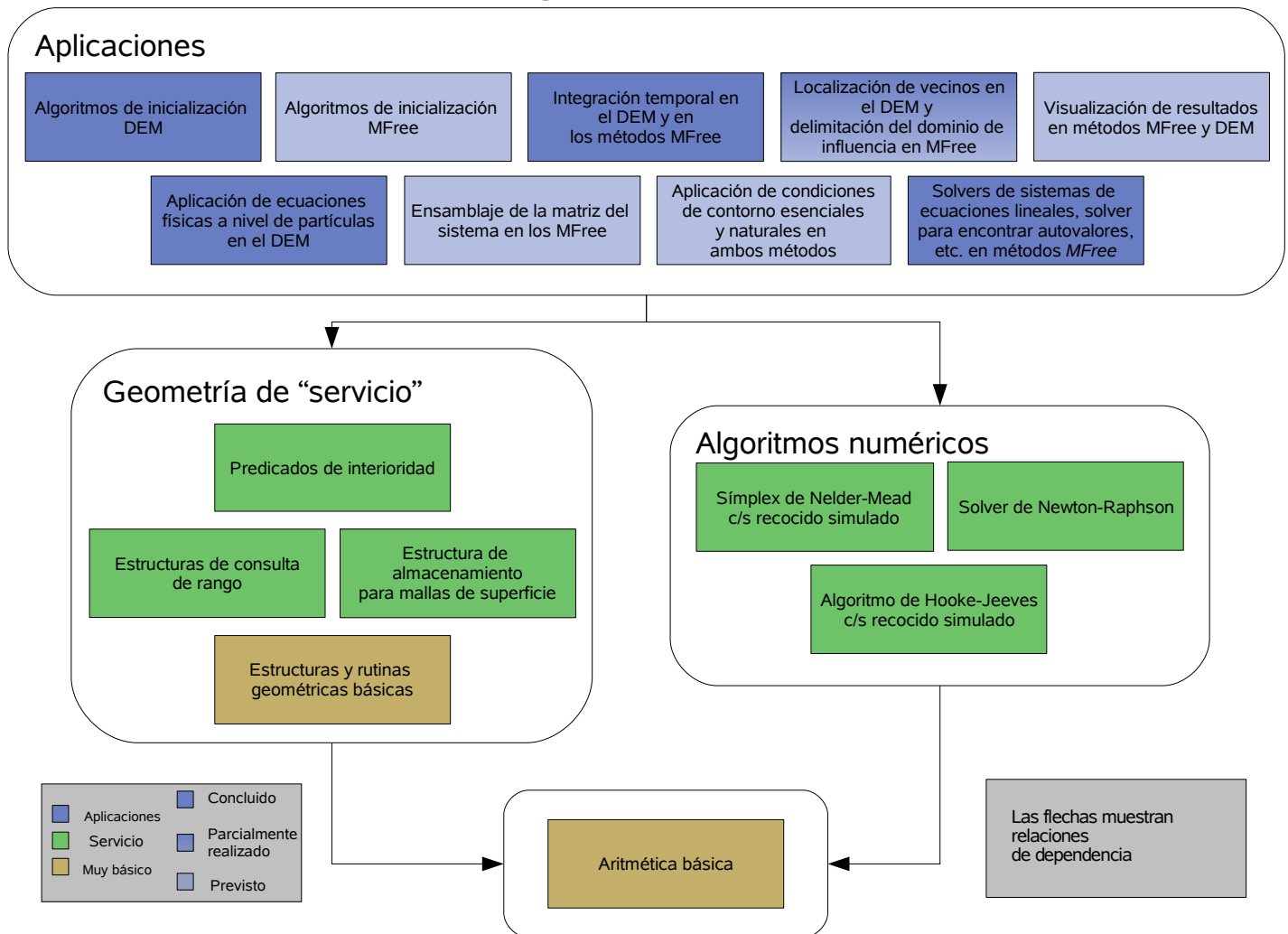
v Ver (Recarey, 1998)

vi Los parámetros que definen las propiedades del cuerpo en un punto son por ejemplo: tensor de tensiones, flujo de calor, entropía, potencial termodinámico, etc.

vii El uso de plantillas en C++ puede hacer insufrible al compilador en términos de tiempo de compilación La CGAL está formada por plantillas.

viii Otro sistema de control de versiones

Capítulo 2 Detalles algorítmicos



2.1 Organización.

NoMS es un producto en evolución. El primer objetivo es lograr un producto de investigación. Como parte del mismo, existe una infraestructura realmente básica, que podría formar parte de otros softwares en aplicaciones muy lejanas o distintas de la actual. Puede esperarse que existan pues, para otros fines, formulaciones e implementaciones sobre estos temas, que las mismos estén bien estudiadas y sean un punto común para muchos desarrolladores. En este epígrafe, estos grupos de conceptos se conocerán como muy básicos.

Ya más cercano al problema existe un grupo de necesidades. En algunos casos las mismas tienen puntos comunes con otras áreas, pero no siempre, y en cualquier caso necesitan ser adaptadas para el propósito de NoMS, y se espera que las aplicaciones finales dependan intrínsecamente de estos servicios.

Sobre las propias aplicaciones, estas representan el valor de uso del software. Son básicamente lo delineado en el primer capítulo respecto a métodos de partículas y sin mallas. La investigación sobre las mismas y su implementación correcta son el objetivo de NoMS.

El diagrama que encabeza este capítulo muestra las partes conceptuales de las que consta el producto, debe tenerse en cuenta que *no* representa la estructura del software, sino de los conceptos y algoritmos necesarios en él. Las secciones en marrón simbolizan la parte más básica, y en este diagrama, vuelve a recalcarse, en cuanto a cuestiones teóricas. Las secciones en verde representan componentes especializados de la armazón lógica.

De los contenidos de las cajas marrón y verde, existen elementos que han sido tomados *verbatim* de la literatura y las implementaciones existentes, mientras que otros han sido adaptados con características novedosas. Los epígrafes de este capítulo están dedicados a profundizar en estos últimos, es decir, en los asuntos donde existe algún aporte o nueva implicación.

Las cajas azules representan las aplicaciones finales del software. Algunas han sido implementadas y probadas satisfactoriamente, otras están en camino en el momento que se escribe este documento, y unas últimas existen meramente en planes. Aunque de hecho las cajas azules representan aplicaciones del producto, su formulación es completamente novedosa en muchos casos; por lo que este capítulo entra en detalles sobre algunas de ellas.

2.2 Estructuras y rutinas geométricas básicas.

En lo que sigue se enuncian algunos predicados geométricos. Son necesarios en el trabajo con mallas, organización de objetos dentro de estructuras de consultas de rango, determinación de distancias, etc. También las aplicaciones finales requieren las estructuras geométricas definidas en este epígrafe, por ejemplo, para representar las partículas o los puntos en los casos respectivos de los métodos DEM y MFree.

En un curso de geometría analítica básica se dan suficientes contenidos como para precipitar a un programador a implementar predicados de geometría computacional con dicha teoría. Sin embargo, la teoría de la geometría analítica no está centrada en las aplicaciones de cómputo y su om-

nipresente precisión limitada. Es por eso que se necesitan enfoques especiales para obtener predicados geométricos robustos numéricamente. En el caso de este epígrafe, se utilizará un enfoque basado en el álgebra vectorial que hasta el momento ha dado muy buenos resultados. Debe notarse que no se encontraron referencias a trabajos similares en la bibliografía.

Los predicados que tienen que ver con distancias, se construyen todos en base al cuadrado de la distancia, lo cual economiza una o más operaciones de radicación en el proceso de cálculos.

2.2.1 El punto.

Un geómetra puede dar varias definiciones de “punto”. No todas son representables para un cómputo de Turing ni tienen el mismo nivel de utilidad. Por ejemplo, la noción de punto que da Euclides en “Elementos” es demasiado intuitiva: “todo lo que es infinitamente pequeño y carente de longitud”.

En el lado opuesto, la palabra “punto” es un concepto abstracto de partida para la axiomática de Hilbert^x, y de primera no tiene nada que ver con números y su representación finita. Aún así, este será el concepto (no la implementación) de punto geométrico que sirva de punto de partida en este documento: “punto” es un ente que cumple con los axiomas de incidencia, orden, congruencia, paralelismo y continuidad resumidos por Hilbert para un espacio geométrico euclídeo.

La realización concreta que se elige para los puntos de un espacio geométrico euclídeo es un espacio vectorial euclídeo de la misma dimensión^x, sea que se denota por Σ . Basándose en el hecho de que todo elemento de un espacio vectorial de dimensión finita puede representarse como una tupla, a cada punto geométrico P n -dimensional puede asociarse una n -tupla sobre el campo escalar de los números reales.

El predicado más notable que involucra puntos solamente es el que tiene que ver con la distancia. Se utiliza el teorema de Pitágoras; y es útil calcular el cuadrado de la distancia, puesto que es un valor que puede reutilizarse en cálculos ulterio-

res y evita hallar una raíz cuadrada.

2.2.2 El vector.

La física conoce desde hace tiempo las magnitudes vectoriales. Se trata de cantidades que se caracterizan por una “dirección”. Geométricamente hablando puede identificarse un vector con un segmento orientado. Sin embargo, “segmento orientado” se corresponde en física con la noción de desplazamiento o distancia, y no encaja bien en la mente identificar una magnitud arbitraria (posiblemente diferencial) con un objeto que va de un punto a otro en el espacio. Por otra parte, la teoría de los espacios lineales introduce los vectores de forma axiomática, y establece que todo vector en un espacio lineal de dimensión finita puede asociarse con una tupla de escalares, asociada con una base.

Con los vectores geométricos se definen las operaciones habituales. Sean dos vectores

$$\mathbf{x} = (x_1, x_2, x_3) \text{ y } \mathbf{y} = (y_1, y_2, y_3) .$$

- El producto de un vector por un escalar:

$$\mathbf{x} t = (t x_1, t x_2, t x_3)$$

- La suma y diferencia :
 $\mathbf{x} + \mathbf{y} = (x_1 + y_1, x_2 + y_2, x_3 + y_3)$;
 $\mathbf{x} - \mathbf{y} = (x_1 - y_1, x_2 - y_2, x_3 - y_3)$.

- El producto escalar
 $\mathbf{x} \cdot \mathbf{y} = x_1 y_1 + x_2 y_2 + x_3 y_3$

- El producto vectorial de dos vectores:

$$\mathbf{x} \times \mathbf{y} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{vmatrix}$$

También es necesario trabajar con el módulo de un vector:

$$\|\mathbf{x}\| = \sqrt{\mathbf{x} \cdot \mathbf{x}}$$

2.2.3 Operaciones entre puntos y vectores.

La separación entre estos dos conceptos, punto y vector, a pesar de que comparten una representa-

ción, se da para simplificar la semántica de las operaciones geométricas. Un punto denota una posición en el espacio, mientras que un vector denota un desplazamiento si se trata de términos geométricos.

Así, puede denotarse el desplazamiento para llegar de un punto a otro como la “diferencia” de los puntos. La operación inversa a la diferencia (paradójicamente), consistente en hallar un punto dado un punto de origen y un vector, podría denotarse con la suma.

Los predicados entre puntos y vectores sirven en multitud de operaciones geométricas.

2.2.4 Recta.

La recta es el lugar geométrico del subespacio lineal generado por un vector y un punto. La representación paramétrica viene dada precisamente por este enunciado:

Los puntos P de una recta que pasa por P_0 vienen dados por

$$P = P_0 + t \mathbf{v} \quad (8)$$

, donde \mathbf{v} es un vector dado.

Se escogió la representación paramétrica, en vez de la canónica, porque puede representar rectas con cualquier orientación, es inherentemente vectorial y no depende explícitamente de la dimensión del espacio geométrico. Esta representación paramétrica es cómoda para asignar coordenadas a los puntos sobre la recta, y de hecho tiene información “de más”, en el sentido de que define una recta orientada y no una recta simple. Los algoritmos que trabajen sobre esta estructura de datos deben tenerlo en cuenta.

Un predicado de interés donde participa la recta es el que tiene que ver con la distancia de un punto a una recta:

Algoritmo 1 Distancia de un punto a una recta

Sea una recta definida por (8) y un punto P

arbitrario. Se desea conocer la distancia entre ellos.

Calcular:

$$\frac{(P - P_0) \times v}{\|v\|} \quad (9)$$

Comentario:

El denominador es el módulo de un vector, lo que requiere que se realice una radicación. Para evitar esto, es más conveniente hallar el cuadrado de la distancia:

$$\frac{[(P - P_0) \times v]^2}{v \cdot v} \quad (10)$$

2.2.5 El rayo.

Data una recta r y un punto A sobre dicha recta, en geometría se conoce por rayo o semirrecta el lugar geométrico de los puntos de r que no son divididos por A . Una recta se divide en dos rayos para cada uno de sus puntos.

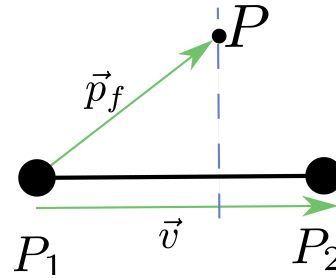
La forma de implementar computacionalmente este concepto es usando la misma expresión (8) pero exigiendo que $t > 0$ (o $t < 0$, se trata de una convención)

2.2.6 Segmento

El segmento se define como el lugar geométrico de los puntos sobre una recta que están “entre” dos puntos dados. El “entre” es el mismo que aparece en los axiomas de orden. Computacionalmente puede representarse con los dos puntos de sus extremos. Una representación finita de los números no permite la implementación del predicado “el punto R se encuentra en el segmento a ”. Por eso este predicado no está implementado. En su lugar, sin embargo, puede implementarse el predicado menos fuerte pero más efectivo “la proyección del punto R cae sobre el segmento a ”:

Algoritmo 2 Proyección de un punto sobre un segmento.

Sea un segmento de P_1 a P_2 . Se quiere saber si la proyección del punto P sobre la recta que pasa por P_1 y P_2 cae dentro del segmento.



Hacer:

$$\begin{aligned} v &= P_2 - P_1 \\ p_f &= P - P_1 \\ s &= \frac{v \cdot p_f}{v \cdot v} \end{aligned}$$

Retornar si s está en el intervalo $(0;1)$

Comentario:

El valor de s es la medida en que el punto está dentro del segmento. Puede calcularse cuál es el punto de proyección exactamente como $P_p = P_1 + s v$

Otro predicado que es necesario con frecuencia es la distancia de un punto a un segmento. Con ayuda del predicado anterior, puede formularse de una manera directa:

Algoritmo 3 Distancia de un punto a un segmento

Sea un segmento de P_1 a P_2 . Se quiere conocer la distancia desde un punto P arbitrario al segmento.

Usando los pasos del algoritmo anterior, puede calcularse s . Hecho esto, puede discriminarse en pasos:

Hacer:

Si s es

- menor que 0 : devolver la distancia desde P hasta P_1
- está entre cero y uno: devolver la distancia desde P hasta la recta que pasa por P_1 y P_2
- mayor que uno: devolver la distancia entre el punto P y el punto P_2

2.2.7 El plano

Es otro de los objetos que presupuestos en el sistema de Hilbert. La realización de este concepto, en concordancia con lo que se ha establecido en las secciones anteriores, debe ser la siguiente.

Para un espacio de dos dimensiones, se supondrá que existe un solo plano que contiene todos los puntos del espacio.

En un espacio de tres dimensiones, la realización del plano se introducirá usando los conceptos de secciones anteriores:

Dado cierto punto P_0 y un vector \mathbf{v} , existe un plano asociado a ellos formado por todos los puntos P tales que:

$$(P - P_0) \cdot \mathbf{v} = 0 \quad (11)$$

Observe que la diferencia de los puntos, en concordancia con lo dicho anteriormente, se asume que es un vector, por lo que el producto escalar tiene sentido.

La definición hecha sirve para identificar con facilidad los semiespacios en los que el plano (11) divide el espacio. En efecto, si un punto se encuentra en el semiespacio abierto I , entonces $(P - P_0) \cdot \mathbf{v} > 0$ y si se encuentra en el semiespacio II , $(P - P_0) \cdot \mathbf{v} < 0$.

La distancia punto-plano puede calcularse con el siguiente procedimiento:

Algoritmo 4 Distancia de un punto a un plano.

Se desea conocer el cuadrado de la distancia entre un plano dado en la forma (11) y un punto arbitrario P .

Calcular:

$$\frac{[(P - P_0) \cdot \mathbf{v}]^2}{\mathbf{v} \cdot \mathbf{v}}$$

Para poder calcular con facilidad la distancia del plano a otras entidades geométricas básicas, se construye un predicado que diga el lugar relativo de la intersección entre el plano y un segmento, aprovechando el sistema de coordenadas que definen dos puntos sobre una recta. Procediendo de esta forma es más fácil luego calcular las coordenadas reales de la intersección, además el procedimiento puede reutilizarse para intersectar rectas, segmentos y rayos de una sola vez. Un detalle de este predicado es que hay que tener cuidado especial para el caso de que los dos puntos definan una recta paralela al plano.

Algoritmo 5 Coordenada de intersección de un plano y una recta

Dado un plano en la forma $(P - P_p) \cdot \mathbf{v} = 0$ y un segmento formado por los dos puntos P_1 y P_2 , hallar la coordenada relativa al segmento $\overline{P_1 P_2}$ en la que el plano y el segmento se cortan.

Hacer:

$$d = (P_2 - P_1) \cdot \mathbf{v}$$

$$n = (P_p - P_1) \cdot \mathbf{v}$$

Si $d \ll n$, tanto como para ser considerada cero, entonces reportar que el segmento y el plano no se intersectan.

En otro caso:

$$\lambda = n/d$$

Retornar λ

Con este algoritmo, es posible reportar la intersección no solo de segmentos, sino de rayos y rectas.

2.2.8 El triángulo

Se trata de la estructura más problemática en sus predicados. Integra las mayas de superficie básicas, y es parte esencial de los predicados de interioridad con estas estructuras. Su carácter acotado lo hace difícil de representar. Para la biblioteca de geometría computacional de NoMS, se escogió representarlo por los tres puntos que lo conforman.

Todos los predicados serán enunciados para el caso tri-dimensional, que es el más complejo. Son automáticamente válidos o triviales para el caso bidimensional, tomando las coordenadas correspondientes iguales a cero.

El predicado más importante implementado es el siguiente:

Algoritmo 6 Relación punto-triángulo

Dado un triángulo por sus tres puntos A_1, A_2, A_3 y un punto arbitrario P , decir la localización de la proyección de este punto P_a en el plano del triángulo con respecto al mismo, y la distancia del punto al triángulo.

Los distritos se determinan tal como se muestra en la figura: de acuerdo a si el punto está en el interior del triángulo, es más próximo a un lado o a un vértice del triángulo. El resultado de este algoritmo debe ser la distancia, el índice del punto característico (o cero para interior) y la clasificación del distrito, en la forma de una de tres constantes VERTICE, LADO, o INTERIOR.

Este es un algoritmo no trivial, que requiere pruebas extensivas, y que no puede documentarse correctamente en un lenguaje matemático simple. Su exposición aquí sería engorrosa. Remitimos al lector al fichero correspondiente en el código fuente para analizar su contenido.

Otro predicado que tiene que ver con el triángulo, que es de importancia en este trabajo, es conocer si un rayo se interseca con un triángulo.

Algoritmo 7 Rayo interseca triángulo

Dado un triángulo por sus tres puntos, conocer si un rayo expresado en la forma 8 con $0 \leq t \leq \infty$ toca al triángulo.

Hacer:

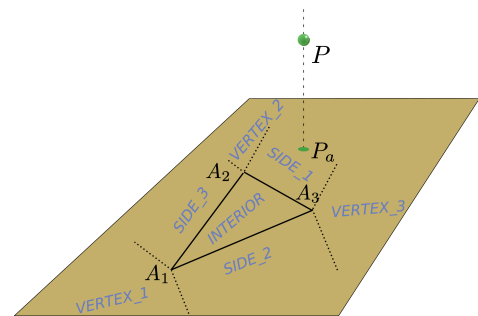
Ejecutar el algoritmo 5 para conocer la coordenada λ de intersección con el rayo del plano que contiene al triángulo. Si esta coordenada Q es negativa, retornar que no ocurre intersección.

En otro caso, utilizar el algoritmo 6 para averiguar la relación de Q con el triángulo, si el distrito es INTERIOR, decretar que ocurre intersección entre el rayo y el triángulo.

El último algoritmo tiene utilidad en predicados que buscan conocer si existe intersección entre mallas. Este algoritmo se basa en una idea simple: si dos triángulos tienen algún punto en común, entonces alguno de los lados de uno de ellos pasa por el interior del otro.

Algoritmo 8 Intersección triángulo-triángulo

Dados dos triángulos por sus sistemas de puntos A_1, A_2, A_3 y B_1, B_2, B_3 , decir si estos tienen algún punto en común.



Hacer:

1. Para $i=1, 2, 3$:

Ejecutar el algoritmo 5 con la recta que pasa por el segmento formado por $A_i A_{i+1}$ y el plano B_1, B_2, B_3 para obtener un punto de intersección Q .

Si existe el punto Q , utilizar el algoritmo 6 para conocer la relación entre Q y el triángulo. Si dicha intersección ocurre, reportar verdadero y terminar.

Algoritmo 8 Intersección triángulo-triángulo

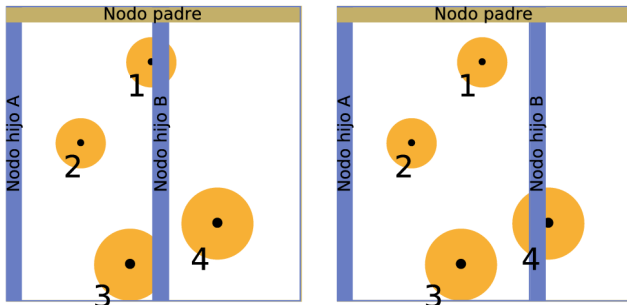
2. Repetir el paso 1 cambiando A_i por B_i ; es decir, invirtiendo los roles de los triángulos.

2.2.9 Paralelepípedo y caja.

En un espacio euclídeo d-dimensional, se denomina paralelepípedo a un conjunto de puntos que se obtiene de la siguiente forma. Sea un conjunto X_0 formado por los puntos (x_1, x_2, \dots, x_d) tales que $a_i \leq x_i \leq b_i$, $a_i \leq b_i$ constantes arbitrarias, con $i=1, 2, \dots, d$. Se denomina paralelepípedo a cualquier objeto que pueda obtenerse de X_0 mediante operaciones de traslación, escalamiento, rotación o esquilamiento.

En geometría computacional, se pueden hacer muchas cosas con el conjunto original X_0 , sin necesidad de transformarlo. Este representa un paralelepípedo con ejes coaxiales. Para distinguirlo del paralelepípedo general, será denominado “caja”.

La representación computacional de una caja puede darse por las dos tuplas (a_i, a_2, \dots, a_d) y (b_i, b_2, \dots, b_d) que coinciden con las coordenadas del punto superior izquierdo anterior (coordenadas menores) e inferior derecho posterior (coordenadas mayores) en alguna orientación relativa del sistema de coordenadas. Otra variante, es dar la primera tupla y la longitud de las aristas. Esta es la que adopta NoMS. La representación de una



Cuando cada nodo hijo tiene exactamente la mitad de uno de los lados del padre.

Cuando los hijos son distintos en proporción

caja, entonces, queda dada por:

$$\text{Una tupla } (a_1, a_2, a_3) \text{ de posición y una tupla } (l_1, l_2, l_3) \text{ de longitud.} \quad (12)$$

El primer predicado de esta sección tiene que ver con el punto medio de una caja. Es bastante simple.

Algoritmo 9 Punto medio de una caja

Dada una caja en la representación (12), se quiere hallar el punto que yace en su centro geométrico

Calcular:

$$(c_1, c_2, c_3) = (a_1 + \frac{l_1}{2}, a_2 + \frac{l_2}{2}, a_3 + \frac{l_3}{2})$$

El otro procedimiento que más o menos se usa con frecuencia es hallar una caja que contenga a las de un conjunto de cajas.

Algoritmo 10 Cubrimiento de un grupo de cajas

Dado un conjunto de cajas B_1, B_2, \dots, B_n , cada una representada en la forma (12) y con tuplas $a_i^k, l_i^k, i=1, 2, \dots, d, k=1, 2, \dots, n$ hallar la caja de menor medida que las contiene a todas.

Hacer:

$$x_1 = x_2 = x_3 = \infty$$

$$y_1 = y_2 = y_3 = -\infty$$

Para k desde 1 hasta n :

Para i desde 1 hasta d :

Si $a_i^k < x_i$ hacer

$$x_i = a_i^k$$

Si $a_i^k + l_i^k > y_i$ hacer

$$y_i = a_i^k + l_i^k$$

Construir una caja con las tuplas

(x_1, x_2, x_3) y
 $(y_1 - x_1, y_2 - x_2, y_3 - x_3)$ y retornarla.

De bastante utilidad en las estructuras de detección de contacto se considera un predicado que pueda decir si dos cajas se interceptan.

Algoritmo 11 Intersección entre cajas; respuesta booleana.

Sean dadas dos cajas en la forma (12), con la siguiente notación:

- Para la caja 1: las coordenadas menores (a_1, a_2, a_3) y las longitudes de aristas (g_1, g_2, g_3) .
- Para la caja 2: las coordenadas menores (b_1, b_2, b_3) y las longitudes de aristas (h_1, h_2, h_3) .

Se desea conocer si las dos cajas tienen al menos un punto en común.

2.3 Estructuras de consulta espacial.

Las estructuras de consulta espacial permiten localizar objetos en el espacio. Es posible establecer relaciones entre los distintos tipos de consulta espacial. En particular, para el funcionamiento de

NoMS pueden tomarse como base estructuras que implementan las consultas intervalo-intervalo. Se trata de una clase de objetos que puede responder a la siguiente pregunta básica:

Dada una caja B_0 en el espacio, ¿cuáles son los objetos que se solapan con ella?

Para formalizar con mayor facilidad, se hará alusión frecuente al concepto de *tipo*. Es necesario notar que *tipo* tiene significados distintos en teoría de conjuntos, lógica formal y computación. Dentro de la propia rama de la computación tiene varios significados. En este epígrafe se utilizará un concepto propio con muchos elementos intuitivos:

Sea que existe un conjunto de objetos sobre los cuales pueden definirse operaciones. Entonces, cada objeto puede ponerse en una o más categorías, denominas tipos, de forma tal que todos los elementos de una categoría tengan definido cierto subconjunto mínimo de operaciones.

En NoMS, las estructuras de consulta de rango se utilizan a través de una interfaz, lo que permite variar la implementación de forma transparente. Hasta el momento, NoMS tiene implementado dos estructuras de consulta intervalo-intervalo, y encima de ellas un adaptador que hace posible realizar consultas de punto y de rango. Como estas consultas se conocen en general por “consultas de rango”, la propia estructura será referida por momentos con este nombre.

Una de las estructuras básicas en NoMS se basa en una tabla de dispersión, y la otra en un árbol.

Estas estructuras están orientadas al trabajo permanente en memoria. Existen en la documentación (Argawal, 1997; , Berckman, 1990) referencias a estructuras creadas para el trabajo en medios de almacenamiento secundario, se tiene pensado incorporar alguna de ellas a NoMS próximamente.

2.3.1 Estructura de consulta por tabla de dispersión.

La idea de esta estructura estriba en crear regiones de tamaño fijo para almacenar puntos de registro

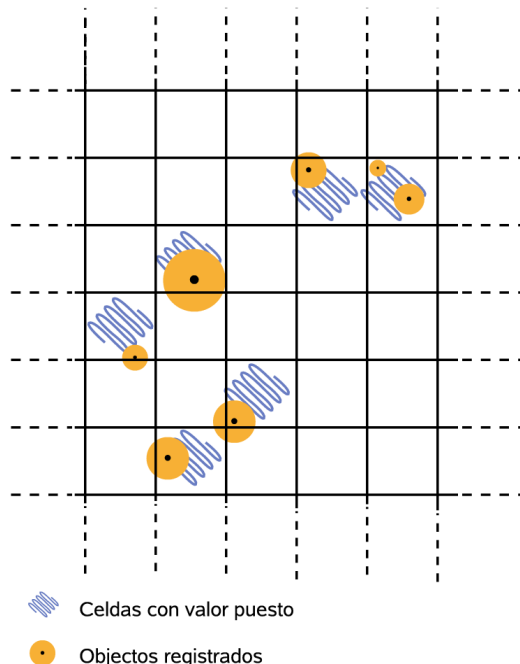


Ilustración 1: Estructura de consulta de rango por tabla de dispersión

de los objetos que potencialmente entran en la consulta. Esta idea aparece en los trabajos de *Mujinza* (ver *Mujinza* 1999), quién la proclama superior a los árboles usados tradicionalmente.

De regiones de registro se elige una partición^{xi} de cajas en el espacio. En la variante más simple, todas las aristas tienen el mismo tamaño, y por tanto se trata de cuadrados/cubos (dependiendo de la dimensión del espacio).

En el esquema se muestra la disposición para un espacio de dos dimensiones.

La ventaja de este esquema es que solo se utiliza almacenamiento para las celdas que tienen algún contenido (marcadas en azul en el diagrama), puesto que el *grid* solo existe de forma lógica. Su principal desventaja es el tamaño fijo de las celdas, que restringe el tamaño relativo de los objetos contenidos.

A continuación se da una definición formal utilizando el caso particular de tres dimensiones. La adaptación a dos dimensiones es directa:

1. Se escoge un tamaño de celda mayor que la caja circundante del mayor de los objetos destinados al almacenamiento, por ejemplo, r , y un origen de coordenadas, que será denotado por $P_0 = (p_x, p_y, p_z)$.
2. Para toda tupla de números enteros (n_x, n_y, n_z) se considera que hay una celda formada por una caja con coordenadas menores $(p_x + n_x r, p_y + n_y r, p_z + n_z r)$ y longitud de arista r .
3. Se escoge una función de dispersión $\varphi: \mathbb{Z}^3 \rightarrow \mathbb{N}$. Esto es muy sencillo y se darán mayores detalles en el capítulo 3.
4. Se crea un diccionario basado en una tabla de dispersión cuyas llaves sean tuplas de enteros de la forma descrita en el punto anterior. Los valores del diccionario son contenedores generales para el tipo T de objetos que se quieren almacenar. En adelante, se denotará con el mismo símbolo T el conjunto formado por todos los

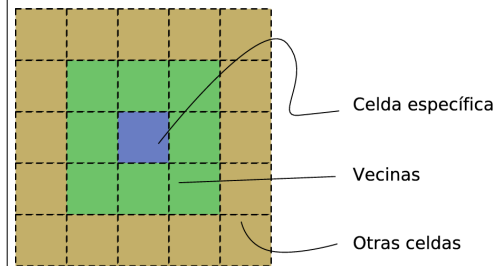
objetos de dicho tipo.

5. Se construye una función $\pi: T \rightarrow \mathbb{R}^3$ que permita asociar a cada objeto un punto de registro, y una función Λ que dado un objeto $t \in T$ devuelva la caja de medida (volumen, en tres dimensiones) mínima que contenga completamente a t .

Con esto es suficiente. La tupla de objetos enumerados $D = (P_0, r, \varphi, T, \pi, \Lambda)$ será conocida como estructura de consulta de rango por tabla de dispersión. A continuación se detallan los algoritmos necesarios para manejar dicha estructura.

El primero de ellos tiene que ver con la posibilidad de listar todas las celdas que son vecinas inmediatas de una celda dada; es un algoritmo auxiliar. Se dará solo el objetivo del algoritmo, puesto que su implementación es trivial.

Algoritmo 12 Hallar el conjunto de celdas vecinas a una celda dada.



Dada una celda B_i , se desean hallar todas sus vecinas, tal como se muestra en la figura.

El siguiente algoritmo tiene que ver con la inserción de objetos en la estructura. La lógica de este algoritmo es bastante simple.

Algoritmo 13 Inserción de objetos en una estructura de consulta de rango por tabla de dispersión.

Sea que se quiere insertar el objeto t en la estructura

Hacer:

Hallar el punto de registro $P_t = \pi(t)$

del objeto dado.

Hallar una tupla de enteros, correspondiente a P_t . Para hallar esta tupla de enteros, tomar cada uno de los números

$$r_t^i = \frac{x_t^i}{r} . \text{ Estos números son en general}$$

reales. Cada real está entre dos enteros, sea el n^i el entero a la izquierda de r_t^i . Con esto, es posible construir la tu-

$$\text{pla } K = (n^1, n^2, n^3) .$$

Utilizar la tupla del párrafo anterior como llave K en la tabla de dispersión. Si

K ya forma parte de la tabla de dispersión, su valor es una lista. En esta lista se agrega t . Por otra parte, si K no forma parte de la tabla de dispersión, se crea una nueva lista con t como único elemento y se agrega a la tabla de dispersión bajo la llave K .

Comentario:

No hay nada fuera de lo común en la rutina de

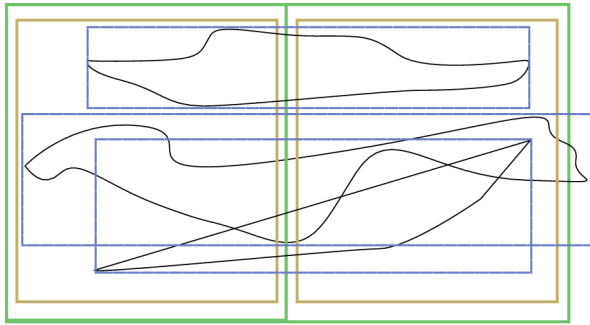


Ilustración 2: Caso en que una subdivisión no mejora la cantidad de objetos en los contenedores de los nuevos nodos hoja.

inserción. Puede modificarse esta rutina para agregar t a todas las celdas que se solapan parcialmente con el objeto. En ese caso, a costo de mayor consumo de memoria, se obtiene una leve ganancia en rapidez en el proceso de búsqueda.

El algoritmo de consulta de intervalo-intervalo

funciona obteniendo todas las celdas afectadas por la caja de consulta y sus vecinas, y revisando luego los objetos en dicha caja, de forma individual, uno por uno.

Algoritmo 14 Consulta intervalo-intervalo

Dada una caja $B = ((b_1, b_2, b_3), (l_1, l_2, l_3))$ y una estructura de consulta de rango $D = (P_0, r, \varphi, T, \pi, \Lambda)$, reportar el conjunto de objetos $\{t_i\}$ que se solapan al menos parcialmente con la caja B .

Hacer:

Hallar el conjunto de cajas S_1 que tocan al menos parcialmente a B . Esto puede hacerse con d ciclos anidados, siendo d la cantidad de dimensiones del espacio; cada uno de los ciclos se comienza en el valor entero inmediatamente inferior a $\frac{b_i}{r}$.

Hallar el conjunto de cajas S_2 formado por S_1 y por todas las vecinas a cada caja de S_1 ; esto se logra utilizando el algoritmo 12 para cada una de las celdas y uniendo conjuntos luego.

Con todos los objetos $\{t_i\}$ en el conjunto de celdas S_2 hacer:

si $\Lambda(t_i) \cap B \neq \emptyset$ (usar algoritmo 11) añadir t_i al conjunto de retorno.

Estos son los dos algoritmos esenciales a formular sobre una estructura de consulta de rango, aunque no son los únicos. En la práctica también hace falta un algoritmo para eliminar elementos ya insertados, o para enumerar todos los elementos de la estructura. Estos no se detallan aquí; pueden realizarse fácilmente con la información provista hasta el momento y son operaciones triviales, para más detalles ver el código fuente.

Un comentario final sobre esta estructura. El uso de una tabla de dispersión permite realizar las operaciones de localización de las celdas en un tiempo mínimo (complejidad unitaria). Esto hace

eficiente a esta estructura de búsqueda para espacios increíblemente grandes. Sin embargo, el número de celdas que es necesario procesar entonces, aún para una región de consulta pequeña, es significativo.

2.3.2 Estructura de consulta de intervalo-intervalo por Kdtree.

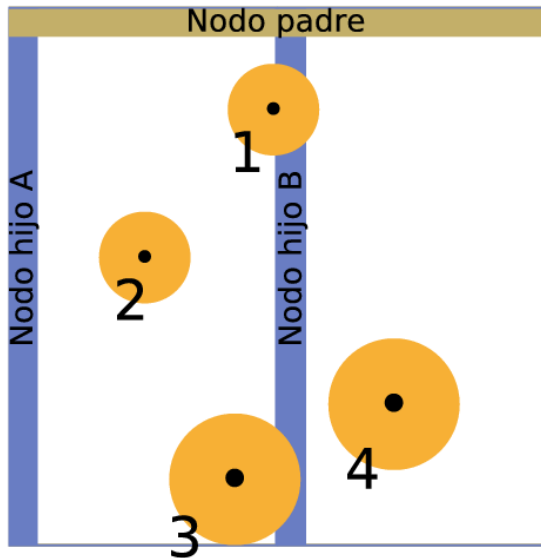
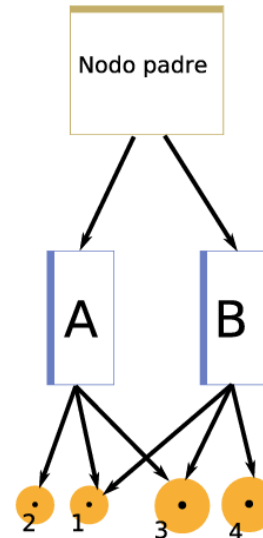


Ilustración 3: A la izquierda, la forma en que se distribuye un rectángulo entre los nodos de un kdtree. A la derecha, la forma en que queda el árbol asociado.

definitiva son las que distinguen una implementación de otra. En la segunda parte se detallan las políticas particulares que hacen del kdtree implementado en NoMS una estructura con ventajas notables sobre sus predecesores.

Kdtree genérico.

La idea detrás de esta estructura de datos es aso-



Primero una nota sobre el nombre: proviene del inglés pero allí tampoco tiene sentido, es una regularización que se ha hecho del nombre original, que venía siendo *k-dimensional-tree*. En este documento se usará *kdtree*, que es una notación breve y conveniente.

Esta estructura se basa en crear una partición jerárquica [limitada] del espacio. En el caso de la implementación que se presenta aquí, el kdtree trabaja sobre un área/volumen base, que tiene forma de caja. En cuanto a estructura de datos, la representación del kdtree es un árbol binario. Todos los nodos representan una caja. Si un nodo *b* es hijo de un nodo *a* entonces la caja de *b* se encuentra contenida en la caja de *a*.

Esta sección se divide en dos partes. En la primera, se introduce de forma general un kdtree, definiendo puntos de inserción para políticas^{xiii} que en

ciar a cada nodo de un árbol una caja, y que la relación de paternidad desde el punto de vista del árbol, sea una relación de inclusión desde el punto de vista de las cajas asociadas a los nodos. Esta idea es compartida con otra estructura, conocida como quadtree/octree. La diferencia entre ambas es el número de hijos que tiene cada nodo. En el caso del kdtree son solo dos, mientras que en el quadtree/octree son cuatro/ocho, dependiendo de la cantidad de dimensiones del espacio (de ahí deriva el nombre de esta última estructura). Sucede que en el kdtree, los hijos de un nodo representan la división de la caja del nodo padre según un eje; dicho eje se va alternando por niveles de profundidad.

Estas políticas importantes definen el comportamiento de un kdtree:

1. La proporción relativa η de la división de la caja asociada a un nodo con respecto a los nodos hijos. Esta decisión puede depender de la distribución particular de los objetos en el momento de la división, puede ser un número fijado de antemano o puede variar dinámicamente.
2. En qué momento se considera que un nodo hoja está lleno y debe subdividirse. Esta decisión puede depender de la cantidad de objetos que existen en el nodo en el momento de la división, o de los resultados potenciales de la propia división.
3. En qué momento se considera que dos nodos hoja hermanos están lo suficientemente vacíos y deben unirse en uno solo, sustituyendo al padre por un nodo hoja. Esta decisión es similar a la del paso anterior, y puede depender de los mismos factores.

Las tres políticas serán conocidas, de forma abreviada por los caracteres p_η , p_s y p_j . También es necesario asignar una caja para el nodo raíz, sea esta B_0 , un tipo T de objetos contenidos, y una función Λ que dada una instancia de T devuelve una caja. La configuración del sistema puede anotarse como una tupla:

$$E = (p_\eta, p_s, p_j, T, \Lambda) \quad (13)$$

Las tres primeras componentes en esta tupla son los ajustes que se le pueden hacer a cada implementación del kdtree, la cuarta es una instancia de datos, y la quinta es una función asociada al tipo de los datos del contenedor.

La organización de un árbol se realiza en nodos. En el caso de un kdtree, existen dos tipos de nodos: los que son internos, y los que son nodos hoja. Los nodos internos no tienen instancias de T como hijos, solo nodos hoja. Los nodos hoja, por el contrario, no tienen otros nodos de hijos, y sí tienen asociado cada uno un contenedor de objetos del tipo T . Los algoritmos de inserción y consulta se definen de forma recursiva sobre la estructura de nodos.

El algoritmo de inserción ilustra el comportamiento recursivo del kdtree:

Algoritmo 15 Inserción de objetos en una estructura de búsqueda por kdtree.

Dada una instancia t de T y una estructura de búsqueda en la forma (13), insertar el objeto t en la estructura de búsqueda.

Comentario:

La naturaleza recursiva del algoritmo hace más natural que este se enuncie en dos partes, una para los nodos internos del árbol y otra para los nodos hoja.

El preámbulo de la inserción es encontrar la caja que limita al cuerpo:

- $B = \Lambda(t)$

Nodo interno:

Los nodos internos no tienen responsabilidad en insertar directamente las instancias de T , sino en delegar esta operación a sus hijos. Sea un nodo interno N_I y sus dos nodos hijos:

N_1 y N_2 . Sea que la caja asociada a N_i es B_{N_i} , y la de sus nodos hijos B_{N_1} , B_{N_2} :

- Si B se interseca con B_{N_1} (cosa que puede determinarse utilizando el algoritmo 11) invocar el algoritmo de inserción (este algoritmo) sobre N_1 .
- Si B se interseca con B_{N_2} invocar el algoritmo de inserción (este algoritmo) sobre N_2 .

Nodo hoja:

Aquí es donde realmente tiene lugar la incorporación del objeto t a un contenedor. Sea

N_H el nodo hoja, y $C_T^{N_H}$ el contenedor de objetos asociado a este nodo.

- Si al agregar el objeto t se cumple p_s , es necesario:
 - convertir N_H en un nuevo subárbol, formado por un nuevo nodo interno y dos nuevos nodos

hojas, tal como se muestra en la figura. Las cajas asociadas a los nuevos nodos hoja se obtienen dividiendo la caja original por el eje correspondiente al nivel y según la política p_η . En esta operación, deben redistribuirse los objetos asociados al nodo original insertándolos (invocación recursiva de este algoritmo) desde el nuevo nodo interno.



- Insertar el objeto t en el nuevo nodo interno, utilizando recursivamente este algoritmo..
- Agregar el objeto t al contenedor C_T .

Este es el algoritmo de consulta de intervalo-intervalo para el *kdtree*:

Algoritmo 16 Consulta intervalo-intervalo en el *kdtree*

Dada una caja de consulta B y una estructura en la forma (13), reportar todas las instancias de T contenidas al menos parcialmente en B .

Comentario:

La naturaleza recursiva del algoritmo hace más natural que este se enuncie en dos partes, una para los nodos internos del árbol y otra para los nodos hoja.

Nodo interno:

Los nodos internos no tienen responsabilidad en reportar directamente las instancias de T , sino en delegar esta operación a sus hijos. Sea un nodo interno N_I y sus dos nodos hijos: N_1 y N_2 . Sea que la caja asociada a

N_I es B_{N_I} , y la de sus nodos hijos B_{N_1} , B_{N_2} :

- Si B se interseca con B_{N_1} (cosa que puede determinarse utilizando el algoritmo 11) invocar el algoritmo de consulta (este algoritmo) sobre N_1 .
- Si B se interseca con B_{N_2} invocar el algoritmo de consulta (este algoritmo) sobre N_2 .
- Reunir el resultado de la búsqueda sobre los nodos N_1 y N_2 y reportarlo.

Nodo hoja:

Estos son los verdaderos responsables de reportar qué instancias actualmente se encuentran contenidas en B . Sea que el nodo hoja N_H tiene contenedor asociado C^{N_H} , y que dicho contenedor referencia los objetos t_1, t_2, \dots, t_n :

- Inicializar la lista de resultados L en
- vacío
- Para t_i con $i=1, 2, \dots, n$ hacer:
 - Encontrar $B_i = \Lambda(t_i)$
 - Si al aplicar el algoritmo 11 este indica intersección entre las cajas B_i y B , entonces agregar t_i a la lista de resultados.

No se detallará el algoritmo de eliminación del *kdtree*, por cuanto es similar y funciona en el mismo espíritu que los dos detallados hasta aquí. Baste notar que en dicho algoritmo se utiliza la última de las políticas enumeradas, p_j , para decidir en qué momento tiene lugar el colapso de dos nodos hoja hermanos y su padre en un solo nodo hoja.

Kdtree en NoMS.

El *kdtree* en NoMS es una estructura de datos implementada desde cero, con políticas fijadas de antemano. Esta segunda parte describe brevemente las políticas concretas, dictadas por los siguientes requerimientos:

1. Los objetos contenidos tienden a variar en

el tiempo.

Sobre la política de proporción de subdivisión

p_n : es fija; cuando un nodo se subdivide sus hijos tienen cajas asociadas idénticas entre sí. Es

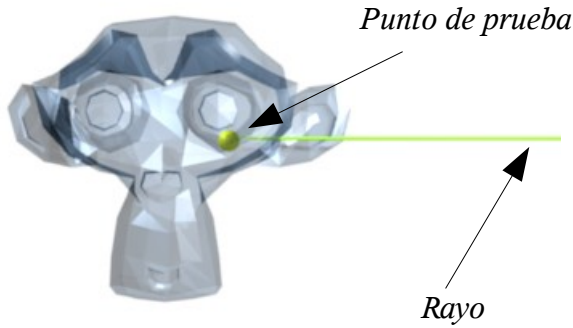



Ilustración 4: Malla y punto interior

decir $\eta = \frac{1}{2}$. La ventaja de esta aproximación es que es óptima en el caso promedio, simple y eficiente, atendiendo a la variabilidad de la localización de los objetos en el kd-tree durante el tiempo de vida de la estructura y los algoritmos de aplicación que la utilizan.

Sobre la política del momento de subdivisión

p_s : una primera aproximación, paramétrica y por tanto con cierto grado de flexibilidad, consiste en establecer una cantidad tope de objetos K en el contenedor asociado con cada nodo hoja. El problema con este enfoque es que en algunos casos la subdivisión no disminuye sensiblemente la cantidad de objetos en cada uno de los nuevos nodos hoja, ver figura 2. La solución es mantener un contador con la cantidad de objetos que permanecerían en ambos nodos, k_{12} , y observar antes de hacer una subdivisión que se cumpla

$\frac{k_{12}}{K} \leq q < 1$, siendo q cierto umbral de subdivisión. 

2.4 Predicado de interioridad para mallas.

Saber si un punto es interior a un cuerpo es otro

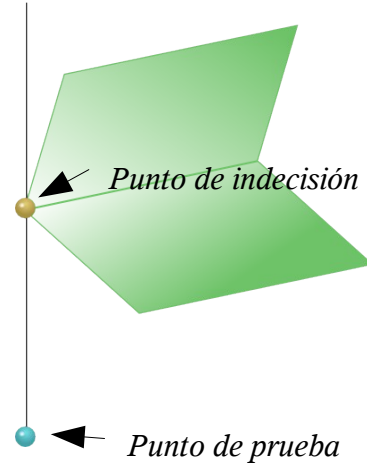


Ilustración 5: Punto de indecisión en cuanto a conteo de intersecciones

de los problemas geométricos presentes en la construcción del sistema. En teoría, NoMS podría usar no solo mallas, sino también CSG^{xiii} para delimitar los cuerpos de interés. Pero hasta ahora solo ha habido tiempo para ocuparse de uno de los dos tipos de contornos, y se prefirió comenzar desde el inicio con un trabajo consistente con mallas de superficies, que es lo que exporta la mayor parte de los softwares de diseño asistido por computadora.

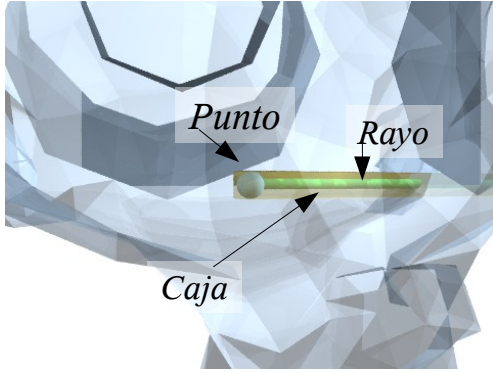


Ilustración 6: Fragmento de la intersección rayo - malla, donde se muestra la caja asociada en naranja.

En una aproximación intuitiva el problema puede plantearse como encontrar si un punto es interior a una malla de superficie o no. Se consideran mallas de “buen comportamiento”: sin puntos múltiples, completamente cerradas, etc. Este tipo de mallas se conoce como mallas STL, y deben distinguirse especialmente de las mallas de superficie necesarias en la simulación con el método de elementos finitos.

Existen varios algoritmos para implementar este predicado. El más simple y conocido procede enviando un rayo desde el punto de prueba en cualquier dirección y contando el número de intersecciones con la malla; si es par se considera que el punto es exterior, y si es impar, se considera que el punto es interior. Este algoritmo tiene varias desventajas, la más criticada es que puede darse el caso de que el rayo sea toque a la malla en un conjunto no numerable de puntos (por ejemplo, el rayo coincida durante parte de su trayectoria con un segmento o una cara de la malla), o que toque a la malla sin atravesarla (ver ilustración 5). Si el ordenador funcionase con aritmética exacta, la probabilidad de una eventualidad de este tipo para una malla STL sería infinitamente más pequeña que la probabilidad de funcionamiento correcto del algoritmo. Aún con aritmética inexacta (simple doble precisión), nunca se ha observado que el algoritmo falle debido a un evento de este tipo. Para tener seguridad de que no ocurran problemas debido a direcciones “especiales”, el rayo en la implementación en NoMS siempre se toma en

una dirección ligeramente perturbada con respecto a uno de los ejes de coordenadas.

La otra desventaja de este algoritmo, es que en teoría es necesario analizar todas las caras de la malla buscando intersecciones. La implementación en NoMS se ahorra este problema, gracias al uso de las estructuras de consulta de rango introducidas en el epígrafe anterior. El algoritmo completo se detalla a continuación:

Algoritmo 17 Determinación de si un punto es interior a una malla.

Se desea conocer si los puntos de un conjunto $\{P_1, P_2, \dots, P_m\}$ con coordenadas $P_i = (x_i, y_i, z_i)$, $i=1, 2, \dots, m$ yacen en el interior de una malla formada por un sistema de triángulos.

Comentario:

Una de las variables libres que toma la solución es un número k que puede servir de índice de coordenadas; es decir, para un espacio tridimensional $k \in \{1, 2, 3\}$; otro parámetro de la solución sin demasiada influencia es una constante positiva pequeña ϵ . Para clarificar un tanto la forma del algoritmo, se supondrá que k tiene un valor fijo $k=1$. El problema se ha presentado con relación a un sistema de puntos porque en efecto, en las aplicaciones se trata de conocer la respuesta para unos cuantos puntos y no uno en particular. Solo así tiene sentido la etapa de:

Preproceso:

- Preparar una estructura de búsqueda por consulta de rango D útil para el tipo de los triángulos. Ver epígrafe 2.3 sobre estructuras de consulta de rango.
- Para $i=1, 2, \dots, n$ hacer:
 - Insertar T_i en D .

Predicado para cada punto $P_i = (x_i, y_i, z_i)$:

- Para el punto P_i y cierto número k construir una caja B_i con coordenadas menores $Q_- = (x_i - \epsilon, y_i - \epsilon, z_i - \epsilon)$ y coordenadas mayores

$$Q_+ = (\infty, y_i + \epsilon, z_i + \epsilon)$$

- Usando una consulta intervalo-intervalo sobre la estructura D , obtener todos los triángulos que se solapan al menos parcialmente con la caja B_i . Sea este conjunto S_1 .
- Está claro que los únicos triángulos que pueden tocar al rayo que parte desde el punto P_i en la dirección $(1, 0, 0)$ se encuentran en el conjunto S_1 . Contra estos triángulos y el mencionado rayo, se cuenta la cantidad n_i de intersecciones, usando el algoritmo 7.
- Si n_i es par, se retorna que el punto está fuera de la malla, y si es impar, se retorna que el punto está dentro de la malla.

2.5 Conclusiones parciales.

En este capítulo se muestran algunas de las soluciones algorítmicas a tareas imprescindibles en el

área de la simulación con método de partículas y método sin mallas. Específicamente, en la parte de geometría computacional las soluciones intentan ser sencillas y generales, esperando obtener con esta filosofía un comportamiento robusto.

En la parte de estructuras de consulta de rango se ilustra el funcionamiento del *kdtree*, que es la estructura usada hasta este momento en NoMS. Esta rama, sin embargo, requiere un esfuerzo de investigación e implementación posterior y más profundo, puesto que es el cuello de botella del procesamiento en la mayoría de las aplicaciones.

ix Citado en (Efrémov, 1998).

x Aquí se conocerá como “espacio geométrico” al conjunto de objetos que cumplen los axiomas de cierta geometría, y no es necesariamente un espacio vectorial.

xi Partición: dado un conjunto S , se denomina partición de S a un sistema de conjuntos T_1, T_2, \dots que cumplen

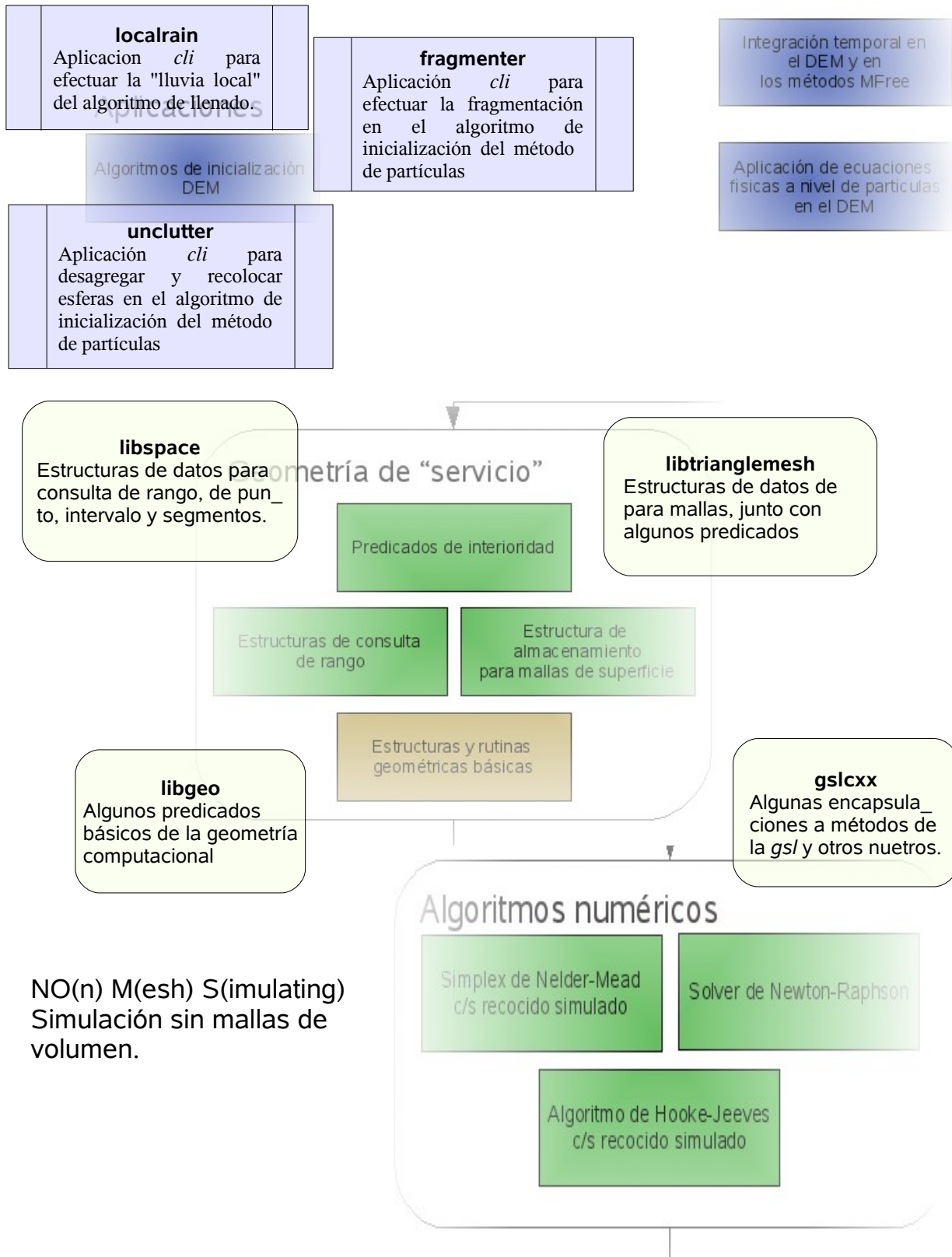
$$S = T_1 \cup T_2 \cup \dots \text{ y para } i \neq j \text{ se tiene}$$

$$T_i \cap T_j = \emptyset$$

xii Política: viene del argot de los patrones de diseño en programación. Entiéndase “forma de hacer las cosas”.

xiii CSG: figura definida por la aplicación de operadores booleanos a primitivas geométricas básicas.

Capítulo 3 Notas sobre implementación



3.1 Organización del capítulo

Este no es un capítulo para un manual de usuario. No existe un manual para una interfaz de usuario en NoMS, al menos hasta ahora, dado el carácter del software. En su lugar existe una guía para el desarrollador y una referencia. Ambos forman un documento aparte de más de 400 páginas. Dicho documento contiene, entre otras cosas, los diagramas de clases del proyecto. Por favor, el lector interesado consultar (NoMS, 2006).

Este capítulo tiene dos objetivos principales. El primero y más importante trata de documentar todas las decisiones de desarrollo relativas a NoMS, desde el punto de vista del especialista en computación. El segundo objetivo es introducir la estructura general, en cuanto a servicios y por directorios del código fuente, del proyecto.

El primer epígrafe trata sobre la cuestión de las licencias, y es un complemento natural al epígrafe correspondiente en el capítulo 1. A continuación se explica por qué se escogió C++ como lenguaje de programación, y las implicaciones. El siguiente paso tiene que ver con una descripción pormenorizada de las bibliotecas externas que forman el basamento de NoMS. También se da alguna información sobre las herramientas satélites que utiliza y promueve el producto. Por último, se explican algunas cuestiones más técnicas, como el asunto del manejo de memoria.

En el segundo objetivo de este capítulo, se presenta un diagrama y una lista, ambos orientados a esclarecer, a grandes rasgos, la estructura física y lógica del proyecto. Una vez más, se hace necesario dirigir al lector interesado al manual de referencia, donde podrá encontrar diagramas y tablas más detallados, además de una pormenorizada descripción de cada una de las clases, archivos y directorios que conforman el proyecto.

3.2 Software sin ataduras

En el momento en que se escribe este documento, NoMS es un producto en desarrollo y sin demasiado valor para usuarios finales. Si alcanza, con el tiempo, el grado de madurez necesario, podría

ser de interés a una audiencia un poco más amplia, quizá formada por grupos especializados dedicados al diseño y a gestión de obras de ingeniería. Estos grupos están formados mayormente por personal interesado en el valor de uso del producto y no en su desarrollo, por lo que para tales usuarios NoMS debe ser distribuido utilizando un esquema comercial.

Por otra parte, muchas de las partes constitutivas de NoMS podrían ser de interés a una comunidad mucho más amplia con habilidades de programación. En ese caso está la biblioteca de geometría computacional y la biblioteca de estructuras de consulta espacial. Sería deseable darle a esas piezas vida propia, liberándolas quizá bajo un modelo de código abierto. Esto, por el momento, solo son ideas y planes. Las decisiones finales probablemente no dependen de quién escribe.

La consideración de estos aspectos proscribire el uso de dos tipos de licencias. En primer lugar no pueden usarse licencias comerciales, hay demasiadas razones para no usarlas. En segundo lugar, no pueden usarse productos bajo licencia GPL, si es que se desea alguna variante de distribución comercial.

Sin embargo, la consideración de una gama importante de licencias abiertas intermedias está bien. Por ejemplo, no hay problemas con usar LGPL, MIT, etc.

En lo que sigue, se analiza la licencia que acompaña a cada una de las dependencias de NoMS. Sobre el propio producto, es todavía código interno y no existe una decisión sobre qué modelo de distribución podría seguir, pero como ya se dijo, probablemente distintas partes sigan modelos distintos.

3.3 Lenguaje de programación, C++.

La programación orientada a objetos ha probado ser una forma efectiva de atacar el problema de la complejidad del software y de gastar recursos de cómputo. En la elección de un lenguaje de programación es necesario alcanzar un compromiso entre ambos aspectos. En ese sentido, C++ es una

buena opción, que además goza de gran popularidad. Su trabajo con plantillas, si bien algo primitivo y fuera de época, no deja de ser esplendorosamente potente.

La combinación plataforma-compiler escogida fue *linux/gcc*, por ser los más barato y potente razonablemente disponible. El compilador de C++ presente en *gcc* es una de las implementaciones más completas y conformes al estándar existentes hasta el momento. Su licencia es GPL, pero los trabajos compilados con él no son cubiertos por dicha licencia.

3.4 Bibliotecas empleadas

3.4.1 Boost

Boost (BOOST, 2006) es un conjunto de bibliotecas para C++, mayormente basadas en plantillas. Cubre un amplio rango de aplicaciones. Dentro de NoMS, sus usos fueron los siguientes:

- Gestión de objetos con propiedad compartida: *shared_ptr*, *intrusive_ptr* y *weak_ptr*.
- Persistencia: *boost.serialization*.
- En las herramientas *cli*, para procesar la línea de comandos con *boost.program_options*.
- Para el trabajo con clausuras, utilizando *boost.function*.
- Para reportar progreso en el cómputo, utilizando *boost.progress_display*.
- Para hacer interfaz con lenguajes scripts de alta productividad, con *boost.python*

La experiencia con *boost* fue enriquecedora, en cuanto permitió profundizar en la programación orientada a conceptos y la metaprogramación.

3.4.2 Gsl

Excelente biblioteca de métodos numéricos. Habrá que prescindir de ella en un futuro cercano, puesto que se distribuye bajo licencia GPL y en este caso los términos de la misma aplican efectivamente en NoMS. Muchos de los algoritmos que

inicialmente se adoptaron de ella ya han sido sustituidos por implementaciones propias. En particular, en algún momento se usaron algoritmos de programación convexa y búsqueda directa. En estos momentos todavía se emplea de la misma lo siguiente:

- La implementación del generador de números aleatorios. No es un problema cambiarla, puesto que el algoritmo es público y la sentencia de *copyright* de su autor explícitamente lo permite.
- El adaptador para obtener distribuciones de números aleatorios.
- El algoritmo de Nelder-Mead.

En algún momento alguien dijo que LAPACK es mucho más eficiente en trabajos con matrices que *gsl*, que por qué no usar LAPACK en su lugar. Es una consideración perfectamente válida, que se tendrá en cuenta tan pronto como se comiencen a utilizar matrices en NoMS.

3.4.3 Base de datos empotradas. Sqlite

NoMS no tiene, al menos hasta el momento, necesidades complejas de almacenamiento/recuperación de datos, al menos de la forma en que lo hacen las bases de datos relacionales. Todo lo más que ha hecho falta hasta ahora es un sistema cómodo para almacenar sistemáticamente los resultados de corridas y juegos de parámetros de calibración. En el futuro será necesario un sistema para organizar de forma cómoda para el usuario los resultados de cómputos parciales.

A estas necesidades se ajusta perfectamente *Sqlite* (SQLITE, 2006), una biblioteca para bases de datos empotradas. Se caracteriza por su eficiente sencillez, aún cuando la funcionalidad que brinda ha sido más que suficiente para las necesidades explicadas. Un rasgo positivo y necesario a partir de la versión 3 de esta biblioteca es la posibilidad de trabajar con *blobs*, es decir, datos binarios sin estructura.

Sqlite no se distribuye bajo ninguna licencia, solo bajo una sentencia de *copyright* bastante permissi-

va.

El principal problema que se encontró durante el trabajo con esta biblioteca fue que está diseñada para C, y en general el trabajo con ella en su forma más directa es un tanto engorroso, cuando menos; además el manejo del tiempo de vida de los recursos es completamente manual. Para darle la vuelta a estas cuestiones fue necesario escribir dos capas, una encima de otra y ambas encima de *Sqlite* que la encapsularan para C++ y *Python* respectivamente. Es lo que en el proyecto se conoce como *pysqlite3*.

3.5 Herramientas de construcción de software.

Cuando un proyecto de programación comienza a crecer en complejidad, necesita más y más sistemas de infraestructura que agilicen y humanicen el trabajo de desarrollo. Así, por ejemplo, para compilar un “*Hello World*” clásico de C++, todo lo necesario es un editor y el propio compilador. Cuando el proyecto crece un poco más, ya es imposible navegar el código fuente sin un *ctags*, y si la mente del programador se sobrecarga con un poco más de complejidad, entonces es inevitable utilizar un IDE. Por este momento también aparece la necesidad de coordinar de alguna forma el proceso de sincronización de los ficheros objeto, bibliotecas y ejecutables con los cambios en el código fuente. Cuando el plazo de desarrollo del proyecto se extiende por par de meses, el código fuente comienza a ser un recurso extremadamente valioso y requiere cada vez mayor vigilancia; con frecuencia se hace necesario efectuar limpiezas en su estructura y probar variantes nuevas y viejas. También sucede que se incorporan nuevos desarrolladores. Para entonces ya es imprescindible el uso de un sistema de control de versiones, un sistema de construcción de la documentación y un sistema de manejo de operaciones variadas, específicas a las características del proyecto.

Las siguientes secciones dan una breve panorámica de cómo se enfrentó ese proceso en NoMS.

3.5.1 Autotools y Scons

Un ejecutable es el resultado de compilar unos cuantos archivos de código fuente. Lo mismo para una biblioteca. Si cambia uno de los archivos del código fuente, entonces el ejecutable o la biblioteca tiene que ser actualizado. Tradicionalmente esta ha sido la función de una herramienta conocida como *make* en el mundo Unix. Sin embargo, en proyectos complejos el uso de *make* no es suficiente, puesto que deja en las espaldas del desarrollador el problema de describir las reglas de compilación para cada biblioteca y ejecutable, las dependencias entre ellos, la estructura por directorios, la localización de las dependencias de tiempo de compilación y edición de enlaces, etc.

Es allí donde entra otro grupo de herramientas, conocidas en el mundo del software libre como *autotools* y formado por las utilidades individuales *Autoconf*, *Automake* y *Libtool*. Dichas herramientas manejan directamente la configuración de las fuentes y la generación de los ficheros de reglas para *make*. Las reglas generadas incluyen no solo los detalles para la compilación del proyecto, sino también para generar archivos de distribución y para efectuar la instalación del programa construido por el sistema.

Autotools tiene dos inconvenientes principales. El primero es que está demasiado ligado a la plataforma Unix, por lo que depender de él exclusivamente para manejar la actualización del código fuente no es buena idea. El otro inconveniente es que estas herramientas ponen demasiado en la habilidad de Unix de crear procesos de una forma eficiente y rápida. Cuando se trabaja con la misma filosofía en Windows, el tiempo que consumen las herramientas se hace prohibitivo.

Estos dos problemas dieron lugar a la necesidad de utilizar un sistema adicional para manejar el proceso de construcción: *Scons*. Esta herramienta es mucho más moderna y está preparada para correr en un variado número de plataformas. No utiliza *make* sino que efectúa directamente el proceso de actualización de las bibliotecas y los ejecutables, por lo que el proceso de compilación se hace mucho más eficiente y ligero.

En la práctica se decidió usar un paradigma mo-

Las bibliotecas/ejecutables representadas en los cuadrados en azul dependen de las bibliotecas básicas representadas en amarillo

fragmenter

Aplicación *cli* para efectuar la fragmentación en el algoritmo de inicialización del método de partículas

localrain

Aplicación *cli* para efectuar la "lluvia local" del algoritmo de llenado.

unclutter

Aplicación *cli* para desagregar y recolocar esferas en el algoritmo de inicialización del método de partículas

libspace

Estructuras de datos para consulta de rango, de punto, intervalo y segmentos.

libgeo

Algunos predicados básicos de la geometría computacional

libdebugaids

Servicios de traza y depuración.

libtrianglemesh

Estructuras de datos para mallas, junto con algunos predicados

portabledocuments

Manejo de serialización y persistencia.

gslcxx

Algunas encapsulaciones a métodos de la *gsl* y otros nuestros.

libw_spherefillers

Encapsulación para *Python* de los algoritmos de inicialización del método de partículas

libw_geo

Encapsulación para *Python* de las rutinas geométricas

libw_space

Encapsulación para *Python* de las rutinas de consulta espacial

libw_trianglemesh

Encapsulación para *Python* de las rutinas de consulta espacial

delo-ejecutor que permitiera utilizar cualquiera de las dos utilidades mencionadas, mientras que el desarrollador tendría que actualizar los archivos correspondientes (modelo) en un solo lugar. Hubiera sido posible prescindir por completo de *Autotools*, pero no era buena idea tirar por la borda todo el soporte que generaciones de programadores (y sus herramientas) le han dedicado.

3.5.2 *Kdevelop*

Para el momento en que había más de 10 archivos de código fuente la navegación entre ellos era bastante engorrosa, para agilizar el trabajo fue necesario comenzar a utilizar un IDE. Se analizaron algunas variantes, entre ellas el CDT/Eclipse (CDT, 2006). Este proyecto ha recibido mucha atención de algunos medios, pero en la práctica no puede atender proyectos de más de unos pocos ficheros sin interrumpirse de forma inesperada.

Del otro lado, estaba *Kdevelop* (KDEVELOP, 2006), con un esfuerzo de desarrollo mucho más constante. Este fue el IDE que sustentó el trabajo todo el tiempo. Sus características positivas son las siguientes:

- Manejo de la configuración del proyecto de forma automática.
- Modelo interno del código fuente, lo que permite navegar el código, localizar símbolos^{xiv} y autocompletar. Además, fuerte integración con *ctags*, de forma que el usuario dispone de al menos dos formas de navegar el código.
- Navegador de clases.
- Ambiente *multi-tabs* que simplifica mucho el trabajo con varios archivos de código fuente.
- Navegador de documentación. Dicho navegador, aparte de ser personalizable, provee por defecto acceso a 44 referencias más toda la documentación en forma de páginas *info* y *man* instalada en el sistema, con índice por contenidos, por palabras clave y búsqueda a través de *htdig*.
- Acceso inmediato a las funciones de

Doxygen para mantener la documentación del proyecto.

No fue posible evaluar otros ambientes de desarrollo, pero en cuestión de C++ y código abierto parece no tener rivales. *Kdevelop* se distribuye bajo licencia GPL, pero en el modo en que se usa la licencia no se propaga al código fuente de NoMS.

3.5.3 *Python*

Este fue el lenguaje escogido para hacer los pequeños trabajos auxiliares de configuración del proyecto (PYTHON, 2006). Dichos trabajos son:

- Cambiar la *url* del repositorio.
- Generar “distribuciones” del proyecto.
- Supervisar y refactorizar el modelo del código fuente. De este modelo ya se habló en la sección 3.5.1, resulta que el mismo se trabaja como un sistema de objetos en *Python*.

La decisión de usar *Python* viene dada porque es el mismo lenguaje sobre el que está construido *Scons* y porque es el lenguaje de “alta productividad” que se utiliza para manejar a NoMS desde arriba, así que con esto se disminuía el número de dependencias del producto. Por otra parte, el lenguaje en sí es muy sencillo y potente. No solo admite programación orientada a objetos, sino de una forma muy simple y eficaz incorpora rasgos de la programación funcional. *Python* se distribuye bajo una licencia propia conocida como PSFL 2.0, que permite su uso y redistribución siempre que se mantengan las proclamas de derecho de autor.

3.5.4 *Doxygen*

Esta es la herramienta que se utiliza para extraer la estructura del proyecto y la documentación de sus partes. Es una herramienta libre y de código abierto (DOXYGEN, 2006). Se caracteriza porque puede generar documentación en gran variedad de formatos, incluyendo hipertexto, RTF y látex. También puede procesar conveniente

Python, de esta forma toda la documentación puede generarse con una sola herramienta.

La documentación generada por *Doxygen* también puede utilizarse para navegar el código fuente.

3.5.5 Subversion

Subversion es el sistema usado para el control de versiones. Por control de versiones se entiende la posibilidad de mantener y utilizar la historia del código fuente y hacer bifurcaciones y entremezclas en la misma. *Subversion* es el sucesor natural en conceptos del clásico sistema (y querido para tantos programadores) conocido como CVS. *Subversion* es una herramienta libre y una de las historias de éxito del software libre. Tiene características técnicas relevantes (SUBVERSION, 2006):

- Todas las características de CVS
- Todos los directorios, archivos y movimientos tienen metadatos.
- Las entregas son realmente atómicas. Ninguna parte de la entrega tiene efecto hasta que la entrega completa termina.
- Integración con *Apache* y *Webdav*.
- Servidor de red independiente.
- La bifurcación y el etiquetado no replican archivos, por lo que son operaciones baratas.
- Selección del formato de almacenamiento para el repositorio.

La licencia que acompaña a *Subversion* es de estilo BSD, que solo exige que se mantengan las proclamas sobre derechos de autor y la propia licencia en redistribuciones posteriores del software.

3.6 Otras herramientas, auxiliares o satélites.

3.6.1 Python

Tal y como se mencionó antes, *Python* también se utiliza como lenguaje para manejar las bibliotecas

de NoMS desde arriba. Esto puede lograrse fácilmente con la ayuda de *boost.python*. En alguno de los epígrafes siguientes se darán más detalles.

3.6.2 Povray

En lo que NoMS se hace de un sistema de post-proceso propio o logra adaptar uno de los ya existentes, todo el análisis visual de resultados se ha hecho con esta herramienta. Muchos de los gráficos que aparecen en esta tesis fueron realizados con *PovRay*. Su nombre está formado por la abreviatura de *Persistence Of Vision Raytracer*. Es un producto que normalmente se utiliza en la producción de animaciones e imágenes estáticas de gran realismo (POVRAY, 2006). Este software tiene dos licencias: una para usuarios finales y otra para redistribución. Por un lado no es de interés redistribuir *PovRay* ni ningún trabajo derivado de él, mientras que la primera licencia solo sostiene proclamas de no garantía.

3.7 Decisiones de diseño

3.7.1 Manejo de memoria.

C++ es un lenguaje que no provee un esquema de manejo de memoria automático. Cada objeto debe ser destruido explícitamente en algún punto, de modo que la memoria que ocupa pueda ser reclamada. En programas que corren por pocos segundos y que no son intensivos en datos, es perdonable que el diseñador olvide de vez en cuando disponer algunos punteros. No es el caso para programas largos o que manejan instancias de datos grandes. NoMS cae en el último grupo.

Para el manejo de memoria, existe un primer nivel de decisión que consiste en considerar si todo el trabajo con memoria va a hacerse manualmente o de forma al menos parcialmente automática. Manejar la memoria de forma totalmente manual significa que por cada *new* que se haga debe existir en algún punto un *delete*, en la práctica, para un proyecto grande esto es excesivamente engorroso. Para el caso de NoMS, este primer nivel de decisión se resolvió optando por la segunda variante, utilizar la mayor parte del tiempo un es-

quema de manejo de memoria semiautomático.

Una vez hecho esto se pudo pasar al segundo nivel de decisión, relativo al sistema de manejo de memoria que se emplearía. Puesto que C++ no tiene ninguno, pues había que elegir entre alguno de los artificialmente posibles. Fueron consideradas dos variantes: el uso de punteros inteligentes y el uso de un recolector de basura conservativo, en particular, el popular *boehm-gc*.

La primera opción permite reclamar toda la memoria tan pronto como esta se va liberando, pero tiene una sobrecarga mayor en tiempo de corrida que la primera, y en espacio de almacenamiento. Por su parte, la segunda falla en reclamar toda la memoria (por su carácter “conservativo”), si bien es verdad que esos fallos son mínimos.

Ante este escenario, se decidió elegir por la certeza, es decir, el uso de punteros inteligentes. Este proceder se ve favorecido por la existencia en *boost* de un conjunto de plantillas para crear punteros inteligentes, la biblioteca *boost.smart_ptr*. El uso de punteros inteligentes se basa en contadores de referencias, y tiene una desventaja muy conocida, el asunto de las referencias circulares. Sucede de que si dos objetos se referencian mutuamente a través de este mecanismo, directa o indirectamente, su contador nunca alcanzará el valor cero y aún si ya no son visibles desde el conjunto raíz^{xv} no se destruirán. La única forma de trabajar alrededor de esto es teniendo cuidado, aunque por suerte el asunto de las referencias circulares aparece con relativamente poca frecuencia.

Otro asunto es el espacio de almacenamiento par el contador de referencias. Hay dos opciones: almacenarlo en la misma clase o externo. La variante preferida con *boost.smart_ptr* es almacenarlo de manera externa, de esta forma no es necesario modificar la clase a la que se quiere apuntar y en verdad esta aproximación es bastante buena en la mayoría de los casos. Una situación en que este enfoque no sirve, sin embargo, es cuando se intenta implementar el patrón factoría y los objetos fabricados deben portar una referencia a la factoría. Entonces no queda otra solución que mantener el contador de referencias interno. Este fue el caso que se presentó al diseñar *pysqlite3*.

3.7.2 Persistencia

Los cálculos en NoMS suelen ser largos, y es engorroso repetirlos. Por otra parte, los ensambles de objetos suelen ser bastante complejos. Entonces no queda otra cosa que recurrir al almacenamiento auxiliar para salvar resultados intermedios o definitivos. Otro escenario en el que se ha trabajado es la transmisión de objetos entre nodos de un ordenador paralelo por paso de mensajes o *cluster*.

Los requerimientos para un herramienta (biblioteca) que maneja la serialización son complejos, puesto que tiene que ser capaz de navegar a través de punteros de C y punteros inteligentes, mantener la unicidad de objetos y trabajar de forma transparente con los contenedores de la STL. También debe entender de herencia y polimorfismo.

Con estos requisitos, no es posible elegir. Hasta este momento, solo existe una implementación que los cumple: *boost.serialization*. Este implementación utiliza los rasgos más complejos del trabajo con plantillas en C++ discernir y diferenciar tipos. En el código de NoMS, puede encontrarse en la función miembro paramétrica *serialize* de todas las clases portadoras de datos.

3.7.3 Interfaz con lenguaje *script*.

La filosofía de C++ de poner declaraciones de clases en archivos con extensión .h; y de poner en los mismos archivos implementaciones de plantillas, conduce en proyectos grandes a un aumento desmedido en la cantidad de código que debe procesar el compilador por cada fichero de código fuente. Existen medidas y técnicas de programación para contrarrestar este fenómeno, pero aún tomándolas los tiempos de compilación suelen dispararse. Por esta razón, el ciclo de desarrollo habitual para C++, edición/compilación/prueba-depuración crece en el tiempo, cosa que afecta la productividad el programador.

Una parte importante de la solución es evitar usar C++ para cuestiones que (por razones de eficiencia) no lo requieran. De allí viene la idea de utilizar un lenguaje *script* para manejar la funcionalidad.

dad implementada en bibliotecas, a un nivel tan alto que no afecte el tiempo de ejecución del programa en sí, es decir, dejando el grueso del cómputo del lado de C++. El lenguaje *script* que se usó, tal y como ya se dijo, fue *Python*.

Para exportar la funcionalidad de las bibliotecas en C++ se exploraron tres vías. La primera consiste en hacerlo directamente, “a mano”. Si bien los desarrolladores de *Python* siempre han tenido en cuenta este aspecto, el trabajo se vuelve monótono y repetitivo.

Otra forma es utilizando alguna herramienta especializada. Específicamente se evaluó *Swig*, una utilidad bastante popular y potente que se ha venido empleando para este propósito desde hace muchos años. El problema con *Swig* es que representa una regla más en el sistema de compilación, además, la herramienta necesita que se le especifiquen datos como la signatura de tipos de cada función libre o método que se vaya a exportar; tampoco permite al código en C++ manipular los objetos en *Python* convenientemente.

La última forma, que fue la que finalmente se adoptó, consiste en usar *boost.python*, una biblioteca en *boost* diseñada para este fin. Esta biblioteca no tiene ninguna de las desventajas de *Swig*, y además le evitaba una dependencia adicional al proyecto. Dentro del código fuente, pueden encontrarse el trabajo con *boost.python* dentro del directorio *externization/topython*.

3.8 Estructura física del proyecto.

3.8.1 Lista de directorios.

- *Configurationreader* Manejo de configuración
- *externization* Manejo del alcance externo
 - *classkeeper* Manejo de plugins (no finalizado)
 - *topython* Funcionalidad exportada hacia *Python*.
- *gslcxx* Encapsulaciones y extensiones a

gsl.

- *include* Ficheros de configuración de tiempo de compilación.
- *libbuckard* Predicados de geometría computacional de John Buckardt
- *libdebugaids* Biblioteca con utilitarios de traza y depuración
- *libgeo* Biblioteca de geometría computacional
- *libspace* Biblioteca con estructuras de consulta espacial.
- *libtrianglemesh* Biblioteca con estructuras para el manejo y la transformación de mallas.
- *portabledocuments* Manejo de persistencia a través de las distintas aplicaciones
- *stochasticity* Clases para manejo especializado de persistencia.
- *workers* Interfaz cli para alguna funcionalidad de NoMS
 - *fileinfo* Extrae y muestra metadatos en los archivos de NoMS
 - *fragmenter* Aplicación cli para efectuar la parte correspondiente del algoritmo de inicialización del método de partículas. Ver capítulo 4.
 - *localrain* Aplicación cli para efectuar la segunda parte del algoritmo detallado en el capítulo 4.
 - *povprinter* Aplicación para representar con *PovRay* el contenido de varios archivos de NoMS
 - *simulators* Directorio destinado a contener interfaces cli para los distintos tipos de algoritmos de simulación física que se implementen. Actualmente contiene la parte física del método de partículas.
 - *statsdiscover* Aplicación cli para verificar el correcto funcionamiento del algoritmo de inicialización del

método de partículas.

- *unclutter* Aplicación *cli* para efectuar la última fase del algoritmo de llenado en el método de partículas.

Una de las cosas interesantes que saltan a la vista en esta enumeración, es la existencia de una interfaz formada por tres herramientas *cli* para realizar la implementación del algoritmo detallado en 4. En realidad, existe otra interfaz mucho más flexible que fue la que se utilizó en la obtención de las tablas y diagramas que aparecen en el capítulo 4, y que está implementada utilizando la encapsulación de las bibliotecas desde *Python*.

3.9 Conclusiones parciales.

NoMS ha sido desarrollado con herramientas

abiertas y sin costo económico. Se ha hecho el mayor esfuerzo en utilizar componentes y bibliotecas que no pesen más tarde económica o legalmente.

La utilización de sistemas libres y abiertos ha tenido ventajas colaterales: cuando han aparecido dificultades o fenómenos ininteligibles, ha sido posible seguir el comportamiento hasta el código fuente de las herramientas o bibliotecas, y en algunos casos (pocos en verdad) se ha modificado este para satisfacer las exigencias del problema concreto que aborda NoMS.

xiv A partir de la versión 3.4

xv Conjunto raíz: en la parte de las ciencias de la computación que se dedica al estudio de técnicas de manejo de memoria, se denomina conjunto raíz al conjunto de punteros y referencias en almacenamiento automático (pila) o regiones estáticas del programa.

Capítulo 4 Una aplicación: inicialización para el método de partículas y el de puntos.

4.1 *Nota introductoria*

El propósito de este capítulo es ilustrar la primera aplicación que se ha hecho con NoMS, y en cierta forma la más acabada. Su objetivo es demostrar que NoMS puede ser usado en la solución de problemas para los que está diseñado, y hasta el momento, con ventajas. Se espera que a lo largo de la exposición teórica y la presentación de los resultados de esta aplicación, el lector sea capaz de percibir los problemas que NoMS ya resuelve.

Sin embargo, sobre la propia aplicación que se ilustra en este capítulo, **no** es objetivo de esta tesis discurrir sobre sus resultados, ni si resultan mejores entre todas las posibles aplicaciones dedicadas a resolver este problema. Tampoco es objetivo hacer pruebas o aseveraciones terminantes sobre la complejidad de los algoritmos.

4.2 *El problema*

La etapa de inicialización del método de partículas consiste en poner partículas en el dominio del problema Γ , o en una parte de él si se trata de métodos mixtos. El dominio del problema puede darse de distintas formas. En esta aplicación, el problema que se abordará es el llenado total del dominio, útil para el método de partículas solo. Las partículas serán esféricas, porque como ya se notó en el capítulo 1, otras formas son inviables hasta el momento por su costo computacional.

Existen parámetros que la población inicial de partículas esféricas debe cumplir, y que están dados como condiciones iniciales en el problema de

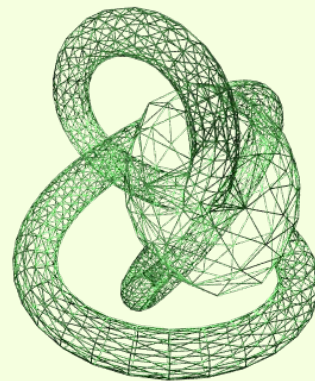
simulación^{xvi}. Se trata de restricciones de dos tipos. La primera tiene que ver con el radio de las partículas, y formalmente está dada por una función de densidad de probabilidades $\phi(r)$ sobre dicho radio. La segunda tiene que ver con la distancia entre partículas, que se asumirá positiva (no solapamiento) y distribuida normalmente con media d y desviación σ .

Sobre el dominio Γ , puede tener cualquier forma, siempre que sea acotado y cerrado; para que el problema sea practicable computacionalmente tiene que cumplir algunos otros requisitos mínimos (por ejemplo, estar formado por una cantidad finita de subdominios conexos) que no se discutirán aquí. Son de interés, sin embargo, dos predicados geométricos, relacionados con el dominio y su clausura $\bar{\Gamma}$:

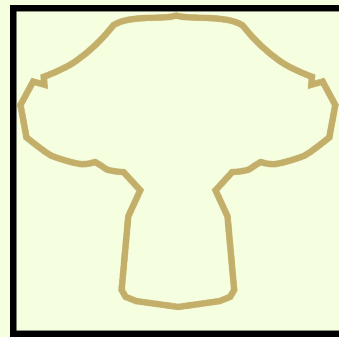
1. Un predicado π_{\in} que permita conocer si un punto del espacio pertenece al dominio Γ .
2. Un predicado $\pi_{|s|}$ que permita conocer la distancia desde un punto P hasta el punto más cercano de la clausura $\bar{\Gamma}$.

Con ayuda de los dos predicados anteriores, puede construirse un tercer predicado π_s definido de la siguiente forma:

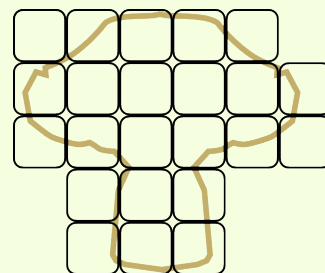
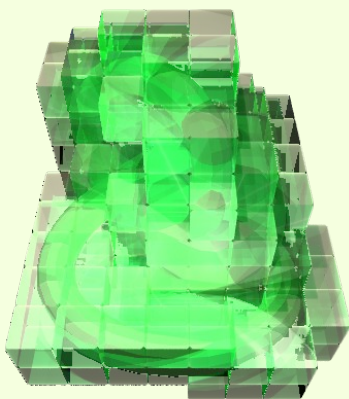
Diagrama 1: Malla, caja, partición en celdas y celdas que solapan.



La entrada del algoritmo consiste en una malla de superficie que define el dominio



Luego es necesario hallar la caja de menor volumen que contiene totalmente al dominio.



Se crea un cubrimiento para la caja. Un cubrimiento de un conjunto S es un sistema de conjuntos T_i cuya unión contiene a S totalmente.

Con esto están creadas las condiciones para plantear el problema de la aplicación:

Construir un algoritmo que dado un dominio Γ y dos predicados π_{\in} y $\pi_{|s|}$ con la funcionalidad argumentada antes, sea capaz de crear una población inicial de esferas dentro del dominio con función de densidad de distribución de radios $\phi(r)$, y con la distancia entre ellas con media d y desviación σ .

4.3 Idea general

El objetivo de esta sección es una idea de la secuencia de pasos de esta aplicación, y el por qué de cada paso.

El algoritmo procede en tres etapas fundamentales. La primera es el particionamiento en celdas. Con este paso se hace el preproceso necesario para trabajar el dominio por fragmentos, e influye tanto sobre el dominio como su frontera. Su función es restringir luego el procesamiento a secciones del espacio de que se solapan con el dominio y que tengan una forma predeterminada: celdas cúbicas. Es esencial en el funcionamiento óptimo de las etapas posteriores.

La segunda etapa tiene que ver con la generación inicial de esferas para cada celda. Durante esta etapa se generan unos pocos “lotes” predeterminados de esferas empacados en celdas, que ya cumplen con las restricciones sobre la distribución de los radios y las distancias. Estos se estampan sobre las celdas del particionamiento en el paso anterior. Tanto la construcción de los lotes como el estampado se hacen simultáneamente, por lo que esto se considera una sola etapa. El resultado es un cubrimiento del cuerpo con esferas, cuya principal desventaja es la existencia de “espacios vacíos” sobre las juntas de las celdas. Desde otro punto de vista, la segunda etapa ejecuta una operación de calibración para la tercera.

La tercera etapa tiene dos funciones: eliminar las esferas exteriores al dominio y eliminar los “espacios vacíos” en las juntas.

Las siguientes secciones detallan cada una de las etapas de esta aplicación.

4.4 Primera etapa: particionamiento en celdas.

En la primera sección se dijo que el dominio Γ era acotado. Entonces, es posible construir una caja con volumen mínimo que lo contenga totalmente. Sea B_{Γ} la menor de tales cajas. En la práctica, la forma de construir B_{Γ} depende concretamente del dominio Γ . A continuación un algoritmo capaz de construir dicha caja a partir de una malla:

Algoritmo 18 Caja envolvente de malla

Sea una malla formada por un sistema de triángulos $\{T_i\}$; se quiere construir una caja de volumen mínimo que contenga a todos los elementos de $\{T_i\}$.

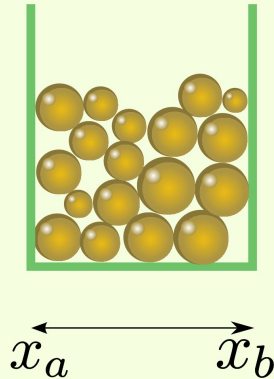
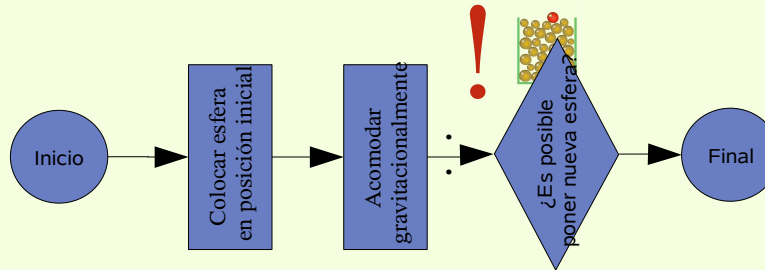
Hacer:

- Inicializar $x_{\max} = y_{\max} = z_{\max} = -\infty$ y $x_{\min} = y_{\min} = z_{\min} = \infty$.
- Para T_i en $\{T_i\}$:
- Para $P_k = (x_k, y_k, z_k)$ vértice del triángulo T_i :
 - Si $x_k < x_{\min}$, hacer $x_{\min} = x_k$
 - Si $y_k < y_{\min}$, hacer $y_{\min} = y_k$
 - Si $z_k < z_{\min}$, hacer $z_{\min} = z_k$
 - Si $x_k > x_{\max}$, hacer $x_{\max} = x_k$
 - Si $y_k > y_{\max}$, hacer $y_{\max} = y_k$
 - Si $z_k > z_{\max}$, hacer $z_{\max} = z_k$
- Construir una caja con coordenadas menores $(x_{\min}, y_{\min}, z_{\min})$ y coordenadas mayores $(x_{\max}, y_{\max}, z_{\max})$

Una vez que se tiene B_{Γ} , se elige un lado de arista b para el sistema de celdas, y se construye un cubrimiento de celdas $\{B_i\}$ sobre B_{Γ} . Esto puede hacerse en tres ciclos anidados, no es difícil darse cuenta cómo, para el caso tridimensional.

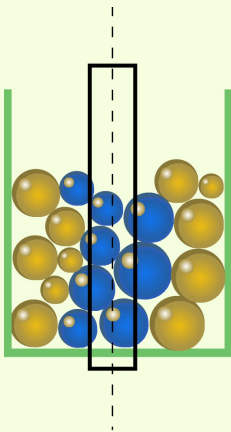
El siguiente paso tiene que ver con descartar los elementos de $\{B_i\}$ que no se relacionen con el dominio Γ . Para su ejecución, en el caso de

Diagrama 2: Un paso en la lluvia local.

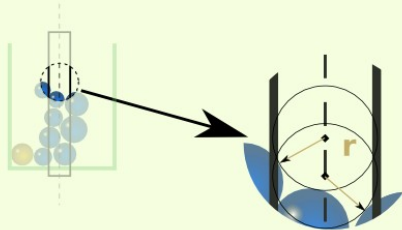


Estado medio del algoritmo. El objetivo de cada paso es, desde este estado, agregar una nueva esfera. Para esto:

- Usando un generador de números aleatorios con la distribución pedida $\phi(r)$ obtener un radio r para la esfera que está a punto de insertarse.
- Escoger una posición x aleatoria en el rango $[x_a + r, x_b - r]$. En el caso tridimensional, se busca una región en el plano equivalente a la cara correspondiente en la celda menos el radio.



- Se construye una caja B_q de base $2r$ y centro x , que se extienda verticalmente desde la base de la celda hasta su techo.
- Se realiza una consulta de rango con forma de rectángulo/ortohedro. Con esto se obtiene un conjunto de esferas S_q que limitan el movimiento desde $y = +\infty$ en la dirección negativa de este.
- Para cada esfera elemento de S_q , con radio r_i y coordenadas (x_i, y_i) (ó (x_i, y_i, z_i) , pero el caso tridimensional es una extensión simple del caso bidimensional) hacer:



- Hallar el número $t_i = \sqrt{(r_i + r)^2 - (x - x_i)^2} + y_i$
- De todos los números t_i hallar el mayor $y = \max_{i \in S_q} t_i$, y colocar la nueva esfera en las coordenadas iniciales (x_i, t_i) .

un dominio cuya frontera es una malla se necesita de un predicado especial:

Algoritmo 19 Intersección caja-triángulo

Dado una malla formada por un sistema de n triángulos $\{T_i\}$ y un sistema de m cajas $\{B_k\}$, hallar cuales cajas caen dentro de la malla, o se solapan parcialmente con ella.

Comentario:

Aquí se utilizará el aparato definido en el algoritmo 17, en particular la estructura de consulta de rango D y la funcionalidad que tiene este algoritmo de decidir si un punto se encuentra dentro o no de la malla.

Predicado para cada

caja B_k , $k=1, 2, \dots, m$:

- Utilizar el aparato definido en el algoritmo [17], en particular su estructura de consulta de rango, para conocer si el punto medio de B_k (hallado de acuerdo al algoritmo 9) pertenece o no a la malla. Si pertenece, entonces el trabajo para esta caja puede darse por terminado y se retorna verdadero.
- Utilizando la estructura de consulta de rango para los triángulos, hallar el subconjunto S_1 formado por todos los triángulos de la malla que se tocan parcialmente con B_k ; esto es equivalente a una consulta de rango simple.
- Construir un conjunto S_2 de triángulos formado por una pareja de triángulos por cada cara de B_k .
- Averiguar, utilizando el algoritmo 8, si alguno de los elementos de S_1 toca a alguno de los elementos de S_2 . De ser así, retornar verdadero. En otro caso, retornar falso.

Este algoritmo se aplica a la posible intersección de cada una de las celdas con la malla, y se descartan aquellas que quedan totalmente afuera.

El resultado de esta etapa es un conjunto de cel-

das E que se solapan parcialmente con el dominio. Puesto que todas las celdas son iguales, también puede decirse que el resultado es un conjunto de índices. El diagrama 2 resume de modo visual su ejecución.

4.5 Segunda etapa: lluvia local.

El algoritmo le debe su nombre a esta etapa. Comienza generando un conjunto de lotes de partida, y luego ejecuta un ciclo que pasa sobre todo el conjunto E aplicando en unos casos uno de los lotes existentes y en otros generando un nuevo lote, de forma que se mantenga una diversidad de configuraciones lo suficientemente amplia.

La generación progresiva de lotes es necesaria atendiendo a un hecho simple: si se toman m lotes como patrón, con una cantidad promedio de n esferas cada uno, la cantidad total de muestras de la distribución de radios sería fija y aproximadamente igual a $n * m$, lo que no parece buena idea desde el punto de vista estadístico.

Algoritmo 20 Poniendo la generación inicial de esferas.

Se toman como parámetros de entrada la cantidad de lotes almacenados m y la probabilidad α de generar un nuevo lote durante el proceso de llenado.

Hacer:

- Crear los m lotes iniciales.
- Para cada celda que se quiera llenar:
- Generar aleatoriamente un índice $i=1, 2, \dots, m$
- Generar un evento aleatorio y ver si se cumple α . En caso de ser positivo, llenar la celda actual con un nuevo lote y poner este en el almacén en la posición asegurada por el índice i . En caso de ser negativo, simplemente usar el lote almacenado en la posición i para llenar la celda actual.

Como ya se dijo en la primera sección de este epí-

grafe, un lote es un conjunto de esferas empacadas en una celda. El algoritmo podría proseguir solamente conociendo la densidad promedio de esferas, aunque se beneficia en términos de eficiencia si dispone de un estimado para las posiciones iniciales. La mejor forma de conseguir esto es usando un método de deposición, de allí el nombre de “lluvia”. Puesto que el volumen de la celda es bastante pequeño, no es costoso llenarla por este método, además, el mismo se ejecuta en una fracción ínfima del conjunto de todas las celdas, con lo que se logra minimizar el costo computacional de esta etapa.

Este algoritmo se puede emplear en dos y tres dimensiones, la implementación que se encuentra en NoMS es genérica por lo que puede emplearse en un espacio de cualquier número de dimensiones. Sin embargo, la forma de representar la frontera del dominio, cuando esta es un conjunto de elementos simpliciales, cambia según el número de dimensiones; y hay que hacer las modificaciones pertinentes a las partes del algoritmo que tocan la frontera. Por una cuestión de tiempo y utilidad se presenta la versión tridimensional.

Para los propósitos de esta explicación, los diagramas serán esquemas bidimensionales bastante simplificados.

El algoritmo inicia buscando una posición inicial para la nueva esfera, como se muestra en el diagrama 2. A continuación, desde esa posición inicial, se ejecuta una etapa de optimización sobre una función L basada en la siguiente idea:

- Existe un campo gravitatorio que actúa en la dirección de las y y aporta a la función L una componente en la forma $L = \dots + b y$, donde b es cierta constante y los tres puntos ocupan el lugar del resto de la expresión.
- Entre dos esferas existe una fuerza de repulsión, que se anula a cierta distancia s_c . Dicha fuerza, por cada partícula (esfera) p le aporta un componente a L :

$$V_p = \begin{cases} 0 & \text{en otro caso} \\ A(s - s_c)^2 e^{-\alpha s} & \text{si } s - s_c > 0 \end{cases} \quad (14)$$

Con esto se puede llegar a una expresión completa para L , el valor de la función que se debe optimizar:

$$L = \sum_{p \in N} V_p + b y \quad (15)$$

En esta etapa no se toman en cuenta las paredes del cuerpo (sistema de caras, función, etc), sino las de la celda, es decir, un ortoedro regular con un patrón simple para el cuál es posible usar la misma expresión (15). De esta forma, la función construida es suficientemente suave como para aplicar un minimizador basado en derivadas. El algoritmo de optimización que se escogió fue el gradiente conjugado de *Fletcher-Reeves* (GSL, 2004), con algunas modificaciones. Dicha modificación es, básicamente, limitar la longitud del paso en cada iteración del ciclo de modo que no pueda “pasarse por arriba” de una esfera ya colocada. Con esto se destruye la convergencia cuadrática del método en puntos lejanos al mínimo, pero se preserva en puntos suficientemente cercanos, que prescriban un paso pequeño.

En la práctica se probaron otras variantes:

- El *simplex* de Nelder-Mead. Excelente convergencia, pero no resultaba fácil adecuarlo al hecho de que ya existen esferas en el espacio de configuraciones, que actúan como restricciones de tipo muy complejo. Además, no usa la información de derivadas que es barata de obtener dada la forma de la función L .
- El algoritmo de *Fletcher-Reeves* sin modificar: el mismo problema respecto a las restricciones, este no es un algoritmo para

optimización con restricciones.

- El algoritmo de *Hooke-Jeeves*. Este es de búsqueda directa y puede ser fácilmente adaptado a la existencia de restricciones complejas, pero tampoco toma en cuenta las derivadas.

Existen enfoques tradicionales para la optimización con restricciones, pero estos se refieren a restricciones en forma analítica y no pudieron ser aplicados con éxito en esta instancia concreta. Aún así, resultaba atractiva la posibilidad de emplear algoritmos con convergencia cuadrática. El camino seguido permitió alcanzar un compromiso entre convergencia cerca del punto de extremo y respeto a las restricciones.

Otro aspecto que debe tratarse con algún detalle es el relativo a la selección de los coeficientes α y b en las expresiones (14) y (15); el valor del coeficiente A permite variar el peso relativo de cada interacción en contraposición con la gravedad. El coeficiente α permite variar la suavidad con que la función V_p se anula, mientras que b , junto con A permite controlar la distancia promedio entre esferas. Puesto que A y b sirven para lo mismo, en la práctica se escogió un valor fijo para uno de ellos, por ejemplo, $A=1$. La relación entre b y el resto de los coeficientes se obtiene alineando la fuerza de repulsión a que da lugar el gradiente de (14) en la misma dirección y sentido opuesto a la fuerza que resulta de (15), de lo que resulta una expresión de la siguiente forma:

$$b = 2A(d - s_c)e^{-\alpha d} - \alpha A(d - s_c)^2 e^{-\alpha d} \quad (16)$$

En la misma, d es la distancia esperada entre esferas y s_c es la distancia a la cual deja de actuar el campo de (14).

4.6 Última etapa: eliminación de las esferas en el exterior del dominio y de los huecos.

Los dos algoritmos existentes en esta etapa son

los más antiguos en la concepción de esta aplicación, cada uno de ellos data de al menos un año. El segundo de ellos, es además el algoritmo con el costo computacional dominante. Pudiera parecer por esta última razón que es el más importante en el método, pero en verdad es completamente inútil sin la fase de calibración descrita en la sección anterior. Sucede que este algoritmo solo alcanza su carácter más efectivo y sencillo si se da una distribución inicial de esferas suficientemente densa y en el interior del dominio. Aunque la restricción de no-solapamiento no es necesaria sobre dicha distribución inicial, no hay manera de conocer de antemano qué fracción del volumen necesitarán abarcar las esferas generadas, ni cómo satisfacer esa restricción en términos de la densidad de la nube de puntos inicial. Por otra parte, el ajuste de estos parámetros se hace de forma implícita en la etapa anterior.

Eliminar las esferas en el exterior del dominio es bastante sencillo. Basta con usar el predicado π_s para cada una de las esferas puestas en el algoritmo 3.

La parte más compleja es la relacionada con la eliminación de los “espacios vacíos”, cosa que se logra redistribuyendo un tanto las esferas en las celdas. Se hace con la ayuda de la misma función L definida en la sección anterior, pero ignorando el componente gravitatorio. Por otra parte, en esta fase se toman en cuenta las paredes del cuerpo, cosa que impide el uso de un minimizador con derivadas (para fronteras formadas por conjuntos de simpliciales, la función de distancia desde un punto no es continua en general). Los métodos de optimización elegibles son, en este caso, el *simplex* de *Nelder-Mead* y el de *Hooke-Jeeves*. En la práctica, se probaron las siguientes variantes:

- Método de *Nelder-Mead*. Los resultados fueron satisfactorios, y esta fue la variante elegida.
- Método de *Hooke-Jeeves*. Los resultados fueron satisfactorios pero es ligeramente menos eficientes que el anterior.
- Método de *Nelder-Mead* con perturbación estocástica programada (recocido simula-

do). Tiene relativamente el mismo costo que la primera variante, pero no se lograron progresos significativos en cuanto a la cantidad de iteraciones necesarias para lograr convergencia. De frente a la duda, decide por la simplicidad.

En esta etapa, no es necesario conocer nada sobre la existencia de las celdas, pero es la etapa dominante en cuanto a costo computacional; así que es buena idea mantener la información de particionamiento con el propósito de distribuir el procesamiento en un entorno paralelo.

terior sobre la formulación del algoritmo. A continuación, una explicación detallada de los parámetros y su influencia en el funcionamiento del algoritmo. En párrafos posteriores, se discuten los resultados empíricos de la corrida de esta aplicación con distintos parámetros.

Este primer grupo tiene que ver con los valores que pueden tomar los radios:

- **Distribución:** la función $\phi(r)$ especificada en el primer epígrafe. Aquí, sin embargo, bajo estos términos se entenderá su

Tiempo contra cantidad de esferas

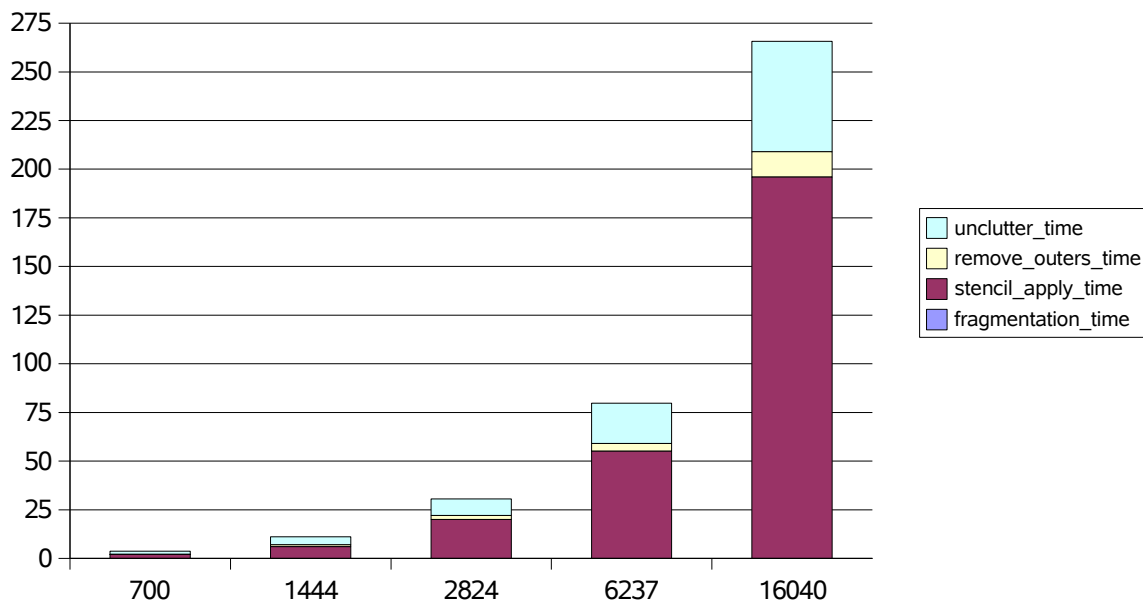


Illustration 7: La conclusión principal que permite extraer este gráfico es que, cuando aumenta el número de esferas y se mantiene constante el tamaño relativo de la celda, la proporción de los tiempos por operación es la misma. También puede notarse como en este caso, con un tamaño de celda relativamente grande, el tiempo dominante es el tiempo de generación y aplicación de stencils.

4.7 Resultados

Este algoritmo contiene unos cuantos parámetros numéricos que deben ser ajustados por el usuario. El ajuste de los parámetros es de interés a los usuarios finales de esta aplicación y por tanto no será cubierto en este documento.

En algunos casos, dichos parámetros solo tienen sentido si están en cierto rango. Cada uno de estos parámetros fueron mencionados en el epígrafe an-

terior sobre la formulación del algoritmo. A continuación, una explicación detallada de los parámetros y su influencia en el funcionamiento del algoritmo. Es decir, se trata de una categoría.

- **Radio mínimo:** el mínimo radio que pueden tener las esferas.
- **Radio máximo:** el radio máximo que pueden tener las esferas.
- Otros parámetros de la distribución; por ejemplo, en el caso de la distribución nor-

Tiempo por número de esferas

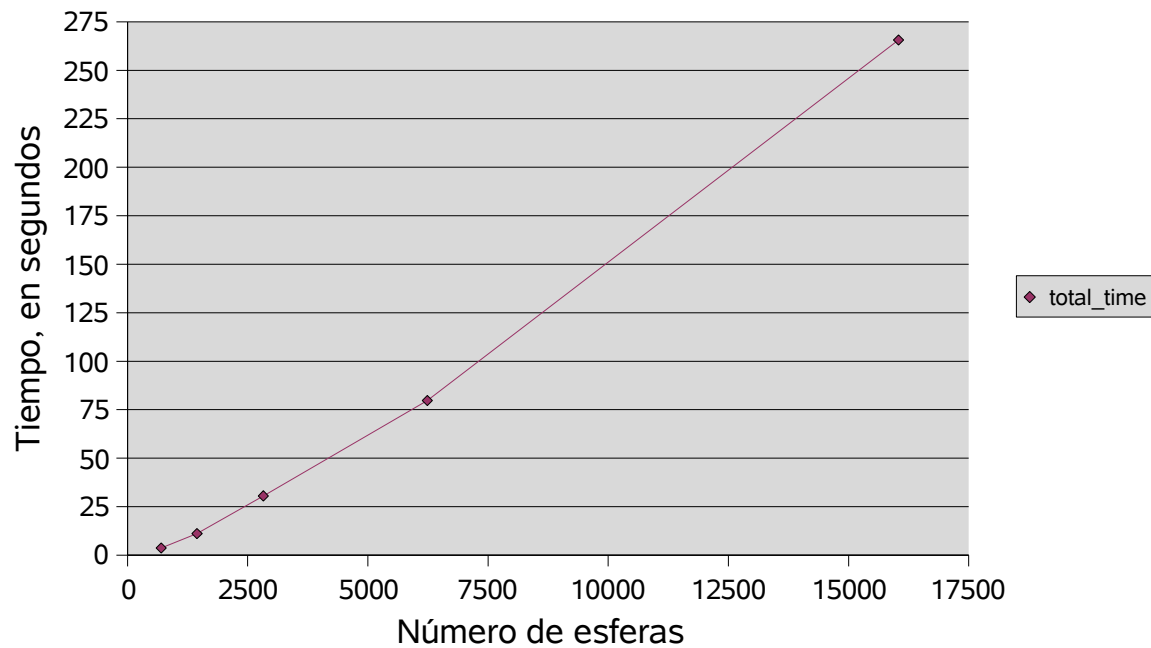


Ilustración 8: El tiempo total contra el número de esferas

mal es necesario además especificar la distribución estándar.

El siguiente grupo está relacionado con las características intrínsecas del algoritmo. Pueden usarse

Porcentaje de tiempo por operación

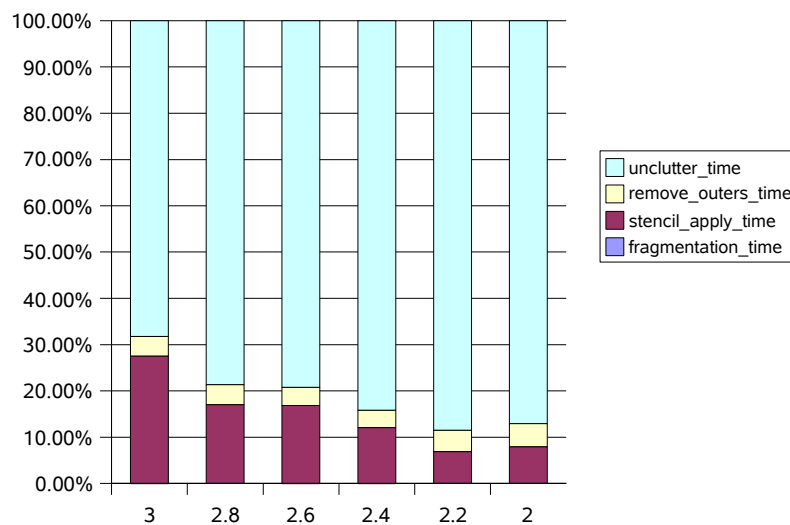


Ilustración 9: Al disminuir el tamaño de celdas, aumenta su número y disminuye el por ciento de tiempo entregado a la aplicación de stencils, mientras se hace predominante el tiempo de reacomodo.

Volumen relativo ocupado por las esferas, contra el tamaño de celda.

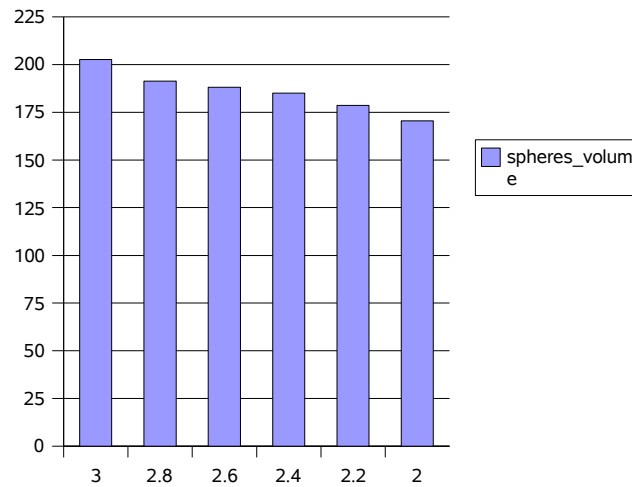


Ilustración 10: Debido al aumento de las fronteras entre celdas, el volumen relativo que ocupan las esferas disminuyen ligeramente con la disminución del tamaño de celdas.

para ajustar el tiempo de corrida y la densidad de las esferas .

- **Multiplicador de gravedad:** tiene que ver con el valor del coeficiente b en la fórmula (15) .
- **Tamaño de la celda:** el tamaño de la

Esto resume el grupo de parámetros de entrada. Por otra parte, la corrida del algoritmo permite conocer un grupo de indicadores, además de su resultado (las esferas generadas), que son:

- **Tiempo de fragmentación:** tiempo empleado creando la partición en celdas.

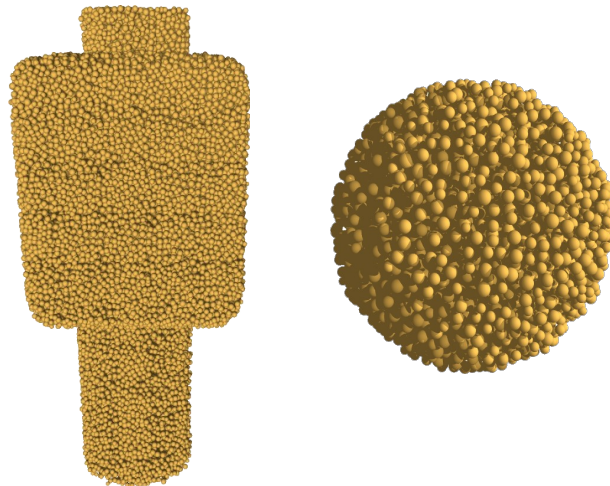


Ilustración 11: Aspecto visual del sistema después de la generación

arista de cada celda cúbica en el particionamiento.

- **Tiempo de generación y aplicación de stencils:** tiempo empleado por el algoritmo generando y aplicando stencils.

- **Tiempo para remover esferas exteriores:** el llenado por particiones deja algunas esferas en el exterior. Este es el tiempo que se necesita para removerlas
- **Tiempo para *re-aleatorizar* la distribución de esferas:** es el tiempo necesario para eliminar grietas.
- **Densidad:** la relación del volumen de las esferas al volumen total del cuerpo.

El análisis de todas las variables es bastante complejo, y un estudio más detallado puede hacerse en un trabajo posterior. Además, es de interés a personal especializado. Las siguientes secciones tratan de ilustrar lo que concierne al algoritmo mismo, y no pretende ser, de ninguna forma, un estudio detallado de alguna cuestión física o empírica en general.

4.7.1 Tamaño de celda fijo.

Esta sección trata sobre como varían los tiempos, número de esferas, etc cuando se mantiene fijo el tamaño de celda y se varían otros parámetros. Se vuelve a recalcar que tiene propósito ilustrativo, por lo que se omiten análisis detallados, el uso de la estadística, etc.

Así, por ejemplo, la ilustración 2 muestra el tiempo total contra el número de esferas. Sería muy buen objeto de un estudio posterior conocer si esto es lineal o lineal-logarítmico.

4.7.2 Tamaño de celda variable

El siguiente gráfico muestra el comportamiento al disminuir el tamaño de celda, al menos en cuanto al tiempo por cada fase del algoritmo. El aumento del número de celdas trae aparejado una menor

densidad (o volumen total) de esferas para la misma distribución de radio como puede observarse en la ilustración 5. Sin embargo, como se verá en la sección que sigue, la densidad de esferas es muy fácil de ajustar cambiando el multiplicador gravitatorio.

4.7.3 Variando el multiplicador gravitatorio.

Las siguientes ilustraciones muestran como afecta la variación del coeficiente gravitatorio a la densidad general de esferas en el cuerpo que se está llenando. Ver ilustración 6.

4.8 Conclusiones parciales.

El objetivo de este capítulo fue mostrar la viabilidad del desarrollo de aplicaciones dentro de NoMS; un marco de desarrollo que debe hacer posible la investigación e implementación de nuevos y viejos métodos de partículas y sin mallas en la mecánica.

La aplicación particular que se discutió funciona correctamente. En sí, logra superar por ejemplo, en cuanto a calidad en los resultados y rapidez, el código de otros investigadores orientado al método de partículas, al menos en su primera fase, la generación del medio.

Esta no es la única aplicación implementada sobre NoMS, en estos momentos el resto del marco para la simulación orientada a partículas se encuentra en un estadio de desarrollo bastante avanzado.

xvi No confundir aquí el término “condiciones iniciales” con su significado en la teoría de las ecuaciones diferenciales.

Conclusiones

Este trabajo ha permitido introducir a NoMS, un sistema para asistir la investigación en el campo de métodos de partículas y sin mallas. Su concepción fue, y mientras se trabaje en él continuará siendo, un problema vasto, por el gran número de cuestiones que abarca. Para el desarrollo de métodos de partículas y sin mallas se requiere el trabajo de equipos multidisciplinarios, que aúnen los esfuerzos de especialistas en mecánica computacional, en computación y en matemática aplicada. Aún dentro de cada una de las esferas mencionadas existen ramas de trabajo distintas cuya conjugación es fundamental.

En esta tesis fue objeto de estudio el trabajo con la geometría computacional, en una aproximación que intentó describir todos los predicados geométricos básicos, utilizando el enfoque del álgebra vectorial. La implementación de algunos de estos predicados no es 100% robusta, porque existen “casos perversos” en los que la precisión numérica los puede hacer fallar. Sin embargo en la práctica nunca se encontró uno de estos fallos; sería necesario experimentar a escalas más grandes para documentar este sujeto.

También se hizo la descripción de par de estructuras de consulta espacial implementadas en NoMS, una basada en tablas de dispersión y otra que es una variación del conocido *kdtree*. En este último caso, es importante notar que el *kdtree* se usa tradicionalmente para consultas de rango y la adap-

tación hecha permite realizar consultas intervalo-intervalo sobre él. Existe un esquema más general que el empleado para realizar consultas intervalo-intervalo utilizando contenedores de puntos, pero es sin duda menos potente puesto que requiere un tamaño fijo en los objetos. Este esquema, precisamente, se utilizó en el caso de la estructura por tabla de dispersión.

En la práctica, hay motivos para pensar que se necesita un trabajo más profundo en el campo de las estructuras de consulta espacial, puesto que son el paso obligado de muchos algoritmos. Esta aseveración viene de la experiencia directa con los algoritmos y de intercambios realizados con especialistas del área.

En el tercer capítulo se exploraron las herramientas y bibliotecas de software disponibles, haciendo hincapié no solo en las características técnicas de estas herramientas, sino también (en contra de lo que ya es costumbre entre nosotros) en los aspectos legales y económicos de su uso.

Por último, se validó la utilidad general del sistema con una aplicación concreta, la inicialización del método de partículas y la generación de puntos para los métodos sin mallas.

Como producto de software, NoMS es un intento de unir las mejores enseñanzas de la ingeniería de sistemas con el andamiaje teórico de los modernos métodos de simulación.

Recomendaciones

Los resultados de tres años de trabajo son, en verdad, solo el punto de arrancada. Es enorme el número de tareas por tratar en el futuro. Al escribir estas últimas palabras, me vienen a la mente estas:

- Mejorar el compromiso entre flexibilidad y facilidad de uso de la biblioteca de geometría computacional.
- Seguir avanzando en el área de consultas de rango.
- Crear una interfaz de usuario elaborada de forma que sea accesible a un mayor número de usuarios, en particular, aquellos sin conocimientos de computación interesados en su aplicación como mera herramienta ingenieril.
- Mejorar la documentación, de forma que la curva de aprendizaje sea más suave.
- Crear una versión del sistema para Windows
- Incorporar el trabajo de paralelismo que se ha venido haciendo paralelamente, de forma que NoMS pueda correr en clusters de computadoras
- Limpiar algo más el código base, sobre todo la cuestión de los espacios de nombres, que durante la evolución del sistema quedó con algunas incongruencias.
- Ampliar la funcionalidad disponible desde Python.

Bibliografía.

- Agarwal, Pankaj, et al..(1997) "Geometric Range Searching and Its Relatives" Report of Research at Center for Geometric Computing, Department of Computer Science
- Alcides Viamontes Esquivel.(2006) "NoMS Reference Manual" Technical Documentation
- Berkman, Norbert.(1990) "The R*-tree: An efficient and Robust Access Method for Points and Rectangles" Report of Research at Matsachussets Technological Institute
- [HCITASCA] Corporation Authorship.(2002) "Sales Publicity" online, see at www.hcitasca.com
- [GSL] Non Governmental Organization/ Software Foundation Web Site. Page reviewed in (2006) year. "GNU Scientific Library" Available at <http://www.gsl.org>
- [PYTHON] Non Governmental Organization/ Software Foundation Web Site. Page reviewed in (2006) year. "Python Software Foundation" Available at <http://www.python.org>
- [BOOST] Non Governmental Organization/ Software Foundation Web Site. Page reviewed in (2006) year. "Boost" Available at <http://www.boost.org>
- [POVRAY] Non Governmental Organization/ Software Foundation Web Site. Page reviewed in (2006) year. "Porvray" Available at <http://www.povray.org>
- [KDEVELOP] Non Governmental Organization/ Software Foundation Web Site. Page reviewed in (2006) year. "KDevelop" Available at <http://www.kdevelop.org>
- [SUBVERSION] Non Governmental Organization/ Software Foundation Web Site. Page reviewed in (2006) year. "subversion.colabnet" Available at <http://subversion.tigris.org>
- [SQLITE] Non Governmental Organization/ Software Foundation Web Site. Page reviewed in (2006) year. "Sqlite" Available at www.sqlite.org
- [CGAL] Non Governmental Organization/ Software Foundation Web Site. Page reviewed in (2006) year. "CGAL" Available at <http://www.cgal.org>
- [ECLIPSE] Non Governmental Organization/ Software Foundation Web Site. Page reviewed in (2006) year. "Eclipse CDT Project" Available at <http://www.eclipse.org/cdt>
- Efremov, I.(1988) "Geometría Superior" Book published by Editorial MIR
- Group of Authors.(2004) "GNU Scientific Library, Reference Manual" Technical Documentation
- Hanson, Eric H, et al..(1992) "The Interfal Skip List: A data structure for finding all intervals that overlap a point." Report of Research at Center for Geometric Computing, Department of

Computer Science

- Itasca Technical Staff.(2002) "Particle Flow Code in 2 Dimensions, Theory and Background" Technical Documentation
- Khamel, Ibrahim, et al..(1994) "Hilbert R-tree: An improved R-tree using fractals" Article in Proceedings of the 20th VLDB Conference, Santiago de Chile
- Liu, G.R..(2003) "Mesh Free Methods" Book published by CRC Press
- Lohner, R, et al.(2004) "A general advancing front technique for filling space with arbitrary objects" Article in International Journal for Numerics Methods in Engineering
- Lohner, R, et al.(1996) "Three-dimensional grid generation by the advancing front method" Article in International Journal for Numerics Methods in Engineering
- Lohner, R, et al.(1998) "An advancing front point technique" Article in International Journal for Numerics Methods in Engineering
- Mujinza, Ante.(1999) "The combined Finite Discrete Element Method" Book published by John Wiley & Sons, Ltd
- O'Rourke, Stephen.(2002) "Computational Geometry in C" Book published by John Wiley & Sons, Ltd
- [OSI] Non Governmental Organization/ Software Foundation Web Site. Page reviewed in (2006) year. "Open Source Initiative" Available at <http://www.opensource.org>
- Recarey, Carlos A..(2003) "Informe de Estancia en Cimne" Report of Research at CIMNE
- Recarey, Carlos A..(1999)"Modelación no estacionaria de las estructuras y el terreno" Ph.D thesis at UCLV
- Serón, FJ, et al..(1989) "Elastic Wave propagation with the Finite Element Method" Book published by John Wiley & Sons Ltd
- Vaughtan, Gary, et al..(2006) "GNU Autoconf, Automake and Libtool" Book published by Web Site
- [Weisstein] Weisstein, Eric H.(2001) "Taylor Series." online, see at <http://mathworld.wolfram.com/TaylorSeries.html>
- Worm Mortensen, Christian.(2003) "Fully-dynamic orthogonal range reporting on RAM" Report of Research at The It University of Copenhagen
- [IBM-128] Zeskel, Paul.(2003) "C/C++ development with the Eclipse Platform" online, see at <http://www-128.ibm.com/developerworks/opensource/>
- Zheng et Al.(2006) "Distributed Segment Tree: Support of Range Query and Cover Query over DHT" Article in The 5th International Workwhop in Peer To Peer Systems Proceedings