Universidad Rey Juan Carlos

Departamento de Ciencias de la Computación,
Arquitectura de la Computación, Lenguajes y Sistemas Informáticos y
Estadística e Investigación Operativa

Simulation and rapid development of virtual deformable objects for training applications

Tesis doctoral

Author: D. Jorge Gascón Pérez

Director: Dr. Miguel Angel Otaduy

Escuela Técnica Superior de Ingeniería Informática

6 de Octubre, 2014

Dr. Miguel A. Otaduy, Profesor titular de la Universidad Rey Juan Carlos, con NIE 72447035W.

CERTIFICA: Que D. Jorge Gascón Pérez, licenciado en Ingeniería Informática, ha realizado en el Departamento de Ciencias de la Computación, Arquitectura de la Computación, Lenguajes y Sistemas Informáticos y Estadística e Investigación Operativa, bajo su dirección, el trabajo correspondiente a la tesis doctoral titulada:

Simulation and rapid development of
virtual deformable objects for training applications

Revisado el presente trabajo, estima que puede ser presentado al tribunal que ha de juzgarlo. Y para que conste a efecto de lo establecido en la normativa reguladora del tercer ciclo de la Universidad Rey Juan Carlos, autoriza su presentación.

Móstoles, 6 Octubre 2014.

Fdo.: Miguel A. Otaduy

Contents

List of Figures

List of Tables

Abstract

Virtual Reality (VR) applications are employed for creating virtual worlds that are interesting in several domains. Some of these domains are: industrial design, entertainment, professional training, among others. The VR applications focused on training programs are a way to instruct professionals without the need of putting actual people and resources under riskier conditions. As the goal of these applications is to emulate reality, they have to be user-immersive. That means that they have to produce realistic response at interactive rates. Nevertheless, creating a VR aplication is a highly demanding task, its development requires a multidisciplinary team of professionals, and it also needs specialized tools. In our dissertation we have implemented a framework for rapid development of VR applications. This framework, called BlenderCave, is useful for creating VR applications capable of running in distributed, multi-display setups, such as these that are in CAVEs and Powerwalls.

As several VR applications rely on physics-based simulators, we present some contributions to enhance them. Specifically, we have designed new algorithms for some of the most important processing stages that a physics simulator performs. Regarding the simulation stage that deforms virtual objects, we have developed a fast technique that deforms dense volumetric objects. This deformation is performed by means of parallelized rasterization of embedding tetrahedral meshes, using graphics hardware. Enhancing this same stage, we have implemented a method for simulating coupling between objects that are joined together. In order to illustrate our method, we have created a haptic demo where complex human anatomy, with several couplings, is simulated interactively. Finally, the last of our contributions addresses the simulation stage that computes contact handling. Since adhesion is a very common phenomenon in nature, specially in biological forms, we have implemented a novel method to simulate adhesive contact. Our method robustly simulates adhesion using contact constraints and it is easily integrable with existing contact handling libraries.

Resumen

En ciencias aplicadas, uno de los mayores retos es la verificación de hipótesis. Por ello, a la hora de testear premisas es necesario crear y refinar conceptos, prototipos y pruebas. Los prototipos sirven para evaluar los nuevos productos y servicios bajo diversas condiciones, entre ellas los tests de viabilidad y aceptación del público. El problema es que su desarrollo requiere mucho tiempo y esfuerzo, y necesitan de instalaciones y maquinaria costosas. Los prototipos, en función de las pruebas a las que sean sometidos, pueden llegar a ser descartados, dañados o incluso destruidos, implicando un desperdicio de esfuerzo y materiales. Otras actividades, como el adiestramiento de profesionales, requieren la intervención de personas reales, el empleo de recursos y el uso de maquinaria real, lo que implica unos costes y riesgos considerables. Concretamente, el entrenamiento de los profesionales médicos implica aprender y practicar sobre pacientes reales, con los riesgos que ello supone. Además, ciertas dolencias son difíciles de reproducir, limitando el aprendizaje que se puede adquirir.

Afortunadamente, gracias al desarrollo de la informática gráfica, ahora es posible crear escenarios y modelos virtuales, lo cual brinda incontables ventajas en numerosos campos. No solo es posible crear prototipos virtuales, que permiten rápidas pruebas y modificaciones, sino que también es posible crear simuladores de entrenamiento y sistemas de planificación de resultados. Las aplicaciones de Realidad Virtual (RV), en combinación con los simuladores basados en físicas, son capaces de emular la realidad. Si bien, para ser completamente útiles han de ser totalmente inmersivos y dar respuesta al usuario a ritmos interactivos.

Antecedentes

Crear aplicaciones de Realidad Virtual (RV) que además simulen físicas es una exigente tarea sujeta a la resolución de numerosos problemas. De todos ellos hay tres que se tratan de resolver

en esta tesis, y son: Deformación de objetos virtuales, simulación de interacciones de contacto y composición de escenarios virtuales. Esta sección muestra las técnicas más relevantes que hasta la fecha han lidiado con estos problemas.

Deformación de objetos virtuales

Muchos objetos de la naturaleza sufren deformaciones debido a fuerzas e interacciones entre ellos. En esta sección centraremos nuestra atención en la simulación de objetos deformables elásticos. En informática gráfica hay varias formas de representar un objeto virtual, las dos representaciones más populares son la malla poligonal y los volúmenes densos de datos.

Deformación basada en mallas poligonales

Una malla poligonal representa la superficie de un objeto virtual, que es la parte que el usuario ve y la que interactúa con otros cuerpos en la escena. Para representarla se utiliza una estructura de datos llamada malla poligonal. Esencialmente, una malla poligonal es una colección de vértices, aristas y facetas que representan la superficie exterior de un cuerpo virtual. Este tipo de representación es muy popular en la informática gráfica, ya que existe hardware específico utilizado para renderizar mallas de forma acelerada. Además, es muy fácil deformar estas mallas, pues simplemente hay que cambiar las posiciones de los vértices para definir la deformación producida. Por contra, conseguir gran detalle implica aumentar el número de facetas, lo que implica utilizar mallas muy densas.

Deformación basada en volúmenes densos de datos

Aunque la representación de objetos mediante mallas poligonales es una técnica muy utilizada, algunos objetos virtuales tienen estructuras internas complejas que serían muy costosas de representar utilizando mallas. Este hecho dificulta su uso en ciertos entornos, como el de la informática médica. Las rejillas 3D regulares son una forma muy popular de almacenar información volumétrica. Este tipo de representación se utiliza con frecuencia en imagen médica, ya que es útil para representar las formas biológicas. Estos tipos de estructuras de datos tienen muchas ventajas, como la capacidad de ilustrar las estructuras internas de un objeto volumétrico de un sólo vistazo. Aunque muchas formas biológicas son inher-

entemente deformables, los algoritmos actuales para visualizar estas estructuras de datos no están optimizados para datos deformables. Y aunque ha habido muchos enfoques intentando deformarlas, todas ellas tienen numerosos inconvenientes. Esencialmente, nuestro enfoque utiliza la rasterización 3D de tetraedros, que es un algoritmo que toma como entradas el volumen denso de datos original, una malla de tetraedros envolvente y un campo de deformación aplicado sobre dicha malla. Como salida este algoritmo genera un nuevo volumen de datos deformado, adaptado a la malla contenedora deformada. Hay varias técnicas de rasterización [FLB+09, LK11, LHLW10, FLB+09, SCS+08, EL10], sin embargo nuestro enfoque combina ideas de todos ellos. El método que proponemos deforma el volumen completo a partir de mapeado de texturas 3D. También utilizamos un algoritmo de culling para evitar procesar los voxels que quedan fuera de cada tetraedro rasterizado.

Acoplamiento entre cuerpos rígidos y deformables

Uno de los problemas que se resuelven en esta tesis es la simulación de los objetos virtuales que están conectados entre si. Utilizamos un enfoque basado en restricciones de contacto, similar a [SSIF07], que utiliza muelles conectores. Además, empleamos un solver basado en gradiente conjugado, que resuelve islas de contacto, de forma similar a [PO09]. Respecto a simulación médica, hay trabajos anteriores que simulan modelos de la mano [SKP08], el cuello [LT06], la cara [SNF05], el torso [TSB+05, DZS08], o la parte superior del cuerpo [LST09]. Para las deformaciones de nuestra demo del hombro se utiliza simulación de elementos finitos, utilizando formulación co-rotacional [MKN+04]. Para aumentar el rendimiento utilizamos varias mallas de distintas resoluciones, unidas mediante una malla envolvente de tetraedros [MBTF03, SBT06, NKJF09]. Para la resolución del contacto se utilizan restricciones, cuya explicación viene resumida en el apartado siguiente.

Simulación de contacto

Una de las partes más importantes en la simulación de físicas es la resolución del contacto. Resolver correctamente el contacto evita las interpenetraciones entre objetos y permite la simulación de otros efectos como la repulsión, la fricción, la adherencia y muchos otros. Esencialmente, la simulación del contacto se divide en dos componentes: Detección de colisiones, que detecta intersecciones entre las superficies de los objetos y tratamiento del contacto, que se

encarga de evitar que haya interpenetraciones, es decir, que los objetos queden bien separados sin intersecar.

Detección de colisiones

Respecto a la detección de colisiones (CD, en sus siglas en inglés), hay un trabajo muy extensivo en [TKH+05, Sch13]. El objetivo principal de un algoritmo de CD es detectar la intersección de objetos lo más rápido posible. Este tipo de algoritmos toma como entrada un conjunto de objetos, y su misión es la de testear cada primitiva de cada objeto contra todas las demás. Por lo que el número máximo de tests geométricos sería de N^2, siendo N el número de todas las primitivas de todos los objetos de la escena. A fin de minimizar este inmenso número de tests, un algoritmo de CD se divide en dos fases:

Broad-Phase [CLMP95], que determina rápidamente los pares de cuerpos que potencialmente están colisionando, descartando rápidamente todos los demás.

Narrow-Phase es la que recibe pares de candidatos a colisionar y se encarga de hacer los tests geométricos que calculan la intersección exacta.

Hay otras formas de clasificar las colisiones, como es la clasificación temporal, por un lado están los métodos discretos, que calculan la colisión de los cuerpos en un instante determinado, y por otro los métodos contínuos [BFA02, RKLM04, ORC07, TCYM09], que son más exhaustivos porque tienen en cuenta las trayectorias de cada primitiva dentro de un lapso de tiempo determinado. El resultado que genera un algoritmo de detección de colisiones es un conjunto de datos sobre las primitivas afectadas por las colisiones. Estas después se utilizarán en la etapa de resolución del contacto, explicada a continuación.

Resolución del contacto

Los métodos de resolución de contacto usan como entrada las colisiones detectadas en la etapa de detección de colisiones. La misión de esta etapa es eliminar dichas colisiones, dejando a los objetos bien separados. Si esta etapa no funcionase así, los objetos se atravesarían, causando un comportamiento muy poco realista. Existen varios enfoques para resolver el contacto, los dos enfoques más utilizados, por ser los más físicamente correctos, son estos que generan fuerzas de repulsión que obligan a las partes que colisionan a separarse. En concreto, los dos enfoques

más utilizados son los **métodos de fuerzas de penalty** [TPBF87, MW88, WVVS90, TMOT12] y los métodos basados en restricciones [BFA02, DAK04, PPG04, DDKA06, OTSG09].

Penalty-Based Methods:

Los métodos de fuerzas de penalty [MW88, BJ07a, HVS+09a] se emplean en simulaciones interactivas. Esencialmente, este método resuelve las colisiones definiendo una fuerza repulsiva que se aplica a las superficies que están intersecando. Esta fuerza se basa en la profundidad de penetración de ambas superficies. Este enfoque es más impreciso y puede llevar a inestabilidades, por lo que hay numerosas publicaciones tratando de perfeccionar este método y reducir sus inconvenientes [FL01, BFA02, HTK+04, HS04, TKH+05, TSIF05, FBAF08, Dru08, AFC+10].

Contact handling constraints

Las restricciones de contacto [Wri02] son un método más robusto y preciso, y garantizan la no penetración al final de cada paso de simulación. Lo consiguen gracias a formular la respuesta a colisiones como un problema de optimización con restricciones. Este método está basado en un sistema lineal de ecuaciones que utiliza multiplicadores de Lagrange. Cada restricción está definida como una condición que siempre debe cumplirse. De esta forma, cuando el sistema de ecuaciones es resuelto, las fuerzas de repulsión resultantes son las fuerzas exactas a aplicar sobre las superficies que intersecan. Hay varios tipos de algoritmos de restricciones de contacto: Los iterativos [BFA02, GBF03a, MHHR06], que son más sencillos pero menos exactos, ya que resuelven localmente las restricciones una a una, y los LCP [CPS92] (Linear complementarity Problem), que resuelven todas las restricciones a la vez, por lo que son más robustos y mucho más exactos, a costa de ser más costosos computacionalmente. De todos los tipos de LCP, en muchas de nuestras pruebas hemos utilizado anticipación iterativa de restricciones [OTSG09].

Fricción

La fricción, también llamada rozamiento, es una fuerza que se opone al deslizamiento tangencial entre dos superficies en contacto. Si esta fuerza no existiese, las superficies podrían deslizarse sin ninguna resistencia, como sobre hielo. Sobre la fricción existen trabajos anteriores en los que se simula rozamiento en cuerpos rígidos [Lot84, Bar91, ST96, ST00, KEP05], cuerpos articulados [AP97, KSJP08] y cuerpos deformables [BFA02, WVVS90].

Contacto adhesivo

La adherencia es el fenómeno en el cual dos superficies en contacto se quedan parcial o totalmente pegadas. Este fenómeno consiste en una fuerza de dirección normal a las superficies que se opone a las fuerzas de tracción que intentan separarlas. La adherencia proporciona más realismo y riqueza a las simulaciones físicas, especialmente en superficies húmedas, estructuras biológicas, mucosas, etc... Como la adherencia está íntimamente unida al fenómeno de contacto, se ha modelado este fenómeno como una formulación basada en restricciones. Dicha formulación está conectada a un algoritmos LCP de resolución de contacto. En gráficos por computador, la adherencia ha sido modelada anteriormente utilizando fuerzas de penalty [JL93, CJY02, BMF03, WGL04, SLF08]. En el campo de la mecánica, la adherencia fue formulada por Fremond [Fre87], mientras que Raous et al. [RCC99] estableció una relación entre la termodinámica y la adherencia. Un sumario de estos enfoques pueden ser encontrados en el libro de Wriggers [Wri02]. Nuestro enfoque adapta todas estas ideas para crear un algoritmo que simula la adherencia en gráficos por computador. Además, nuestra implementación es muy robusta, ya que utiliza restricciones de contacto para modelar la adherencia. Además, nuestro enfoque es válido para simulación de adherencia en rígidos y deformables y se puede integrar en otras librerías de resolución de contacto existentes, como [BUL, ODE, SOF].

Desarrollo de aplicaciones de RV

Como ya hemos visto, el desarrollo de aplicaciones de RV es un proceso conocido por ser muy exigente y cargado de complejidad. Esta tarea no solamente requiere de una gestión de dispositivos avanzada y de algoritmos de simulación avanzados, sino que también demanda el uso de herramientas especializadas de prototipado rápido. Dichas herramientas son útiles para crear los objetos y escenarios virtuales de la aplicación. También es necesario configurar los objetos de forma que la simulación se inicie en un estado seguro y libre de interpenetraciones. Además, las aplicaciones de RV diseñadas para entornos multipantalla, como las cuevas de realidad virtual, o CAVEs [CNSD93], suponen retos añadidos. Al ser necesario generar fotogramas de más resolución y tener una consistencia de brillo similar [Sto01] entre pantallas. Existen algunas soluciones específicas para el desarrollo de aplicaciones de RV destinadas a CAVEs. Un ejemplo de ellas es la librería CAVELib [CN95], creada por los inventores de las CAVEs

pero limitada para su hardware específico. Hay otros frameworks, como por ejemplo VRJuggler [BJH+01], DIVERSE [KSA+03], Studierstube [SRH03], RB2 [VPL90], DIVE [CH93], dVS [Gri91], Avocado [Tra99], VRPN [THS+01], Equalizer [Equ], MRToolkit [SLGS92], dVise [Ghe97], EAI' Wordltoolkit [Ini97], EON Studio™ [Eon] o INVRS [Ant09]. Respecto al desarrollo de videojuegos, existen también varios motores de juegos que podrían adaptarse para funcionar en cuevas de realidad virtual, como por ejemplo [OGR, OSG, Epi, Cryb, Unib]. Finalmente, existen herramientas tipo sandbox que permiten la creación de contenidos y la definicion de la lógica de la aplicación utilizando editores visuales.

Objetivos

Los objetivos de esta tesis se organizan en dos grupos: por un lado se incluyen nuevas técnicas para el desarrollo rápido de aplicaciones de RV. Por otro lado se han desarrollado mejoras para las simulaciones físicas en las que participan objetos deformables.

Respecto a la creación rápida de aplicaciones de RV, se ha desarrollado un framework llamado BlenderCave. Este framework incluye herramientas visuales que sirven para crear nuevos objetos y escenarios virtuales. Dichas herramientas están inspiradas en las que se utilizan en el desarrollo de videojuegos. Además, hemos diseñado un protocolo de sincronización de red que permite a las aplicaciones funcionar en entornos distribuidos multipantalla, como son las cuevas de realidad virtual (CAVEs) y las pantallas PowerWall.

Respecto a las contribuciones para mejorar los simuladores físicos actuales, se han desarrollado algoritmos que enriquecen la etapa en la que se calculan las deformaciones de los objetos y también nuevos métodos para la etapa de resolución del contacto entre superficies.

Respecto a la fase de deformación de objetos virtuales, presentamos un algoritmo que deforma objetos volumétricos densos de forma interactiva. Dicho algoritmo es capaz de deformar millones de voxels mediante rasterización de mallas de tetraedros. Esto se consigue mediante el uso de hardware gráfico y de una elevada paralelización a nivel de voxel.

Mejorando esta misma fase, se ha implementado un método que simula acoplamientos entre los objetos que están unidos entre sí. A fin de ilustrar dicho método, se ha creado una demo háptica destinada a explorar la anatomía humana, donde los acoplamientos están simulados interactivamente.

Finalmente, con respecto a la etapa de resolución del contacto, también se incluye un nuevo método que simula contacto adhesivo. Este método es muy robusto porque modela la adherencia a partir de definir restricciones de contacto. Dicho algoritmo genera efectos muy interesantes y puede ser integrado en las librerías de resolución de contacto existentes, como [BUL, ODE, SOF].

Metodología

El empleo de herramientas de prototipado rápido, unido a nuevos algoritmos de deformación de objetos y tratamiento del contacto, pueden aumentar el realismo de aplicaciones de RV que simulan la física de objetos deformables.

Para respaldar esta tesis, los próximos capítulos presentarán las siguientes contribuciones, las cuales mejoran la creación de aplicaciones de RV y el funcionamiento de los simuladores basados en físicas:

1. Se ha desarrollado un método para deformar objetos volumétricos de forma interactiva. La deformación se consigue mediante rasterización de una malla de tetraedros envolvente. Dicho método está acelerado por hardware gráfico y es capaz de deformar millones de voxels por segundo. Para demostrar su funcionamiento se ha creado una demo interactiva de palpación médica de un abdomen humano. Todos los detalles de esta contribución se pueden encontrar en el capítulo 3, así como en la publicación [GEP+13].

2. Se ha diseñado un algoritmo para simular contacto adhesivo entre superficies de objetos virtuales. Este es un método muy robusto que define la adherencia de forma unificada, utilizando restricciones de contacto. Todos los detalles de este algoritmo se pueden encontrar en el capítulo 4, así como en la publicación [GZO10].

3. Se ha implementado un sistema para configurar una demo interactiva de palpación médica de un hombro humano. Esta demo muestra toda la anatomía interna del hombro, incluidos sus huesos, músculos y ligamentos más importantes. Se han empleado herramientas visuales que han permitido configurar las características mecánicas de cada órgano, así como los puntos de acoplamiento entre unos órganos y otros. Asimismo, se ha configurado toda la escena virtual para que la simulación comience en un estado seguro y libre de colisiones. Todo este sistema se apoya en herramientas visuales y scripts que realizan tareas específicas. En la publicación [OGG+10] y en el capítulo 5 se explican todos los detalles del proceso realizado.

4. La creación de aplicaciones de RV es una tarea muy exigente. Dicho proceso necesita de un equipo multidisciplinar y herramientas especializadas. Para facilitar esta labor, se ha implementado un framework para el desarrollo rápido de aplicaciones de RV. Este framework, conocido como BlenderCave, combina herramientas visuales con un motor de juegos muy potente y versátil, en constante desarrollo. BlenderCave es útil para crear aplicaciones de RV que funcionan incluso en entornos distribuidos multi-pantalla, como los que se pueden encontrar en CAVEs y Powerwalls. En la publicación [GBEO11] y en el capítulo 6 se muestran todas las características de este framework.

Conclusiones

Desarrollar aplicaciones de RV es una de las tareas más complejas conocidas por toda la industria del software. Ello se debe a que estas aplicaciones tienen que lidiar con algoritmos de simulación muy complejos, también tienen que gestionar todos los objetos virtuales y sus estructuras de datos adjuntas (mallas, texturas, animaciones, etc...), y además tienen que interactuar con periféricos de entrada salida (dispositivos hápticos, pantallas y cascos de realidad virtual, etc...) a una velocidad suficiente para funcionar a ritmos interactivos. Todos estos desafíos requieren el uso de herramientas altamente especializadas y de un equipo multidisciplinar capaz de utilizarlas.

En esta tesis se muestran nuevas herramientas para acelerar la creación de aplicaciones

de RV. Todas las herramientas están integradas en un framework, llamado BlenderCave, que permite crear aplicaciones completas de forma muy rápida. Además, se ha desarrollado un protocolo de sincronización de red que habilita a estas aplicaciones para funcionar en entornos distribuidos multi-pantalla, como los que encontrados en CAVEs y PowerWalls. A pesar de sus bondades, BlenderCave tiene algunas limitaciones: El protocolo de red desarrollado es muy eficiente para sincronizar cambios de estados producidos por eventos, pero necesitaría de mejoras para cambios más complejos, como los producidos en las simulaciones físicas de objetos deformables. Otro punto a perfeccionar sería el reparto de carga en configuraciones en las que la relación entre pantallas y nodos de ejecución no está balanceada equitativamente. No obstante, muchos de estos problemas son específicos de la aplicación a ser desarrollada y en ese nivel pueden ser resueltos. Respecto a las líneas futuras, BlenderCave fue liberado como software libre y desde entonces está siendo desarrollado muy activamente por varios grupos de investigación europeos.

Esta tesis también ofrece innovaciones que mejoran los simuladores físicos existentes. En concreto, nos hemos centrado en las etapas de deformación de objetos y de resolución de contacto. Respecto a la deformación de objetos virtuales, una de nuestras innovaciones consiste en un algoritmo de deformación de objetos volumétricos. Este algoritmo está altamente paralelizado y utiliza hardware gráfico, lo que le permite deformar millones de voxels a ritmos interactivos. Además, el algoritmo cuenta con una técnica de culling que aumenta la velocidad de rasterización en hasta 1.5x veces, dependiendo de la demo. Para ilustrar dicho algoritmo se ha creado una demo interactiva de palpación médica del abdomen humano. El método desarrollado es muy prometedor pero no está carente de limitaciones: el actual algoritmo de culling da lugar a falsos positivos. Afortunadamente, estos falsos positivos no dañan excesivamente el rendimiento general del algoritmo. Otra limitación es el ligero desenfoque provocado por la rasterización de voxels, que podría mitigarse con métodos de filtrado más costosos pero con mejor calidad. Hay algunas líneas futuras que serían interesantes de desarrollar, como por ejemplo la utilización de hexaedros como elementos contenedores, entre otras.

En esta misma etapa de deformación de objetos, se ha diseñado un método para simular acoplamientos entre objetos virtuales que están unidos entre sí. Esta situación es muy común

en anatomía humana, donde órganos de diferentes características (huesos, músculos y ligamentos) están conectados y donde pueden suceder situaciones de contacto complejas de resolver. El algoritmo desarrollado resuelve estas situaciones de forma unificada, calculando las nuevas posiciones de los órganos para que estén libres de colisiones. Actualmente nuestra técnica tiene algunas limitaciones: Otra limitación se debe a las aproximaciones que utilizamos, que alejan la demo de ser anatómicamente exacta. Asimismo, hay órganos, como la bursa, que han sido omitidos, pues de otra forma se dispararía el número de contactos a resolver y la demo dejaría de ser interactiva. Por otro lado, hay muchas líneas futuras que se podrían seguir, como la inclusión de corte de órganos y la sutura, lo que implicaría implementar soluciones para lidiar con cambios topológicos y con contacto con hilo de sutura.

Respecto a la etapa de resolución del contacto, se ha implementado un algoritmo para simular contacto adhesivo. Este tipo de contacto es interesante porque es muy frecuente en la naturaleza, especialmente entre mucosas y otras estructuras biológicas. Nuestro método utiliza restricciones de contacto para calcular la adherencia, por lo que es muy robusto. Sin embargo no está carente de limitaciones: Respecto a la conexión entre adherencia y fricción no hemos encontrado ningún modelo termodinámico en la literatura existente, lo que hacemos en su lugar es aplicar la fuerza más restrictiva de las dos. Otra de las limitaciones es que se requiere almacenar la historia de los contacto ocurridos, a fin de poder calcular la evolución de su adherencia en el tiempo. También hay que comentar que hay problemas de convergencia, ya conocidos, que sufren los solvers utilizados, lo que nos impide garantizar una tasa de actualización mínima. Otra limitación es que el algoritmo solamente puede calcular adherencia en superficies bien definidas, lo que significa que aún no se puede aplicar en fluídos simulados ni en otros tipos de entidades. Como trabajo futuro, sería muy interesante aplicar nuestro método a otro tipo de materiales, como por ejemplo los viscoplásticos.

Finalmente, destacar que el desarrollo de aplicaciones de RV que simulen físicas es un problema muy complejo, pues quedan multitud de desafíos aún no resueltos. Aunque los resultados de esta tesis muestran que nos dirigimos hacia mejores herramientas y procesos, aún estamos lejos de conseguir animaciones indistinguibles de la realidad, suponiendo que ello sea posible.

Acknowledgements

Six years ago I could not imagine that today I would complete a PhD having as advisor Miguel A. Otaduy.

I want to thank him for the lessons learned, specially those that inspired all of us to improve ourselves, for forcing us to aim to higher goals, for teaching us to be ambitious, for never giving up although everything appears to fail, for all the discussions, and for helping me to reach our deadlines at time, without surrender.

To be honest, there are several advices I have received from him during this time, and I am pretty sure that all of them will be essential in my next steps I hope to accomplish in the next years. For me, the most valuable lesson learned during this time is “With enough work and dedication, nothing is impossible”.

Thanks to the rest of the members of my Ph.D. committee by sharing their experience in the form of invaluable guidance. In addition, I wish to thank to the reviewers that provided feedback for each one of our publications.

I would like to acknowledge the support and encouragement of Professor Luis Pastor and the entire GMRV (Grupo de Modelado y Realidad Virtual) group at the Universidad Rey Juan Carlos. Doing a Ph.D. is much easier when you are surrounded by people so passionate about graphics and virtual reality. Thanks to every teacher in the Master of Computer Graphics, Video Games and Virtual Reality.

Thanks to Sara C. Schvartzman, Álvaro Pérez, Javier Zurdo, José María Bayona, José

Miguel Espadero, Rosell Torres, Carlos Garre, Eder Miguel Villalva, Sofía Bayona, Ángel Rodriguez and Luis Pastor for their collaboration in our publications. These publications would not have been possible without them.

Regarding BlenderCave framework, I would like to thank Julian Adenauer, Damien Touraine and Brian FG Katz, for being the first contributors to BlenderCave and for your interest to transforming it into the excellent virtual reality engine for CAVEs that currently is.

I would like to thank all the contributors to all the open source projects that have enabled us to develop our projects, demos and software libraries, you are too many to be listed here, thank you!

I would like to thank Ton Roosendaal, and all the Blender community, for its incredible work developing Blender 3D, my favorite 3D authoring suite that many times has been useful for our projects. Thanks to Martinsh Upitis ('martinsh' from BlenderArtists.org), author of some scenes we have used in our demos for BlenderCave and thanks to Stefan Roettger's volume library and the OsiriX DICOM Viewer's site, for being so generous sharing the volumetric models that have been useful for our deforming algorithm.

Finally, I wish to thank every member in the GMRV group, because they are the best teammates that one could wish. Specially to José Miguel Espadero, Álvaro G. Pérez, Sara C. Schvartzman, Iván Alduán, David Miraut, José María Bayona, Javier Zurdo, Marcos Novalbos, Carlos Garre, Laura Raya, Luis Miguel Serrano, Angela Mendoza, Eder Miguel Villalva, Rosell Torres, Jaime González, Juan Pedro Brito, Richard S. M., Loïc Corenthy, Alberto Sánchez, Gabriel Cirio, Óscar Robles, Caroline Larboulette, Jorge Lopez-Moreno, Sofía Bayona, José San Martín, Pablo Toharia, Marcos García, Susana Mata, Francesco Cosco, Jesús Pérez, Aaron Sújar Garrido and many others, you know who you are.

Without their brilliant ideas, suggestions, research discussions and support, this work would have not been possible.

I deeply appreciate the professional opportunity given by GMV, not only supporting my research, but also giving us so many flexible and innovative work guidelines, and for allowing us to publish the developed algorithms. Thanks to Gonzalo Mora and Beatriz Tierno for clinical advice and the Healthcare team at GMV for discussions.

I also thank all the faculty and staff of the Department of Computer Science at the University Rey Juan Carlos, for making ETSII school such a wonderful place to work at. And I thank Ministry of Education and Science of Spain for its granted funds that have allowed me to focus on my research.

Getting this far involved considerable work, and there are many people who have made it easier by letting me be their friend and by supporting me at all times.

I want to thank first Unai Fuente, for his huge optimism and his particular point of view about life. And my buddies David, Alfonso, Luis, Fer and Jorge, for all these fun and crazy meetings watching sub-genre movies and playing mus and all of those fun boardgames. Thanks to my buddies Sergio, Hector and Alberto Gómez, for these good trips and conversations about everything and nothing. And thanks to Luisete, Fran, Dario, Victor, and the entire GAEM, for showing me another fun meaning of the word *adventure*. Thanks to my friends Gabriel, Alvarito, Ángel Daniel, Arturo, Carlos and many others. Thanks to every member of the cultural association Ágora, for all these fun trips and jokes.

But also all my relatives and other friends, who always filled me with energy when I went home, and then I was able to keep working hard back in URJC, most of them redefined the world *hospitality*.

I would like to appreciate to Pablo of the Vivero de Empresas de Móstoles for his support, encouragement and understanding, I hope to follow and accomplish my next goals and dreams with his help.

None of this work would have been possible without the support of immediate family and

close friends. I would specially like to thank Asun, Jorge, Paco, Mila, Carmen, Sacra, Antonio and Arturo for their support and love.

My deepest thanks to my parents, Fermín and Chari, my sister Victoria and her partner Fran, and my grandparents, Rosario and Tomás, for their love, care and encouragement in the moments of weakness, Muchas gracias.

This thesis would not have been possible without the funding from URJC - Comunidad de Madrid (proj. CCG08-URJC/DPI-3647), the Spanish Science and Innovation Dept. (projects TIN2007-67188, TIN2009-07942, PSE-300000-2009-5, IPT-2012-0401-300000 and TIN2012-35840), and the EU FEDER fund.

Chapter 1

Introduction

One of the greatest challenges in applied sciences is how to evaluate and how to verify hypothesis. In this way, in order to check premises it is necessary to develop concepts, prototypes and other kind of tests, and to make refinements afterwards. Disciplines like architecture, commerce and industrial engineering have relied on mock-ups in order to evaluate designs, behaviors and acceptance under specific conditions. The development of these prototypes often requires the use of expensive processes, facilities and machinery. In order to check performance, robustness and durability, some parts of the prototypes need to be manufactured employing expensive materials. Frequently these parts are fatigued, damaged or destroyed in the process, specially in durability tests of engines or in crash tests of vehicles. Several industries require expensive structures only to test one or few properties. That happens, for instance, in aeronautics and automobile industries, that require the use of wind tunnels and other expensive facilities in order to test and refine aerodynamics. Architecture and commerce require regulatory compliance and acceptance tests, usually from long and expensive market studies before refining the prototypes and beginning the production. Regarding design acceptance, usability and other characteristics, some of the prototypes are designed for the interaction with humans. As result, several of these prototypes are discarded or replaced by others, and unfortunately, the manufacture of these mock-ups requires several resources and time.

Other activities, like training for professionals who use heavy machinery, imply expensive and dangerous training programs because they have to use the actual machinery under real circumstances. That requirement implies working under riskier conditions, that demand higher costs and taking unnecessary threats. Other kinds of learning, specially training programs fo-

cused on surgeons and other medical professionals, require operating on living patients, which leads to additional risks over their lives and integrity. Moreover, some diseases are rare or hard to reproduce, making harder the learning of their recovery procedures. Finally, operating on cadavers or other specimens requires preservation facilities and other expensive equipment.

Fortunately, thanks to the increasing computation power of modern computers, the development of new design techniques like Computer-aided design (CAD) and disciplines like Computer-generated imagery (CGI), it has been possible to create virtual models of the products to be developed, saving time and resources. The use of virtual models of a product has many advantages: not only is possible to evaluate the design and appearance of a model before building it, but also is easier and cheaper to perform modifications, frequently in an interactive manner. Recently, with the appearance of new human-machine interfaces and haptic manipulators is possible to create immersive scenarios for potential users to operate virtual tools and to manipulate virtual objects. The innate consequences of this new technology, called Virtual Reality (VR), are revolutionary: Not only is possible to develop powerful immersive training applications, that provide a faster and more complete learning for professionals, but also it provides new procedures for interactive prototyping and huge savings of time and resources. In figure 1.1 we can find several domains where virtual reality is successfully employed nowadays.

Figure 1.1: Examples of applications where Virtual Reality is successfully employed.

The development of physics-based numerical simulators brings to the VR users several new possibilities. Not only they give to engineers powerful tools to evaluate robustness and other characteristics of the model and its components, but also they give a realistic way to produce physics-based accurate animations of bodies, frequently at interactive rates. These simulators have the capability to emulate real-nature physics in order to simulate forces, velocities, temperatures and other magnitudes. There is a wide range of simulators depending on the kind of objects and magnitudes to be simulated. The kind of simulator that we use for our demos is able to compute forces and interactions between rigid bodies and deformable models. For each object, it can predict effects like forces, accelerations, velocities and positions. In addition, for deformable objects it also can compute stress, strain and deformations. In order to deal with interactions between objects, a simulator has several components, like collision detection algorithms and contact handling methods. As result they are able to simulate accurate contact simulation, repulsive forces to have all the objects well-separated, friction and other effects.

Finally, more advanced simulators can compute interaction between solid or deformable objects and fluids, like air flow, smoke or water. As result, with these simulations is possible to reproduce the behavior of virtual models as if they were real, before/instead of building them in real scale.

There are several commercial simulators focused on training professionals in activities like vehicles and heavy machinery driving, manipulation of dangerous goods or surgery training for medical professionals. These technologies are also useful in the entertainment industry. Currently is very common to find physics-based simulators used for the generation of impressive visual effects in films and video games, in most of them we can find simulations of fluids, fire, smoke, fragmentation of objects and many more, as shown in figure 1.2.

This dissertation is mainly focused on medical training VR applications. These training applications allow professionals to practice virtual surgery operations, like palpation, manipulating haptic tools over simulated organs. With the aid of these applications is possible to practice surgery exercises of cases that might be difficult to reproduce in real world. For instance, there exist simulators that implement a complete solution for arthroscopic surgery training, as shown in figure 1.3. Thanks to these applications, professionals can improve their skills doing exercises and reproducing specific cases, using almost the same tools applied to real arthroscopic interventions. Each exercise has performance evaluation tools built-in, allowing to measure the

Figure 1.2: © Warner Bros Pictures. In these two scenes of the film *Pacific Rim* (2013) we can find simulations of rigged articulated bodies (left) and rigged soft bodies with simulation of virtual muscles (right). In this film both characters are animated by simulation of physics while they interact with other fluid simulations, like water, rain and smoke.

performance and expertise of each professional. Other set of VR tools applied to medicine are the planning applications. This kind of systems allow to predict the effects of a medical treatment over a patient without compromising his/her integrity. These planning applications are extremely useful and can save lives, as they can compute the correct treatment doses to be applied to the patient according the characteristics of his/her disease. Some examples of planning medical applications can be found in radiotherapy, because they can compute the correct amount of radiotherapy dose that has to be applied to the tissues affected by cancer.

In order to have a completely immersive simulator, we need some requirements to be accomplished, like a virtual scenario populated with highly detailed virtual assets. They also have to be animated in a physically realistic manner at interactive rates. If some of these conditions are not met, the sense of immersion is lost and the application becomes useless.

Developing this kind of applications implies the accomplishment of many challenges, not only it is necessary to model (or to capture) all the virtual assets required in the scene, but also it is necessary to define their mechanical properties. Many performance and behavior settings need to be adjusted and these requirements take time and several tests before finding an accurate, robust and fast enough simulation. A graphics rendering engine is required for rendering all the objects in the scene and showing a visual response to the user. In addition, a physics simulator is needed to simulate the behavior of each virtual body and to solve the interactions between all the simulated objects. Typically this kind of interactions are solved using a collision detection

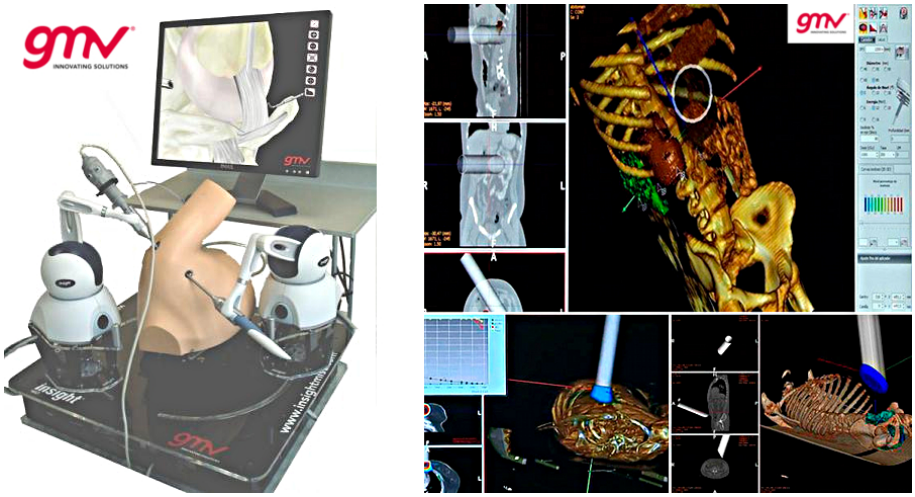

Figure 1.3: © GMV. Insight: Shoulder arthroscopic simulator designed for training arthroscopic surgery (left). Radiance: IORT Radiotherapy planner designed for computing the correct radiotherapy dose after a surgical removal of a tumor (right).

algorithm, -that detects intersections between objects-, working in tandem with a contact solver, -that computes repulsion forces in order to eliminate those intersections detected before-. All of these algorithms are designed carefully to solve these interactions in a robust manner at interactive rates.

This dissertation is focused on two problems: simulation of deformable objects and fast development of VR applications that simulate them.

We will then briefly present our contributions and outline the contents of this thesis.

1.1 Volumetric representation of 3D objects

Typically, in Computer Graphics, a virtual object is represented by its surface. This surface is what the user sees and it is the part of the object that interacts with other objects. The surface of an object is represented by a polygon mesh, which essentially is a collection of vertices, edges and faces. Polygonal meshes have several advantages: they are able to render fast because there exist specialized hardware, and they can be deformed easily. Nevertheless, biological structures have internal structures with several details inside, as shown in figure 1.4. Actually, it is not practical to represent these objects using polygonal meshes, and instead we use volumetric representations.

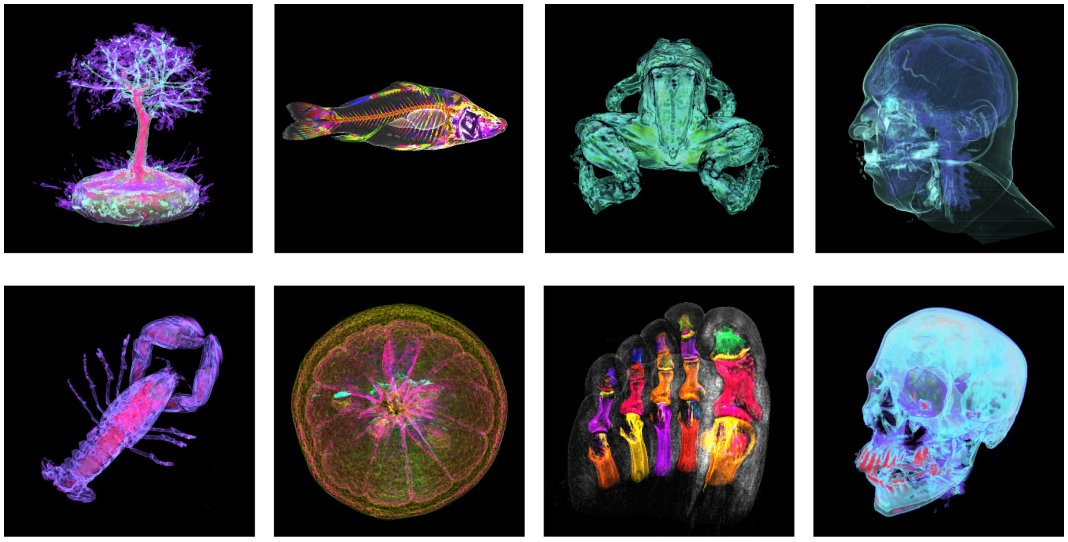

Figure 1.4: Some volumetric examples from Stefan Roettger's volume library. All of these models are captured from real specimens.

There exist algorithms to represent these dense data volumes, using raytracing techniques for visualization. Fortunately, these techniques allow not only to display the complete volumetric object but also it is possible to discriminate parts of it showing only specific structures inside. Unfortunately, these rendering algorithms are not optimized for the representation of deformable volumes. In order to fix this problem, we have implemented a novel 3D rasterization algorithm that generates new deformed volumes.

1.2 Simulation of adhesive contact

It is necessary to simulate interactions between objects in a robust manner. To that end it is necessary to detect contacts between objects and solve them in order to prevent interpenetrations. Otherwise, some objects could go though each other, producing an unrealistic behavior. Typically, contact solving methods require collision detection algorithms. These algorithms detect geometrical intersections between the primitives of the surface of each virtual object. The more detailed virtual objects are, more intersection tests have to be performed, because there is a test between each pair of primitives. The maximum number of geometric tests is N^2, being N the number of primitives of all the objects in the simulation. Modern collision detection algorithms use one or more culling algorithms, aimed to discard tests between false positives as fast as

possible. The most popular techniques are based on bounding volume hierarchies (BVH) or in spatial partitioning. BVH algorithms consist in bounding parts of each model inside volumes like Spheres, Axis Aligned Bounding Boxes (AABBs), Oriented Bounding Boxes (OBBs) or other kind of more complex volumes. Essentially, two primitives are free of collision if they belong to two bounding volumes that do not intersect, so the geometric test between them can be avoided (culled). If two volumes do not intersect, it is not necessary to check collision between their children volumes. Spatial partitioning works in a similar manner, but in this case the space is partitioned in a grid, that can be regular or not. Primitives that are in different grid cells do not collide, so geometric intersection checks between them can be avoided.

When two objects are in contact, the next step consists in performing collision detection between every primitive of those objects, following the same techniques introduced before. After that, each detected collision is sent to the contact handling algorithm, which computes repulsion forces in order to separate the affected primitives. A robust contact handling method is capable of preventing all the detected interpenetrations.

A popular manner to compute the exact forces to prevent collisions is using constrained contact handling. This method formulates and solves a constrained optimization problem. Each intersection is a kind of constraint, that is, a mathematical element that determine conditions that always have to be met. In this way, when the problem is solved, the exact repulsion forces are computed and the affected objects finish the current time step remaining well-separated and free of interpenetrations.

There are other interesting phenomena related to the contact itself, like friction. Friction is a force that opposes to tangential movement between contacting surfaces and it is explained in detail in subsection 2.2.3.

Another interesting phenomenon in relation to contact is adhesion. Adhesion is a potential energy produced by the contact between sticky surfaces. This energy produces a force that opposes to traction forces that might try to separate or slide the glued surfaces. In figure 1.5 we can see some examples of sticky objects in the real world. Adhesion is also present in the human skin, because of the existence of humidity on its surface.

Adhesion is a very common phenomenon in nature, and it provides additional realism and richness to physics-based animations. Adhesion is present where biological structures are involved, like wet surfaces, mucous membranes and others. As far as we know, our adhesion

Figure 1.5: Some examples of adhesion phenomena in the real world.

model is the first physically based model to be applied to computer graphics simulations.

Essentially, adhesion is a potential energy stored between two surfaces in contact. This energy increases when these surfaces are compressed against each other, and it decreases when there is traction or sliding between both surfaces. When adhesion energy is completely dissipated those surfaces are separated.

In this dissertation, an algorithm to simulate adhesion is formulated and implemented. In addition, our method is designed for its easy integration in other constrained-based contact handling algorithms.

1.3 Authoring of virtual reality applications

The development of VR applications is a multidisciplinary process well-known to be complex. A VR application requires the most efficient algorithms and a set of components (also called engines) that manage hardware devices, graph scene, rendering of virtual objects, physics sim-

ulation and many others. In addition, the creation of assets demands the use of rapid authoring tools. These tools are useful for modeling, texturing, animating and configuring each virtual body, and for compositing the entire scene. One of the problems that implies the use or third party tools is the exchange of information between the VR application and the used tools, in this way the use of exportation scripts allows the conversion of the original file format of each asset to the application's native format. After that, a stack of tests is required in order to find the most adequate mechanical parameters for each object, just like for obtaining an interactive, plausible simulation.

In other activities, like development of video games, a set of tools called sandboxes are becoming quite popular, specially for providing a powerful set of tools that accelerates the composition of each game level. Taking these ideas as inspiration, we have developed a fast prototyping framework, called BlenderCave. BlenderCave is a visual tool, based on a popular modeling suite, that provides a very fast pipeline for creation of VR applications. Detailed information about BlenderCave is shown in chapter 6.

Authoring tools, like 3D modeling suites, are specially useful for configuring the assets used in the VR application. Thanks to these tools, we created a set-up to configure the virtual muscles, bones and ligaments of our human shoulder demo, shown in chapter 5.

With these tools, it is possible to configure the initial state of each object in order to keep them well-separated from each other at the beginning of the simulation. Otherwise the simulator could not start in a safe configuration and the contact handling routines could not perform properly at all.

1.4 Contributions of this Thesis

The goals of this dissertation are: the development of techniques for easing the creation of VR applications and new contributions to improve the behavior of deformable bodies in physic-based simulations.

Regarding the rapid creation of VR applications, we have developed a framework called BlenderCave. This framework includes tools that are inspired from the ones employed in the state-of-the-art video game development. In addition, we have designed a network synchronization protocol for enabling these applications to work in distributed multi-display systems.

With respect to our contributions for improving physics-based simulations, in this thesis we have made some advances in specific processing stages of a simulator. Specifically we have designed algorithms that enrich the stages of object deformation and contact handling.

Regarding the objects deformation stage, we show an innovation consisting in a parallelized algorithm that deforms dense volumetric objects. This algorithm employs graphics hardware and is able to deform millions of voxels by rasterization of tetrahedral meshes. Enhancing this same stage, we have implemented a method that simulates coupling between objects that are joined together. In order to illustrate our method, we have created an interactive haptic demo for exploring human anatomy, where many couplings are simulated interactively.

Finally, with respect to the contact handling stage, we include a novel method to robustly simulate adhesive contact, using contact constrains. Our algorithm can be integrated in existing constrained contact handling libraries, like [BUL, ODE, SOF].

The following thesis statement summarizes the contributions of this dissertation.

1.4.1 Thesis statement

The employment of fast prototyping tools, joined with new algorithms of collision detection and contact handling, can improve realism and efficiency of VR Applications that simulate physics of deformable objects.

To support this thesis, the next chapters will present the following contributions:

1. A method to deform dense data volumes interactively (chapter 3). In this chapter we introduce a fast method for hardware-accelerated rasterization of volumetric objects. This method is interactive and capable of deforming millions of voxels per second. In order to demonstrate its performance and applications, an interactive demo with a volumetric human abdomen has been implemented.

2. A fast algorithm that implements adhesive contact between surfaces. This unified method is based on contact constraints, bringing more realism and robustness to the interactions between objects (chapter 4). This technique is robust and it can be integrated in existing constrained contact handling libraries easily.

3. A set-up system that allows to configure each parameter of our interactive demo for shoulder palpation, applied to medical purposes (chapter 5). In this chapter a medical palpation

demo of the human shoulder is presented. This medical application has all the human shoulder organs (bones, ligaments and muscles) configured with diverse mechanical parameters. In order to define all the contact couplings present between organs, a complete set-up system is implemented. In this way we use a visual modeling tool to define all of these parameters and to deform the organs in an initial pose to start the simulation in a safe, collision-free configuration. In addition, a set of exportation scripts have been implemented to make our set-up inter-operable with the demo. Finally, our shoulder demo has support for haptic tools and it is designed to be fully interactive.

4. A framework for fast authoring of Virtual Reality (VR) applications for CAVEs and other multi-display devices (chapter 6). This chapter presents a new visual tool for quick development of applications. This suite combines a game engine and several integrated tools that ease the deployment of VR applications in distributed multi-screen environments.

1.4.2 List of publications

Fortunately, all the contributions presented in the next chapters have been successfully published in notorious conferences. Some of our contributions were published in international conferences like the ACM SIGGRAPH / Eurographics Symposium on Computer Animation, an others were published in national conferences, like the Congreso Español de Informática Gráfica (CEIG). We show details about all of these publications in the following paragraphs:

1. Our method to deform dense data volumes interactively, described extensively in chapter 3, was published in [GEP+13].

2. Our algorithm to simulate adhesive contact between surfaces, explained in chapter 4, was published in [GZO10].

3. The framework we have developed for the authoring of a human shoulder medical demo, shown in chapter 5, was published in [OGG+10].

4. BlenderCave, our tool for fast creation of distributed VR applications, explained in chapter 6, was published in [GBEO11].

1.5 Outline

In the next chapter we present previous work done in the fields of collision detection, contact solving and development of VR applications.

The rest of the thesis will focus on describing and analyzing our contributions.

Chapter 3 presents our novel method for deforming volumetric objects, using an accelerated rasterization algorithm.

Chapter 4 describes an algorithm to simulate adhesive contact between surfaces, using robust techniques based on constrained contact handling.

Chapter 5 presents a framework that we have developed for the authoring of a human shoulder demo designed for medical palpation.

Chapter 6 presents our distributed Rapid Application Development (RAD) tool that we have implemented for fast creation of VR applications for CAVEs and other multi-display devices.

Finally, Chapter 7 discusses the results and future lines of work.

Chapter 2

Related work

Virtual Reality (VR) applications that simulate Physics is a demanding domain that tries to solve many problems, three of them that are very important are: Deformation of virtual objects, simulation of interactions between objects with different properties (rigid, deformable) and integration of assets into VR applications using authoring tools. This chapter discusses the most relevant techniques developed in the past that deal with these problems.

2.1 Deformation of virtual objects

Every object in nature suffers deformations due to forces and interactions with other objects. The deformation degree depends on the material stiffness of the object, therefore a very stiff material could suffer not readily apparent deformations although the applied forces were high. Soft materials, on the other hand, can be deformed easily by the action of forces and other interactions. Among the types of deformable objects, we call *elastic objects* the bodies that recover their original shape after these deforming forces disappear, and the so called *plastic objects* are those that cannot recover totally their original shape after the effects of forces are gone. Most of the deformable bodies in nature are a mixture of these two types, depending on the magnitude of the forces applied over them, so a body could have an elastic behavior under small forces and suffer plastic deformations if the applied forces overpass a specific threshold.

Although we can find more details about general deformation modeling [NMK+05], or modeling of cloth [HB00, CK05], in this section we will focus exclusively on the deformation phenomenon itself, regardless of the kind of deformable object we are taking into consideration.

In Computer Graphics, there are several ways to represent a virtual object, two of the most popular techniques are polygonal meshes and dense volumes of data. In the next subsections we will explain their characteristics and how they can be deformed.

2.1.1 Deformation based on surface meshes

The surface is the part of the object the user sees and the part that interacts with other objects. Essentially, a polygonal mesh is a collection of vertices, edges and faces that represent the external shape of a body. The vertices are points with a position in 3D space, each edge relates two vertices and it is drawn using a line that joins them. Finally, each face is a plane that is limited by three edges (triangular) or four edges (quadrilateral), in figure 2.1 we can see an example of a triangular mesh.

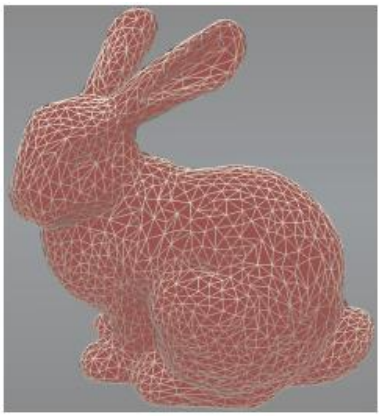

Figure 2.1: An example of a surface mesh: Stanford Bunny

Polygonal meshes have many advantages, since the shape of the mesh is defined by the positions of its vertices, only a change in the coordinates of some of them is needed to produce a deformation. Deformation models can perform this task computing new positions of vertices in a realistic manner.

The polygonal mesh is a definition quite popular in Computer Graphics and it is extensively used in VR applications and video games. Most of the efforts done in Computer Graphics are focused on increasing the rendering speed, realism, richness and interactivity of this kind of representation. In addition, dedicated acceleration hardware is available to speed-up the rendering of bodies defined by polygonal meshes. Nevertheless, the representation of highly detailed objects involves the definition of highly tessellated meshes, and that demands high

computation power for rendering all of these triangles at interactive rates. Fortunately, modern Graphics Processing Units (GPUs) allow the parallelized processing and display of millions of triangles per second. In addition modern models of GPUs are programmable, allowing the execution of specific programs (shaders) that manipulate every single property of the geometry to be displayed.

2.1.2 Deformation based on dense volumetric data

Although polygonal meshes are a popular technique for rendering virtual objects, some entities have internal complex structures that would be difficult or very expensive to represent by this manner. Regular 3D grids are a popular way to store dense volumetric data, like biological forms, notably in medical imaging. These types of data structures have many advantages, like allowing to illustrate the internal structures of a volumetric object in a single view (Figure: 2.1.2).

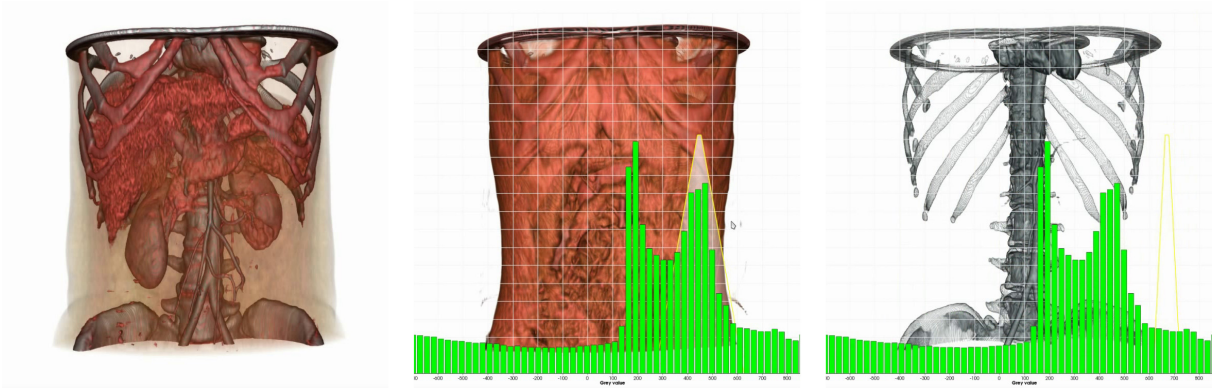

Figure 2.2: A dense volume of data representing a human torso. This data was captured by tomography from a real patient (left). 3D grids allow to store all the internal structures and organs. Thanks to visualization algorithms based on raycasting it is possible to select the information to be illustrated (center and right).

Although many forms are inherently deformable, current volume visualization algorithms are not optimized for deformable data grids. On the other hand, there are many application fields where it might be convenient to deform volumetric models interactively, like surgical planning and animation of volumetric characters (figure: 2.3).

Figure 2.3: Applications of interactive volumetric deformation: Surgical planning (left) and animation of volumetric characters (right).

There have been several approaches to obtain deformable volumetric data. One of the most popular techniques consists in the segmentation and creation of surface meshes using the volumetric materials as input, since polygonal meshes are easier to manipulate. Unfortunately, current segmentation algorithms are not accurate enough and human manipulation is still required. Automatic segmentation is, in general, a hard and unsolved problem. Another drawback of the segmentation is the noticeable loss of detail in the resultant polygonal mesh in relation to the original data grid.

Moreover, there exist other approaches to deform dense volumes, like 3D tetrahedra rasterization. Essentially, 3D tetrahedra rasterization is an algorithm that takes as inputs an undeformed data grid, an embedding tetrahedral mesh and a deformation field discretized on the mesh; as output it generates a new deformed data volume adapted to the deformed embedding mesh.

Our approach for 3D rasterization is based on previous rasterization techniques. Rasterization of geometric primitives to a grid data structure is a largely studied problem, as it constitutes a key element of current GPU rendering algorithms [FLB$^+$09, LK11]. They rasterize triangles into a 2D grid, and there are mainly two approaches to parallelize the process.

One approach is to parallelize on a triangle basis. Each processor handles one triangle, computes an axis-aligned bounding box (AABB) around the triangle, and then processes internal pixels testing for inclusion in the triangle [LHLW10, FLB$^+$09]. The second approach is to parallelize on a tile basis. A first step assigns triangles to a tile of pixels, and then pixels are processed in parallel testing the list of triangles [SCS$^+$08, EL10].

Our approach for 3D rasterization of tetrahedra combines ideas from these two approaches. We parallelize at the voxel level, but we produce candidate voxels based on the AABB of the rendered tetrahedron. In addition, we face different problems than traditional triangle rasterization algorithms. As opposed to triangle rasterization, in our setting the tetrahedral mesh constitutes a partition of space, hence only one tetrahedron covers each grid point. At the same time, the 3D AABB of a tetrahedron produces many more false candidate voxels than the false candidate pixels produced by the 2D AABB of a triangle.

3D rasterization of tetrahedra has also been studied, although the currently published algorithms work by traversing scan planes and scan lines [RSF+04]. This approach is difficult to parallelize with effective load balancing, whereas our proposed method produces extremely uniform workload across processors.

Yet another related problem is the voxelization of triangle meshes. Current parallel approaches parallelize the voxelization on tiles, and construct an A-buffer per tile as a first culling approach [SS10, Pan11]. Surface voxelization, although connected to volume voxelization, also suffers different difficulties. Primitives occupy fewer voxels, but their AABBs produce many more false candidate voxels.

One of the problems that needs to be solved as part of our algorithm is a tetrahedron-cube intersection test. One possibility is to extend existing methods for triangle-square intersection [AMA05]. Another possibility is to build on the general separating axis test for simple convex primitives [GLM96]. However, we exploit the fact that, in our problem, cubes are actually cells of a grid, and we design a faster algorithm that works in two steps: grid point classification followed by conservative tetrahedron-cube intersection test.

Our tetrahedral rasterization algorithm is intended as a method for volume data deformation. Other techniques have also been used for this purpose, such as deforming planes with semi-transparent textures that are rendered front to back [NFP10]. Instead, we propose a method that deforms the full volume data and allows the application of volume raycasting. More similar to our approach is the deformation method of Goksel and Salcudean [GS09], who also map the deformation of a tetrahedral mesh using a texture mapping approach. However, their method to map deformed tetrahedra to voxels follows a scanline approach, and their interactive visualizations are limited to 2D images. Yet another possibility would be to apply volume raycasting on the deformed tetrahedral mesh [KWW01, GW06], but the resolution of the tetrahedral mesh is

too low in our case, and rendering a high-resolution tetrahedral mesh would be very inefficient.

2.1.3 Coupling between rigid-deformable bodies

One of the major issues that our work addresses is the efficient coupling of objects with various mechanical properties, as seen in figure 2.4.

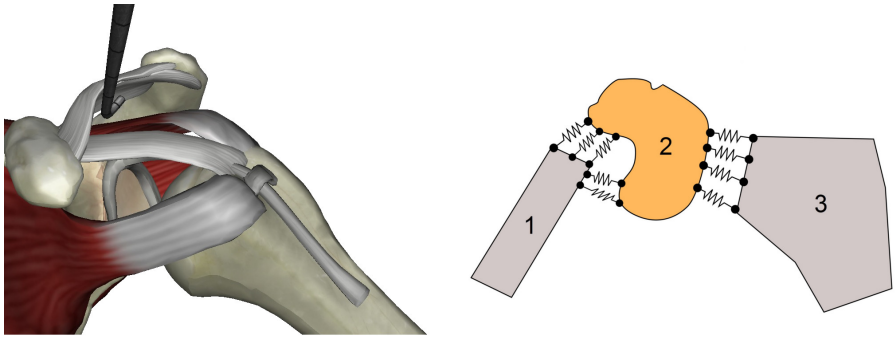

Figure 2.4: VR palpation demo of a human shoulder (left) and a conceptual representation of a contact island, that is a set of couplings between three bodies with assorted mechanical properties (right). Couplings were modeled to connect objects together, like virtual bones with ligaments and muscles. In this way, each coupling is composed of binding springs.

This has been addressed for coupling constraints by [SSIF07] using binding springs. We follow a similar approach and solve binding springs efficiently in a global conjugate gradient solver. Coupling through contact constraints is often addressed in the game engine field, with the runtime creation of contact islands [PO09].

Couplings are necessary for keeping virtual bodies and other structures joined together, specially at interactive medical demos. Biomechanical modeling has seen a lot of success recently in computer graphics for the simulation of body parts such as the hand [SKP08], the neck [LT06], the face [SNF05], the torso [TSB+05,DZS08], or the complete upper body [LST09]. These models rely on highly detailed discretizations and geometrically accurate modeling of the anatomy, which imposes severe restrictions on their applicability to interactive simulation.

For interactive simulation of deformations, methods based on the linear co-rotational finite element formulation [MKN+04] are perhaps the ones that give a best balance between performance, robustness, and measurement-based parameterization. This last aspect is important when the behavior of a model should approximate the behavior of a real structure. Linear

co-rotational FEM models can be accelerated by decoupling the simulation mesh from the visualization or collision detection mesh, using embedded meshes [MBTF03, SBT06, NKJF09].

Another important aspect for a biomechanical simulation is contact handling. As shown in subsection 2.2.2, two main approaches exist. Even though they may have a higher computational cost, constraint-based methods provide higher robustness under stability issues. For the case of rigid bodies, they are used on several open-source libraries [BUL, ODE]. It is also worth pointing out the SOFA open-source library [SOF], which, similar to our work, supports soft-tissue contact. In chapter 5 we explain extensively the features of our interactive human shoulder haptic demo, applied to training and medical palpation.

2.2 Simulation of contact

Due to the last advances in computer graphics and hardware processing power of modern CPUs and GPUs, several interactive applications, like virtual scenarios and video games, are beginning to rely on physics simulators. These simulators allow to enhance the realism and behavior of the interactive animations of the virtual characters and objects being simulated.

One of the most important parts in the simulation of physics is the proper handling of interactions between virtual bodies. Handling contacts correctly prevents interpenetrations between objects and allows the simulation of additional realistic effects like repulsion, friction, coupling, adhesion and many others. The correct simulation of contact is a crucial problem in computer graphics, haptics and robotics. Essentially, contact simulation is split in two components: Collision detection, that detects intersections between the surfaces of the virtual objects, and Contact Handling, that computes repulsion forces in order to make all of these interpenetrations disappear. The goal of these two components is essentially to keep bodies free of interpenetrations, that is, well-separated from each other.

2.2.1 Collision detection

Regarding collision detection (CD), there is an extensive work on CD and contact computation [TKH+05, Sch13]. The main goal of a CD algorithm is to detect intersection of bodies as fast as possible. A collision detection algorithm uses as input a set of objects, its mission is to test each primitive of each object versus each other primitive. As said in chapter 1 the maximum number

of geometric tests could be up to N^2, being N the number of primitives of all the objects in the scene. In order to minimize this huge number of tests, a collision detection algorithm is divided into two phases:

1. **Broad-Phase**: This phase determines which pairs of bodies are potentially colliding. Many culling algorithms and data structures are defined to discard quickly false positives. A very popular approach that resolves this phase is the Sweep-and-prune method [CLMP95].

2. **Narrow-Phase**: Given a pair of colliding objects, this phase generates a set of possibly colliding primitives. Then, computes the geometric primitive tests between each pair of primitives to obtain the set of contacts. A primitive test must perform intersection tests between the face of a primitive A and the edges of primitive B, and vice-versa.

There is also an additional classification for detecting collisions, not only at broad phase but also at narrow phase. This classification is based on when the collisions are detected. Discrete methods check for collisions or penetration at a particular instant of time. Nevertheless, the virtual objects in the simulation are usually in motion, so a discrete method may miss collisions if the objects are moving quickly or there exist thin shells (e.g. a thin wall, a piece of cloth) in the scene. In this case, the algorithm may not notice when two bodies go through one another, because the query is called before and after the intersection.

Continuous methods are regarded as more robust as they test for collisions between two discrete time instances and compute the first time of contact [BFA02,RKLM04,ORC07,TCYM09]. Although they require more complex tests, these methods linearly approximate the trajectory of the vertices in between frames, transforming the primitive test into an intersection test between two prisms. Moreover, the acceleration structures used for detecting collisions also need to acknowledge this trajectory. Regarding primitive continuous tests, they would have to check for intersections between edges and planes of these both prisms.

Due to the characteristics of our demos, in all the experiments of this dissertation we have found that discrete collision detection algorithms are valid enough for our purposes, nevertheless, all of our techniques can be adapted to continuous collision detection if it is required.

Collision detection algorithms return a set of contacts. Each contact is a data structure that can be defined in several ways, depending on the information required by the contact handling

algorithm in use. In our case each contact between two bodies A and B is defined by the next properties (more details in figure 2.5):

- S_a and S_b: The two intersecting surface primitives of objects A and B, respectively.
- $\mathbf{N}_a$ and $\mathbf{N}_b$: The two normal vectors of primitives S_a and S_b, in same order.
- δ: Penetration depth, it is the minimum distance required to separate S_a and S_b in order to eliminate their intersection.
- $\mathbf{P}$: Point of one of the surfaces where the penetration depth was computed.
- $\mathbf{P}'$: Closest point in the surface of the penetrated object, usually it is computed projecting $\mathbf{P}$ on the other surface.

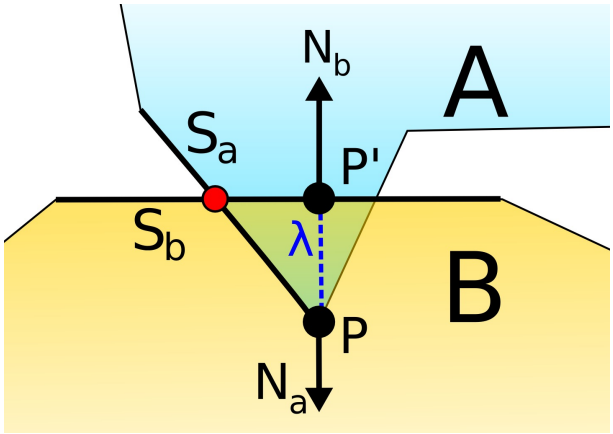

Figure 2.5: In this example, the CD algorithm in use has detected an intersection between surface primitives S_a and S_b (red dot). Performing a geometric intersection test between these primitives reveals the penetration depth λ and the points $\mathbf{P}$ and $\mathbf{P}'$, that are essential for the contact handling algorithm to solve the interpenetration.

In the next section, the most popular contact handling approaches will be discussed.

2.2.2 Contact handling

The Contact Handling methods, also called collision response methods, use as input the detected contacts in the Collision Detection phase and try to restore a non-penetrating state in all the colliding bodies. Otherwise, the intersecting bodies may have unrealistic behaviors, like

going through one another. Although there exist several approaches, the most physically accurate methods are those that compute repulsion forces and apply them to intersecting surfaces, aiming them to separate each other in a more realistic manner. Essentially, collision response algorithms can be classified in many kinds, based on the nature of the response there are two categories [WB97]:

- Penalty-based methods [TPBF87, MW88, WVVS90, TMOT12].
- Constraint-based formulations [BFA02, DAK04, PPG04, DDKA06, OTSG09].

Although in literature we can also find Impulse-based methods [Mir96, BFA02, GBF03b], they are just a subgroup of constraint-based methods.

In addition, collision response can be classified in the local or global effect of the forces, or in the manner the contacts are processed (sequential or simultaneous).

In general, penalty-based methods are fast and suitable for real time interactive applications, where processing speed is more important than accuracy. Constraint-based methods, on the other hand, result in a more robust and plausible simulation at the cost of extra computation.

Apart from mere collision between objects, there exist in nature several other contact interactions that might bring more richness and realism to physics simulations. For instance, most of the actual surfaces in real world have friction. There is also coupling between objects with diverse mechanical properties. In addition, some types of surfaces have an adhesive behavior, specially some biological structures.

Although there exist several other phenomena, we have researched these three effects that surely enhance the quality of the contact interactions. In the rest of this section we will enumerate all the related work that, as far as we know, we have found in literature.

Penalty-Based Methods:

Penalty-based methods [MW88, BJ07a, HVS+09a] are widely used on interactive simulations, specially when very high performance is a major goal [BJ07b]. These kind of methods can be useful also on complex applications such as cloth simulation [BW98b, CK02]. Essentially, a penalty-based method tries to restore a non-penetration state when two or more virtual objects intersect. Its manner to perform consists in defining a repulsive force field which is applied to the penetrating surface areas, based on their penetration depth.

Traditionally, penalty-based approaches suffer from multiple problems and many of them have been addressed in several publications. One of them consists in how to define the penetration depth, another is how to establish the direction and magnitude of each repulsion force, the reason is because repulsive forces and penetrations are not continuous between time steps, that fact can generate inconsistencies, as we will see below.

Regarding the definition of penetration depth, the simplest one is the one that defines a point $\mathbf{x}$ that belongs to object A and it penetrates object B. The penetration depth $\delta(\mathbf{x})$ is the distance to the closest point to $\mathbf{x}$ on the surface of B. In this way, the basic penalty energy is typically defined as:

$$E(\mathbf{x}) = \frac{1}{2} k \, \delta(\mathbf{x})^2 \tag{2.1}$$

Where k is the stiffness constant. The penalty force can be computed as the gradient of the penalty energy [TMOT12], i.e.,

$$\mathbf{F}(\mathbf{x}) = -\nabla E(\mathbf{x}) = -k \, \delta(\mathbf{x}) \nabla \delta(\mathbf{x}) \tag{2.2}$$

Given the above definition of penetration depth, the (negative) gradient of penetration depth equals the unit surface normal $\mathbf{n}$ at the closest point on the surface of the penetrated object. The penalty force can be computed simply as:

$$\mathbf{F}(\mathbf{x}) = k \, \delta(\mathbf{x}) \, \mathbf{n} \tag{2.3}$$

This definition is very similar to the Hooke's law, as shown in figure 2.6.

As explained before, computing the penetration depth as the distance from the penetrating point to the closest point to the surface of the penetrated object may bring several problems to the simulation, like jittering (that is high-frequency deviation from the expected path), oscillatory behavior [Dru08], instability (i.e., unbounded growth of physical quantities such as momentum), inconsistency (i.e. penetrating points with opposing penalty forces) and other issues. Some approaches [HTK+04, MH05, ZKVM07, BJ08] try to mitigate these drawbacks with different techniques.

Regarding the magnitude of each repulsion force to be applied, penalty-based methods cannot guarantee to avoid all the interpenetrations, that is because repulsive forces need to be very stiff or non-linear [TPBF87]. The problem is that higher contact stiffness increases the risk of instability, in particular because the forces are discontinuous.

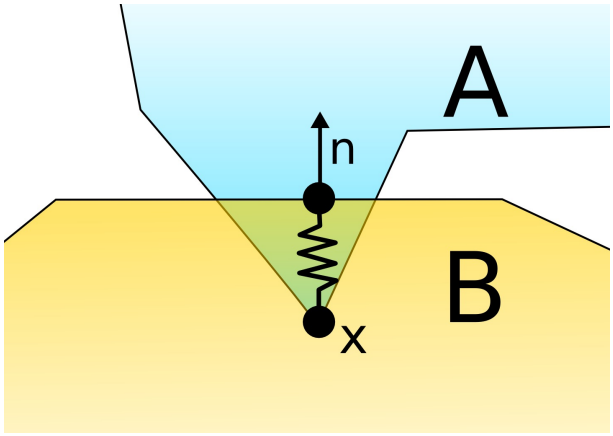

Figure 2.6: Illustrative example of equation 2.3. In penalty method basic formulation, also called *discrete penalty force*, interpenetrations are resolved using forces simulated by virtual bilateral springs of rest length of 0. When the spring tries to shrink to its rest position, it pulls both surfaces in order to reduce/eliminate interpenetration.

Computing appropriate repulsive forces is a challenging issue [BFA02, HTK+04, Dru08]. Fortunately there exist promising approaches like distance fields [FL01, HTK+04, TKH+05], approaches like the barrier method [HVS+09b], filtering methods for haptic devices [ESJ97] or volume-based penalty methods [HS04, TSIF05, FBAF08, AFC+10].

Contact handling constraints

Constrained contact handling is a kind of method that is robust, accurate and guarantees non-penetration at the end of each timestep. Its formulation models collision response as a constrained optimization, based on Lagrange multipliers.

Constrained contact handling approach has gained popularity for simulation of rigid bodies [Bar94, RKC02, PPG04, KEP05, DDKA06, Erl07, KSJP08, OTSG09, CAR+09], and since the work of Baraff and Witkin [BW92] it also is applicable for simulation of deformable bodies. One of the advantages of this method, as well as the non-penetration guarantee, is that also allows accurate handling of friction and stacking. For more details applied to the field of computational mechanics, refer to the book of Wriggers [Wri02] for a comprehensive treatment.

A contact constraint is a condition that always has to be fulfilled. Typically, a contact handling algorithm defines an optimization problem that not only solves the dynamics of each virtual body, but also every constraint plugged into the problem is taken in account.

The technique that we employ in our experiments uses a linear system of equations, based on Lagrange multipliers, where every contact detected has been defined as a constraint. In this way, when this system is solved, the resultant forces will obey every defined constraint of contact. In figure 2.7 we can see an example of two bodies colliding, where the equation 2.4 formulates their constraint of contact.

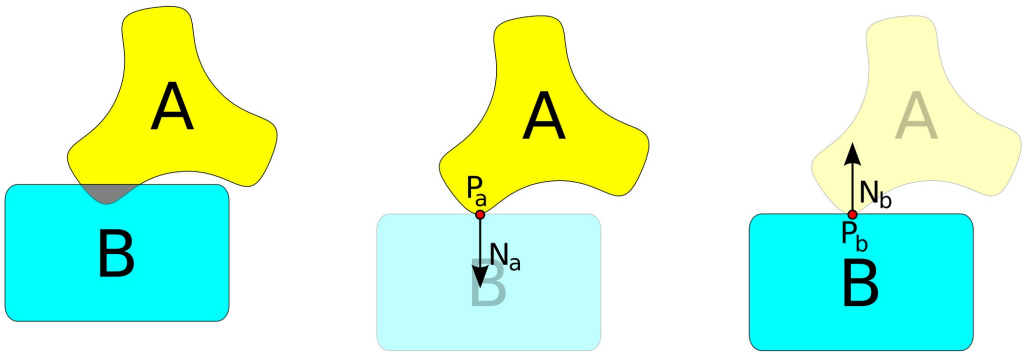

Figure 2.7: This figure shows an example of two virtual objects colliding (left). In order to define a contact constraint, one point and its normal is defined per colliding surface (center and right). The contact constraint is formulated in equation 2.4.

$$\mathbf{N}_a(\mathbf{P}_b - \mathbf{P}_a) \geq 0 \tag{2.4}$$

Typically, constrained contact handling response algorithms can be classified in the manner the contacts are processed: Iterative handling, which solves constraints one at a time, and Linear complementarity problems (LCP), that solves all the contacts simultaneously.

Iterative constraint handling methods [BFA02, GBF03a, MHHR06], formulate and solve each constraint sequentially and locally on the colliding areas [BFA02, MHHR06, SBT07]. Since the work of Shinar et al. [SSF08], these methods are suitable for handling contact for deformable and rigid bodies. This kind of handling works well with small time steps and with a low number of constraints per time step. Nevertheless, it may produce spurious energy growth due to excessive local deformation at colliding zones. Bridson et al. [BFA02] alleviated this effect by adding cloth relaxation steps to collision response.

On the other hand, linear complementarity problems (LCP) [CPS92] use an implicit time-stepping method that robustly guarantees the enforcement of the constraints at the end of the time step [ST96]. That works because all the existing constraints at current time step are globally formulated and solved simultaneously. A simultaneous formulation guarantees that the solution to one constraint will not violate others.

The formulation of an implicit LCP defines a linear system that includes all the constraints to be held. Solving this system gives us the magnitudes of the repulsive forces to be applied. These forces actually eliminate all the found intersections at the current time step.

The formulation of a contact LCP requires the computation of all pair-wise contact force effects, a process referred to as constraint anticipation [Bar96] or computation of the Delassus operator [DDKA06].

In our examples, we have used iterative constraint anticipation [OTSG09]. The basic dynamics formulation that we employ in our experiments is defined as follows, beginning by the 2^{nd} Newton's law:

$$\mathbf{M}\dot{\mathbf{v}} = \mathbf{F} \tag{2.5}$$

Adding numerical integration, the previous equation is re-defined depending on the new velocities $\mathbf{v}$:

$$\mathbf{A}\mathbf{v} = \mathbf{b} \tag{2.6}$$

In this formulation, all the constraints are solved simultaneously, so that every constraint like equation 2.4 is linearized, written in terms of velocity and grouped in the matrix $\mathbf{J}$, all in a single equation system like:

$$\mathbf{J}\mathbf{v} - \mathbf{c} \geq 0 \tag{2.7}$$

Defining a system similar to Lagrange multipliers, constraints forces are added to the basic dynamics formulation:

$$\mathbf{A}\mathbf{v} = \mathbf{b} + \mathbf{J}^T\lambda \tag{2.8}$$

$$\mathbf{J}\mathbf{v} - \mathbf{c} \geq 0 \tag{2.9}$$

Where $\mathbf{J}^T$ is a matrix that stores every unit vector of every contact repulsive force, and λ are their unknown magnitudes.

As result, this is the classic contact complementarity problem:

$$\mathbf{Av} = \mathbf{b} + \mathbf{J}^T \lambda \tag{2.10}$$

$$\lambda \geq 0 \perp \mathbf{Jv} - \mathbf{c} \geq 0 \tag{2.11}$$

The $\perp$ symbol means *complementarity*, and it requires at least the fulfillment of one of the two conditions it joins (left or right).

Solving the Constrained Problem implies to single-out the velocities:

$$\mathbf{v} = \mathbf{A}^{-1}(\mathbf{b} + \mathbf{J}^T \lambda) \tag{2.12}$$

And plug them into the constraints:

$$\mathbf{Jv} - \mathbf{c} \geq 0 \tag{2.13}$$

We obtain an LCP:

$$\mathbf{B}\lambda \geq \mathbf{d} \tag{2.14}$$

Where:

$$\mathbf{B} = \mathbf{JA}^{-1}\mathbf{J}^T \tag{2.15}$$

$$\mathbf{d} = \mathbf{c} - \mathbf{JA}^{-1}\mathbf{b} \tag{2.16}$$

Our preferred linear solver is Projected-Gauss-Seidel (PGS), this solver allows to project λ_i as it visits each single equation in the system. The applied non-penetration projection described as follows:

$$\text{if } \lambda_i < 0 \text{ then } \lambda_i \leftarrow 0$$

PGS algorithm solves each row i^{th} of $\mathbf{B}\lambda \geq \mathbf{d}$ as shown:

For the i^{th} row:

$$B_{ii}\lambda_i + \overline{\mathbf{B}}_i \overline{\lambda} \geq d_i \tag{2.17}$$

Where:

$\overline{\mathbf{B}}_i$: is $\mathbf{B}_i - \{\mathbf{B}_{ii}\}$, that is the row $\mathbf{B}_i$ without the i^{th} column.

$\overline{\lambda}$: is $\lambda - \{\lambda_i\}$, that is the vector λ without the i^{th} row.

Finally, solving each constraint force λ_i (and projecting it to 0 if needed):

$$\lambda_i = max(0, \frac{d_i - \overline{\mathbf{B}}_i \overline{\lambda}}{B_{ii}}) \tag{2.18}$$

Although LCP systems have higher computation cost, they solve robustly all the dynamics of the virtual bodies and all the contacts happened in the simulation. In simulations with rigid bodies only, these systems are very fast because each rigid body has few degrees of freedom (dofs) and the resultant linear system to be solved is small. The problem arises when deformable bodies are present in the simulation. Deformable objects usually have many dofs and implicit constraints. As result, that implies the solution of larger linear systems, which become excessively expensive.

In order to speed-up computation, several approaches have been developed, many of them trying to rely in precomputations, for instance using linear [PPG04] or global co-rotational deformation models [DDKA06]. Other techniques try to exploit linear system solvers maintaining an active constraint set, like Raghupathi and Faure [RF06]. Other approaches try to make approximations, like the use of compliance warping (Saupin et al. [SDCG08]), designed to accelerate the formulation of the Delassus operator. Finally, there is a recent approach which uses reduced deformation models, like the one used in Kaufman et al. [KSJP08], developed to accelerate non-penetration and frictional LCPs in a staggered manner for rigid bodies.

2.2.3 Friction

Friction is a force that opposes to tangential movement between two surfaces in contact. If friction force did not exist, surfaces might slide without resistance, like over ice.

As far as we know, the most popular mathematical model for friction computation is Coulomb's cone, shown in equation 2.19.

$$\mathbf{F}_r <= \mu \mathbf{F}_n \tag{2.19}$$

Where $\mathbf{F}_n$ is the normal force applied from one surface over the other, and μ is the Coulomb's friction coefficient, which depends on the properties of the surfaces in contact. Remember that the maximum magnitude of $\mathbf{F}_r$ cannot be greater than $\mu\mathbf{F}_n$.

There are works in computer graphics that implement friction in rigid body simulations [Lot84, Bar91, ST96, ST00, KEP05], multi-body and articulated rigid bodies [AP97, KSJP08]. There are also friction algorithms for deformable bodies, like cloth [BFA02] or finite element methods (FEM) [WVVS90]. Finally, friction is not modeled only for constrained contact handling, but also there are penalty-based approaches [YN06] and impulse-based algorithms [KSK97] that implement it.

2.2.4 Adhesive contact

Adhesion (also referred to as *stiction*) is the phenomenon which keeps two contacting surfaces glued together. Hence, this is a force that opposes to forces (traction or tangential) that might try to separate those surfaces. In figure 1.5 several real-world examples of sticky objects are shown. Adhesive contact gives more realism and richness to the simulations, specially between wet surfaces, biological structures, mucous membranes and others.

As adhesion is closely related to the contact phenomenon, that we have introduced in subsection 2.2.2, we have modeled this phenomenon as a constraint-based formulation, that is plugged to a LCP algorithm of contact handling. The features of our technique are exhaustively explained in chapter 4.

In computer graphics, adhesion has been modeled before in a way similar to penalty forces [JL93, CJY02, BMF03, WGL04, SLF08]. When adhesion takes place, a bilateral spring is set between contact points. As we will discuss later, under traction our adhesion constraints can also be regarded as springs, but under compression they are not active, and we completely enforce non-penetration instead. Another important difference between our constraint-based adhesion and typical adhesive springs is our physically-based model for decohesion.

On the other hand, the formulation of adhesion using constraints was largely developed in the field of contact mechanics by Fremond [Fre87], while Raous et al. [RCC99] developed a thermodynamics background and the connection to friction. A summary can be found in the book of Wriggers [Wri02]. Similar to adhesion, other phenomena, such as puncture [CAR+09], can be modeled using constraints in conjunction with contact.

Our approach takes many of these ideas and adapts them to computer graphics, specifically to constraint-based contact simulations. Our implementation is suitable for rigid and deformable bodies and it is easy to integrate it into existent contact handling frameworks, like [BUL, ODE, SOF] and others. As said before, more details about our method are shown in chapter 4.

2.3 Development of VR applications

The development of VR applications is a process well-known to be demanding and full of complexities. This multidisciplinary task not only requires an advanced device management and the most efficient state-of-the-art algorithms, but also it demands the use of rapid authoring tools. These tools are useful for the creation of assets (virtual objects) and for the composition of the entire simulation scene. In addition, the employment of exportation scripts enables developers to migrate each asset to the simulator and to check each object's mechanical properties in the actual simulation environment. To summarize, the development of VR applications requires the definition of a pipeline, as shown in figure 2.8.

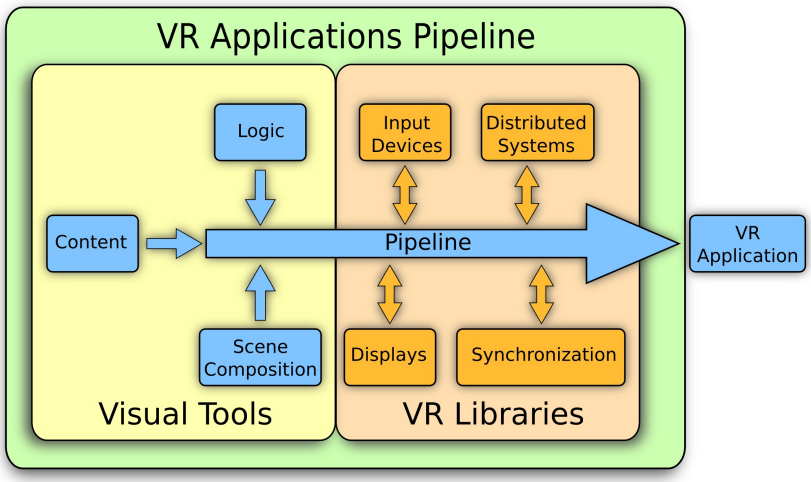

Figure 2.8: A pipeline applied for the development of VR multi-screen applications.

Several physics-based applications, like our shoulder palpation demo mentioned in chapter 5, rely heavily on robust collision handling algorithms, that is because most of the time several virtual bodies are colliding each other. These algorithms require the bodies to start the simulation well-separated from each other, in an inherently stable configuration. In order to

accomplish these requirements, adapted authoring tools and custom file formats and scripts are required for a proper mechanical configuration of the affected assets.

Regarding VR applications that are executed in large-area display systems, they allow more immersion for the user, but, on the other hand, they also present additional challenges in their development. Large-area displays provide a high degree of immersion in VR applications, often making the VR experience more compelling to the user. This kind of displays can typically be achieved in two ways. One way is to project the image onto a large passive screen using a powerful projector, such as in a PowerWall, a CurvedScreen, a Workbench [KBF+95] or subsequent similar designs. Another way is to tile multiple smaller displays.

In the first way, the difficulty is to achieve high resolution images, while in the second way the difficulties are due to color and brightness consistency [Sto01]. Tiled displays are also more expensive and require more computational resources. A CAVE system [CNSD93] shares some similarities with both approaches, as it uses multiple projectors to project images on several passive screens and give the user a sense of full immersion in a virtual environment.

The output display is just one of the hardware components involved in the design of a VR application. Typically, VR applications also involve various input devices, and possibly non-visual output, for example haptic response. Multiple researchers have developed solutions based on abstraction layers that free the developer from dealing with implementation details about I/O devices.

An early example is CAVELib [CN95], created by the inventors of the CAVE and later turned into a popular commercial tool. CAVELib's major limitation is its strong connection to specific hardware configurations. A more recent and active VR design framework is VR-Juggler [BJH+01]. It provides excellent support for I/O and it is managed at a programming level. There are many other VR development frameworks that provide abstraction layers, such as DIVERSE [KSA+03], Studierstube [SRH03], RB2 [VPL90], DIVE [CH93], dVS [Gri91], Avocado [Tra99], VRPN [THS+01], Equalizer [Equ], MRToolkit [SLGS92], or dVise [Ghe97]. A more recent framework, INVRS [Ant09], extends traditional abstraction capabilities to networked virtual environments.

Some VR authoring frameworks also provide easy-to-use graphical interfaces for application development. AMIRE [GHR+02] is an authoring tool for mixed reality, whose interesting aspect is that the design of the virtual environment may be performed in an intuitive manner

in a mixed reality setting. EAI' Wordltoolkit [Ini97] enables novice developers to quickly prototype an application, however, it expects the application data to come from CAD packages. Finally, the commercial framework EON Studio™ [Eon], from EON Reality Inc, allows authoring through a graphical user interface. It is a complete package for a CAVE environment, however it lacks flexibility for developers.

In the creation of a VR application, modeling and rendering the virtual environment plays an important role as well. There are many high-quality options for content creation and rendering. Multiple rendering engines [OGR, OSG, Epi, Cryb] allow the creation of highly realistic yet real-time visualization applications without low-level knowledge of the most advanced CG algorithms. Some months after finishing our study of alternatives and developing BlenderCave, the developers of a render engine called Unity [Unib] also coded additional functionality in order to allow VR applications to run in multi-display systems. VR device abstraction frameworks [BJH+01, THS+01, Ant09] ease I/O services, communications and process management aspects. Finally, virtual sandbox tools allow content creation and the definition of the application's logic in easy-to-use visual editors.

In chapter 6 we have a detailed analysis of several state-of-the-art rendering engines. In section 6.1 we discuss some of the most relevant ones in detail. In addition, a comparative of their capabilities is shown in table 6.1.

Chapter 3

Fast Deformation of Volume Data Using Tetrahedral Mesh Rasterization

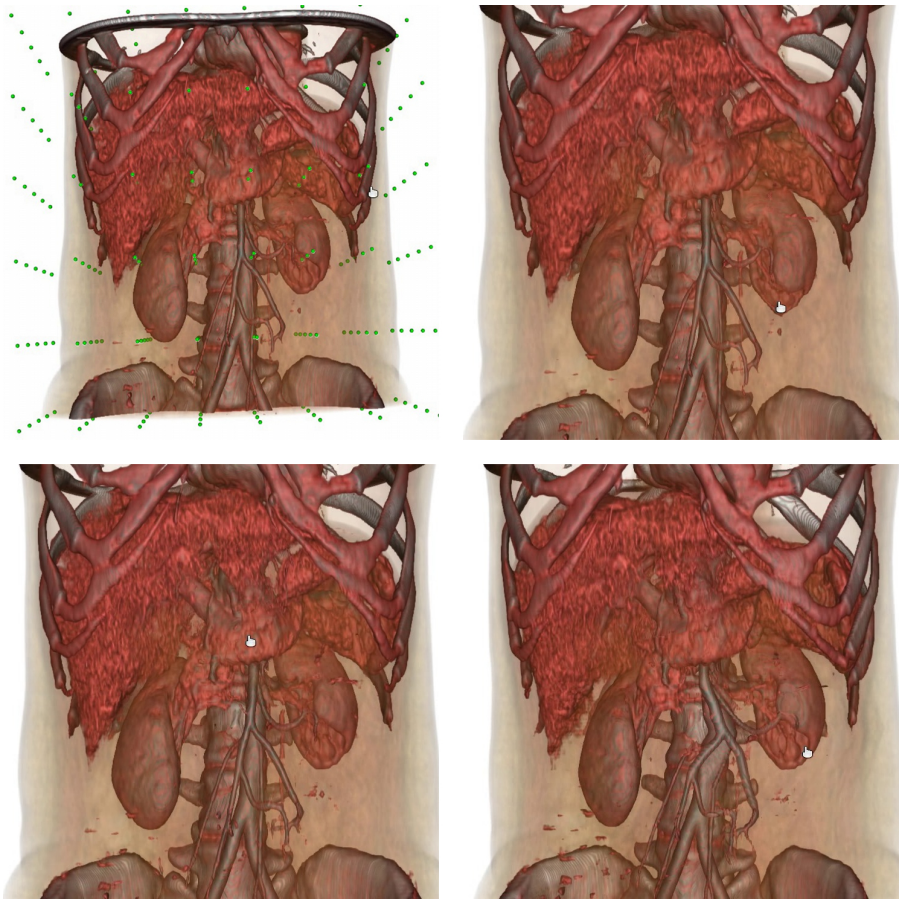

Figure 3.1: 3D medical image with the nodes of a tetrahedral mesh overlaid (top left). The next three snapshots show interactive deformations of a kidney, the heart, and abdominal vessels. The $256 \times 160 \times 122$ volume is deformed at 67fps.

As introduced in section 1.1, and explained in more detail in the previous chapter (section 2.1), regular 3D grids are a popular way to store dense volumetric data. They are often used to capture and represent volumetric information of anatomy or other biological forms, as those shown in figure 1.4, notably in medical imaging [Son00]. Volume rendering offers a convenient way to illustrate in a single view the internal structures of solid volumetric objects stored in regular 3D grids [RSHRL08].

Elastic deformations, on the other hand, are typically solved by discretizing the deformation field on a Lagrangian mesh, i.e., a mesh that moves and deforms along with the material, as this enables trivial mass conservation and, as seen in section 2.1, deforming a mesh is a trivial task. To deform and manipulate volume data, e.g., for medical planning applications, the data is typically segmented and meshed, and the deformed data is visualized using surface meshes [SM00]. As a result, the visualization of the deformed data misses the full-resolution volumetric detail present in the original 3D grid.

Instead, we propose a method that takes as input a deformation field discretized on a tetrahedral mesh, and uses it to warp the original volume data into a new 3D grid. Our method is independent of the technique used to compute the deformation (See [NMK+06] for a survey of deformation techniques). The data in the resulting grid can be visualized with standard volume rendering algorithms to reveal its full volumetric detail.We pose the problem of warping the volume data as tetrahedral mesh rasterization with 3D texture mapping, and the central contribution of our work is an extremely fast method for tetrahedral rasterization. As an example, the human torso data in the first image of this chapter, with 4.99M voxels ($256 \times 160 \times 122$), is deformed at 67fps. In other examples, our method allows the deformation of volume data with over 20 million voxels at interactive rates.

In chapter 2 (subsection 2.1.2) we have discussed the previous work related to our deformation approach. And in Section 3.1 we describe a massively parallel 3D image warping approach based on barycentric mappings. To further accelerate rasterization, we introduce a parallel culling algorithm described in Section 3.2. We conclude the chapter with a discussion of performance and results.

All the contributions shown in this chapter were published in [GEP+13].

3.1 3D Grid Warping

The input data to our method is a regular 3D voxel grid G_0, where the voxels store a scalar field c_0 (which could be extended to vector or tensor fields). In addition, the method takes as input a tetrahedral mesh M_0, which may partially or completely embed the grid. Given a deformed tetrahedral mesh M_1, we wish to compute a deformed scalar field c_1 on an output 3D voxel grid G_1. We assume that the deformation field is linearly interpolated inside each tetrahedron.

We propose a massively parallel method to compute the deformed scalar field c_1, by rasterizing the deformed tetrahedra onto the output grid G_1, and assigning values of the scalar field as a texture mapping process. We trivially define the assignment of deformed scalar values through barycentric mappings inside each tetrahedron. Formally, given a point with barycentric coordinates $\mathbf{b}$ inside a tetrahedron T, and with undeformed (resp. deformed) position $\mathbf{x}_0$ (resp. $\mathbf{x}_1$), we define two barycentric mappings: $\beta_0 : \mathbf{x}_0 \rightarrow \mathbf{b}$ and $\beta_1 : \mathbf{x}_1 \rightarrow \mathbf{b}$. Given matrices $\mathbf{X}_0$ and $\mathbf{X}_1$ whose columns are formed respectively by the undeformed and deformed positions of the nodes of T, and a vector of ones $\mathbf{1}$, the mappings β_0 and β_1 are defined as the linear transformations

$$\mathbf{b} = \beta_0(\mathbf{x}_0) = \begin{pmatrix} \mathbf{X}_0 \\ \mathbf{1}^T \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{x}_0 \\ 1 \end{pmatrix} = \mathbf{B}_0 \bar{\mathbf{x}}_0, \tag{3.1}$$

$$\mathbf{b} = \beta_1(\mathbf{x}_1) = \begin{pmatrix} \mathbf{X}_1 \\ \mathbf{1}^T \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{x}_1 \\ 1 \end{pmatrix} = \mathbf{B}_1 \bar{\mathbf{x}}_1. \tag{3.2}$$

Here, $\bar{\mathbf{x}}$ represents a point $\mathbf{x}$ in homogeneous coordinates.

In practice, to assign the deformed scalar value of a voxel with position $\mathbf{x}_1$, we simply fetch the scalar value from the input point $\mathbf{x}_0$ with the same barycentric coordinates. This operation is formally defined as $c_1(\mathbf{x}_1) = c_0(\beta_0^{-1}(\beta_1(\mathbf{x}_1)))$. In our results, we have implemented the texture read operation $c_0(\mathbf{x}_0)$ as a trilinear interpolation of voxel values. Note also that $\mathbf{x}_0$ and $\mathbf{x}_1$ denote point positions in world coordinates, and they need to be transformed to and from device coordinates for texture access operations. This transformation accounts for deformations that change the overall size of the volume data, as well as anisotropic voxel spacing in the input data.

Next, we describe the massively parallel rasterization of the complete grid. We start with a basic algorithm, and in the next section we describe the addition of culling for improved efficiency. First, for each tetrahedron, we compute matrices of barycentric mappings $\mathbf{B}_0^{-1}$ and

$\mathbf{B}_1$. We also compute an AABB for each deformed tetrahedron. Then, we process all voxels inside the AABB of each deformed tetrahedron, and compute their barycentric coordinates $\mathbf{b}$ following Eq. (3.2). For each voxel, we test if it lies inside its corresponding tetrahedron or not using the barycentric coordinates. If it does, then we compute and write the deformed scalar value.

The computation of barycentric mappings and the AABBs of tetrahedra are executed on the CPU. Processing the voxels, on the other hand, is remarkably amenable to GPU architectures. Our voxel rasterization algorithm is outlined in Algorithm 1. It barely suffers divergence, as the voxels that do not follow the main flow, i.e., those that lie outside the tetrahedron, are simply discarded. Texture look-ups are not coalesced, but they enjoy high cache coherence.

Algorithm 1 GPU voxel rasterization algorithm

INPUT: *thread_id*, *block_id*, c_0

OUTPUT: c_1

corner = GetAABBCorner(*block_id*)

size = GetAABBSize(*block_id*)

$\mathbf{x}_1 =$ *corner* + ComputePosInAABB(*thread_id*, *size*)

$\mathbf{B}_1$ = GetB1(*block_id*)

$\mathbf{b} = \mathbf{B}_1 \bar{\mathbf{x}}_1$

if IsOutsideTet($\mathbf{b}$) **then**

  discard thread

end if

$\mathbf{B}_0^{-1}$ = GetB0Inv(*block_id*)

$\bar{\mathbf{x}}_0 = \mathbf{B}_0^{-1} \mathbf{b}$

c_1 = GetTrilinear(c_0, $\mathbf{x}_0$)

3.2 Hierarchical Culling

The volume of a tetrahedron is just $\frac{1}{6}$ of the volume of a prism defined by one of its corners and the three incident edges. This fraction of volume suggests that most of the voxels inside the AABB of a tetrahedron fall outside the tetrahedron itself. In fact, we found that, with the AABB-based rasterization described in the previous section, only 14.3% of the candidate voxels

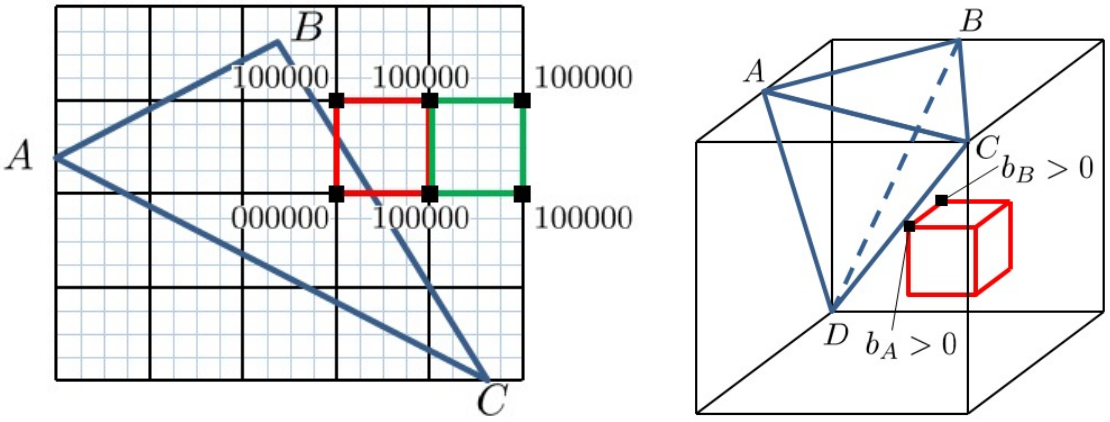

Figure 3.2: Left: Examples of grid point masks for a triangle (A,B,C). The green cell can be culled because the barycentric coordinate of A is < 0 for its 4 vertices. Right: In 3D, our culling algorithm may produce false positives for cells close to an edge of a tetrahedron, such as the red cell in the figure.

fall inside their corresponding tetrahedron. Next, we describe a hierarchical culling approach that reduces dramatically the voxels to be rasterized.

3.2.1 Grid Point Masks

For each tetrahedron in the deformed mesh M_1, we define a coarse grid with a spacing of N voxels. Each coarse cell encloses N^3 voxels of the output grid G_1, and if a cell does not intersect the tetrahedron, then its complete batch of enclosed voxels can be culled. Instead of testing for exact cell-tetrahedron intersection, we compute a spatial classification of coarse grid points, and then apply a conservative culling of coarse cells.

Based on barycentric coordinates $\mathbf{b} = (\ b_A \quad b_B \quad b_C \quad b_D\ )^T$ for a tetrahedron (A,B,C,D), we define 8 half-spaces $b_A < 0$, $b_A > 1$, $b_B < 0$, $b_B > 1$, $b_C < 0$, $b_C > 1$, $b_D < 0$, and $b_D > 1$. Then, for each coarse grid point, we compute an 8-bit mask where each bit classifies the point w.r.t. one of the 8 half-spaces.

3.2.2 Cell Culling

Our culling algorithm is based on the following theorem. Given the 8 half-spaces of a tetrahedron as defined above, and the 8 vertices of a coarse cell, if there is at least one half-space such

that all 8 vertices lie inside, then the tetrahedron and the cell do not intersect. As a corollary, all voxels inside the cell can be culled and do not need to be rasterized.

Based on this theorem and the readily available grid point masks, the culling of cells can be trivially executed as follows. For each cell, we perform a logical AND operation of the masks of its 8 vertices. The cell can be culled if the value of its mask is not 0. Fig. 3.2-left shows examples of point and cell mask computations.

In 2D, cell-triangle culling is exact if all 4 vertices of a cell are inside the AABB of the triangle. In 3D, cell-tetrahedron culling is conservative and may produce false positives for cells close to an edge of the tetrahedron, as shown in Fig. 3.2-right. However, as we show in our results, the number of false positives is small in practice, and we achieve a good trade-off w.r.t. culling cost.

Our algorithm would easily allow additional levels of hierarchical culling and an octree-based refinement strategy. However, our results indicate that, with just one level of hierarchical culling, rasterization of valid voxels becomes the bottleneck; therefore, more sophisticated culling would not yield additional speed-up.

3.2.3 Implementation Details

First, we process the coarse grid points of all tetrahedra and compute their masks. Subsequently, we process the cells of all tetrahedra and compute their masks too. Although these two procedures are highly amenable to GPU computation, we have found that a multi-core CPU implementation is fast enough and culling is not a bottleneck compared to rasterization, as we present in our results. After the computation of grid point masks and cell masks, we rasterize in parallel on the GPU all voxels of cells that cannot be culled. Each cell is treated as an AABB, and hence we follow the procedure already presented in Algorithm 1. In our tests, we found optimal performance by making the cell size the same as the CUDA block size. Once culling is executed, we construct on the CPU look-up tables that map each valid cell, through its corresponding CUDA *block_id*, to the AABB's size and location, and the barycentric mappings $\mathbf{B}_1$ and $\mathbf{B}_0^{-1}$.

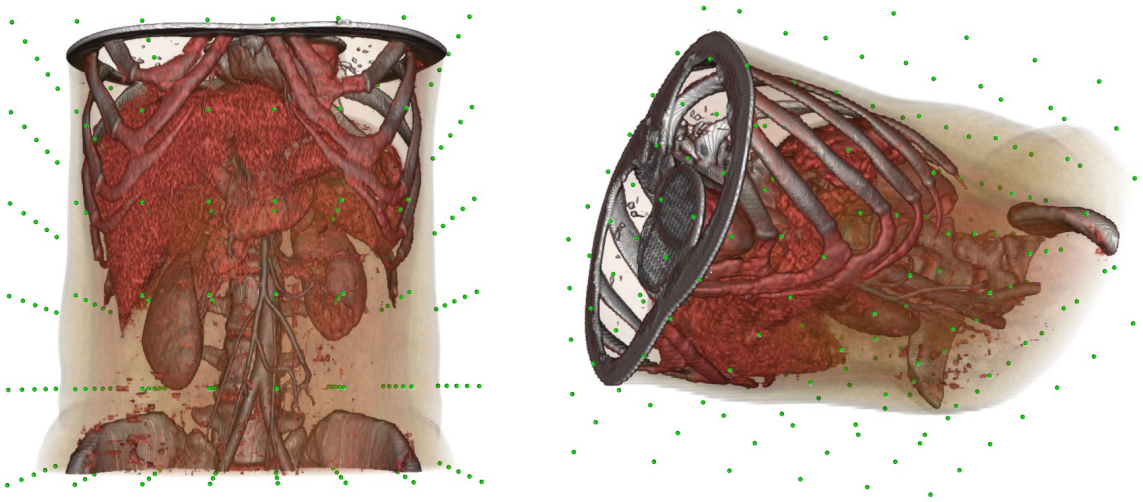

Figure 3.3: Torso model for performance analysis. Left: in its initial configuration; Right: rotated 45deg around two orthogonal axes.

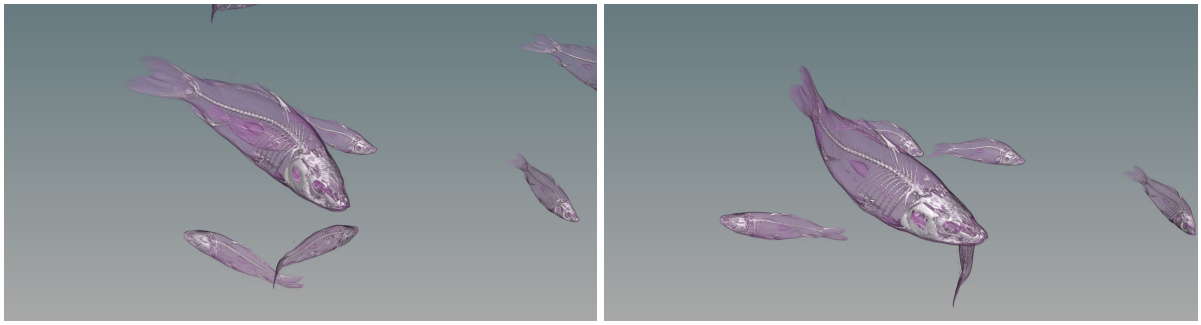

Figure 3.4: Animation of swimming carps with raycasted volume visualization. Each carp is represented using a $204 \times 202 \times 512$ volume and deformed using a 35-tetrahedra mesh. Our rasterization algorithm runs at 57.87ms per carp.

3.3 Results and Evaluation

We have tested our fast rasterization algorithm on several volumetric deformation examples. All examples were executed on a 3.40GHz 8-processor Intel i7 CPU with 32GB of RAM, and a NVIDIA GeForce GTX 680 GPU with 2GB of RAM. Our parallel GPU rendering algorithm was coded using CUDA. We observed optimal processor occupancy with a CUDA block size of 512, and optimal balance between culling and performance trade-off with a cell size of $8 \times 8 \times 8 = 512$, i.e., equal to block size. For volume rendering, we have used VTK [SML04].

3.3.1 Performance Analysis

To evaluate the performance of our algorithm, we have designed a controlled deformation example, tested under various resolutions and settings. Fig. 3.3 shows two snapshots of a volumetric anatomical model, in its upright initial configuration (left), and rotated 45 degrees around two orthogonal axes (right). The model is meshed with an axis-aligned regular tetrahedral mesh, and the rotation creates a misalignment of axes and tetrahedral edges, increasing the volume of AABBs of tetrahedra. We have tested tetrahedral meshes ranging from 40 to 5000 tetrahedra, and volume data with resolutions ranging from $128 \times 128 \times 128$ to $512 \times 512 \times 512$.

First, we have evaluated the performance of our rasterization algorithm with no culling, as described in Section 3.1. Most of the time spent on rasterization is devoted to the computation of barycentric coordinates for voxels that fail the barycentric coordinate test. In fact, only 14.3% of the voxels processed in the GPU pass the barycentric test and are actually updated.

With our culling algorithm described in Section 3.2, on the other hand, 51% of the voxels processed in the GPU pass the barycentric test and are actually updated. The total rasterization time achieves a speed-up of up to 1.5×. Fig. 3.5 shows graphs of total rasterization time per frame vs. volume data size, with and without culling. The speed-up becomes larger with larger datasets. For this comparison, we used a tetrahedral mesh with 1080 tetrahedra.

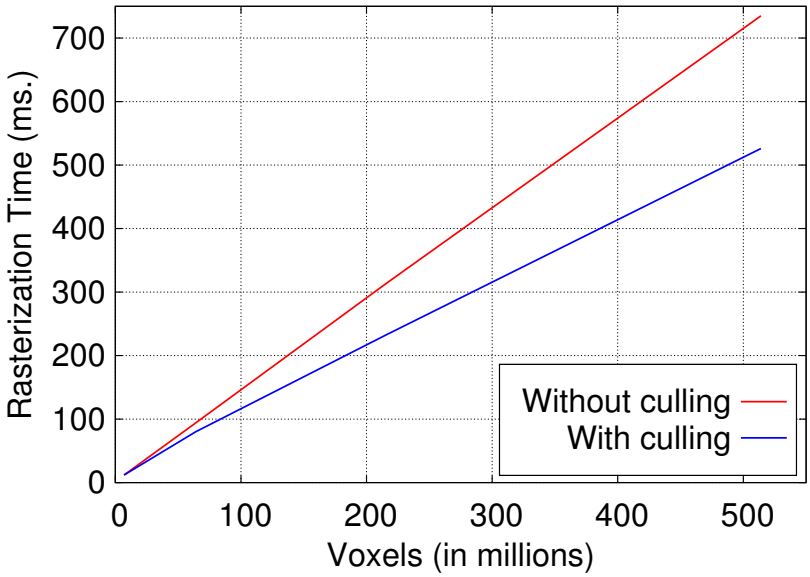

Figure 3.5: Evaluation of performance (with and without culling) vs. volume resolution for the torso model in Fig. 3.3. We used a tetrahedral mesh with 1080 tetrahedra.

Our voxel batch culling takes only 7% of the total cost on average, and culls away 68% of the unnecessary voxels. As described in Section 3.2, we have used a parallel CPU implementation for the computation of grid point masks and cell masks. On an 8-core machine, it achieves up to 2.6× speed-up over a single core implementation.

We have also evaluated the influence of the tetrahedral mesh resolution on performance, as shown in Fig. 3.6. The plot shows the total rasterization time per frame vs. tetrahedral mesh size, with and without culling, for the rotated configuration shown in Fig. 3.3. The data in the plot was collected for a volume with 233M voxels after the rotation. As expected, with culling, performance tends to decrease with denser tetrahedral meshes, as more tetrahedra need to be processed, and the cell resolution reduces the culling efficiency.

3.3.2 Simulation Examples

Our first two examples show the potential of our technique for animation purposes. Fig. 3.4 shows 8 carp models swimming, due to a scripted procedural deformation applied to their embedding tetrahedral meshes. Each carp is modeled using 35 tetrahedra, and the volume dataset consists of 21M voxels in the undeformed state. Each carp is rasterized in 57.87ms on average. Fig. 3.7 shows 6 oranges falling and rolling on a plane. We have modeled the deformation using a linear corotational finite element model [MG04], and we have simulated frictional contact with the plane considering only collisions of the nodes of the tetrahedral mesh. Each orange is modeled using 160 tetrahedra, and the volume dataset consists of 6.3M voxels in the undeformed state. Each orange is rasterized in 15.64ms on average.

But, in addition to animation, our work is largely motivated by medical planning applications, with the possibility to provide an interactive volume manipulation and deformation tool. As a test example, we present the interactive deformation of the torso model in Fig. 3.1. Manipulating the original volume data is appealing for medical applications, as the full detail of the data is still available during the deformation, and the volume rendering settings can be dynamically tuned. The example uses a finite element model with 700 tetrahedra, and a volume dataset with 4.99M voxels. The full model is rasterized in 14.86ms on average. Note that even though the deformations may be localized, we rasterize the full volume to test the performance of our algorithm.

We have modeled the inhomogeneity of tissue properties using standard tables to con-

vert opacity values into mechanical parameter values, and we compute nodal masses and per-element stiffness matrices by integrating per-voxel mechanical parameters. Nevertheless, the material properties and the actual deformations do not intend to appear realistic; we simply demonstrate the performance of our method and the interaction possibilities.

3.4 Conclusion

In this chapter we have presented an algorithm to efficiently deform volumetric data by rasterizing an embedding tetrahedral mesh. The simplicity of the method is key for its high performance, as it enables processing all target voxels in parallel with very simple operations and practically no divergence. To further accelerate rasterization, we apply efficient multi-core CPU culling as a first step.

Our method suffers some limitations, such as the existence of false positives during culling. However, these false positives do not hurt performance significantly. Another limitation is the smoothing introduced by trilinear interpolation of input data. More costly filtering approaches would produce higher quality results.

From an applied point of view, our method allows interactive editing, manipulation, and

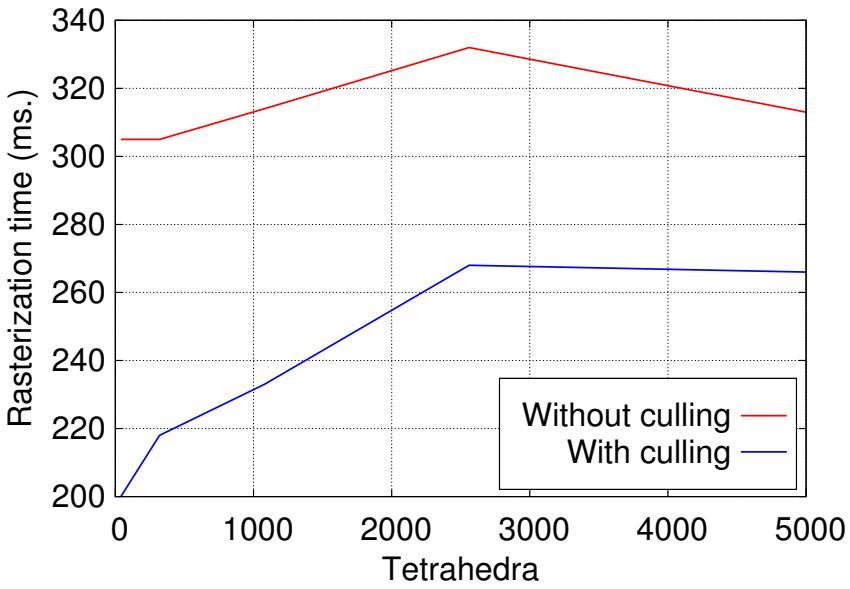

Figure 3.6: Evaluation of performance (with and without culling) vs. number of tetrahedra for the torso model in Fig. 3.3. We used a volume with 233M voxels after the rotation.

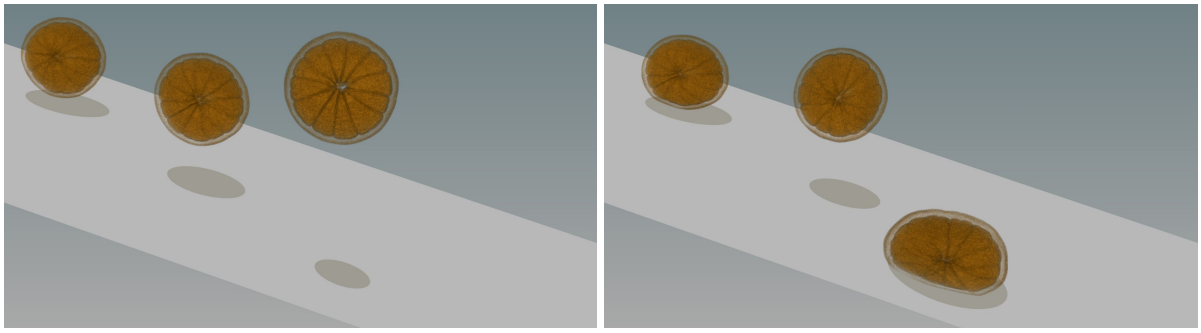

Figure 3.7: Deformable oranges bounce and roll on a plane. Each orange is represented using a $198 \times 199 \times 160$ volume and deformed using a 160-tetrahedra mesh. Our rasterization algorithm runs at 15.64ms per orange on average.

deformation of dense volume data. Its applicability could be extended by handling other types of mesh elements and basis functions, such as trilinear interpolation in hexahedra. This extension would require modifications to the mapping function and the culling algorithm.

Currently, our rasterization algorithm is being used by the GMRV group in the context of an INNPACTO project in collaboration with company GMV. The method is being applied for the deformation of breast imaging data, and there is currently a submission under review.

Chapter 4

Constraint-Based Simulation of Adhesive Contact

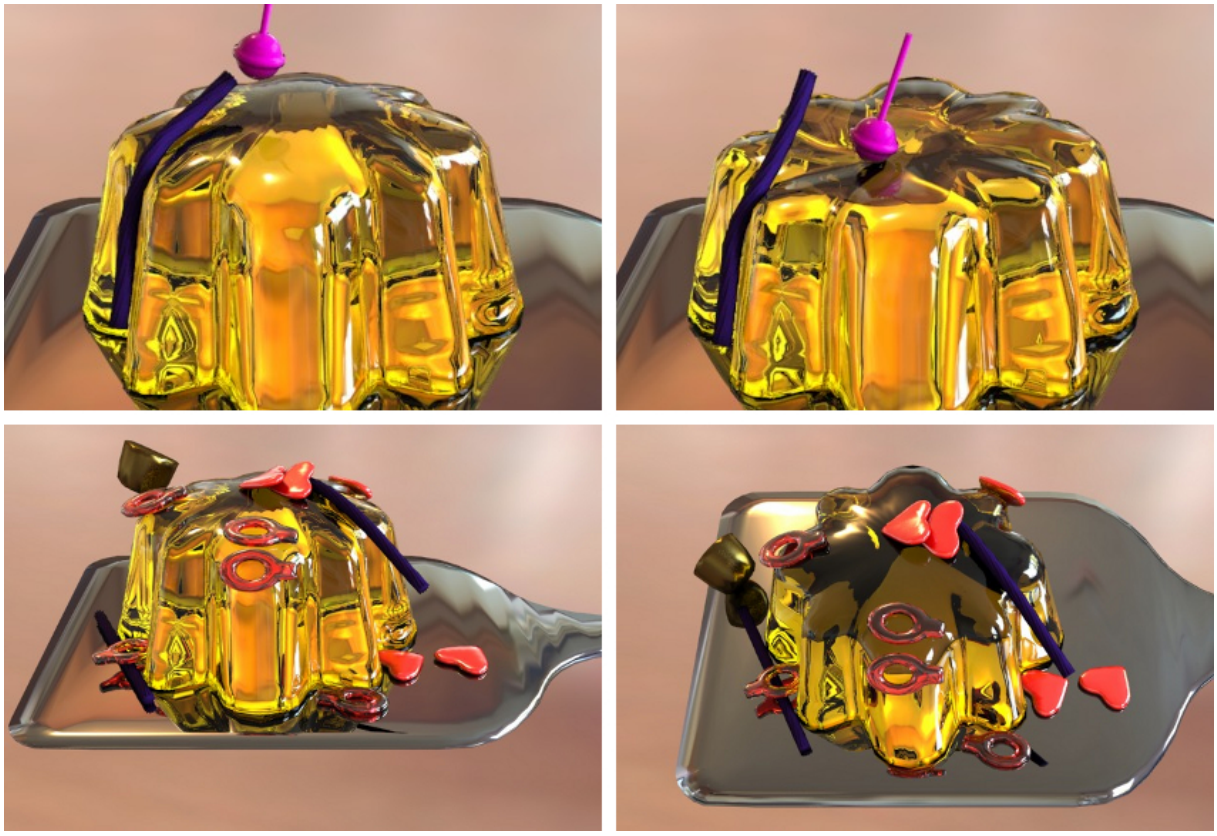

Figure 4.1: Pieces of candy with diverse adhesion coefficients fall on top of a block of Jell-O.

In the previous chapter we showed a volumetric representation used for deformable anatomic models and other biological forms. In this chapter we describe a phenomenon very common

between biological surfaces called adhesion. Adhesion is an effect intrinsically related to contact that was introduced in the chapter 1 (subsection 1.2). This phenomenon can be regarded as a thermodynamic effect in which a potential energy is stored at the interface between two surfaces. Debonding two surfaces that are adhered requires a traction force high enough to release the adhesion energy [RCC99]. In addition, we have referred to other approaches regarding adhesion in the chapter 2 (subsection 2.2.4). More recently in computer graphics, adhesion is commonly handled in the simulation of viscoplastic materials using continuum models (See [BWHT07, WTGT09] for two ways of merging viscoplastic materials due to adhesion).

Instead, in this chapter we are interested in modeling and simulating adhesion at a coarser scale, in order to efficiently handle sticking effects at the interface between rigid and/or elastic objects. Some examples of these kind of objects can be found in figure 1.5. We follow a constraint-based formulation, inspired by adhesion models described in the computational mechanics literature [Wri02, Fre87, RCC99]. Our main contribution is an algorithm for efficiently handling adhesion as part of constrained dynamics simulation.

Given a constraint-based formulation of contact dynamics (Section 4.1), and a formulation of adhesion using unilateral constraints (Section 4.2), we have developed an algorithm (Section 4.3) for seamlessly integrating adhesion constraints into state-of-the-art constraint-based contact solvers. A priori, this integration is not trivial, because, unlike non-penetration constraints, adhesion constraints are formulated in terms of both contact force and the separation at the contact interface. When formulating these constraints implicitly (a condition for large time steps), they become non-linear, thereby complicating the solution of the system. However, we present an algorithm that elegantly handles implicit adhesion constraints in the context of a projected-Gauss-Seidel solver for linear complementarity problems.

Our approach is general, and it handles rigid bodies, volumetric elastic bodies, thin shells such as cloth, and their combinations, as shown in our examples. Once the mathematical formulation is developed, integrating adhesion in state-of-the-art constraint-based contact solvers is simple and efficient, allowing interesting effects with low effort.

4.1 Constraint-Based Contact

In this section, we describe the underlying constrained dynamics formulation where we include the formulation of adhesion constraints. We first describe a general formulation of the constrained dynamics problem, and then we discuss its solution using a PGS solver.

4.1.1 Formulation

Given state and velocity vectors $\mathbf{q}$ and $\mathbf{v}$ that group the coordinates and velocities of all objects in a scene, we target constrained dynamics formulations of a general form

$$\mathbf{M}\dot{\mathbf{v}} = \mathbf{F}, \tag{4.1}$$

$$\dot{\mathbf{q}} = \mathbf{G}\mathbf{v}, \tag{4.2}$$

$$\mathbf{g}(\mathbf{q}) \geq 0. \tag{4.3}$$

$\mathbf{M}$ denotes de mass matrix and $\mathbf{F}$ is the force vector, $\mathbf{G}$ relates the velocity vector to the derivative of the generalized coordinates ($\mathbf{G}$ is typically identity for deformable bodies, but not for rigid bodies [Sha89]), and $\mathbf{g}$ is a vector of constraints. In our examples, we have used linear corotational finite element models [MKN+04], mass-spring cloth [BFA02], and rigid bodies. We formulate contact constraints by executing continuous collision detection between state updates. The general formulation is valid for other constraints such as joints, although we did not test them in our examples.

We assume that the dynamics equations of the system are discretized and linearized, which yields a constrained velocity formulation of the form:

$$\mathbf{A}\mathbf{v} = \mathbf{J}^T\lambda + \mathbf{b}, \tag{4.4}$$

$$\mathbf{0} \leq \lambda \quad \perp \quad \mathbf{J}\mathbf{v} \geq \mathbf{c}. \tag{4.5}$$

The system dynamics may be discretized with explicit integrators or implicit integrators with force linearization (see [BW98a] for the formulation of $\mathbf{A}$ and $\mathbf{b}$ under implicit Backward Euler discretization). λ represents contact impulses at the constraints, while Eq. (4.5) describes non-penetration as linear complementarity constraints. In our examples, we used an implicit position-level LCP, linearized to yield velocity constraints as shown here. This type of formulation (including friction, which is omitted here for readability) can be found, for example,

in [DDKA06]. Specifically, we have followed the approach of [OTSG09] for the discretization of both dynamics and contact constraints.

The complete system from Eqs. (4.4) and (4.5) constitutes a mixed linear complementarity problem (MLCP). With friction, the system remains an MLCP if friction constraints are expressed using a linearized version of Coulomb's friction cone. We align the friction cone at each contact with the direction of the unconstrained tangential velocity. The MLCP can be transformed into the following LCP:

$$\begin{aligned} &\mathbf{0} \leq \lambda \quad \perp \quad \mathbf{B}\lambda \geq \mathbf{d}, \text{ with} \\ &\mathbf{B} = \mathbf{J}\mathbf{A}^{-1}\mathbf{J}^T, \quad \mathbf{d} = \mathbf{c} - \mathbf{J}\mathbf{A}^{-1}\mathbf{b}. \end{aligned} \tag{4.6}$$

In our examples, we have used *iterative constraint anticipation* [OTSG09], a variant of this formulation that produces a sparse matrix $\mathbf{B}$ by nesting two relaxation solvers. The approach described in this chapter for including adhesive constraints into the LCP is independent of the way in which the LCP is formulated, but, for deformables objects with many degrees of freedom and many contacts, iterative constraint anticipation provides better performance in practice.

The i^{th} constraint in Eq. (4.6) represents the implicit non-penetration constraint of the i^{th} contact, after linearization and time integration. Then, the gap function g_i at the i^{th} contact at the end of the time step can simply be expressed (up to linearization) as

$$g_i = \Delta t(\mathbf{B}_i\lambda - d_i) \tag{4.7}$$

$\mathbf{B}_i$ and d_i represent the i^{th} rows of $\mathbf{B}$ and $\mathbf{d}$, respectively.

4.1.2 Solution

We consider the solution of the LCP problem above using a PGS solver. Then, when a PGS iteration reaches the i^{th} contact, the constraints for that contact can be expressed as:

$$0 \leq \lambda_i \quad \perp \quad B_{ii}\lambda_i - \overline{d}_i \geq 0, \tag{4.8}$$

$$\text{with } \overline{d}_i = d_i - \overline{\mathbf{B}}_i\overline{\lambda}_i. \tag{4.9}$$

$\overline{\lambda}_i$ contains all values of λ but λ_i. It combines values from the current iteration of PGS (up to the i^{th} entry), with values from the previous iteration (after the i^{th} entry). $\overline{\mathbf{B}}_i$ is defined accordingly, by removing B_{ii} from $\mathbf{B}_i$.

During each iteration of PGS, the i^{th} contact is handled as follows:

1. Compute $\lambda_i^* = \frac{\bar{d}_i}{B_{ii}}$.

2. Project $\lambda_i = \max(\lambda_i^*, 0)$.

4.2 Formulation of Adhesion

The thermodynamics model of adhesion by Raous et al. [RCC99] defines an elastic potential energy at a contact interface as a function of the separation gap g and an adhesion intensity β (with both terms squared). The adhesion intensity term captures the thermodynamic effect that, under traction, internal adhesion energy can be released as heat. This thermodynamics model has two implications when developing a computational algorithm for simulating adhesion: (i) it defines a constraint law that relates the maximum adhesive force to the contact gap and the adhesion intensity, and (ii) it defines a physical law for debonding, i.e., the time-dependent reduction of the adhesion intensity due to heat release. In this section, we describe the adhesion constraint law and the debonding law, as well as our own model for bonding, i.e., the time-dependent increase of the adhesion intensity under compression.

4.2.1 Adhesion Constraints

An adhesion constraint implies that the traction force must be smaller than a maximum defined by the adhesion intensity. Together with the non-penetration constraint, adhesive contact can be formulated with the following complementarity constraints [RCC99, Wri02]:

$$0 \leq -p_i + C_i \beta_i^2 g_i \quad \perp \quad g_i \geq 0 \tag{4.10}$$

where $\beta_i \in [0,1]$ is the adhesion intensity, C_i is the adhesion stiffness, and p_i is the traction. The value of the adhesion stiffness depends on the materials and the local properties of the contact interface.

In order to handle adhesion in the tangent plane, we use a box model that accounts separately for the normal adhesion and tangential adhesion along two orthogonal directions. We set a local frame on each contact, using the normal $\mathbf{n}$, the direction of unconstrained tangential velocity, $\mathbf{t}$, and the binormal $\mathbf{b} = \mathbf{n} \times \mathbf{t}$. The adhesion constraint can then be expressed for each contact impulse and gap function independently. Tangential adhesion can be regarded as a model similar

to Coulomb friction, with the difference that the magnitude of the tangential force is limited by the adhesion intensity, instead of the magnitude of the normal force.

4.2.2 Bonding

Bonding and debonding model the evolution of the adhesion intensity as a function of the contact traction/compression. In the case of bonding, we account for a bonding rate, r, and a compression value for saturation, p_0. Adhesion intensity will grow as long as compression is exerted, until saturation is reached. Specifically, our bonding model is formulated as:

$$\dot{\beta}_i = r \max(p_i - \beta_i p_0, 0) \tag{4.11}$$

4.2.3 Debonding

For the debonding model, we follow the linear case in the thermodynamic adhesion model [RCC99, Wri02]. Considering the adhesion stiffness C and the gap function g, debonding starts taking place once $Cg^2\beta$ reaches a maximum adhesion energy W. The term $Cg^2\beta$ is obtained by differentiating a thermodynamic energy $1/2Cg^2\beta^2$ w.r.t. β. Please see [RCC99] for the full details. During debonding, adhesion decreases at a rate of $\frac{1}{\eta}$, where η is a viscosity parameter. Formally, we can write the debonding model for the i^{th} contact as:

$$\dot{\beta}_i = \frac{1}{\eta} \min(W - C_i g_i^2 \beta_i, 0) \tag{4.12}$$

4.3 Algorithm

We describe now our algorithm for including adhesion constraints in the contact solver outlined in Section 4.1. We start by describing the 1D case, and then extend it to the full 3D case including tangential adhesion. Last, we describe the evolution of the adhesion intensity.

4.3.1 Implicit Adhesion Constraints: 1D Case

Our goal is to execute a PGS step similar to the one in the non-adhesive case (Eq. (4.8)). In the adhesion constraint in Eq. (4.10), contact traction is related to the gap function, hence implicit adhesion constraints do not allow for a simple computation of a projection value. Contact

traction is related to the contact impulse by $p_i = -\frac{\lambda_i}{\Delta t A_i}$, with A_i the local contact area. Accounting for the implicit definition of the gap function from Eq. (4.7), an adhesion constraint can be reformulated in terms of the contact impulse as:

$$0 \leq \lambda_i + \Delta t A_i C_i \beta_i^2 g_i \quad \perp \quad \mathbf{B}_i \lambda \geq d_i \tag{4.13}$$

In order to compute the local area, at edges and vertices we store the averaged area of incident triangles, and at each contact we select the smallest area from those of the two primitives involved in contact. In this way, contact traction is less sensitive to mesh resolution.

By substituting the implicit gap function from Eq. (4.7), we obtain the following implicit linearized adhesion constraint:

$$0 \leq \lambda_i + \Delta t^2 A_i C_i \beta_i^2 (\mathbf{B}_i \lambda - d_i). \tag{4.14}$$

A priori, the constraint depends on all values of contact impulses λ, but we are interested in its evaluation in one PGS step. Then, we can substitute the evaluation of the right-hand-side of the PGS step given by Eq. (4.9):

$$0 \leq \lambda_i + \Delta t^2 A_i C_i \beta_i^2 (B_{ii} \lambda_i - \overline{d}_i) \tag{4.15}$$

It suffices to single out λ_i in order to express the implicit adhesion constraint on the contact impulse. By analogy with Eq. (4.8), in the adhesive case the complementarity constraint in the PGS step turns into:

$$\frac{\Delta t^2 A_i C_i \beta_i^2 \overline{d}_i}{1 + \Delta t^2 A_i C_i \beta_i^2 B_{ii}} \leq \lambda_i \quad \perp \quad B_{ii} \lambda_i - \overline{d}_i \geq 0 \tag{4.16}$$

As demonstrated, since $\overline{d}_i$ is readily computed during the PGS iteration, applying implicit adhesion constraints effectively reduces to modifying the projection values of the PGS solver.

4.3.2 Full 3D Adhesion

With the inclusion of tangential adhesion, the contact impulse at the i^{th} contact can be represented as a vector $\lambda_i = (\lambda_{i,n}, \lambda_{i,t}, \lambda_{i,b})^T$, with the tangential impulses aligned with the pre-contact tangent velocity and the binormal. Using a Block-PGS relaxation solver, the $\mathbf{B}_{ii}$ block of the $\mathbf{B}$ matrix is now

$$\mathbf{B}_{ii} = \begin{pmatrix} B_{ii,nn} & B_{ii,nt} & B_{ii,nb} \\ B_{ii,tn} & B_{ii,tt} & B_{ii,tb} \\ B_{ii,bn} & B_{ii,bt} & B_{ii,bb} \end{pmatrix} \tag{4.17}$$

By analogy with the case with normal adhesion only, we can write now the implicit normal gap in the case of full 3D adhesion, in the context of the Block-PGS solver:

$$g_{i,n} = \Delta t(B_{ii,nn}\lambda_{i,n} - \overline{d}_{i,n}), \tag{4.18}$$
$$\text{with } \overline{d}_{i,n} = d_{i,n} - \overline{\mathbf{B}}_{i,n}\overline{\lambda} + B_{ii,nt}\lambda_{i,t} + B_{ii,nb}\lambda_{i,b}.$$

Tangential and binormal gaps can be expressed in a similar way. By inserting these implicit expressions into the adhesion constraints expressed as in Eq. (4.13), we can single out the contact impulses and formulate the projection values for the Block-PGS solver. At the projection step, tangential and binormal adhesion are handled slightly differently than in the normal direction, because forces must be constrained in positive and negative directions.

Eventually, the algorithm for Block-PGS with implicit adhesive constraints can be outlined as follows:

1. Compute $\overline{d}_{i,n} = d_{i,n} - \overline{\mathbf{B}}_{i,n}\overline{\lambda} + B_{ii,nt}\lambda_{i,t} + B_{ii,nb}\lambda_{i,b}$.

2. Compute $\lambda^*_{i,n} = \frac{\overline{d}_{i,n}}{B_{ii,nn}}$.

3. Project $\lambda_{i,n} = \max(\lambda^*_{i,n}, \min(\frac{\Delta t^2 A_i C_i \beta_i^2 \overline{d}_{i,n}}{1+\Delta t^2 A_i C_i \beta_i^2 B_{ii,nn}}, 0))$.

4. Compute $\overline{d}_{i,t} = d_{i,t} - \overline{\mathbf{B}}_{i,t}\overline{\lambda} + B_{ii,tn}\lambda_{i,n} + B_{ii,tb}\lambda_{i,b}$.

5. Compute $\lambda^*_{i,t} = \frac{\overline{d}_{i,t}}{B_{ii,tt}}$.

6. If $\lambda^*_{i,t} < 0$, $\lambda_{i,t} = \max(\lambda^*_{i,t}, \min(\frac{\Delta t^2 A_i C_i \beta_i^2 \overline{d}_{i,t}}{1+\Delta t^2 A_i C_i \beta_i^2 B_{ii,tt}}, 0))$.

7. Else, $\lambda_{i,t} = \min(\lambda^*_{i,t}, \max(\frac{\Delta t^2 A_i C_i \beta_i \overline{d}_{i,t}}{1+\Delta t^2 A_i C_i \beta_i B_{ii,tt}}, 0))$.

8. Do for $\lambda_{i,b}$ similarly as for $\lambda_{i,t}$.

It is convenient to include a friction model, and, out of tangential adhesion and friction, apply the most restrictive projection. This projection refers to the value used for projection in step 6 above. We have used Coulomb's friction model with a 4-sided pyramid approximation. Same as for tangential adhesion, we align the pyramid in every time step to the unconstrained relative velocity at each contact.

4.3.3 Adhesion Evolution

After a complete step of the constrained dynamics solve, we evolve the adhesion intensity β at all contacts. First, we determine the compression or traction state, and apply the bonding or debonding model, as appropriate. Normal compression may increase bonding, while normal traction may decrease bonding. Tangential adhesion forces, on the other hand, always tend to decrease bonding (if they exceed the debonding energy). In case of normal traction, we compute the total adhesive traction $p = \|(p_n, p_t, p_b)\|$ and apply the debonding law in Eq. (4.12). In case of normal compression, we compute the tangential adhesive traction $p = \|(p_t, p_b)\|$, and add simultaneous debonding and bonding effects.

Given the time-derivative of the adhesion intensity, $\dot{\beta}$, we have used a simple explicit Euler integrator in order to compute the adhesion intensity for the next time step. We found that, for our examples, interesting adhesion effects take place with rather slow bonding and debonding dynamics, hence a simple explicit integrator sufficed.

After computing the adhesion coefficient for the next time step, we eliminate contacts where debonding has completely taken place. Eq. (4.12) models a first order system that never reaches $\beta = 0$, hence we apply full debonding when the gap function at a contact grows beyond a threshold. This threshold is set based on a reference gap value, as discussed next along with our results.

4.4 Results

Thanks to the implicit formulation of adhesion constraints, we were able to simulate in a stable manner adhesion stiffness values in the range of 10^6 to 10^8N/m^3 with time steps between 1ms and 5ms.

Our simulation examples show the application of our algorithm to mass-spring cloth (Fig. 4.2), and combined rigid and deformable bodies (Fig. 4.1). We have implemented a demo of swinging cloth that is shown in Fig. 4.2, we demonstrate that the overall behavior of adhesion varies little under varying mesh resolution. The *candy* demo depicts rich bonding/debonding effects, and we have also applied our algorithm to a facial animation setting (Fig. 4.3), where the lips of a character briefly stick to each other when opening the mouth. All our examples were rendered using YafaRay.

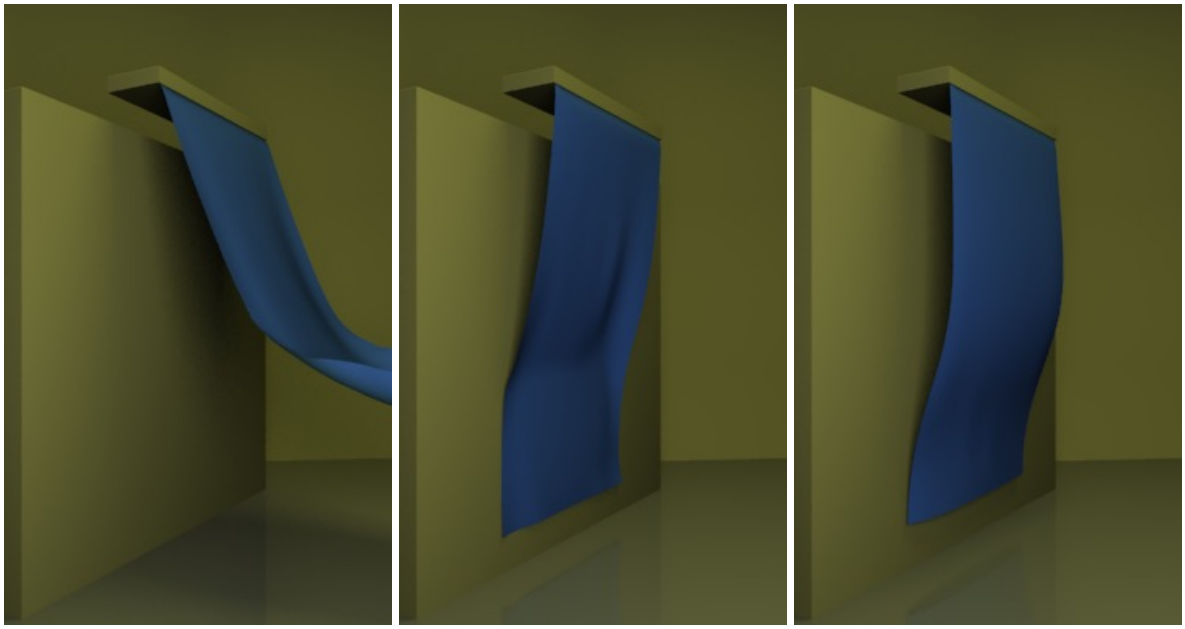

Figure 4.2: A swinging cloth hits a wall and adheres to it until it slowly starts debonding.

We have executed our demos on a 1.8-GHz Intel Core 2 Duo processor PC with 2GB RAM. In the *candy* demo, the deformable objects are meshed with a total of approximately 10K tetrahedra, and the triangle meshes involved in continuous collision detection consist of a total of 28K triangles. The adhesion properties are dominated by the adhesion stiffness of the Jell-O, which is $4 \times 10^7 \text{N/m}^3$, and the Coulomb friction coefficient is $\mu = 0.3$ for all objects in the scene. As shown in Fig. 4.4, the average number of contacts in the simulation is 151, and a maximum of 531. The 25-second simulation takes 20 minutes to compute (1.6 seconds/frame or 0.4 seconds/timestep with 8ms timesteps), which we consider is reasonably fast for a constrained deformation problem of the size described. We also computed the same simulation without adhesion constraints, and it took 13 minutes (1 second/frame). The main difference for the cost is not the convergence rate, which is almost the same in both cases, but the number of contacts. Without adhesion, contacts break easier, and the average number of contacts is 97, and a maximum of 330, as shown also in Fig. 4.4.

4.5 Discussion

In this chapter, we have shown a model for adhesive contact that can be efficiently integrated into existing constraint-based contact solvers. It retains the robustness of constraint-based con-

Figure 4.3: Simulation of an opening and closing mouth, with adhesion taking place at the lips.

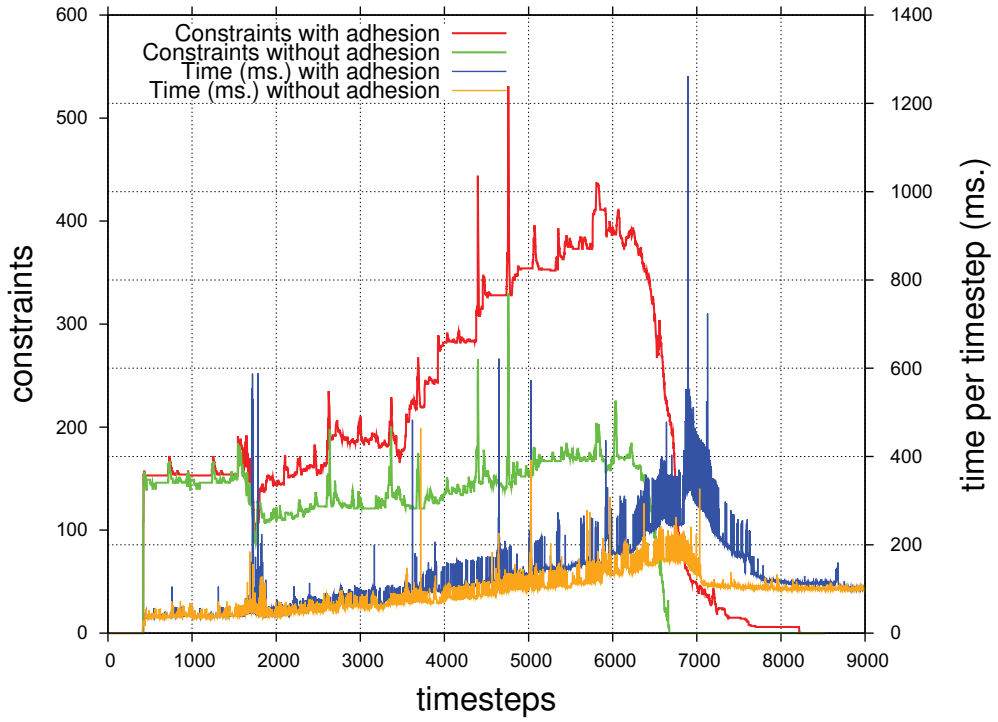

Figure 4.4: Comparison of number of contact constraints and timings per timestep for the *candy* demo from Fig. 4.1 with and without adhesion.

tact while allowing for rich and versatile adhesion effects under a diverse range of object types.

Moreover, the adhesion model incorporates a thermodynamics formulation of debonding

from the mechanics literature. Connected to this feature, one limitation in our work is that the formulation of bonding and the connection between friction and adhesion are not sustained by a comparable thermodynamics approach. Regarding friction, we obtained plausible results by selecting the most restrictive constraint out of Coulomb friction and tangential adhesion, as discussed in Section 4.3.2. Another limitation of our algorithm is that it requires contact tracking, not present in some of the available rigid body dynamics simulators, in order to evolve the value of the adhesion intensity across frames.

Although this was not a major problem in our examples, relaxation solvers, such as Gauss-Seidel, may suffer from slow convergence at times. This is a general limitation in constraint-based contact formulations, and more efficient LCP solvers are still an issue under investigation.

Our model handles only well-defined interfaces between rigid and deformable bodies. Therefore, another interesting extension to our work would be to integrate it with other materials, such as viscoplastic ones, for which adhesion produces very interesting effects.

As far as we know, our method was extended in later publications, like [AO11], for cohesion simulation in granular materials, using Smooth Particle Hydrodynamics (SPH).

Chapter 5

Modeling and Simulation of a Human Shoulder for Interactive Medical Applications

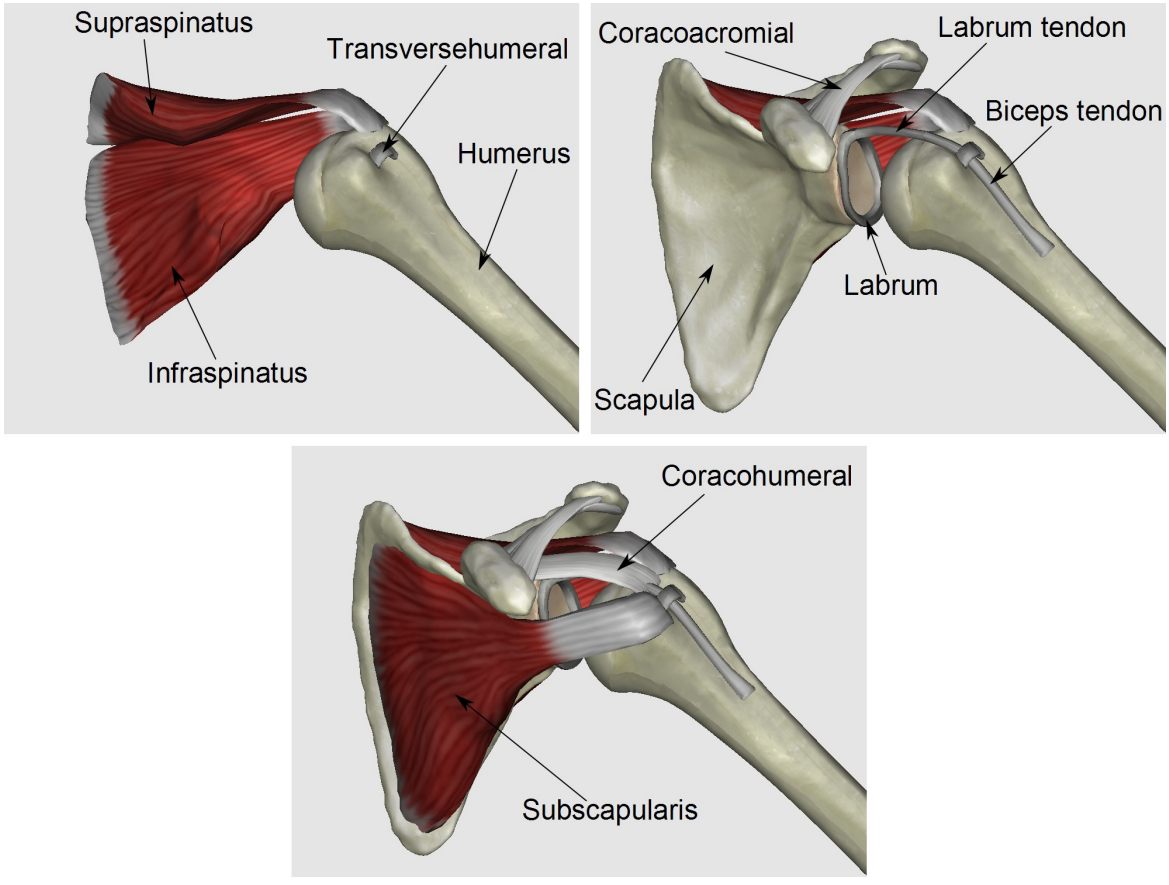

Figure 5.1: Layered display of the anatomical parts of the shoulder simulated in our examples.

In the previous chapter, a constrain-based formulation was presented for adhesive contact. In this chapter, contact constraints will also be used, but this time they are applied to virtual reality simulators, operating interactively.

Virtual reality simulators provide medical practitioners with a non degradable, versatile, and realistic environment, in which novices may learn and try as much as desired. There is a vast amount of examples of medical simulators, as discussed in various survey papers [LTCK03, LHS06, BSHW07].

But the true explosion of medical simulators is still to happen, and one of the major obstacles for this explosion is the difficulty to simulate in an interactive yet realistic manner the internal human anatomical structures. These anatomical structures present challenges such as a very diverse mechanical behavior, ranging from hard bone to soft fat tissue, and intricate contact situations. Contact, with its associated problems of collision detection, collision response, and friction handling, is often a computational bottleneck and, more importantly, a task with an unpredictable computational cost, which severely complicates interactive simulation.

In this chapter, we present a combination of representations, simulation methodology, and algorithms, geared at producing efficient yet plausible simulation of intricate internal human anatomy. One of the key ingredients for our simulation methodology, described in Section 5.1, is the choice of appropriate representations for dynamics simulation, contact handling, and visualization. A generalized definition of dynamics representation allows us to handle the binding of surface representations, as well as the coupling of anatomical parts, all in an elegant and unified manner. More information regarding contact coupling, just like other related approaches, can be found in chapter 2 (subsection 2.1.3). In Section 5.2 we discuss a modeling pipeline to produce all the representations and the coupling. And the second key ingredient of our methodology, described in Section 5.3, is a contact handling algorithm that accounts in a unified yet efficient manner for diverse dynamic representations, their couplings, and contact constraints.

Human joints are one of the situations where the challenges of biomechanical simulation arise constantly, therefore we have selected the shoulder as the target example for demonstrating our results. Specifically, we show application of interactive shoulder simulation to virtual arthroscopy [BGMFA06] and physiotherapy palpation [DLB94], as depicted in Fig. 5.6. With our simulation methodology and carefully selected representations, we obtain interactive contact, deformations, and even haptic feedback, on intricate situations involving multi-way contact

of several anatomical parts.

All the work in this chapter implied a team effort of several people, where the author of this dissertation was focused on the model creation pipeline, explained in subsection 5.2.2.

5.1 Representations

We use separate representations for the three main tasks in the simulation, namely, dynamics, collisions, and visualization, which allows us to treat each task efficiently. The dynamics representation acts as the link between all three representations, and it completely defines the state of the collision and visualization representations. In this section, we describe the three representations, as well as the generic data structures and mathematics for binding them together.

5.1.1 Dynamics, Collisions, Visualization

In our algorithm, we define a *contact object* as the elementary dynamic object with distinct mechanical properties (e.g., a bone, a ligament, etc.). Its dynamics representation consists of a state vector $\mathbf{q}$ and a velocity vector $\mathbf{v}$ that fully define the dynamics of an object. In the general case, the velocity and state vectors are related as $\dot{\mathbf{q}} = \mathbf{G}\mathbf{v}$. All contact objects share a common interface from the software engineering point-of-view, which allows us to handle them all in a unified manner in the simulation algorithm to be described in Section 5.3. For rigid bodies, the state vector $\mathbf{q}$ consists of the position of the center of mass and a quaternion for the orientation, while the velocity vector $\mathbf{v}$ consists of the linear and angular velocities of the body. For deformable bodies, we use a linear co-rotational finite element formulation [MKN$^+$04], with objects discretized using tetrahedral meshes. Then, the state vector is formed by the positions of mesh nodes, and the velocity vector is formed by the velocities of the nodes.

For collisions and visualization, we represent each contact object using a collection of triangle meshes. The topology of these meshes is fixed, and their geometry is fully defined by the positions of their vertices, which in turn are defined by the dynamics representation as we describe next.

5.1.2 Binding Dynamics and Surfaces

The position of every vertex in a triangle mesh is computed using a generic *point* entity. In essence, a point binds a vertex and a generic contact object, by a definition of the vertex position as $\mathbf{p} = f(\mathbf{q})$. For rigid bodies, the vertex position is defined as $\mathbf{p} = \mathbf{c} + \mathbf{R}\mathbf{r}$, where $\mathbf{c}$ and $\mathbf{R}$ are the position and orientation of the rigid body, and $\mathbf{r}$ is the position of the vertex in the body's local reference system. For deformable bodies, we assume each surface vertex to be embedded inside a tetrahedron. Then, the vertex position is defined as $\mathbf{p} = \sum_{i=1}^{4} w_i \mathbf{q}_i$, where the 4 $\mathbf{q}_i$ values are the positions of tetrahedral nodes, and the w_i are barycentric coordinates. Full details about the construction and embedding of the tetrahedral mesh are given in the next section.

The point entity also relates the velocities of mesh vertices and the velocity vector of a contact object, by simple differentiation of the position, $\dot{\mathbf{p}} = \mathbf{J}\mathbf{v}$, with $\mathbf{J} = \frac{\partial \mathbf{p}}{\partial \mathbf{q}}\mathbf{G}$. Each specific point entity, depending on the type of contact object it acts on, stores this relationship in a compact manner. For deformable objects, for example, $\mathbf{J}$ is a sparse matrix where the only non-zero columns are those due to the tetrahedral nodes that affect the surface vertex. It is important to point out that velocities are always linearly related.

But one of the main features of our generic point entity definition is that it allows us to define forces and constraints on surface vertices in a unified manner, independently of the type of contact object. Given the relationship between the velocity vector and the velocity of a vertex, $\mathbf{J}$, forces on a contact object can be computed from surface forces as $\mathbf{F_q} = \mathbf{J}^T \mathbf{F_p}$. The linear relationship between velocities and between forces is the classic *manipulator Jacobian* from robotics.

For visualization purposes, we store mesh connectivity in the GPU by exploiting buffer objects. Every time a contact object is modified, we simply need to send the state vector $\mathbf{q}$ to the GPU.

5.2 Modeling

In this section, we list the anatomical parts of the shoulder that we have accounted for in our simulations, and we describe our modeling pipeline to produce the simulation scenario where the various representations of the different parts are integrated.

Id	Anatomical part	Render	CD	FEM	Couplings
1	Scapula	10224	105	NA	3, 4, 5, 6(x2), 7, 9
2	Humerus	5189	112	NA	2(x2), 3, 4, 5, 7, 8(x2), 11
3	Supraspinatus	1716	86	87	1, 2
4	Infraspinatus	2136	96	129	1, 2
5	Subscapularis	2424	100	158	1, 2
6	Coracoacromial	1632	146	65	1(x2)
7	Coracohumeral	522	60	86	1, 2
8	Transversehumeral	264	28	NA	2(x2)
9	Labrum	636	120	96	1, 10
10	Labrum Tendon	128	128	122	9, 11
11	Biceps Tendon	264	264	NA	2, 10

Table 5.1: List of anatomical parts simulated in the arthroscopy example, with the number of triangles of their visualization surface (Render), number of triangles of the collision surface (CD), number of tetrahedra of the dynamic representation (FEM), and list of couplings to other parts (indicated by their ids). 'x2' means that the coupling with another anatomical part takes place at two locations. The number of tetrahedra does not apply for rigid bodies.

5.2.1 Description of Anatomical Parts

The anatomy of the shoulder is full of diverse parts such as bones, muscles, ligaments, cartilages or tendons. The result is a compact and heterogeneous volume where it is not easy for unexperienced persons to discern one part from another. In order to develop useful and interesting applications, we need to simplify the anatomical complexity and focus on those parts that are meaningful for our application.

For example, in the case of arthroscopy, there are some bulky parts that are not interesting for the surgeon, such as the subacromial bursa or the deltoid muscle. In Fig. 5.1, we show the parts that we have accounted for in our arthroscopy example. We model the bones, i.e., the scapula and humerus, as rigid bodies. Moreover, due to their limited range of motion, we also model the biceps tendon and the transversehumeral as rigid bodies. All other parts are modeled as soft bodies. Table 5.1 indicates pairs of parts that are coupled using zero-length coupling springs. Note that some parts are coupled at two different locations to the same bone. When a large surface of a soft body is coupled to a bone, we disable collision detection between the two objects, since there is no relative motion between them. We do this, for example, in the coupling between the scapula and the labrum.

For physiotherapy palpation, on the other hand, the criterion for selecting the interesting parts is almost the opposite as for arthroscopy, because the practitioner focuses mainly on the outer anatomical layers. Therefore, for physiotherapy palpation we incorporate the deltoid muscle.

5.2.2 Model Creation Pipeline

The input to our pipeline is a set of triangle meshes that describe the surfaces of the various anatomical parts. These meshes can be obtained by scanning real parts with a standard 3D scanner, or by manual authoring. We use these meshes as the visualization representations in our examples.

For collision handling purposes, we apply standard mesh simplification techniques to obtain low-resolution approximations that can be efficiently handled interactively. Then, for the soft-tissue parts, we create the tetrahedral meshes that define the dynamic representation by embedding both the visualization and collision meshes. We start by enclosing the surface meshes

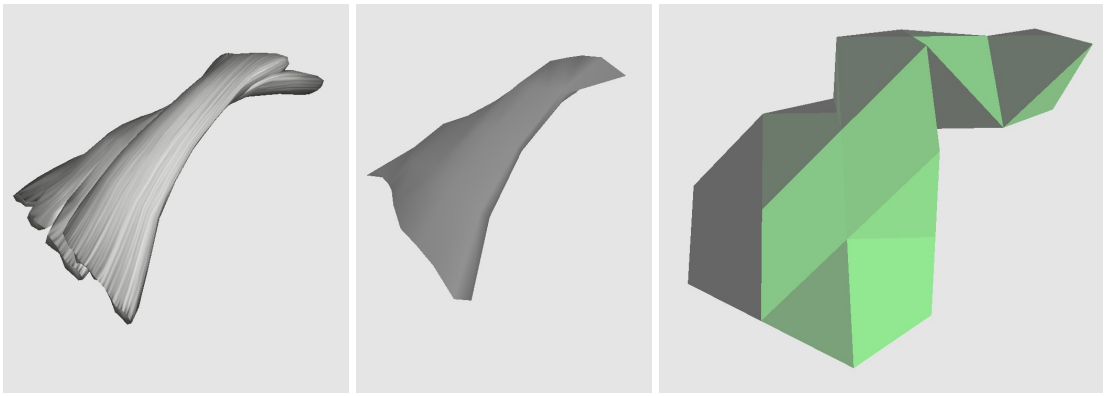

Figure 5.2: From left to right, visualization, collision, and dynamic meshes for the coracoacromial ligament.

with a bounding box, we subdivide it regularly to the desired cell resolution, decompose each cubic cell into five tetrahedra, and finally we eliminate those tetrahedra that do not intersect the volume enclosed by the visualization and collision meshes. Thanks to the embedding-based simulation, plausible deformations are possible even with rather coarse tetrahedral meshes. Fig. 5.2 shows the visualization, collision, and dynamics meshes for the coracoacromial ligament. The resolution of all meshes for the arthroscopy example is listed in Table 5.1.

In order to define couplings between a soft body and a rigid bone, we manually select tetrahedral nodes of the soft body that should be coupled to the bone, and we set zero-length binding springs at those nodes. Both end-points of a binding spring are fully defined by the state of their corresponding contact objects, using the point entity defined in Section 5.1.2. For couplings between soft bodies, we manually select tetrahedral nodes from both bodies, and set binding springs at those locations. Nodes from two bodies a and b are typically not collocated, therefore, one end-point of the binding spring is defined directly by the state of a tetrahedral node, while the other end-point is defined through barycentric interpolation inside the enclosing tetrahedron in the other body.

Due to the intricate layout of anatomical parts, it would be a daunting modeling task to ensure that all parts are intersection-free at their undeformed state. Instead of enforcing this, we allow the objects to intersect in the undeformed state, but we define an initialization state where they are intersection-free. We do this by deforming each tetrahedral mesh (and thus the visualization and collision meshes as well) using a cage-based deformation technique [JMD+07], as depicted in Fig. 5.3. In the simulation, the tetrahedral meshes are then initialized at a deformed

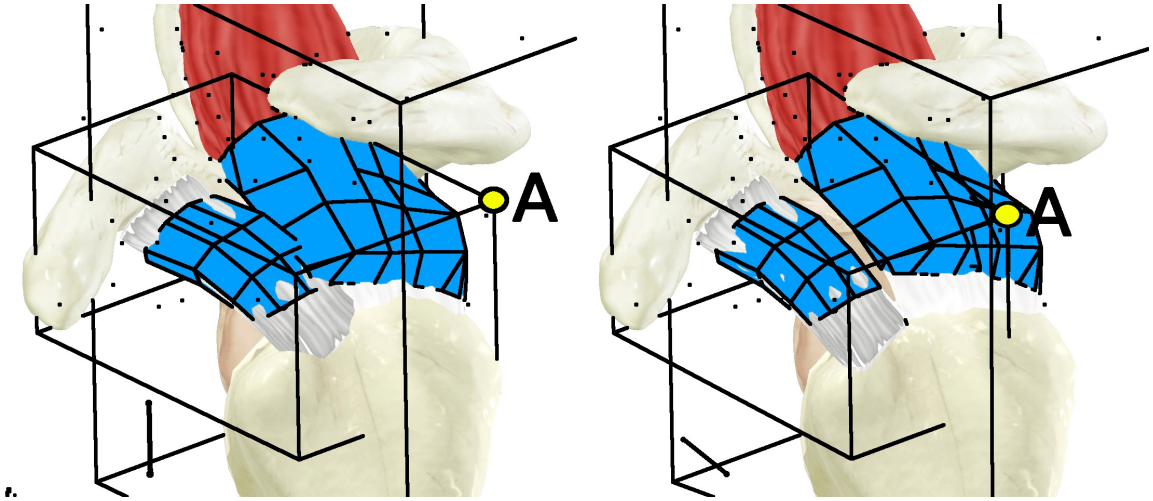

Figure 5.3: Cage-based deformation of the coracohumeral ligament to ensure a collision-free initial state. In blue, the collision meshes. Notice how in the undeformed state (on the left), the collision meshes are intersecting.

state, and as soon as the simulation starts they move to a minimum energy situation. As this work is performed by several authors, the author of this thesis was the person responsible for modeling and creating these intersection-free set-ups.

5.3 Simulation Algorithm

In this section we explain the main features of the algorithm that allows the simulation, in a unified yet efficient manner, of complex anatomical scenarios composed of objects with diverse mechanical behavior. At the same time, this algorithm handles elegantly coupling and contact constraints, making them independent of the objects that they act on.

5.3.1 Implicit Integration of Dynamics

Given state and velocity vectors $\mathbf{q}$ and $\mathbf{v}$ that group the state and velocity of all contact objects in the scene, the dynamics of the simulation are discretized with the ODEs:

$$\begin{aligned} \mathbf{M}\dot{\mathbf{v}} &= \mathbf{F} \\ \dot{\mathbf{q}} &= \mathbf{G}\mathbf{v} \end{aligned} \tag{5.1}$$

where $\mathbf{M}$ denotes the mass matrix and $\mathbf{F}$ is the force vector. We numerically integrate the ODEs using the (implicit) backward Euler method with linear approximation of forces, which yields a

velocity update

$$\begin{aligned} &\mathbf{A}\mathbf{v} = \mathbf{b}, \qquad \text{with} \\ &\mathbf{A} = \mathbf{M} - \Delta t \frac{\partial \mathbf{F}}{\partial \mathbf{v}} - \Delta t^2 \mathbf{G} \frac{\partial \mathbf{F}}{\partial \mathbf{q}}, \\ &\mathbf{b} = \Delta t \mathbf{F}(\mathbf{v_0}, \mathbf{q_0}) + \left(\mathbf{M} - \Delta t \frac{\partial \mathbf{F}}{\partial \mathbf{v}} \right) \mathbf{v_0}. \end{aligned} \tag{5.2}$$

Eq. (5.2) is solved using a Conjugate Gradient (CG) solver, and the result is the unconstrained velocity of the contact objects.

In a simulation with three independent objects, Eq. (5.2) can be writen as:

$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{A}_{33} \end{pmatrix} \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{pmatrix} \tag{5.3}$$

All non-diagonal terms of $\mathbf{A}$ are zero because there are no couplings (i.e. no forces) acting between different objects in the simulation. Our algorithm makes use of this fact and uses the CG solver efficiently, solving Eq. (5.2) for each object independently.

5.3.2 Coupling Islands

For each pair of anatomical structures that are solidly attached, we define a *coupling* between their corresponding contact objects. Our definition of coupling is general and is able to handle any pair of contact objects. Specifically, each coupling consists of two general *semicouplings* and, from a software engineering perspective, we define different semicoupling implementations based on the types of contact objects in our simulation.

For each coupling, we set zero-length springs between the two coupled contact objects, as shown in Fig. 5.4. These springs add new forces to the system, modifying the structure of the terms in Eq. (5.2). With coupled objects as in Fig. 5.4, the new system structure is:

$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{0} \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{A}_{23} \\ \mathbf{0} & \mathbf{A}_{32} & \mathbf{A}_{33} \end{pmatrix} \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{pmatrix} \tag{5.4}$$

The system is no longer block-diagonal, and the right-hand side $\mathbf{b}$ is also modified. Given a zero-length spring between two points $\mathbf{p}_a$ and $\mathbf{p}_b$, the spring force acting on point $\mathbf{p}_a$ is $\mathbf{F}_{\mathbf{p}_a} =$

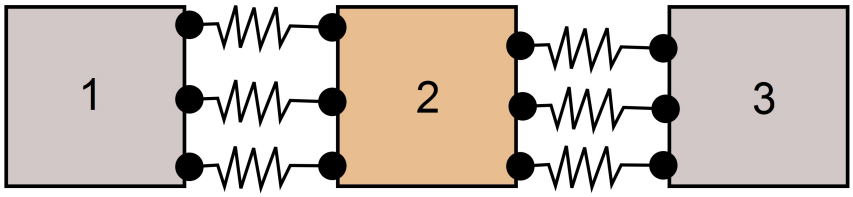

Figure 5.4: Example of coupled objects using springs. Objects (1) and (3) represent rigid bodies, while object (2) represents a deformable body.

$-k(\mathbf{p}_a - \mathbf{p}_b)$, and the force acting on object a is $\mathbf{F}_{\mathbf{q}_a} = \mathbf{J}_a^T \mathbf{F}_{\mathbf{p}_a}$. Off-diagonal terms in $\mathbf{A}$ are due to non-zero derivatives of the form $\frac{\partial \mathbf{F}_{\mathbf{q}_a}}{\partial \mathbf{q}_b} = k\mathbf{J}_a^T \frac{\partial \mathbf{F}_{\mathbf{p}_b}}{\partial \mathbf{q}_b}$. These derivatives can be efficiently computed in a unified manner for arbitrary couplings making use of our point entities defined in Section 5.1.2.

The solution to the velocity update can no longer be executed by doing an independent CG solve for each object. A naïve approach would then compute one global CG solve for the complete system, but we optimize this by identifying sets of objects that are coupled to form a *coupling island*. Two contact objects a and b belong to the same coupling island if and only if they share at least one coupling. Then, an independent CG solve can be executed for each coupling island.

Assuming that couplings are not dynamically created or eliminated, we define coupling islands as a preprocess. In the modeling stage described in Section 5.2, we first define the vector of contact objects, and then a vector of coupling entities. Once all structures and interactions are defined, we initialize coupling islands with individual contact objects, and we grow these coupling islands by traversing the vector of coupling entities.

At runtime, we need to assemble the system $(\mathbf{A}, \mathbf{b})$ of each coupling island prior to the CG solve. In order to do this, we first assemble the system matrices and right-hand-sides of the individual contact objects and coupling entities, and then we merge them.

5.3.3 Contact Islands

In order to handle contact efficiently yet robustly, we follow the constraint-based formulation in [OTSG09]. Given a pair of contact points $\mathbf{p}_a$ and $\mathbf{p}_b$, we define a non-penetration constraint as an algebraic inequality $g(\mathbf{p_a}, \mathbf{p_b}) = \mathbf{n}^T (\mathbf{p_a} - \mathbf{p_b}) \geq 0$, where $\mathbf{n}$ is the contact normal.

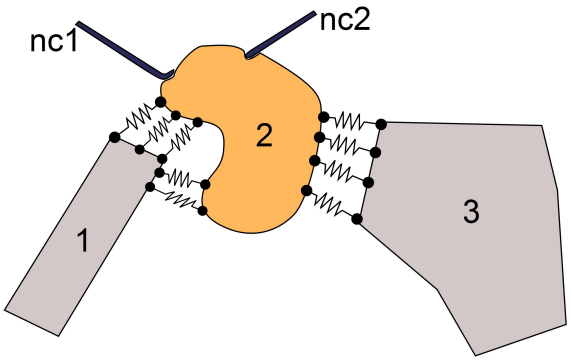

Figure 5.5: A contact island composed of two individual contact objects, i.e., the tools ($nc1$) and ($nc2$), and one coupling island, formed by contact objects (1), (2) and (3).

Constraints are then formulated semi-implicitly and transformed into velocity constraints like $\mathbf{J_a v_a} + \mathbf{J_b v_b} \geq \mathbf{c}$, where the Jacobians are computed using the general point entities described in section Section 5.1.2. These velocity constraints, together with Signorini's contact condition [DDKA06], are added to Eq. (5.2), leading to a Mixed Linear Complementarity Problem (MLCP), where contact forces are expressed as $\mathbf{J}^T\lambda$. The full MLCP that defines the constrained velocities can be expressed as:

$$\begin{aligned} &\mathbf{Av} = \mathbf{J}^T\mathbf{v} + \mathbf{b}, \\ &\mathbf{0} \leq \lambda \perp \mathbf{Jv} \geq \mathbf{c}. \end{aligned} \tag{5.5}$$

We solve this MLCP using an iterative scheme composed of two nested loops, as explained in [OTSG09]. First, a Jacobi iteration over the velocities defines the outer loop and decomposes matrix $\mathbf{A}$ into its diagonal and lower and upper triangular parts, $\mathbf{A} = \mathbf{D_A} - \mathbf{L_A} - \mathbf{U_A}$. With this decomposition, the iterative MLCP is transformed into its corresponding LCP:

$$\begin{aligned} &\mathbf{0} \leq \lambda \perp \mathbf{B}\lambda \geq \mathbf{d}, \quad \text{with} \\ &\mathbf{B} = \mathbf{J}{\mathbf{D_A}}^{-1}\mathbf{J}^T, \\ &\mathbf{d} = \mathbf{c} - \mathbf{J}{\mathbf{D_A}}^{-1}\left(\mathbf{b} + \left(\mathbf{L_A} + \mathbf{U_A}\right)\mathbf{v}\right). \end{aligned} \tag{5.6}$$

Then, the LCP is solved to compute λ using the Projected Gauss-Seidel (PGS) method (which represents the inner loop), and the current iteration of the constrained velocity is obtained.

Instead of solving one large MLCP for the complete scene, we identify *contact islands* and formulate and solve one MLCP for each contact island. Two coupling islands belong to the

same contact island if and only if they share at least one contact constraint. For the example in Fig. 5.5, where a single contact island is composed of two individual contact objects and one coupling island, the system matrix can be written as

$$\mathbf{A} = \begin{pmatrix} \mathbf{A_c} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A_{nc1}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{A_{nc2}} \end{pmatrix}$$

with the submatrix for the coupling island expressed as

$$\mathbf{A_c} = \begin{pmatrix} \mathbf{A_{11}} & \mathbf{A_{12}} & \mathbf{0} \\ \mathbf{A_{21}} & \mathbf{A_{22}} & \mathbf{A_{23}} \\ \mathbf{0} & \mathbf{A_{32}} & \mathbf{A_{33}} \end{pmatrix}$$

It is important to note that the off-diagonal terms of matrix **A** do not have much impact on the efficiency of the constrained solve, as opposed to the unconstrained solve discussed earlier. The reason is that the lower and upper triangular parts of **A** affect only the right-hand side of the LCP, **d**, as shown in Eq. (5.6), and are not visited during the iterations of PGS.

In contrast to the unconstrained update, the system matrix for each contact island is not assembled explicitly. Instead, an object-oriented processing is followed, accessing each object's data when required by the iterative solver. Based on an implementation of contact islands that deals with contact objects directly, we have extended it to seamlessly deal with coupling islands. From a software engineering perspective, we achieve this through abstraction, by ensuring that coupling islands share the same interface as contact objects.

5.4 Results

We have executed our experiments on a quad-core 2.4 GHz PC with 3 GB of memory (although we have only used two cores, for the visual and haptic loops) and a GeForce 8800 GTS. For haptic rendering, we have used the method by [GO09]. All the modeling tasks and the cage-based deformation have been executed using Blender 3D, while the real-time renderings have been performed with OGRE.

Fig. 5.6 shows images of the two applications where we have tested our interactive shoulder simulator, arthroscopy and physiotherapy palpation. Our arthroscopy example does not include

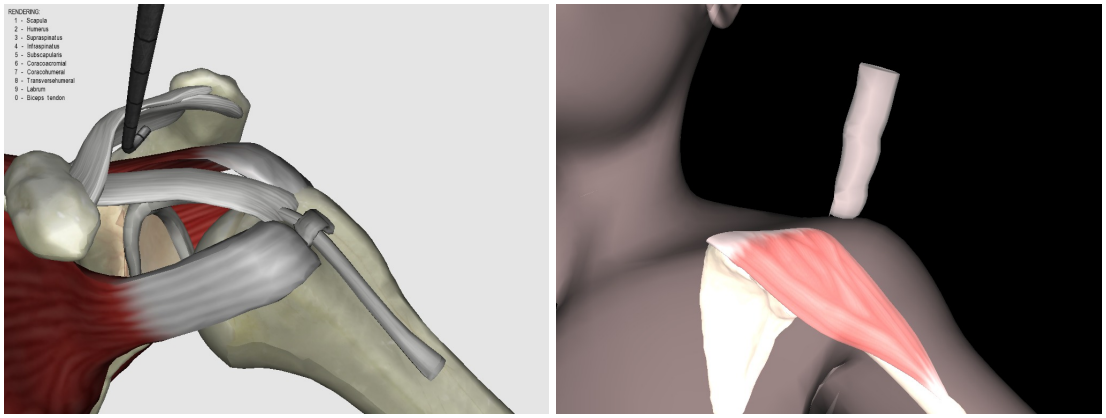

Figure 5.6: Interactive simulations in virtual arthroscopy (left) and physiotherapy palpation (right).

a realistic portal-based interaction [BGMFA06], but this was not the purpose of our work. In the palpation application, we simulate two finger models, one touching external geometry and another one the internal anatomy. The finger models are linked through a spring that models skin stiffness.

In the arthroscopy example, the scene has 50 contacts at rest-state. During some sample haptic interactions that we performed, the average number of contacts was 65, and it reached a maximum of 153. The complete simulation runs at an average of 50 fps. Note that we ensure a 1 kHz haptic update rate thanks to a multi-rate haptic rendering approach.

5.5 Discussion and Future Work

The main conclusion that can be extracted from our results is that our simulation methodology is successful in achieving interactive simulation of complex human joints for medical applications. Key to this success are the optimization of representations, and the efficient handling of couplings and contacts in the constraint-based dynamics solver.

The approximations that we carry out have some implications on the accuracy of the simulation. For example, the couplings between the various parts are not fully anatomically correct, and sometimes we partially eliminate the possibility of muscles to slide on top of bones. In order to fully simulate the shoulder anatomy, we would need to incorporate the bursa, but this structure produces even more intensive contact situations that would be difficult to handle interactively. The fact that our contact handling is based on iterative solvers prevents us from

guaranteeing a certain minimum frame rate, although we did not see this to be a problem in practice.

Besides addressing the limitations of our approximations, there are many possible avenues for future work. We are working together with both arthroscopy and physiotherapy experts to assess the quality of our models and guide further developments. In arthroscopy, many medically interesting interactions are related to tissue cutting and to suture. Therefore, a fully fleshed arthroscopy simulation would require interactive simulation of topological changes and contact with thread models.

In the next chapter, we present a model creation pipeline more general than the one shown in subsection 5.2.2. This new pipeline has tools that enable developers to create complete VR applications, capable of being executed in distributed multi-display environments.

Chapter 6

BlenderCAVE: Easy VR Authoring for Multi-Screen Displays

In the previous chapter, we presented a VR interactive simulator applied to medical purposes. One of the highlights of this simulator is the model creation pipeline that we used to configure each virtual organ. Thanks to this pipeline was possible to create three representations for each body (meshes for collision detection, simulation and rendering) and to deform them in order to become collision-free at the beginning of the simulation. In this chapter, another asset creation pipeline is presented, this was already introduced in the chapter 1 (section 1.3). This pipeline is focused at the creation of VR applications for multi-display environments.

Today, virtual reality (VR) is present in many and very diverse application domains, some of them are illustrated in figure 1.1. VR developers are expected to have expertise not only in computer graphics (CG), but also in the problem domain and in the development of sophisticated software systems. Many of them are explained in chapter 2, section 2.3. Such systems may require handling multiple processes, message passing, dynamic memory management, and a variety of process synchronization techniques. But, due to the hardware aspects of VR, developers may also be concerned with low-level issues such as device drivers for particular I/O devices or techniques to generate multiple stereoscopic views. In addition, VR applications involve an artistic component, specially in order to improve the realism and immersion of the virtual worlds generated. Therefore, they also need the contribution of artists who model, texture and animate virtual characters and objects. To ease the job of VR developers, it is desirable to integrate content creation tools, CG engines, and hardware abstraction layers that make the

development independent of specific I/O devices, and also allow preliminary testing on devices different than the ones of the final product.

As shown in section 2.3 of chapter 2, the needs of VR developers have already been answered to a large extent by existing tools. However, developers still miss tools that close the complete creation pipeline, allowing an automatic deployment of an application designed on a sandbox onto the final VR hardware.

Figure 6.1: Two students play a collaborative game on a CAVE. The video game (both art and logic) was quickly created using the visual editing tools of Blender. Then, our easy-to-integrate BlenderCAVE framework manages a distributed rendering architecture based on the Blender Game Engine that generates the video output for all screens in the CAVE. The stereoscopic response was deactivated in order to capture this figure.

This work introduces a framework, which we refer to as BlenderCAVE, for the easy development of multi-screen VR applications on a virtual sandbox. After an evaluation of existing content editing tools and CG render engines (detailed in Section 6.1), we have opted for the Blender Game Engine (BGE) [Blea] as the base engine for our framework. Then, the framework includes two major components. First, a simple virtual camera setup, described in Section 6.2, defines the content to be output on each screen. Second, as described in Section 6.3, a distributed architecture and a lightweight communication protocol manage the application state and synchronize the output on the various screens.

We demonstrate BlenderCAVE through several applications displayed on an immersive CAVE projection system [CNSD93]. The highlight of the applications is a multi-player videogame

(shown in Fig. 6.1), which was quick-to-design and easily ported from a desktop environment to the CAVE.

6.1 Requirements and Selection of the Base Engine

Our approach to the design of a framework for easy development of multi-screen VR applications has been to augment one of the many existing high-quality CG render engines with functionalities to efficiently support multi-screen projection. In this section, we discuss the desired features of the base engine, we compare several high-quality engines, and we present the particular features that steered our decision toward Blender Game Engine (BGE).

From the CG point-of-view, the engine should support state-of-the-art rendering and animation features: programmable shaders, animations based on bones, simulation of rigid and deformable bodies, generation of stereoscopic video output, and extensibility through plug-ins and scripts.

On top of these features, it is desirable if the render engine includes an integrated 3D WYSIWYG scene compositor, i.e., a sandbox. We look for a solution that allows modeling, texturing and animating directly the objects that will be included in the VR application.

Other desirable features include multi-platform availability, as well as the possibility to execute the VR application on distributed heterogeneous systems. Source code access enables the possibility to implement new capabilities when necessary, and a well-stablished community of developers and artists is a good indication of further evolution of the engine, thus favoring a longer life-cycle of the VR application.

	Unreal 3	CryEngine 2	EON Studio	Ogre3D 1.6.5	BGE 2.49
Stereoscopy	yes	no	yes	experimental	yes
Multi Screen	no	no	yes	no	tiled window
Shaders	Cg	Cg	no	Cg / GLSL	GLSL
Animation Support	external	external	external	external	included
Physics	PhysX	PhysX	no	several	Bullet
Logic Graphs	KissMet	Lua	yes	no	Python / bricks
Scripting	KissMet	Lua	VBScript / JavaScript	no	Python
Modeller	external	external	external	external	included
Scene Compositor	UnrealEd	CryEngine Sandbox	external / graph	external / code	included
Plugins	yes	yes	yes	yes	yes
Source Code	no	no	no	yes	yes
License	proprietary	proprietary	proprietary	GPL	GPL

Table 6.1: Comparison of render engines based on their capabilities.

In our search for an engine that fulfills all or most of the desired features, we have evaluated in detail the following list of high-quality render engines: Unreal 3™ [Epi] from Epic Games Inc, CryEngine 2™ [Cryb] from Crytek GmbH, EON Studio™ [Eon] from EON Reality Inc, and the open source engines Ogre 3D (v. 1.6.5) [OGR], and BGE (v. 2.49) [Blea]. Other possible engines that we have considered are: proprietary game engines such as Unity™ [Unib], from Unity Technologies, Id Tech 5™ [Id] from Id Software, and Unigine™ [Unia] from Unigine Corp, or open source engines such as OpenSceneGraph [OSG], Crystal Space [Crya], Irrlich 3D Engine [Irr], and Id Software's Quake IdTech 3 [Id].

Our list is clearly not comprehensive, but we believe that it covers a set of highly representative engines. We chose Unreal, CryEngine, EON, Ogre and BGE for detailed evaluation for various reasons. Unreal and CryEngine are high-end engines used for top-class commercial video games, which is a good indication of their quality. EON, on the other hand, is an engine particularly oriented to multi-screen VR setups. And, finally, Ogre and BGE offer high-quality CG with the addition of open-source advantages. Moreover, BGE provides a content creation framework that could greatly ease application design. Table 6.1 compares these five engines based on CG quality features, integrated content creation and control possibilities, and further extensibility.

After the evaluation, we decided to select BGE as our base engine. The main reason is its interesting balance of high-quality render engine with integrated content creation and sandbox. Another positive feature is its ease of extensibility. It contains a large API based on Python,

and it allows the connection of specific device drivers by implementing a binding between the driver's library (as long as it is written in C/C++) and a Python class. Moreover, source code access allows the implementation of additional capabilities.

Given our target application, i.e., creation of VR applications for multi-screen displays, we also pay special attention to the capabilities of BGE in terms of stereoscopic and multiple video output. In its standard version, BGE supports five built-in stereoscopy modes (Pageflip, Syncdouble, Anaglyph, Side-by-Side and VInterlace), which can be toggled using a GUI control [MSO+09]. BGE has single-window output, but it is possible to switch a built-in mode and draw many windows on the same screen in a tiled manner, or direct each window to a different screen on a multi-screen system. At first, this feature seemed attractive for our application, but it does not scale well as the number of windows increases. As it will be described later in Section 6.3 we discarded BGE's built-in multi-screen functionality, and we designed a distributed architecture.

6.2 Virtual Camera Setup

Our BlenderCAVE framework includes two main components, a virtual camera setup that defines the output for each display, and a distributed architecture to manage the application. This section describes the virtual camera setup, including the definition of camera frustums, and a master-slave navigation approach.

6.2.1 Configuration of Camera Frustums

Given a VR scene and a target multi-screen display, we define a *Virtual Camera Cluster* (VCC) that associates one virtual camera to each screen. The correct compositing of the images on the multi-screen display requires a careful selection of parameters for each camera and a synchronized transformation of all cameras as the user navigates through the scene. Our prototype implementation is limited to planar screens, and then the configuration of each camera reduces to adjusting the values of ModelView, Projection and Viewport transformations. BGE includes five additional video output modes for dome-shaped screens [Bleb], which would allow supporting also dome-shaped multi-screen displays.

The VCC maps the geometry and topology of the projection system to the camera frustums

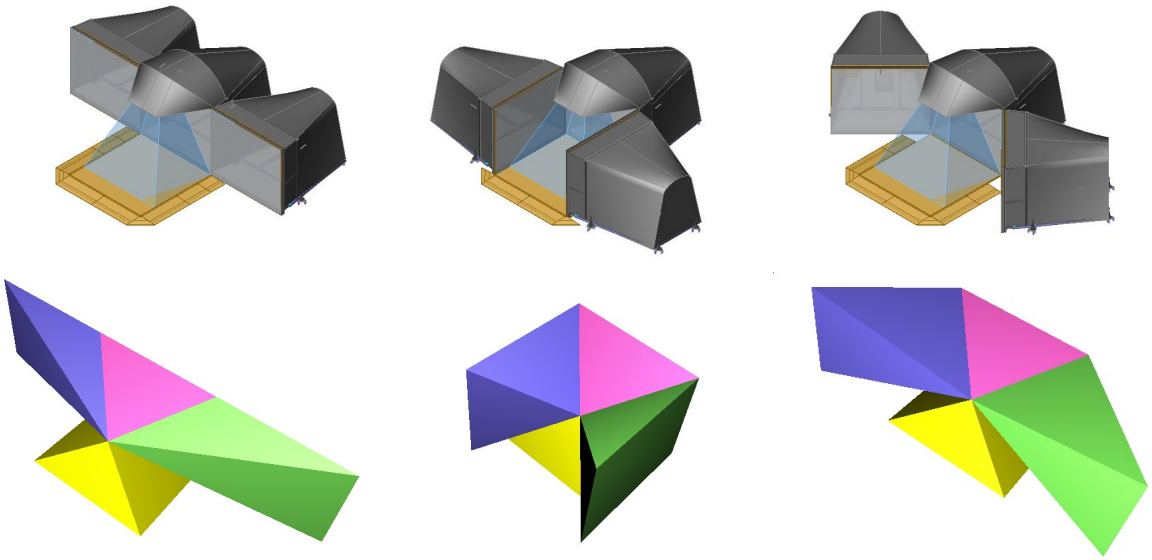

Figure 6.2: From left to right, three possible setups for our reconfigurable 4-wall CAVE: wall, cube, and amphitheatre. The top row shows the screens and the mirror-based projection system. The bottom row shows frustum setups for an observer located at the center of the CAVE. If the observer's position is tracked, all four frustums need to be dynamically reconfigured, otherwise only the right and left frustums need to adapt to the CAVE's configuration.

on BGE. All the cameras of the VCC are located at the same point, called *local origin*, which corresponds to the position of the observer in the VR scene. If the VR installation includes a tracker of the observer, its position is mapped to the local origin. The four corners of the near planes of the various cameras are configured (up to a scale factor) with positions and orientations that respect the relative transformation between the observer and the screens in the real world. Then, the local origin and the corners of the near planes define the perspective angles of the cameras frustums. With this approach, the images captured by the various cameras correctly match at the borders of the screens, the union of the frustums covers all the visible volume in the VR scene, and the frustums do not intersect. Fig. 6.2 shows three possible configurations of the VCC for the particular type of configurable 4-wall CAVE used in our experiments.

6.2.2 Master-Slave Navigation

We assume that the physical screen setup remains invariant while the VR application is in use, therefore, the relative transformation between the various camera frustums depends only on the

local origin, i.e., the observer's position w.r.t. the physical setup. Based on this observation, we perform camera navigation in a master-slave manner, computing the transformation of a master camera based on the user's navigation, and then computing the transformation of the slave cameras relative to the master. In our experiments, we have selected the frontal camera as master camera. In BGE, the master-slave VCC is programmed as a hierarchy, represented schematically in Fig. 6.3, which includes the user, the master camera, and the slave cameras.

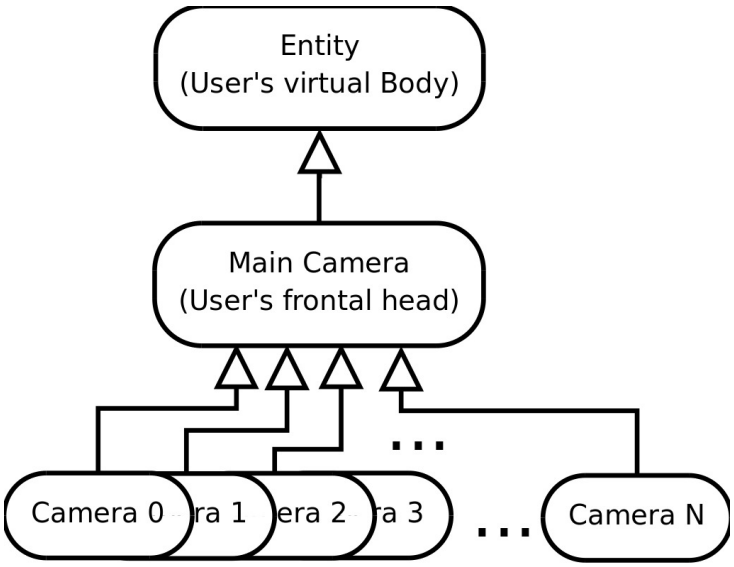

Figure 6.3: BGE model of the master-slave camera hierarchy. The master camera is connected to the user entity, and all other cameras are defined relative to the master camera.

In our prototype implementation, we use a first-person navigation mode. In the VR scene, the user is represented as an invisible human-height character contained on a simple bounding box. This bounding box constitutes the *user entity* in BGE. Each camera in the master-slave VCC has an additional entity, and they are all connected in a hierarchical manner to the user entity.

During navigation, the user moves and orients the virtual workspace through the VR scene, and these transformations are applied to the user entity. Then, the camera transformations are computed automatically based on the tracked local origin. User navigation could be controlled in various ways: using a mouse, a keyboard, a wiimote, etc. Additional controls or keys can be assigned to gear other motions of the virtual character, such as jumping, crouching, or leaning. BGE also handles reactive collisions with the environment during camera navigation, using the bounding box of the user as collision primitive.

6.3 Communication System Architecture

Given a VR application designed on BGE, the target multi-screen display system, and the VCC that defines the camera-screen correspondence, we have designed a distributed rendering architecture that controls the video output of each screen. In this section, we describe the elements that conform this architecture and their main features, paying special attention to the following issues: maintaining a consistent application state across all elements, synchronizing the video output on all screens, and responding to external inputs (i.e., managing input peripherals).

6.3.1 Master-Slave Distributed Architecture

To compute the video output for the multiple screens, we have designed a (possibly heterogeneous) distributed architecture, with one PC per screen. On each PC, we execute one instance of BGE, which computes the image as seen from one of the cameras in the VCC, and outputs it to the corresponding screen.

To synchronize the rendered output, we set a common refresh rate on all BGE instances. The refresh rate is maintained both at the application level, i.e., for logic and physics updates, and at the GPU render level. Even though our architecture supports the use of heterogeneous PCs for the various screens, in our prototype implementation the refresh rate is limited by the slowest machine. For future versions of the framework, one could consider balancing the rendering load among the various PCs in a more efficient way.

We manage the application state and handle peripheral inputs following a master-slave approach. The PC in charge of rendering the master camera in the VCC plays the role of master in our architecture, and it communicates state changes and user input to the other PCs. BGE provides simple tools to program the application logic as a state machine. In particular, it provides logic bricks that react to events, and these logic bricks act on the application state. Events may be produced by internal logic conditions or by input devices.

6.3.2 Communication Protocol

We consider two different communication modes in our master-slave architecture: normal operation and initialization. During normal operation, the master handles events produced internally by its own application logic as well as events produced by input peripherals. If an event is trig-

gered, the master communicates this event simultaneously to all slaves. When a slave receives a packet, it updates its local version of the user entity in the VCC, computes the new position and orientation of its render camera, and a script routine triggers the event and executes its corresponding logic bricks. All slaves execute the same application logic, therefore by reacting to the same event as the master, and given that they maintain the same refresh rate, all local copies of the application state remain synchronized.

In our experiments, we used as peripherals standard keyboards and mice. Then, the information to be communicated by the master consists of: the position and orientation of the user entity, the orientation of the master camera, the keys that have been pressed or released, and the current position of the mouse. This information can be coded in very short messages. In our CAVE, the PCs are connected by less-than-half-meter-long gigabyte ethernet cable, which allowed us to send messages using UDP Multicast without package loss. The combination of small messages, fast network, and little protocol overhead, produce negligible system latency, as discussed in detail in the next section. The communication protocol could be extended to handle other types of events, such as random events or large state modifications due to physically based simulations. Given the small message size and negligible latency in the current prototype, there is plenty of room for additional communication payload.

The initialization mode is executed when a new slave PC joins the system. In this situation, the slave PC sends a request for a full-state update to the master, and the master responds with a standard UDP message containing the full state information. In addition to camera settings and input events, the full state may contain information such as the configurations of all moving objects in the scene, clip and frame numbers for animated characters, internal attributes and flags, etc. Moreover, at initialization, a slave needs to identify the particular camera in the VCC that it should render. We solve this issue by assigning to each slave camera in the VCC an *id* corresponding to the network address of the slave PC in charge. Then, a slave can discriminate its camera information simply by comparing the camera *id* with its own network address.

In our experiments, discussed in detail in the next section, the application did not suffer synchronization issues. In applications with a complex logic, however, it might be convenient to execute periodic full-state synchronizations.

The communication protocol is executed by BGE as a Python script. This guarantees a complete transparency of the communications across platforms, and enables the use of het-

erogeneous machines, with different operating systems. In our tests we have combined nodes running Microsoft Windows and Ubuntu Linux with no problems. All the packages are coded using a Python dictionary format, and we use the cPickle library to serialize Python objects to plain text and viceversa.

6.4 Implementation and Experiments

In this section, we describe first our CAVE-like installation and other hardware details. Then, we discuss the process for setting up a VR application using BlenderCAVE. Finally, we discuss the test applications we have implemented, as well as performance results.

6.4.1 Our Visualization System

The visualization system used to test BlenderCAVE is a RAVE II (Reconfigurable Advanced Visualization Environment), a CAVE-alike system with four screens designed by Fakespace Inc. The main difference with a conventional CAVE system is that the side screens of the RAVE can be reoriented to create different configurations of the immersive space (see Fig. 6.2). Each display module has a screen of dimensions 3.75 x 3.12 meters.

The displays use active stereo projectors and CrystalEyes shutter stereoscopic glasses. The projectors are driven by a cluster of 4 PCs with NVIDIA Quadro FX 4500 graphics cards, and connected through a Gigabyte Ethernet network. The PCs also carry GSync hardware to synchronize the pageflip on all the graphic cards. The cluster can execute Windows XP or Ubuntu Linux, both of which have been used on BlenderCAVE tests.

6.4.2 Setting up and Running BlenderCAVE

BlenderCAVE is programmed as a set of scripts that control the VR application logic on BGE. Given a certain VR application on BGE and a cluster of PCs that send video output to a multi-screen display, setting up BlenderCAVE to drive the multi-screen display is an extremely easy task. First, one needs to include the VCC hierarchical entity on every instance of the VR application. The BlenderCAVE scripts are associated to the VCC, hence they are automatically included. Note that the camera settings of the VCC should be adjusted to match the specific

Figure 6.4: Mountain scene (15727 triangles) displayed on a CAVE. The scene is rendered with shadow mapping, multi-texturing (9 layers), 4 normal mapping passes, and screen-space ambient occlusion.

display, as described in Section 6.2.1. If a display installation is permanent, then the VCC may be defined only once and imported in multiple applications. Setting up BlenderCAVE to drive our CAVE system takes less than two minutes once the VCC is defined.

Additionally, one needs to set the camera *ids* for the various PCs, as described in Section 6.3.2, and adjust the basic rendering settings of BGE (i.e., fullscreen rendering and activation of the pageflip option). In our examples, we used a 1024×768 resolution for each screen and a refresh rate of 100Hz, but higher resolutions and refresh rates are supported.

6.4.3 Test Applications

We have tested BlenderCAVE on three different VR applications. Two of these applications, the mountain scene in Fig. 6.4 and the shark scene in Fig. 6.5, intend to demonstrate the easiness to create applications with high-quality graphics and render them on a CAVE. The third application, the game in Fig. 6.1, gives a glimpse of the great possibilities for CAVE-oriented application development. In all the images shown in this chapter, stereo output was disabled, but the system runs in full stereo mode.

The mountain scene in Fig. 6.4 is composed of 15727 triangles and shows dynamic shadow

computation using shadow maps as the sun rises and sets. We used a high-res shadow buffer of 2048×2048 pixels. More interestingly, the mountains are render using multi-layer texturing, with 9 texture layers, 4 simultaneous normal mapping passes, and an additional pass of screen-space ambient occlusion.

The sharks in Fig. 6.5 are composed of 7502 and 5197 triangles each, and are animated using bones and a skinning technique. Both sharks are rendered using a diffuse texture and a pseudo environment map.

Fig. 6.5 also demonstrates the effectiveness of our VCC camera setup. Notice how the images of the sharks are projected onto the seams and corners of the CAVE, and there is barely any noticeable distortion. In this example, the location of the physical camera is being used as local origin for the VCC.

Our last test scene is a collaborative videogame, shown in Fig. 6.1, where two users fight against a group of zombie skeletons. The floor is rendered using normal mapping, and the skeletons (8609 triangles each) are animated using bones and predefined animation clips. The application maintains the target frame rate (50Hz, stereo) with up to 15 skeletons, for a total of 135553 triangles in the scene.

The videogame was initially designed as a single-player game using the Blender content creation tool, with logic bricks and Python scripting. The major result proved with this scene was that BlenderCAVE allowed extremely simple adaptation of the videogame to a multi-screen display, i.e., our CAVE. Porting the videogame to BlenderCAVE required the definition of the VCC (approximately 1 hour, including tests, but this needs to be done only once if the configuration of the CAVE is static), adding the BlenderCAVE scripts to the application (done in just 1 minute), and, of course, installing the application in all PCs in the architecture.

We have also measured the communication latency and bandwidth in our system. Thanks to the use of UDP Multicast and our event-driven protocol, the network traffic is very low. We measured the total traffic from the master to all slaves in situations with frequent camera motion and button-click events, and we reached peak packet sizes of just 12kB. To measure network latency, we timed the round trip of a packet between the master and a slave, which peaked at just 16μs. As a conclusion, with our Gigabyte Ethernet network, communication latency is not an issue.

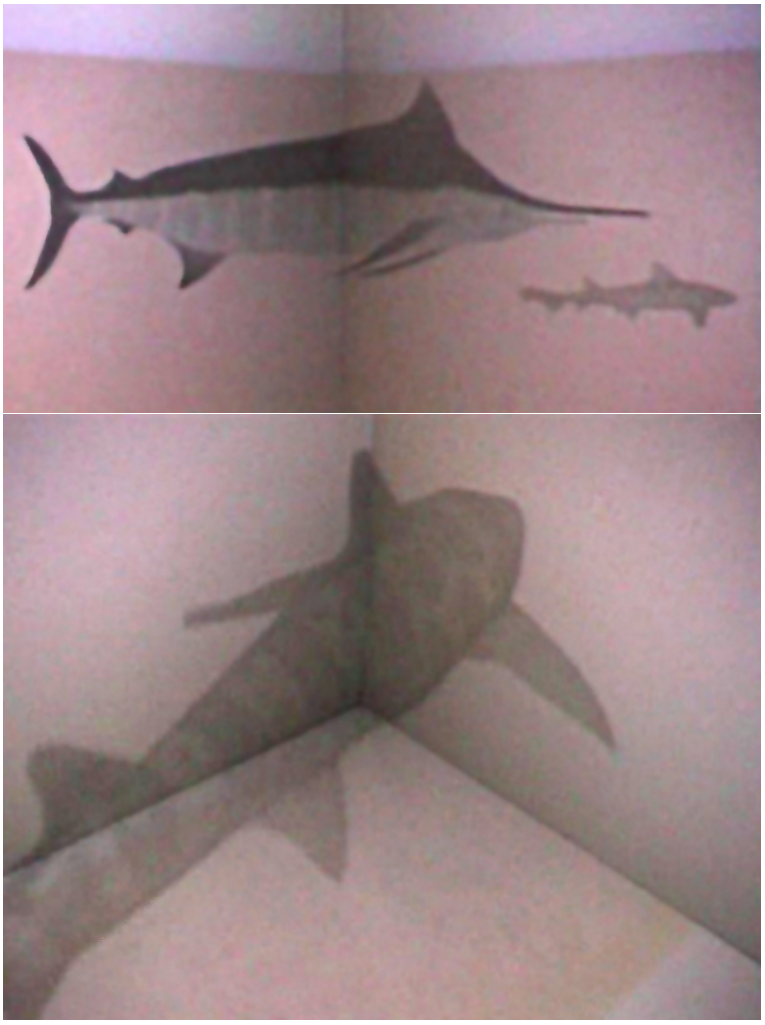

Figure 6.5: Shark scene displayed on a CAVE. Notice the lack of distortion as the images of the sharks are projected onto the seams and corners of the CAVE, demonstrating the effectiveness of the VCC setup.

6.5 Discussion

In the past, the creation of a multi-user VR application for a CAVE entailed the integration of I/O peripherals in the render engine, setting up and coordinating multiple instances of the render engine to drive all screens of the display, and importing art content in the rendering application. This chapter shows that the BlenderCAVE framework allows a much simpler development of complex and interesting VR applications for a CAVE-like display. Using BGE as base render engine, and taking advantage of Blender's content creation possibilities, BlenderCAVE augments the engine to easily direct the video output to a multi-screen display.

There are, however, multiple directions in which the features of BlenderCAVE could be improved or extended. For more general VR applications, it will be necessary to provide support to many I/O peripherals. This can be done by integrating one of the existing VR libraries for hardware abstraction, possibly through Python-based extensions.

As discussed in this chapter, the current communication protocol is particularly efficient for event-driven state changes, but applications with a complex state, such as physically based simulations, may require modifications. Under a complex state, there is a trade-off between distribution of state computations, communication of state updates, and network bandwidth. The current framework is also limited in terms of the tight connection between BGE instances and output screens. Currently, each screen is driven by a different BGE instance, running on a different PC. For tiled displays, it might be convenient to distribute rendering load differently, perhaps with the same machine driving several displays. The critical factor should be the minimization of the computing resources, subject to fulfilling the desired refresh rate, which makes the problem application-dependent.

Since BlenderCave was released as open-source, other research teams have been interested in it and they have contributed with new functionality. For example, the Computer Science Laboratory for Mechanics and Engineering Sciences (LIMSI-CNRS), in France, has redesigned BlenderCave refactoring its API, making it easier to use. In addition, support for different screen configurations has been added [Lim], and some other works have been published [PQTK13b, PQTK13a, PQTK12] extending the functionality of BlenderCave. Finally, other middlewares, like CaveUDK [LCC+12], have been designed taking BlenderCave as inspiration, among others [AMQ+12]. In near future, we expect more people be interested in BlenderCave as though more support for newer peripherals, and more state-of-the-art capabilities, are added to the source code.

BlenderCave source code, templates and examples can be downloaded from:

http://blendercave.limsi.fr/doku.php

(Old site: http://www.gmrv.es/BlenderCave/index.html)

In the next chapter, we present the overall conclusions and the future works of our entire dissertation, as though, we describe the limitations and the future lines of research of all the work presented in previous chapters.

Chapter 7

Conclusion

As seen in previous chapters, developing VR applications is one of the most demanding tasks known in the industry of software. Not only an application is required that can manage complex algorithms for simulation, animation and rendering of virtual bodies, but also it has to deal with the assets and their accompanying data structures (collision detection, physics and visual meshes, textures, rigging information, animation poses and much more). In addition, a VR application has to operate efficiently with I/O peripherals in order to process user input and to produce visual and/or haptic response through them. The response has to be generated as fast as possible and at interactive rates, otherwise, the sensation of immersion could be lost. In order to accomplish these requirements, it is required the employment of multidisciplinary teams of highly-skilled professionals, and the aid of specialized tools.

In this thesis we propose state-of-the-art tools for authoring VR applications. These tools are designed by inspiration of modern methodologies employed in the video game development industry. They employ visual tools and automatized test that allow to save huge amounts of time and resources. In addition, we have implemented a network synchronization protocol that allows applications to be executed in distributed multi-display systems.

Besides, our dissertation offers innovations that enhance realism in physics-based simulations. Specifically, we have focused on the areas of object deformation and in contact handling. The designed methods improve the behavior of each existing object in the simulation, specially if those are deformable and belong to biological forms. Regarding deformation of bodies, one of our innovations consist in a parallelized algorithm to deform dense volumetric objects interactively. This method takes advantage of graphics hardware, being capable of deforming

millions of voxels at interactive rates. In the same area, we have worked in the simulation of couplings. This kind of deformation happens when several objects with different properties are joint together. For instance, in human anatomy, where different organs (bones, ligaments, muscles) are attached together.

Finally, regarding the stage of contact handling, we have implemented a novel algorithm for simulating adhesive contact. This kind of contact is very common in nature, specially in a wide range of objects, like wet surfaces, biological mucous membranes and others. This method is quite robust ans stable, due to its implementation using contact constraints, and it actually enhances the simulations of contact noticeably.

In this chapter we summarize the main results of this thesis, the limitations of our approaches and possible future research directions for overcoming these limitations.

7.1 Summary of Results

Our dissertation offers several solutions to develop complex VR applications and to enhance physic-based simulations. Regarding the development of VR applications, these solutions lie in a set of high-level visual tools that are integrated into a first-class authoring environment. All of these tools have been employed thanks to the achieved experience in the industry of video game development, where it is common to employ a kind of visual tools called *sandboxes*. This type of tools allow the creation and configuration of the virtual objects that will participate in the simulation. In addition, the development environment enables quick tests aimed to verify the correctness of the configurations of all the objects in the scene.

In the past, the creation of a multi-user VR application for a CAVE entailed the integration of I/O peripherals in the render engine, setting up and coordinating multiple instances of the render engine to drive all screens of the display, and importing art content in the rendering application. Our dissertation shows that BlenderCAVE framework allows a much simpler development of complex and interesting VR applications for a CAVE-like display. Using Blender Game Engine (BGE) as base render engine, and taking advantage of Blender's content creation possibilities, BlenderCAVE augments the engine to easily direct the video output to a multi-screen display.

In addition, other innovations this thesis includes are applied to the field of the physics simulators. As known, a physics simulator is composed of several processing stages. Our disserta-

tion incorporates new algorithms applied to some of these stages. Specifically, our contributions address the deformation and contact handling stages. For deforming virtual objects we have developed a novel algorithm to efficiently deform dense volumetric objects interactively, with the assistance of graphic hardware. The deformation is performed by generating a new deformed volume via the rasterization of an embedding tetrahedral mesh. The key for its high performance is the massive parallelization applied to the level of individual target voxel, and with very simple operations and practically no divergence. To further accelerate rasterization, we apply efficient multi-core CPU culling as a first step. This technology enables the deformation and manipulation of complex biological structures, like the ones employed in medicine.

Moreover, we have achieved interactive simulations of complex human joints for medical applications. The keys to the success in the deformation of groups of virtual objects that are joint together, like the organs that lie inside human anatomy, are the optimization of representations, and the efficient handling of couplings and contacts in the constraint-based dynamics solver. Each organ might have diverse properties, ranging from hard bone to soft fat tissue, and intricate contact situations could happen. This technique formulates all the couplings in an unified manner and solves all the affected organs as a group.

Regarding the contact handling stage, we have shown a model for adhesive contact that can be efficiently integrated into existing constraint-based contact solvers. Adhesion phenomenon is very common in nature, as shown in figure 1.5. Our method retains the robustness of constraint-based contact while allowing for rich and versatile adhesion effects under a diverse range of object types. It actually enriches the simulation of contact and it is also suitable for medical VR applications.

The techniques we have presented in this dissertation have focused on efficiently obtaining plausible solutions for deforming volumetric objects, simulating the deformation of complex anatomy and the simulation of adhesive contact. Another important contribution in this thesis is a complete framework of techniques to accelerate the development of all kind of VR applications, even those used in distributed, multi-display setups. In the next section we describe some possible extensions to our works.

7.2 Future Work

The results we have presented in this dissertation show that we are heading towards better tools and frameworks for creating VR applications. In addition, new contributions to the physics simulation pipeline enriches the behavior of virtual objects. The limitations of the approaches originate at the very core of the goals we want to achieve. For example, the addition of new types of objects, like volumetric deformable bodies, and the addition of new simulation techniques, like coupling and adhesion contact, add a noticeable workload to the simulation, making it hard to keep interactive rates. On the other hand, the most important limitation of our algorithms is that they are fast approximations of physically correct simulations. However, we believe that the approaches described in this dissertation form a base-line for further research towards more realistic solutions.

7.2.1 Limitations of the Current Techniques

In previous chapters, we have already discussed several limitations of the techniques we have developed (See Sections 3.4, 4.5, 5.5 and 6.5). Here we list those limitations again:

- Our algorithm to deform volumetric bodies suffers some limitations, such as the existence of false positives during culling. However, these false positives do not hurt performance significantly. Another limitation is the smoothing introduced by trilinear interpolation of input data. More costly filtering approaches would produce higher quality results.

- Regarding our adhesion algorithm, the adhesion model incorporates a thermodynamics formulation of debonding from the mechanics literature. Connected to this feature, one limitation in our work is that the formulation of bonding and the connection between friction and adhesion are not sustained by a comparable thermodynamics approach.

- In the case of friction in adhesive contact, we obtained plausible results by selecting the most restrictive constraint out of Coulomb friction and tangential adhesion, as discussed in Section 4.3.2.

- Another limitation of our adhesion algorithm is that it requires contact tracking, not present in some of the available rigid body dynamics simulators, in order to evolve the value of the adhesion intensity across frames.

- Although this was not a major problem in our examples, relaxation solvers, such as Gauss-Seidel, may suffer from slow convergence at times. This is a general limitation in constraint-based contact formulations, and more efficient LCP solvers are still an issue under investigation.

- About coupling simulation in our demo of the human shoulder, the approximations that we carry out have some implications on the accuracy of the simulation. For example, the couplings between the various parts are not fully anatomically correct, and sometimes we partially eliminate the possibility of muscles to slide on top of bones. In order to fully simulate the shoulder anatomy, we would need to incorporate the bursa, but this structure produces even more intensive contact situations that would be difficult to handle interactively. The fact that our contact handling is based on iterative solvers prevents us from guaranteeing a certain minimum frame rate, although we did not see this to be a problem in practice.

- Finally, regarding BlenderCave, as discussed in the chapter 6, the current communication protocol is particularly efficient for event-driven state changes, but applications with a complex state, such as physically based simulations, may require modifications. Under a complex state, there is a trade-off between distribution of state computations, communication of state updates, and network bandwidth. The current framework is also limited in terms of the tight connection between BGE instances and output screens. Currently, each screen is driven by a different BGE instance, running on a different PC. For tiled displays, it might be convenient to distribute rendering load differently, perhaps with the same machine driving several displays. The critical factor should be the minimization of the computing resources, subject to fulfilling the desired refresh rate, which makes the problem application-dependent.

One limitation we have not included in the list is the initial assumption of working with 2-manifold triangle meshes in the models that are represented in this manner. The adaptation of our approaches for non-manifold meshes is not trivial, and falls out of the scope of this thesis.

7.2.2 Applications and Further Analysis

In the case of our volumetric deformation technique, from an applied point of view, our method allows interactive editing, manipulation, and deformation of dense volume data. It would be interesting to handle other types of mesh elements and basis functions, such as trilinear interpolation in hexahedra. This extension would require modifications to the mapping function and the culling algorithm.

Regarding our method to simulate adhesive contact, it currently handles only well-defined interfaces between rigid and deformable bodies. Therefore, another interesting extension to our work would be to integrate it with other materials, such as viscoplastic ones, for which adhesion produces very interesting effects.

In the matter of our anatomy palpation demo, there are many possible avenues for future work. Following a customer-driven development, we are working together with both arthroscopy and physiotherapy experts to assess the quality of our models and guide further developments. In arthroscopy, many medically interesting interactions are related to tissue cutting and to suture. Therefore, a fully fleshed arthroscopy simulation would require interactive simulation of topological changes and contact with thread models.

Regarding BlenderCave, our framework for authoring VR applications, it is currently actively being developed. Since it was released as open-source, other research teams have been interested in it and have contributed with new functionality. Some teams have adapted BlenderCave to their CAVE topologies and to other I/O devices. In near future we expect multiple directions in which the features of BlenderCAVE could be improved or extended. For more general VR applications, it would be necessary to provide support to newer I/O peripherals. This could be done by integrating existing VR libraries for hardware abstraction, possibly through Python-based extensions or introducing modifications in the source code of BGE. Last, BlenderCAVE could be extended with features that would increase the rendering quality on the CAVE. Such features include color and brightness correction for seamless image continuity across the screens. Fortunately, BlenderCave shares its code base with Blender 3D, therefore, our framework directly benefits of each new contribution applied to improve Blender 3D. In near future we expect to have several cutting-edge capabilities inside the core of our framework.

Developing VR applications that simulate reality is an incredibly complex problem, there are several unresolved challenges and we are still quite far from achieving animations indistin-

guishable from reality, assuming if this is even possible.

Bibliography

[AFC+10] Jérémie Allard, François Faure, Hadrien Courtecuisse, Florent Falipou, Christian Duriez, and Paul G. Kry. Volume contact constraints at arbitrary resolution. *ACM Transactions on Graphics*, 29(3), 2010.

[AMA05] Tomas Akenine-Möller and Timo Aila. Conservative and tiled rasterization using a modified triangle setup. *Journal of Graphics Tools*, 10(3):1–8, 2005.

[AMQ+12] Y Alvarado, M Moyano, David Quiroga, Jacqueline Fernández, and Roberto A Guerrero. A virtual reality computing platform for real time 3d visualization. In *XVIII Congreso Argentino de Ciencias de la Computación*, 2012.

[Ant09] Christoph Anthes. *A Collaborative Interaction Framework for Networked Virtual Environments*. PhD thesis, Institute of Graphics and Parallel Processing at JKU Linz, Austria, Institute of Graphics and Parallel Processing at JKU Linz, Austria, August 2009.

[AO11] Iván Alduán and Miguel A Otaduy. Sph granular flow with friction and cohesion. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics symposium on computer animation*, pages 25–32. ACM, 2011.

[AP97] M. Anitescu and F. A. Potra. Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *Journal Nonlinear Dynamics*, 14(3):231–247, 1997.

[Bar91] David Baraff. Coping with friction for non-penetrating rigid body simulation. In *Computer Graphics (Proceedings of SIGGRAPH 91)*, pages 31–40, July 1991.

[Bar94] D. Baraff. Fast contact force computation for nonpenetrating rigid bodies. *Proc. of ACM SIGGRAPH*, 1994.

[Bar96] David Baraff. Linear-time dynamics using Lagrange multipliers. *Proc. of ACM SIGGRAPH*, 1996.

[BFA02] Robert Bridson, Ronald Fedkiw, and John Anderson. Robust treatment of collisions, contact and friction for cloth animation. *Proc. of ACM SIGGRAPH*, 2002.

[BGMFA06] S. Bayona, M. García, C. Mendoza, and J.M. Fernández-Arroyo. Shoulder arthroscopy training system with force feedback. *Proceedings of Medical Information Visualisation*, pages 71–76, 2006.

[BJ07a] Jernej Barbič and Doug James. Time-critical distributed contact for 6-DoF haptic rendering of adaptively sampled reduced deformable models. In *2007 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 171–180, August 2007.

[BJ07b] J. Barbič and D. L. James. Time-critical distributed contact for 6-dof haptic rendering of adaptively sampled reduced deformable models. *Proc. of ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2007.

[BJ08] Jernej Barbič and Doug L. James. Six-DoF haptic rendering of contact between geometrically complex reduced deformable models. *IEEE Transactions on Haptics*, 1(1):39–52, 2008.

[BJH+01] A. Bierbaum, C. Just, P. Hartling, K. Meinert, A. Baker, and C. Cruz-Neira. Vr juggler: a virtual platform for virtual reality application development. In *Virtual Reality, 2001. Proceedings. IEEE*, pages 89 –96, 2001.

[Blea] Blender. Blender Game Engine Features website. `http://www.blender.org/education-help/tutorials/game-engine/`.

[Bleb] Blender. Dome mode in blender game engine. `http://wiki.blender.org/index.php/Dev:Source/GameEngine/2.49/Fisheye_Dome_Camera`.

[BMF03] R. Bridson, S. Marino, and R. Fedkiw. Simulation of clothing with folds and wrinkles. In *2003 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 28–36, August 2003.

[BSHW07] C. Basdogan, M. Sedef, M. Harders, and S. Wesarg. Vr-based simulators for training in minimally invasive surgery. *IEEE Comput. Graph. Appl.*, 27(2):54–66, 2007.

[BUL] BULLET. Bullet Physics Library. `http://bulletphysics.org/`.

[BW92] D. Baraff and A. P. Witkin. Dynamic simulation of non-penetrating flexible bodies. *Proc. of ACM SIGGRAPH*, 1992.

[BW98a] D. Baraff and A. P. Witkin. Large steps in cloth simulation. *Proc. of ACM SIGGRAPH*, 1998.

[BW98b] David Baraff and Andrew Witkin. Large steps in cloth simulation. In *Proc. of ACM SIGGRAPH*, pages 43–54, 1998.

[BWHT07] Adam W. Bargteil, Chris Wojtan, Jessica K. Hodgins, and Greg Turk. A finite element method for animating large viscoplastic flow. *ACM Transactions on Graphics*, 26(3):16:1–16:8, July 2007.

[CAR+09] Nuttapong Chentanez, Ron Alterovitz, Daniel Ritchie, Lita Cho, Kris K. Hauser, Ken Goldberg, Jonathan R. Shewchuk, and James F. O'Brien. Interactive simulation of surgical needle insertion and steering. *ACM Transactions on Graphics*, 28(3):88:1–88:10, July 2009.

[CH93] C. Carlsson and O. Hagsand. Dive - a platform for multi-user virtual environments. In *Computers and Graphics*, pages 663–669, 1993.

[CJY02] Johnny T. Chang, Jingyi Jin, and Yizhou Yu. A practical model for hair mutual interactions. In *ACM SIGGRAPH Symposium on Computer Animation*, pages 73–80, July 2002.

[CK02] Kwang-Jin Choi and Hyeong-Seok Ko. Stable but responsive cloth. In *Proc. of ACM SIGGRAPH*, pages 604–611, 2002.

[CK05] K.-J. Choi and H.-S. Ko. Advanced topics on clothing simulation and animation, 2005. ACM SIGGRAPH Conference Course Notes.

[CLMP95] Jonathan D Cohen, Ming C Lin, Dinesh Manocha, and Madhav Ponamgi. I-collide: An interactive and exact collision detection system for large-scale environments. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 189–ff. ACM, 1995.

[CN95] C. Cruz-Neira. *Virtual Reality Based on Multiple Projection Screens: The CAVE and its Applications to Computational Science and Engineering*. PhD thesis, University of Illinois at Chicago, Department of Electrical Engineering and Computer Science, 1995.

[CNSD93] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti. Surround-screen projection-based virtual reality: The design and implementation of the cave. In *T. Kajiya, editor, Computer Graphics (SIGGRAPH 93 Proceedings*, pages 135–142, 1993.

[CPS92] R. Cottle, J. Pang, and R. Stone. *The Linear Complementarity Problem*. Academic Press, 1992.

[Crya] Crystal Space. Crystal Space website. `http://www.crystalspace3d.org/main/Main_Page`.

[Cryb] Crytek GmbH. Crytek CryEngine Sandbox. `http://www.crytek.com/cryengine`.

[DAK04] C. Duriez, C. Andriot, and A. Kheddar. Signorini's contact model for deformable objects in haptic simulations. In *Proc. of IEEE Int'l Conf. Intelligent Robots and Systems*, pages 3232–3237, 2004.

[DDKA06] Christian Duriez, Frederic Dubois, Abderrahmane Kheddar, and Claude Andriot. Realistic haptic rendering of interacting deformable objects in virtual environments. *Proc. of IEEE TVCG*, 12(1), 2006.

[DLB94] M. Dinsmore, N. Langrana, and G. Burdea. Issues related to real-time simulation of a virtual knee joint palpation. *Proceedings of Virtual Reality and Medicine, The Cutting Edge*, pages 16–20, 1994.

[Dru08] Evan Drumwright. A fast and stable penalty method for rigid body simulation. *IEEE Transactions on Visualization and Computer Graphics*, 14:231–240, January 2008.

[DZS08] Paul C. DiLorenzo, Victor B. Zordan, and Benjamin L. Sanders. Laughing out loud: Control for modeling anatomically inspired laughter using audio. *ACM Trans. Graph.*, 27(5):125:1–125:8, December 2008.

[EL10] C. Eisenacher and C. Loop. Data-parallel micropolygon rasterization. In *Proc. of Eurographics Short Papers*. 2010.

[Eon] Eon Reality Inc. *Eon Studio Reference Manual*.

[Epi] Epic Games Inc. Epic Games Unreal Engine. `http://www.unreal.com/`.

[Equ] Equalizer. Equalizer: standard middleware to create and deploy parallel OpenGL-based applications. `http://www.equalizergraphics.com`.

[Erl07] Kenny Erleben. Velocity-based shock propagation for multibody dynamics animation. *ACM Trans. on Graphics*, 26(2), 2007.

[ESJ97] R E Ellis, N Sarkar, and M A Jenkins. Numerical methods for the force reflection of contact. *ASME Transactions on Dynamic Systems Measurement and Control*, 119(4):768–774, 1997.

[FBAF08] François Faure, Sébastien Barbier, Jérémie Allard, and Florent Falipou. Image-based collision detection and response between arbitrary volumetric objects. In *Proc. of ACM Siggraph/Eurographics Symp. on Computer Animation*, pages 155–162, 2008.

[FL01] Susan Fisher and Ming C. Lin. Deformed distance fields for simulation of non-penetrating flexible bodies. In *Proc. of Eurographic workshop on Computer animation & simulation*, pages 99–111, 2001.

[FLB+09] Kayvon Fatahalian, Edward Luong, Solomon Boulos, Kurt Akeley, William R. Mark, and Pat Hanrahan. Data-parallel rasterization of micropolygons with de-

focus and motion blur. In *Proceedings of the Conference on High Performance Graphics 2009*, pages 59–68, 2009.

[Fre87] M. Fremond. Adherence des solides. *Journal de Mechanique Theorique et Appliquee*, 6, 1987.

[GBEO11] Jorge Gascón, José M. Bayona, José Miguel Espadero, and Miguel A. Otaduy. Blendercave: Easy vr authoring for multi-screen displays. *SIACG 2011: V IBERO-AMERICAN SYMPOSIUM IN COMPUTER GRAPHICS*, 2011.

[GBF03a] Eran Guendelman, Robert Bridson, and Ronald Fedkiw. Nonconvex rigid bodies with stacking. *Proc. of ACM SIGGRAPH*, 2003.

[GBF03b] Eran Guendelman, Robert Bridson, and Ronald P. Fedkiw. Nonconvex rigid bodies with stacking. *ACM Transactions on Graphics*, 22(3):871–878, July 2003.

[GEP+13] Jorge Gascón, José Miguel Espadero, Alvaro G. Perez, Rosell Torres, and Miguel A. Otaduy. Fast deformation of volume data using tetrahedral mesh rasterization. In *Proc. of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2013.

[Ghe97] S. Ghee. Programming virtual worlds. In *ACM SIGGRAPH 97 Conference, Los Angeles*, 1997.

[GHR+02] P. Grimm, M. Haller, S. Reinhold, C. Reimann, and J. Zauner. Amire - authoring mixed reality. In *Proc. of IEEE International Augmented Reality Toolkit Workshop*, 2002.

[GLM96] Stefan Gottschalk, Ming Lin, and Dinesh Manocha. Obb-tree: A hierarchical structure for rapid interference detection. In *Proceedings of SIGGRAPH 96*, Computer Graphics Proceedings, Annual Conference Series, pages 171–180, 1996.

[GO09] Carlos Garre and Miguel A. Otaduy. Haptic rendering of complex deformations through handle-space force linearization. In *Proc. of World Haptics Conference*, mar 2009.

[Gri91] C. Grimsdale. dvs-distributed virtual environment system. In *In Proceedings of Computer Graphics 1991 Conference*, 1991.

[GS09] Orcun Goksel and Septimiu E. Salcudean. B-mode ultrasound image simulation in deformable 3-d medium. *IEEE Transactions on Medical Imaging*, 28(11):1657–1669, 2009.

[GW06] J. Georgii and R. Westermann. A generic and scalable pipeline for gpu tetrahedral grid rendering. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):1345–1352, 2006.

[GZO10] Jorge Gascón, Javier S. Zurdo, and Miguel A. Otaduy. Constraint-based simulation of adhesive contact. In *Proc. of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2010.

[HB00] D. H. House and D. E. Breen. *Cloth Modeling and Animation*. AK Peters, 2000.

[HS04] S. Hasegawa and M. Sato. Real-time rigid body simulation for haptic interactions based on contact volume of polygonal objects. *Computer Graphics Forum*, 23(3):529–538, 2004.

[HTK+04] Bruno Heidelberger, Matthias Teschner, Richard Keiser, Matthias Müller, and Markus Gross. Consistent penetration depth estimation for deformable collision response. In *Proc. of Vision, Modeling, Visualization*, pages 339–346, 2004.

[HVS+09a] David Harmon, Etienne Vouga, Breannan Smith, Rasmus Tamstorf, and Eitan Grinspun. Asynchronous contact mechanics. *ACM Transactions on Graphics*, 28(3):87:1–87:12, July 2009.

[HVS+09b] David Harmon, Etienne Vouga, Breannan Smith, Rasmus Tamstorf, and Eitan Grinspun. Asynchronous contact mechanics. In *Proc. of ACM SIGGRAPH*, pages 87:1–87:12, 2009.

[Id] Id Software. Quake IdTech 3. `http://www.idsoftware.com/`.

[Ini97] Inition. *Sense8 WorldToolkit R8 Reference Manual*. 1997.

[Irr] Irrlicht. Irrlicht Engine website. `http://irrlicht.sourceforge.net/`.

[JL93] S. Jimenez and A. Luciani. Animation of interacting objects with collisions and prolonged contacts. In *Proc. of the IFIP WG 5.10 Working Conference*, pages 129–141, 1993.

[JMD+07] Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. Harmonic coordinates for character articulation. *ACM Transactions on Graphics*, 26(3):71:1–71:9, July 2007.

[KBF+95] W. Krüger, C.-A. Bohn, B. Fröhlich, H. Schüth, W. Strauss, and G. Wesche. The responsive workbench: A virtual work environment. In *IEEE Computer*, pages 42–48, 1995.

[KEP05] Danny M. Kaufman, Timothy Edmunds, and Dinesh K. Pai. Fast frictional dynamics for rigid bodies. *Proc. of ACM SIGGRAPH*, 2005.

[KSA+03] J. Kelso, S. G. Satterfield, L. E. Arsenault, P. M. Ketchan, and R. D. Kriz. Diverse: A framework for building extensible and reconfigurable device-independent virtual environments and distributed asynchronous simulations. *Presence: Teleoperators and Virtual Environments*, 12(1):19–36, 2003.

[KSJP08] Danny M. Kaufman, Shinjiro Sueda, Doug L. James, and Dinesh K. Pai. Staggered projections for frictional contact in multibody systems. *Proc. of ACM SIGGRAPH Asia*, 2008.

[KSK97] K. Kawachi, H. Suzuki, and F. Kimura. Simulation of rigid body motion with impulsive friction force. In *Assembly and Task Planning, 1997. ISATP 97., 1997 IEEE International Symposium on*, pages 182 –187, aug 1997.

[KWW01] Davis King, CraigM. Wittenbrink, and HansJ. Wolters. An architecture for interactive tetrahedral volume rendering. In Klaus Mueller and ArieE. Kaufman, editors, *Volume Graphics 2001*, Eurographics, pages 163–180. Springer Vienna, 2001.

[LCC+12] Jean-Luc Lugrin, Fred Charles, Marc Cavazza, Marc Le Renard, Jonathan Freeman, and Jane Lessiter. Caveudk: a vr game engine middleware. In *Proceedings*

of the 18th ACM symposium on Virtual reality software and technology, pages 137–144. ACM, 2012.

[LHLW10] Fang Liu, Meng-Cheng Huang, Xue-Hui Liu, and En-Hua Wu. Freepipe: a programmable parallel rendering architecture for efficient multi-fragment effects. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, I3D '10, pages 75–82, New York, NY, USA, 2010. ACM.

[LHS06] P. Leskovsky, M. Harders, and G. Szekely. A web-based repository of surgical simulator projects. *Stud Health Technol Inform*, 119, 2006.

[Lim] Limsi-CNRS. BlenderCave: blender add-on for virtual reality. `http://blendercave.limsi.fr/doku.php`.

[LK11] Samuli Laine and Tero Karras. High-performance software rasterization on gpus. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, HPG '11, pages 79–88, New York, NY, USA, 2011. ACM.

[Lot84] Per Lotstedt. Numerical simulation of time-dependent contact and friction problems in rigid body mechanics. *SIAM Journal on Scientific and Statistical Computing*, 5(2):370–393, 1984.

[LST09] Sung-Hee Lee, Eftychios Sifakis, and Demetri Terzopoulos. Comprehensive biomechanical modeling and simulation of the upper body. *ACM Trans. Graph.*, 28(4):99:1–99:17, August 2009.

[LT06] Sung-Hee Lee and Demetri Terzopoulos. Heads up!: biomechanical modeling and neuromuscular control of the neck. *ACM Trans. Graph*, 25(3):1188–1198, July 2006.

[LTCK03] A. Liu, F. Tendick, K. Cleary, and C. Kaufmann. A survey of surgical simulation: applications, technology, and education. *Presence: Teleoper. Virtual Environ.*, 12(6):599–614, 2003.

[MBTF03] N. Molino, R. Bridson, J. Teran, and R. Fedkiw. A crystalline, red green strategy for meshing highly deformable objects with tetrahedra. *International Meshing Roundtable*, pages 103–114, 2003.

[MG04] Matthias Müller and Markus Gross. Interactive virtual materials. In *Proceedings of Graphics Interface 2004*, pages 239–246, 2004.

[MH05] Mohsen Mahvash and Vincent Hayward. High-fidelty passive force-reflecting virtual environments. *IEEE Transactions on Robotics*, 21(1):38–46, 2005.

[MHHR06] Matthias Müller, B. Heidelberger, M. Hennix, and J. Ratcliff. Position based dynamics. *Proc. of VRIPHYS*, 2006.

[Mir96] B. Mirtich. *Impulse-based Dynamic Simulation of Rigid Body Systems*. Ph.D. thesis, Dept. Elec. Engin. Comput. Sci., Univ. California, Berkeley, CA, 1996.

[MKN+04] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross, and M. Alexa. Point based animation of elastic, plastic and melting objects. *Proc. of Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, 2004.

[MSO+09] S. Martín, J. Suárez, R. Orea, R. Rubio, and R. Gallego. Glsv: Graphics library stereo vision for opengl. *Virtual Reality*, 13:51–57, 2009. 10.1007/s10055-008-0105-y.

[MW88] M. Moore and J. Wilhelms. Collision detection and response for computer animation. *Proc. of ACM SIGGRAPH*, 1988.

[NFP10] Matthieu Nesme, François Faure, and Yohan Payan. Accurate interactive animation of deformable models at arbitrary resolution. *International Journal of Image and Graphics*, 10(2), 2010.

[NKJF09] Matthieu Nesme, Paul G. Kry, Lenka Jeřábková, and François Faure. Preserving topology and elasticity for embedded deformable models. *ACM Trans. Graph.*, 28(3):52:1–52:9, July 2009.

[NMK+05] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxermann, and Mark Carlson. Physically based deformable models in computer graphics. *Eurographics STAR*, 2005.

[NMK+06] Andrew Nealen, Matthias Muller, Richard Keiser, Eddy Boxerman, and Mark Carlson. Physically based deformable models in computer graphics. In *Computer Graphics Forum*, volume 25, pages 809–836. Wiley Online Library, 2006.

[ODE] ODE. Open Dynamics Engine. `http://www.ode.org/`.

[OGG+10] Miguel A. Otaduy, Carlos Garre, Jorge Gascón, Eder Miguel, Alvaro G. Perez, and Javier S. Zurdo. Modeling and simulation of a human shoulder for interactive medical applications. In *Proc. of Congreso Español de Informática Gráfica*, 2010.

[OGR] OGRE. OGRE-Open Source 3D Graphics Engine. `http://www.ogre3d.org/`.

[ORC07] Michael Ortega, Stephane Redon, and Sabine Coquillart. A six degree-of-freedom god-object method for haptic display of rigid bodies with surface properties. *IEEE Transactions on Visualization and Computer Graphics*, 13:458–469, 2007.

[OSG] OSG. OpenSceneGraph website. `http://www.openscenegraph.org/projects/osg`.

[OTSG09] Miguel A. Otaduy, Rasmus Tamstorf, Denis Steinemann, and Markus Gross. Implicit contact handling for deformable objects. *Computer Graphics Forum*, 28(2):559–568, April 2009.

[Pan11] Jacopo Pantaleoni. Voxelpipe: a programmable pipeline for 3d voxelization. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, HPG '11, pages 99–106, New York, NY, USA, 2011. ACM.

[PO09] Eric G. Parker and James F. O'Brien. Real-time deformation and fracture in a game environment. In *2009 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 156–166, 2009.

[PPG04] Mark Pauly, Dinesh K. Pai, and Leonidas J. Guibas. Quasi-rigid objects in contact. *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2004.

[PQTK12] David Poirier-Quinot, Damien Touraine, and Brian FG Katz. Blendercave 3d-s project, opensource architecture adaptation to virtual reality research expectations. *Blender Conference, (Amsterdam), blender.org*, 2012.

[PQTK13a] David Poirier-Quinot, Damien Touraine, and Brian FG Katz. Blendercave: A flexible open source authoring tool dedicated to multimodal virtual reality. *Virtual Reality Conference (JVRC), (Orsay)*, (5):19–22, 2013.

[PQTK13b] David Poirier-Quinot, Damien Touraine, and Brian FG Katz. Blendercave: A multimodal scene graph editor for virtual reality. *International Conference on Auditory Display (ICAD)*, (19):323 –330, 2013.

[RCC99] M. Raous, L. Cangémi, and M. Cocu. A consistent model coupling adhesion, friction, and unilateral contact. *Computer Methods in Applied Mechanics and Engineering*, 177(3-4), 1999.

[RF06] L. Raghupathi and F. Faure. QP-Collide: A new approach to collision treatment. *French Working Group on Animation and Simulation (GTAS)*, 2006.

[RKC02] S. Redon, A. Kheddar, and S. Coquillart. Gauss' least constraints principle and rigid body simulations. *Proc. of ICRA*, 2002.

[RKLM04] Stephane Redon, Young J. Kim, Ming C. Lin, and Dinesh Manocha. Fast continuous collision detection for articulated models. In *Proceedings of the ninth ACM symposium on Solid modeling and applications*, pages 145–156, 2004.

[RSF+04] Antonio J Rueda, Rafael J Segura, Francisco R Feito, Juan Ruiz de Miras, and Carlos Ogáyar. Voxelization of solids using simplicial coverings. In *Proc. of WSCG*, 2004.

[RSHRL08] C. Rezk-Salama, M. Hadwiger, T. Ropinski, and P. Ljung. Advanced illumination techniques for gpu-based volume raycasting. ACM SIGGRAPH Asia Course Notes, 2008.

[SBT06] J. Spillmann, M. Becker, and M. Teschner. Efficient updates of bounding sphere hierarchies for geometrically deformable models. *Proc. of VRIPHYS*, 2006.

[SBT07] J. Spillmann, M. Becker, and M. Teschner. Non-iterative computation of contact forces for deformable objects. *Journal of WSCG*, 2007.

[Sch13] Sara C Schvartzman. Accelerated proximity queries for collision detection and brittle fracture. 2013.

[SCS+08] Larry Seiler, Doug Carmean, Eric Sprangle, Tom Forsyth, Michael Abrash, Pradeep Dubey, Stephen Junkins, Adam Lake, Jeremy Sugerman, Robert Cavin, Roger Espasa, Ed Grochowski, Toni Juan, and Pat Hanrahan. Larrabee: a many-core x86 architecture for visual computing. *ACM Trans. Graph.*, 27(3):18:1–18:15, 2008.

[SDCG08] G. Saupin, C. Duriez, S. Cotin, and L. Grisoni. Efficient contact modeling using compliance warping. *Proc. of Computer Graphics International*, 2008.

[Sha89] Ahmed A. Shabana. *Dynamics of Multibody Systems*. John Wiley and Sons, 1989.

[SKP08] Shinjiro Sueda, Andrew Kaufman, and Dinesh K. Pai. Musculotendon simulation for hand animation. *ACM Trans. Graph.*, 27(3), August 2008.

[SLF08] Andrew Selle, Michael Lentine, and Ronald Fedkiw. A mass spring model for hair simulation. *ACM Transactions on Graphics*, 27(3):64:1–64:11, August 2008.

[SLGS92] C. Shawn, J. Liang, M. Green, and Y. Sun. The decoupled simulation model for virtual reality systems. In *Proceedings of the ACM SIGCHI Human Factors in Computer Systems Conference*, pages pp.321–328, 1992.

[SM00] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.

[SML04] W Schroeder, K Martin, and B Lorensen. The Visualization Toolkit: An object-oriented approach to 3D graphics, 3rd edition. Technical report, Kitware Inc, 2004.

[SNF05] Eftychios Sifakis, Igor Neverov, and Ronald Fedkiw. Automatic determination of facial muscle activations from sparse motion capture marker data. *ACM Trans. Graph*, 24(3):417–425, August 2005.

[SOF] SOFA. Simulation Open Framework Architecture. `http://www.sofa-framework.org/`.

[Son00] Milan Sonka. *Handbook of medical imaging: medical image processing and analysis*. SPIE-International Society for Optical Engineering, 2000.

[SRH03] D. Schmalstieg, G. Reitmayr, and G. Hesina. Distributed applications for collaborative three-dimensional workspaces. *Presence: Teleoperators and Virtual Environments*, 12(1):52–67, 2003.

[SS10] Michael Schwarz and Hans-Peter Seidel. Fast parallel surface and solid voxelization on gpus. *ACM Trans. Graph.*, 29(6):179:1–179:10, 2010.

[SSF08] Tamar Shinar, Craig Schroeder, and Ron Fedkiw. Two-way coupling of rigid and deformable bodies. *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2008.

[SSIF07] Eftychios Sifakis, Tamar Shinar, Geoffrey Irving, and Ronald Fedkiw. Hybrid simulation of deformable solids. In *2007 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 81–90, August 2007.

[ST96] D. E. Stewart and J. C. Trinkle. An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and Coulomb friction. *International Journal of Numerical Methods in Engineering*, 39, 1996.

[ST00] D. E. Stewart and J. C. Trinkle. An implicit time-stepping scheme for rigid body dynamics with Coulomb friction. *Proc. of IEEE ICRA*, 2000.

[Sto01] Maureen C. Stone. Color and brightness appearance issues in tiled displays. *IEEE Computer Graphics and Applications*, 21:58–66, 2001.

[TCYM09] Min Tang, Sean Curtis, Sung-Eui Yoon, and Dinesh Manocha. ICCD: Interactive continuous collision detection between deformable models using connectivity-

based culling. *IEEE Transactions on Visualization and Computer Graphics*, 15(4):544–557, 2009.

[THS+01] Russell M. Taylor II, Thomas C. Hudson, Adam Seeger, Hans Weber, Jeffrey Juliano, and Aron T. Helser. Vrpn: A device-independent, network-transparent vr peripheral system. In *VRST 2001 conference*, 2001.

[TKH+05] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Furhmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, and P. Volino. Collision detection for deformable objects. *Computer Graphics Forum*, 24(1), 2005.

[TMOT12] Min Tang, Dinesh Manocha, Miguel A. Otaduy, and Ruofeng Tong. Continuous penalty forces. *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH)*, 31(4), 2012.

[TPBF87] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. *Proc. of ACM SIGGRAPH*, 1987.

[Tra99] H. Tramberend. Avocado: a distributed virtual reality framework. In *Virtual Reality, 1999. Proceedings., IEEE*, pages 14 –21, March 1999.

[TSB+05] J. Teran, E. Sifakis, S. S. Blemker, V. Ng-Thow-Hing, C. Lau, and R. Fedkiw. Creating and simulating skeletal muscle from the visible human data set. *IEEE Trans. on Visualization and Computer Graphics*, 11(3):317–328, May/June 2005.

[TSIF05] Joseph Teran, Eftychios Sifakis, Geoffrey Irving, and Ronald Fedkiw. Robust quasistatic finite elements and flesh simulation. In *Proc. of the ACM SIGGRAPH/Eurographics Symp. on Computer animation*, pages 181–190, 2005.

[Unia] Unigine Corp. Unigine: multi-platform real-time 3D engine website. `http://unigine.com/`.

[Unib] Unity Technologies. Unity 3D engine website. `http://unity3d.com/unity/`.

[VPL90] VPL Research Inc. *Reality Built for Two: RB2 Operation Manual*. 1990.

[WB97] (Andrew Witkin and David Baraff. Physically based modeling: Principles and practice. *SIGGRAPH 1997 Course*, 1997.

[WGL04] K. Ward, N. Galoppo, and M. C. Lin. Modeling hair influenced by water and styling products. In *Proc. of Computer Animation and Social Agents (CASA)*, pages 207–214, 2004.

[Wri02] P. Wriggers. *Computational Contact Mechanics*. Wiley, 2002.

[WTGT09] Chris Wojtan, Nils Thürey, Markus Gross, and Greg Turk. Deforming meshes that split and merge. *ACM Transactions on Graphics*, 28(3):76:1–76:10, July 2009.

[WVVS90] P. Wriggers, T. Vu Van, and E. Stein. Finite element formulation of large deformation impact-contact problems with friction. *Computers & Structures*, 37(3):319–331, 1990.

[YN06] K. Yamane and Y. Nakamura. Stable penalty-based model of frictional contacts. In *Proc. of IEEE Int'l Conf. on Robotics and Automation*, pages 1904 –1909, 2006.

[ZKVM07] Liangjun Zhang, Young J. Kim, Gokul Varadhan, and Dinesh Manocha. Generalized penetration depth computation. *Comput. Aided Des.*, 39:625–638, 2007.