Diss. ETH No. 17055

Consistent Collision and Self-Collision Handling for Deformable Objects

A dissertation submitted to
ETH Zurich

for the degree of
Doctor of Sciences

presented by
Bruno Heinz Heidelberger
Dipl. Informatik-Ing. ETH
Swiss Federal Institute of Technology, ETH Zurich
born November 17, 1972
citizen of Hochfelden, ZH, Switzerland

accepted on the recommendation of
Prof. Markus Gross, examiner
Prof. Matthias Teschner, co-examiner

2007

"The meeting of two personalities is like a collision of two deformable objects;
if there is any contact, both are transformed."

To my beloved Tatjana

Abstract

Interactive environments with dynamically deforming objects play an important role in physically-based simulation and animation, ranging from computational surgery to computer games. These environments require efficient and robust methods for basic simulation tasks, such as deformation, collision detection and collision response.

This thesis investigates novel collision handling components especially suited for deformable objects. The focus lies on versatile techniques featuring efficient data structures, self-collision support and n-body collision information while still delivering interactive performance. All presented methods aim for visually-plausible and consistent behavior, which is in accordance to the requirements of typical target applications.

The first approach detects collisions and self-collisions of deformable objects based on a variant of spatial partitioning. It employs a hashing scheme which allows for very efficient n-body collision queries between different object primitives, such as vertices, lines, triangles and tetrahedrons.

The second collision detection method generates a volumetric approximation of the intersection volume to detect collisions and self-collisions. The computation of the volumetric representation is done in image-space to take advantage of potential graphics hardware acceleration. The technique allows for several volumetric collision queries: An explicit representation of the intersection volume, a vertex-in-volume check and a self-collision test.

Finally, a technique is proposed that computes consistent n-body penetration depth information in order to reduce collision response artifacts. It considers a set of close surface features to avoid discontinuities and it applies a propagation scheme in case of large penetrations to minimize non-plausible, inconsistent collision information.

A test suite composed of several carefully selected experiments is used to analyze the characteristics and the performance of each presented technique. The methods are also integrated into a hysteroscopy simulator and a complete framework for interactive simulation of dynamically deforming objects. Both applications demonstrate the capability and applicability of the collision handling components presented in this thesis.

Zusammenfassung

Interaktive Umgebungen, wie z.B. Chirurgiesimulationen oder Computerspiele, sind wichtige Anwendungsgebiete der physikalisch basierten Simulation und Animation. Sie benötigen effiziente und robuste Algorithmen zur Berechnung der Simulationsprozesse wie Deformation, Kollisionserkennung und Kollisionsauflösung.

Diese Arbeit untersucht neuartige Komponenten zur Kollisionsbehandlung, die speziell für deformierbare Objekte geeignet sind. Dabei liegt der Fokus auf vielseitig einsetzbaren Methoden, welche, basierend auf effizienten Datenstrukturen, trotz Selbstkollisionserkennung und Vielkörperunterstützung interaktive Laufzeiten erreichen. Alle vorgestellten Algorithmen erzeugen plausibles und konsistentes Simulationsverhalten, um die Bedingungen der typischen Zielapplikationen zu erfüllen.

Der erste Ansatz erkennt Kollisionen und Selbstkollisionen mit Hilfe von Raumunterteilung. Dabei wird ein Hashingverfahren verwendet, welches sehr effiziente Kollisionsanfragen zwischen unterschiedlichen Objektprimitiven, wie z.B. Punkte, Linien, Dreiecke und Tetraeder, erlaubt.

Ein zweiter Ansatz zur Kollisionserkennung erzeugt eine volumetrische Approximation des Schnittvolumens, um Kollisionen und Selbstkollisionen zu finden. Die Berechnung dieser Repräsentation wird im Bildraum durchgeführt und kann daher von Grafikhardware beschleunigt werden. Die Methode erlaubt verschiedenartige, volumetrische Kollisionsanfragen: Eine explizite Repräsentation des Schnittvolumens, eine Überprüfung von Punkt-in-Volumen und ein Test auf Selbstkollision.

Ein weiterer Ansatz berechnet konsistente Eindringtiefen bei Vielkörpersimulationen und reduziert so Artefakte bei der Kollisionsauflösung. Die Methode betrachtet eine Anzahl naheliegender Oberflächen, um unerwünschte Unstetigkeiten zu verhindern. Zusätzlich minimiert ein Propagationsschema nicht plausible und inkonsistente Resultate bei hoher Eindringtiefe.

Um die Eigenschaften und Leistung der vorgestellten Methoden zu validieren, wird eine Reihe von aussagekräftigen Experimenten durchgeführt. Die Ansätze sind auch integriert in eine Chirurgiesimulation und eine vollständige Umgebung zur interaktiven Simulation von dynamisch deformierbaren Objekten. Beide Applikationen demonstrieren die Fähigkeiten und die Anwendbarkeit der in dieser Arbeit präsentierten Kollisionsbehandlungsmethoden.

Acknowledgements

First of all, I would like to thank my advisor Prof. Markus Gross for giving me the opportunity to pursue my Ph.D. in the Computer Graphics Laboratory at ETH Zurich. A special thanks to my co-advisor Prof. Matthias Teschner. It has been a great experience to work with him.

I thank all my fellow colleagues at the Computer Graphics Laboratory who have helped me along the way: Christoph Niederberger, Tim Weyrich, Silke Lang, Richard Keiser, Denis Steinemann, Remo Ziegler, Edouard Lamboray, Stephan Würmlin, Mark Pauly, Miguel A. Otaduy, Doo Young Kwon, Filip Sadlo, Martin Wicke, Christian Sigg, Ronny Peikert, Michael Waschbüsch, Daniel Cotting, Robert Sumner, Mario Botsch, Reto Lütolf, Dirk Bauer, Daniel Bielser, Martin Naef and Matthias Zwicker.

Many thanks go to all my former students who have contributed to this work: Andreas Wetzel, Robert Bargmann, Bernhard Wymann, Jonas Spillmann, Roger Kehrer, Christof Schmid and Daniel Knoblauch.

Finally, I would like to express my gratitude to my family, and especially to my wife, for their constant support during my studies and Ph.D.

This work was supported by the Swiss National Science Foundation (SNSF) as part of the Swiss National Center of Competence in Research on Computer Aided and Image Guided Medical Interventions (NCCR Co-Me) [Com01].

Abbreviations and Acronyms

AABB	Axis-Aligned Bounding Box
ADF	Adaptively Sampled Distance Fields
ARB	Architecture Review Board
BEM	Boundary Element Method
BSP	Binary Space Partitioning
BV	Bounding Volume
BVH	Bounding-Volume Hierarchy
CCD	Continuous Collision Detection
CEGUI	Crazy Eddie's GUI System
CFD	Computational Fluid Dynamics
CPU	Central Processing Unit
CSG	Constructive Solid Geometry
DOP	Discrete-Oriented Polytope
FEM	Finite Element Method
FVM	Finite Volume Method
GJK	Gilbert, Johnson, Keerthi
GPU	Graphics Processing Unit
GUI	Graphical User Interface
KDS	Kinetic Data Structures
KSP	Kinetic Sweep and Prune
LCP	Linear Complementarity Problems
LDC	Layered Depth Cube
LDI	Layered Depth Image
NCCR	National Center of Competence in Research
OBB	Oriented Bounding Box
OGRE	Object-Oriented Graphics Rendering Engine
OR	Operating Room
SAP	Sweep and Prune
SDK	Software Development Kit
SV	Swept Volume
VoI	Volume of Interest

Contents

1 Introduction

Interactive environments with dynamically deforming objects play an important role in physically-based simulation and animation, ranging from computational surgery to computer games. These environments require efficient and robust methods for basic simulation tasks, such as deformation, collision detection and collision response. While efficient deformable models are well-investigated, the handling of collisions between deformable structures only recently gained increasing attention.

1.1 Context

Two major applications in the fields of simulation and animation define the context of this thesis: Surgery simulation and computer games. While all the methods discussed in the following chapters can easily be applied to other environments, the aforementioned context is used to thoroughly analyze and validate the presented approaches.

Surgery simulation is one major application for deformable collision handling. In general, these environments consist of various deformable organs and rigid surgical tools interacting with each other. Collisions between objects occur very frequently and need to be resolved as soon as they are detected to guarantee a realistic look and feel. In addition to collision between multiple objects, various surgical procedures, such as cutting, may lead to self-collision of soft tissue that also have to be taken care of. Since interactive behavior is essential throughout the whole simulation period, the algorithms for deformable collision handling are not only required to be efficient but also robust under all circumstances.

Deformable collision handling methods are also useful in environments with animated objects, such as computer games. Due to the ever increasing computational power, more and more objects in game levels are either animated or physically simulated. Current computer games already feature articulated, skinned objects that are driven by multiple pre-computed motion sequences. So called rag-doll effects further improve the dynamic appearance of these skinned models. The next generation of computer games will support even more general and sophisticated deformable models requiring specific algorithms to detect and resolve collisions at interactive rates.

1.2 Problem Statement

The focus of this thesis lies on the dynamic interaction between deformable objects in interactive simulation environments. Hence the investigated methods should especially be suited for this type of models, such as soft tissue. However, since simulation environments also contain rigid bodies, such as surgical instruments or bones, the methods should not be restricted to deformable objects only. While collision handling for rigid bodies is well-investigated, collision handling for deformable objects introduces additional challenging problems.

Collision, Self-collision and Resting Contact

Deformable objects require a more advanced collision handling approach than rigid objects due to their higher degree of freedom. In order to realistically simulate interactions between deformable objects, all contact points including those due to self-collisions have to be considered. This is in contrast to rigid body collision handling where self-collisions are commonly neglected.

Depending on the applications, rigid body approaches can further be accelerated by only handling one single contact point. This is not possible in case of deformable objects. In addition, the problem of resting contact between deformable objects needs special attention in order to guarantee a stable equilibrium.

Data Structures and Pre-processing

Collision handling approaches for rigid objects usually pre-compute sophisticated and time-consuming data structures once at the beginning of the simulation. This allows for efficient and fast collision detection afterwards. However, the nature of deformable objects require that these pre-processed data structures have to be updated frequently. These updates can be quite extensive in case of large deformations. Therefore, pre-processed data structures are less efficient for deforming objects and their practicability is questionable. This thesis investigates appropriate data structures that allow for efficient updates during the simulation. This also simplifies the handling of topological changes, such as cutting.

The predominant object representations used in current simulation environments with dynamically deforming objects are closed, volumetric tetrahedral meshes as well as potentially open triangular meshes. Both types should be supported by the collision handling components. This requirement imposes specific properties on the underlying data structures which are investigated in this thesis.

Consistent N-body Collision Information

Physically-based simulations typically use discrete time-steps in their simulation loop. This leads to large penetrations, missed collisions, inconsistent collision information and other artifacts that need to be resolved in a robust way. It is not sufficient to just detect the interference of objects. Instead, precise and consistent information such as penetration direction and depth is desired, so that an appropriate collision response can be applied to the involved objects.

Simulation environments usually consist of multiple objects interacting with each other. It is possible that more than two objects collide or are in contact at the same time. To handle these likely events in a robust and plausible way, the collision detection components and their data structures must be queried for such n-body collisions. Then, after the detection, the collision response method should simultaneously resolve all collisions consistently.

Performance

Interactivity is a key characteristic in applications, such as surgery simulation and computer games. These environments typically consist of multiple deforming objects with a few thousand tetrahedrons each. It is required that they perform at interactive speed to be practicable. This defines hard constraints on the performance of all methods, including the collision handling components.

In order to meet the interactive performance criteria, it may be necessary to sacrifice accuracy for speed. Aiming at visually-plausible results, in contrast to physically-correct ones, is a possibility in this context. Calculating the collision information on a simplified approximation of the deformable object is an example of such an approach.

Graphics Hardware Acceleration

With the increasing programmability of commodity graphics processing units (GPUs), these chips are capable of performing more than the specific graphics computations for which they were originally designed. They are now versatile coprocessors and their high speed makes them useful for a variety of applications. Potential acceleration of collision handling techniques through the use of such programmable GPUs, as well as the limitations of such collision handling approaches, must be investigated.

Applications

The collision handling components should be integrated into a framework for interactive simulation of dynamically deforming objects for a thorough analysis and validation. They are also an essential part of a hysteroscopy simulator prototype

developed by project members of the Swiss National Center of Competence in Research on Computer Aided and Image Guided Medical Interventions (NCCR Co-Me) [Com01]. This prototype provides an interactive surgical simulation environment with deformable soft tissue, surgical instruments, haptic feedback and realistic rendering.

1.3 Major Contributions

The goal of this thesis is the design and implementation of novel collision handling components appropriate for physically-based simulation environments with dynamically deforming objects. As a result of the demanding requirements, such as interactive performance and guaranteed robustness, and the assumptions made, such as discrete time-steps and n-body collisions, this thesis concentrates on visually-plausible collision behavior. Still, within the restrictions given by the requirements, it aims for methods that are as physically-correct as possible. Although all investigated algorithms are especially suited for deformable objects, they are not restricted to deformable objects, but also work with rigid bodies.

The contributions of this thesis can be summarized as follows:

- A method is presented that detects collisions and self-collisions of deformable objects based on optimized spatial hashing. This method allows for very efficient n-body collision queries between different object primitives, such as vertices, lines, triangles and tetrahedrons.
- A method is investigated which detects collisions and self-collisions of deformable objects based on a volumetric approximation of the intersection volume. This technique allows for three volumetric collision queries: An explicit representation of the intersection volume, a vertex-in-volume test and a self-collision test. All queries can be accelerated by graphics hardware.
- A method is proposed that eliminates typical errors in the computation of collision information neglected in previous work. The novel method achieves a consistent estimation of the penetration depth leading to a much more robust collision handling. Furthermore, the method addresses the issue of large time-steps and the corresponding collision handling artifacts.
- A complete framework for interactive simulation of dynamically deforming objects and a hysteroscopy simulator are described that consist of all necessary components for deformation, collision detection and collision response. These environments are also used to analyze the performance of all methods presented in this thesis.

Research results from the optimized spatial hashing approach have been published in [Tes03]. The collision detection method based on an LDI decomposition has been presented in [Hei03], [Hei03b] and its extension to self-collision detection in [Hei04]. The technique for the consistent estimation of penetration depth has been published in [Hei04c]. The interactive simulation framework for deformable objects has been used in [Tes04], [Tes05], as well as in [Mue05].

1.4 Outline

The remainder of the thesis is outlined as follows:

- Chapter 2 reviews related work in collision handling. In particular, the applicability of these methods for deformable objects and their resulting collision information is investigated and evaluated.
- Chapter 3 introduces a test suite for collision handling between deformable models. This collection of carefully selected experiments is carried out to analyze the characteristics and the performance of each collision handling technique presented in the following chapters.
- Chapter 4 discusses a spatial partitioning approach that can be used for the detection of n-body collisions and self-collisions between deformable objects. It is based on an optimized spatial hashing technique that efficiently finds intersections between various types of object primitives.
- Chapter 5 presents an image-space collision detection technique based on a volumetric approximation of the intersection volume. Three possible implementations are discussed. The first two variants are accelerated by graphics hardware whereas the third one is a software-only approach.
- Chapter 6 introduces a method to estimate a consistent penetration depth and direction based on the information provided by one of the collision detection approaches presented in the previous two chapters.
- Chapter 7 describes the framework for interactive simulation of dynamically deforming objects and the hysteroscopy simulator prototype. Both applications are built upon components presented in the previous chapters.
- Chapter 8 summarizes this thesis and points out possible directions for future research.

2 Related Work

The process of finding all collisions in a scene is usually divided into a broad phase and a narrow phase [Hub95, Zac01]. The broad phase takes all objects as input and determines groups of objects that potentially collide, so called collision islands. This collision pruning reduces the number of objects that must be simultaneously treated by the near phase, thus improving the overall performance. The near phase performs exact collision tests between all objects within the same collision island. The output is a list of collisions with appropriate collision information that can be used to compute an adequate collision response (see Fig. 2.1).

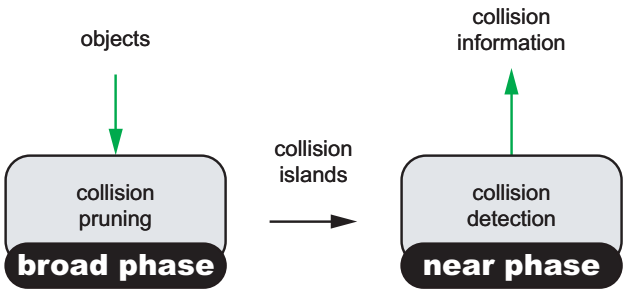


Figure 2.1: The collision detection process is usually divided into a broad phase and a near phase.

This chapter presents one dedicated method that can be used to perform deformable collision pruning during the broad phase in Sec. 2.1. Various collision detection methods for the near phase are presented in Sec. 2.2, followed by a brief discussion in Sec. 2.3 about the resulting collision information. This thesis is then motivated in Sec. 2.5 based on an evaluation of all the reviewed collision handling methods with respect to the target applications (see Sec. 7). The focus throughout the whole chapter lies on methods suitable for deformable models in interactive environments.

2.1 Collision Pruning

Collision pruning in the broad phase is usually done by Sweep and Prune (SAP) or by exact collision detection approaches that are adapted to consider only an

approximation, e.g. the bounding volume, of the models (see Sec. 2.2 for details about this type of methods).

Sweep and Prune, presented in [Coh95], maintains a sorted list for each of the principal axes. The elements in the lists are intervals obtained by projecting the geometry of the objects onto each axis. These lists are regularly updated with current projections and sorted with insertion sort. Whenever a maximum and a minimum swap positions on the list, a pair of intervals either begins or ceases to intersect. A pair of objects can intersect only if their projected intervals intersect on all three axes. Collision islands of objects with overlapping intervals are easily found and maintained during the simulation. Sweep and Prune turns out to be a simple and efficient method when used with deformable objects.

Kinetic Sweep and Prune (KSP) extends the basic method via a kinetic data structure and supports continuous collision detection [Com05]. When paired with continuous intersection tests, it is guaranteed to not miss any collisions even if they happen in between consecutive time-steps.

2.2 Collision Detection

Collision detection methods can be classified into five groups: Bounding-volume hierarchies, spatial subdivision, image-space techniques, stochastic methods and distance fields. Each class and their corresponding collision detection methods are presented in the following sections.

2.2.1 Bounding-Volume Hierarchies

Bounding-volume hierarchies (BVHs) have proven to be among the most efficient data structures for collision detection between rigid bodies. This is mostly due to the fact that BVHs can be pre-computed prior to running the simulation. Unfortunately, this pre-processing is not optimal for deformable objects since their BVHs must be continuously updated during the simulation.

The basic idea of BVHs is to recursively partition the object primitives and build a tree-like data structure. Internal nodes of the tree have links to all their child nodes, whereas leaf nodes hold references to the associated object primitives. Furthermore, each node in the tree contains a bounding volume (BV) that encloses the associated primitives or child nodes with a smallest containing instance of some specified type of shapes.

In the past, a wealth of shape types has been explored, such as spheres [Hub96, Pal95], oriented bounding boxes (OBB) [Got96], discrete-oriented polytopes (DOP) [Klo96], Boxtrees [Zac02, Aga02], axis-aligned bounding boxes (AABB) [Ber97, Lar01], spherical shells [Kri98] and convex hulls [Ehm01]. Sophisticated BV types

such as convex hulls tend to enclose the geometry very tight but the update and collision test are computationally expensive. Simpler types, such as AABBs, can be computed much faster but may result in many unnecessary hierarchy traversal steps due to the poor approximation of the geometry.

Several strategies exist to construct BVHs, such as top-down [Got96], bottom-up [Rou85] and insertion [Gol87]. For deformable objects, it is further advantageous to cluster primitives not only based on their initial position but also on their connectivity (see [Vol94, Vol95, Pro97]). In general, any strategy should try to minimize the volumes of the children to achieve optimal BVHs [Zac02].

For the collision test of two objects, the BVHs are traversed top-down and pairs of tree nodes are recursively tested for overlap. If the overlapping nodes are leaves of the BVH, then the enclosed primitives are tested for exact intersection. If only one node is a leaf while the other one is an internal node, the leaf node is tested against each of the children of the internal node. If, however, both of the nodes are internal nodes, it is tried to minimize the probability of intersection as fast as possible. Therefore, the node with the smaller volume is tested against the children of the larger node (see [Ber97]). This recursive procedure quickly zooms in on pairs of nearby primitives. Large branches of the tree can be skipped as soon as the BV of the corresponding root node is found to be non-overlapping.

In contrast to rigid objects, hierarchies for deformable objects need to be updated in each time-step. The hierarchy can either be refitted or rebuilt. Refitting is much faster than rebuilding, but for large deformations, the BVs tend to be less tight and have larger overlap volumes. Nevertheless, refitting is about ten times faster compared to a complete rebuild of an AABB hierarchy [Ber97].

The number of unnecessary node updates can be reduced using a hybrid method [Lar01]. The top half of the tree is updated bottom-up whereas nodes in the bottom half of the tree are only updated top-down when they are reached during the collision query. The drawback of this method is a higher memory requirement because information about the primitives must not only be stored in the leaf nodes but also in internal nodes.

Other approaches have been proposed to further accelerate the hierarchy update by omitting or simplifying the update process for several time-steps (see [Mez03]). For this purpose, the bounding volumes are inflated by a certain distance. As long as the enclosed primitives do not move farther than this distance, the hierarchy does not need to be updated.

If the deformation of the object is an interpolation between two or more morph targets or a superposition of displacement fields, then the update can be performed at very little extra costs (see [Lar03] and [Jam04]).

2.2.2 Spatial Subdivision

Spatial subdivision is a simple and fast technique to accelerate collision detection in case of moving and deforming objects. In contrast to BVHs, which work in object space, spatial subdivision methods partition the primitives in world space.

The basic idea is to subdivide space into regions, called cells. Each cell maintains a list of object primitives that are fully or partially contained in the cell. The main difficulty in spatial subdivision is the choice of the data structure that is used to represent the 3D space. This data structure has to be flexible and efficient with respect to computational time and memory.

Early spatial subdivision approaches have been proposed for neighborhood queries, e. g. in molecular dynamics. The computation of molecular atom interaction is accelerated with a neighborhood search based on a uniform 3D grid (see [Lev66]). Another related approach is presented in [Rab76], where a hash map is used to represent the 3D grid.

Several spatial subdivision schemes have been proposed for collision detection such as uniform grids [Tur90, Gan00, Zha00], octrees [Gre93, Ban95], BSP trees [Fuc80, Mel00] or kd-trees [Tel91]. Spatial hashing has been applied to collision detection for the first time in [Tur90]. A hierarchical extension was presented in [Mir97] as part of a robot motion planning algorithm which is restricted to rigid bodies. Combining spatial and object subdivision into a hybrid approach has been shown in [Gre99]. A perfect multidimensional hash function can be used to pack sparse primitive data into a compact table while retaining efficient random access [Lef06]. However, this method is limited to rigid objects, since the perfect hash construction is a nontrivial and thus costly operation.

When constructing spatial data structures, the objects can either be considered for subdivision or they can be ignored. Object-dependent approaches, such as BSP trees or kd-trees, split the 3D space based on the position and orientation of well selected object primitives. Whereas the subdivision strategy of object-independent approaches, such as uniform grids or octrees, is solely controlled by global parameters like cell size or subdivision level.

Collision detection based on spatial subdivision is performed by first discretizing the primitives of all objects into the cells. Then, within each cell that contains more than one primitive, exact intersection tests are carried out to find collisions. This clustering into cells highly reduces the number of necessary collision tests since object primitives have to be tested only against other primitives sharing the same cell.

For moving and deforming objects, it is necessary to update the spatial data structures in each time-step. Updating object-dependent data is a lot more time-consuming than object-independent data making the latter a much better choice for

deformable objects. An interesting approach that allows adjacent cells to overlap each other and thus reduce update costs has been presented in [Tha00].

2.2.3 Image-Space Techniques

Recently, several image-space techniques have been proposed for collision detection [Tes05]. Since they usually do not require any pre-processing, they are especially appropriate for environments with dynamically deforming objects. Furthermore, image-space techniques can commonly be implemented using graphics hardware and thus benefit from potential hardware acceleration.

The basic idea is to process projections of objects to accelerate collision queries. This projection is usually done by rendering the object primitives into the frame buffer. Since image-space techniques work with discretized representations, they do not provide exact collision information. The accuracy of the collision detection depends on the discretization error. Thus, accuracy and performance can be balanced in a certain range by changing the resolution of the rendering process.

An early approach to image-space collision detection of convex objects has been outlined in [Shi91]. In this method, the two depth layers of convex objects are rendered into two separate depth buffers. At each pixel the interval from the smaller depth value to the larger depth value approximately represents the object and is efficiently used for interference checking. A similar approach has been presented in [Bac99]. Both methods are restricted to convex objects which makes them not applicable to geometrically complex or even deformable objects.

In [Mys95], an image-space technique is presented which detects collisions for arbitrarily-shaped objects. In contrast to [Shi91] and [Bac99], this approach can also process concave objects. However, the maximum allowed depth complexity is limited. Additionally, object primitives have to be pre-sorted. Due to the required pre-processing, this method cannot efficiently work with deforming objects.

A first application of image-space collision detection to dynamic cloth simulation has been presented in [Vas01]. In this approach, an avatar is rendered from a front and a back view to generate an approximate representation of its volume. This volume is used to detect penetrating cloth particles. A first image-space approach to collision detection in medical applications is presented in [Lom99], where the intersection of a surgical tool with deformable tissue is detected by rendering the interior of the tool.

In [Hof01], an image-space method is not only employed for collision detection, but also for proximity tests. This method is restricted to 2D objects. In [Kim02] and [Kim02b], closest-point queries are performed using bounding-volume hierarchies along with a multi-pass rendering approach. In [Kno03], edge intersections with surfaces can efficiently be detected in multi-body environments. However, the pro-

posed technique is not robust in case of occluded edges. In [Bac02, Gov03, Gov05], several image-space methods are combined for object, sub-object pruning and collision detection. The approach can handle objects with changing topology. However, the setup is highly involved and does not scale well to more complex scenes.

Image-space techniques can be accelerated with graphics hardware. However, due to buffer read-back delays and the limited flexibility of programmable graphics hardware, it is not guaranteed that implementations on graphics hardware are faster than specialized software solutions. As a rule of thumb, graphics hardware should only be used for geometrically complex objects.

While image-space techniques efficiently detect collisions, they are limited in providing information that can be used for collision response by physically-based simulation environments. In many approaches, further post-processing of the provided result is required to compute or to approximate information such as the penetration depth and direction of colliding objects.

2.2.4 Stochastic Methods

So called inexact or stochastic methods have become a focus in collision detection research recently. These approaches are motivated by several observations. First, polygonal models are just an approximation of the true geometry. Second, the perceived quality of most interactive 3D applications does not depend on exact simulation, but rather on real-time response to collisions [Uno97, Sul03]. At the same time, humans cannot distinguish between physically-correct and visually-plausible behavior of objects [Bar96]. Therefore, depending on the application, it can be tolerated to improve the performance of collision detection while degrading its precision.

A naive approach to stochastic collision detection is the selection of random pairs of features as input to an exact intersection test. This method can be augmented by ensuring that the sampling covers features from the entire body and that they are already close enough. Exploiting temporal coherence can further increase the performance, because a pair of close features at the current time-step may still be interesting in the next time-step (see [Lin91]).

Combining stochastic methods with other collision detection approaches permits to efficiently trade accuracy for speed. In [Kim04b], close features of the objects are found by tracking randomly selected pairs of geometric primitives within a hierarchy of DOPs. The bounding-volume hierarchy is used to narrow the regions where random pairs are generated, therefore fewer random samples are necessary. Additionally, by using a lazy hierarchy update, the cost in each time-step can be significantly reduced.

Highly flexible structures, such as strands and cloth, have the possibility of self-colliding at multiple points in space [Bar03]. The general stochastic collision

approach is adapted in [Rag04] to detect collisions and self-collisions for such objects. The proposed method utilizes two optimizations. First, a two-step update method is used for computing the local distance minima for surface structures. Second, collisions are propagated from the collision point in order to provide a robust collision response. When a collision occurs, a recursive algorithm searches the neighborhood for possible collisions where a unique response is applied.

A different approach to stochastic collision detection is presented in [Kle03]. Conceptually, the main idea of the algorithm is to consider sets of polygons at inner nodes of a BVH. But instead of storing the polygons explicitly within the BVH, only the probability of the existence of a pair of intersecting polygons is estimated and stored. This has two advantages. First, the algorithm is truly time-critical. The application can control the runtime of the algorithm by specifying the desired quality of the collision detection. Second, the probabilities can guide the algorithm to those parts of the BVH that allow for faster convergence of the estimate. In contrast to traditional traversal schemes, the algorithm is guided by the probability that a pair of BVs contains intersecting polygons.

2.2.5 Distance Fields

Distance fields specify the minimum distance to a closed surface for all points in the field. The distance may be signed in order to distinguish between inside and outside regions. Representing a closed surface by a distance field is advantageous, because there are no restrictions about topology. Furthermore, the evaluation of distances and normals needed for collision detection and response is extremely fast and independent of the geometric complexity of the object.

Different data structures for representing distance fields have been proposed in the literature. A uniform grid with stored distance information poses the simplest form of a distance field. The drawbacks are the huge memory requirements and the limited resolution when representing objects with sharp features.

In order to overcome these problems, an adaptively sampled distance field (ADF) was proposed in [Fri00]. The data is stored in a hierarchy which is able to increase the sampling rate in regions of fine detail. Although various spatial data structures are suitable in general, ADFs are usually stored in an octree. For collision detection purposes, special care has to be taken in order to guarantee continuity between different levels of the tree. Whenever a cell is adjacent to a coarser cell, its corner values have to be changed to match those of the interpolated values at the coarser cell [Wes99, Bri03].

When using a BSP tree for distance fields, memory consumption can be reduced even further [He97, Wu03]. This is achieved by using a piecewise linear approximation of the distance field, which is not necessarily continuous. Unfortunately,

the construction of a BSP tree is computationally expensive. Another problem may arise from discontinuities between cells, since these cracks are not as easily resolved as for ADFs.

Deformed distance fields have been used to estimate the penetration depth of elastic polyhedral objects [Fis01]. In this method, an internal distance field is created by a fast marching level set method while propagating distance information only within the objects. During collision detection the distance fields are deformed due to the geometry and used for an approximation of the penetration depth. Although the distance field is only partially updated in deforming regions of the object, experiments indicate that this method is not intended for real-time applications.

Updates of a distance field are a common bottleneck of all methods described above. Several image-space approaches exist that accelerate these updates with graphics hardware [Vas01, Hof01, Sig03, Sud04]. The presented results, however, suggest that the generation is still not fast enough for interactive applications, where distance fields of multiple deformable models have to be updated during run-time.

2.2.6 Continuous Collision Detection

Most well-known collision detection methods are discrete. They sample the objects trajectories at discrete time instances and report intersections only. Discrete collision detection algorithms are generally simple to implement and therefore used very frequently in interactive simulations. Unfortunately, they also exhibit various weaknesses. Besides the lack of physical realism due to the penetration, these methods can miss collisions when objects are too thin or moving too fast. Adaptive time-steps or predictive methods can be used to correct the artifacts in off-line applications, but are usually not applicable to interactive applications that require a relatively high and constant frame-rate.

Recently, several algorithms have been proposed for continuous collision detection (CCD). These approaches model the trajectory of the object between successive discrete time instances as a continuous path and check the resulting path for collisions. Different techniques have been used to model the trajectory including linear interpolation between the vertex positions [Hub95, Bri02], screw motion [Kim03] or arbitrary in-between motion [Red04]. At a broad level, the CCD algorithms can be classified into four approaches: algebraic equation solving approaches [Can86, Red00], swept volume (SV) techniques [Abd02], adaptive bisection [Red02, Sch02] and kinetic data structures (KDS) [Aga00]. These approaches have been used to perform CCD at interactive rates for rigid objects [Red02, Kim03] and articulated models or avatars in virtual environments [Red04, Red04b]. However, no efficient algorithms are known for real-time CCD between general deformable models.

2.3 Collision Information

In order to realistically simulate the behavior of colliding objects, an appropriate collision response has to be considered. One idea commonly used in discrete-time simulations is to generate forces which eventually separate colliding objects. These response or penalty forces are computed for penetrating object vertices as a function of their penetration depth which represents the distance and the direction to the surface of the penetrated object. In case of deformable objects, this force computation is intended to reflect the fact that real colliding objects deform each other. The deformation induces forces in the contact area which are approximated with penetration depth approaches in virtual environments. Response forces commonly consider additional features such as friction which is computed as a function of the relative velocity of colliding structures and their penetration depth.

Penetration depth approaches work very well for sufficiently densely sampled surfaces and in case of small penetrations. However, in interactive discrete-time simulations with discretized object representations, these two requirements are rarely met. Depending on the size of the simulation time-step, large penetrations can occur which result in the computation of non-plausible penetration depths and directions. Furthermore, discrete surface representations can result in discontinuous penetration directions.

Contact models and collision information for rigid and deformable bodies are well-investigated. Analytical methods for calculating the forces between dynamically colliding rigid bodies have been presented in [Moo88, Hah88, Bar89, Bar91, Bar93, Bar94, Fau96, Pau04]. These approaches solve inequality-constrained problems which are formulated as linear complementarity problems (LCP). In addition to analytical methods, a second class of collision response schemes is based on so-called penalty forces. These approaches calculate response forces based on penetration depths in order to resolve colliding objects. First solutions have been presented in [Ter87, Pla88]. Penalty-based approaches have been used in simulations with deformable objects, cloth and rigid bodies [Moo88, McK90, Des99]. A third approach which directly computes contact surfaces of colliding deformable objects is presented in [Gas93].

Due to their computational efficiency, penalty-based approaches are very appropriate for interactive simulations of deformable objects. They can consider various elasto-mechanical object properties. Friction and further surface characteristics can also be incorporated. Penalty forces are computed based on penetration depths and there exist many approaches that compute the exact or approximative penetration depth of two colliding objects which is defined as the minimum translation that one object undergoes to resolve the collision. Exact penetration depth computations can be based on Minkowski sums [Cam86, Gui86] or hierarchical object

presentations [Dob93], while approximative solutions based on the GJK algorithm [Gil88] and iteratively expanding polytopes have been presented in [Cam97, Ber01]. Further methods are based on object space discretizations [Fis01], employ graphics hardware [Hof02, Sud04] or introduce incremental optimization steps [Kim04].

While existing approaches very efficiently compute the minimal penetration depth, they do not address inconsistency problems of the result in discrete-time simulations. A solution to this problem are continuous collision detection methods (see Sec. 2.2.6). However, these approaches are computationally expensive compared to discrete collision detection and not appropriate for deformable objects.

2.4 Deformable Models

Deformable models are not in the scope of this thesis but are a necessary prerequisite for any deformable collision detection technique. This section gives a very brief overview over the various approach in this field. A detailed survey can be found in [Nea05].

Many methods and models have been proposed in computer graphics to simulate deformable objects ranging from finite difference approaches [Ter87], mass-spring systems [Bar98, Des99], the Boundary Element Method (BEM) [Jam99], the Finite Element Method (FEM) [Deb01, Mue02, Mue04c], the Finite Volume Method (FVM) [Ter03] to implicit surfaces [Des95] and mesh-free particle systems [Des96, Ton98, Mue04b].

In addition to approaches which mainly focus on the accurate simulation of elasto-mechanical properties, there exist several acceleration strategies. Robust integration schemes for large time-steps have been investigated [Bar98] and multi-resolution models have been proposed [Deb01, Cap02, Gri02]. To further improve the performance, modal analysis approaches have been employed which can trade accuracy for efficiency [Pen89, She02, Jam02]. Further, data-driven methods have been presented where the pre-computed state space dynamics and pre-computed impulse response functions are incorporated to improve the run-time performance [Jam99]. In [Met92], dynamic models have been derived from global geometric deformations of solid primitives such as spheres, cylinders, cones or super-quadrics.

The test suite (see Chap. 3) and target applications of this thesis (see Chap. 7) employ variants of the deformation models presented in [Tes04] and [Mue05].

The first approach considers deformable solids that are discretized into tetrahedrons and mass points. In order to compute the dynamic behavior of objects, forces are derived at mass points from potential energies. These forces preserve distances between mass points, the area of the object surface and the volume of each tetrahedrons. The material properties of a deformable object are described by weighted stiffness coefficients of all considered potential energies. The model considers elastic

and plastic deformation and handles a large variety of material properties ranging from stiff to fluid-like behavior which makes it a suitable method to be used in surgery simulation.

The main idea of the second approach is to replace energies by geometric constraints and forces by distances of current positions to goal positions. These goal positions are determined via a generalized shape matching of an undeformed rest state with the current deformed state of the of mass points. Since points are always drawn towards well-defined locations, the overshooting problem of explicit integration schemes is eliminated. The versatility of the approach in terms of object representations that can be handled, the efficiency in terms of memory and computational complexity and the unconditional stability of the dynamic simulation make the approach particularly interesting for computer games.

Finally, it is important to note that the collision handling methods presented later in this thesis are very general and not bound to a specific deformable model. Therefore, a wide range of applications should be able to directly benefit from this work.

2.5 Evaluation

This section briefly evaluates the different collision detection classes when applied to surgery simulation or other highly interactive environments such as computer games. These kinds of applications define the context of this thesis (see Sec. 1.1) and require efficient and robust collision handling techniques for deformable objects (see Sec. 7.1 and 7.2).

Tab. 2.1 shows the different collision detection classes and their ratings in seven criteria derived from the requirements of deformable collision detection presented in Sec. 1.2. The rating for each criterion is either good (+), neutral (0) or bad (-). Please note that there are several methods in each class and each of them features slightly different strengths and weaknesses (see Sec. 2.2).

No class satisfies all the necessary criteria for an optimal collision detection method for deformable objects. However, spatial subdivision looks like the best overall candidate. Its main weakness is the inherent large memory requirement to store the world cells. This issue motivates the optimized spatial hashing method presented in Chap. 4. It is also based on spatial subdivision but addresses the high memory consumption with the use of a hashing scheme. This leads to an optimal collision detection solution with respect to the aforementioned context. Unfortunately, the method does not map well onto graphics hardware.

Image-space techniques are much better in exploiting the computational power of programmable GPUs. One such method based on LDIs, an approximate volumetric object representation, is presented in Chap. 5. It is motivated by several weaknesses

Class	Def.	Self-coll.	Pre.	N-body	Info.	Mem.	GPU
BV Hierarchies	0	+	-	-	+	0	no
Spatial Subdivision	0	+	+	+	+	-	no
Image-Space Techniques	0	0	+	0	0	0	yes
Stochastic Methods	+	-	+	-	+	0	no
Distance Fields	-	0	-	-	+	-	yes
Opt. Spatial Hashing	+	+	+	+	+	+	no
LDI	+	+	+	+	0	0	yes

Table 2.1: Evaluation of collision detection classes based on the following criteria: Applicability to deformable objects (Def.), self-collision support (Self-coll.), minimal preprocessing (Pre.), n-body support (N-body), collision information quality (Info.), memory usage (Mem.) and graphics hardware acceleration (GPU). The rating is either good (+), neutral (0) or bad (-).

of other graphic hardware accelerated techniques such as insufficient self-collision and n-body support.

An inherent flaw of discrete-time simulations are collision response artifacts such as deep penetrations due to large time-steps. This open issue motivated the development of a post-processing step that highly reduces artifacts and yields more consistent and plausible collision response (see Chap. 6).

3 Test Suite

This chapter introduces a test suite for collision handling between deformable models. The suite consists of six experiments that are carried out to analyze the characteristics and the performance of each collision handling technique presented in the following chapters.

3.1 Overview

The experiments of the test suite are carefully selected to represent a broad range of scenarios that typically appear in interactive simulation environments. The following properties are considered:

- The **scene complexity** of the experiments ranges from a single object to several dozens of objects. This allows for testing the scalability of the collision handling methods with respect to the total number of objects in the scene.
- Varying the overall **geometric complexity** of the models leads to a detailed analysis of the performance when the methods deal with either low-detail or high-detail objects.
- The experiments have different types of **rest states**. Some of them end in a contact-free state whereas other scenes finish with a stack of resting objects.
- The **model variety** in terms of complexity, shape and physical properties is chosen to be different from one experiment to another.
- Detection of **self-collisions** is only enabled in the last two experiments to reflect the fact that many interactive applications do selectively consider them.

3.2 Setup

The test suite is part of an interactive simulation environment for deformable objects which is discussed in Sec. 7.1. This application includes all collision handling methods presented in the thesis and allows for robust measurements of computational cost, number of collisions and other collision information during the simulation.

The objects in the test cases are all deformable and internally represented as tetrahedral meshes. Their visual appearance is improved by adding a high-resolution surface mesh to them. This mesh is deformed based on the underlying, coarser simulation geometry which is further used to perform collision detection and response.

Tab. 3.1 shows the geometric complexity and the physical material properties of the objects used throughout the test suite. The underlying deformable model is based on the work presented in [Tes04] which computes the dynamic behavior of objects based on forces derived at mass points from potential energies. These forces preserve distances between mass points and volumes of tetrahedrons. The material properties of a deformable object are described by weighted stiffness coefficients of these considered potential energies.

Object	Vertices	Tetrahedrons	Vertex	Stiffnesses		Damping
			Mass	Distance	Volume	Coeff.
Head	66	143	0.02	100	10000	0.5
Ball	13	20	0.1	200	200	0.8
Pitbull	2482	7691	0.001	10	10	0
Membrane	242	500	0.01	200	100	0.2
Cow	332	969	0.01	200	500	0.3
Teddy	338	852	0.01	200	5000	0.3
Rod	909	2000	0.01	50	300	0.2

Table 3.1: Objects used in the test cases and their geometric complexity and material properties.

To be able to compare the performance of the different collision handling methods the simulation is pre-recorded and played back. This ensures that the comparisons are based on the same simulation data. The measurements do not include collision detection against the ground and walls of the environment.

3.3 Test Cases

The following Sec. 3.3.1 - 3.3.6 give a detailed description of the six test scenes. A summary of the experiments can be found in Sec. 3.3.7.

3.3.1 Boldie Scene

The focus of this experiment lies on testing the performance of many low-resolution objects that experience multiple collisions and finally find a stable rest state.

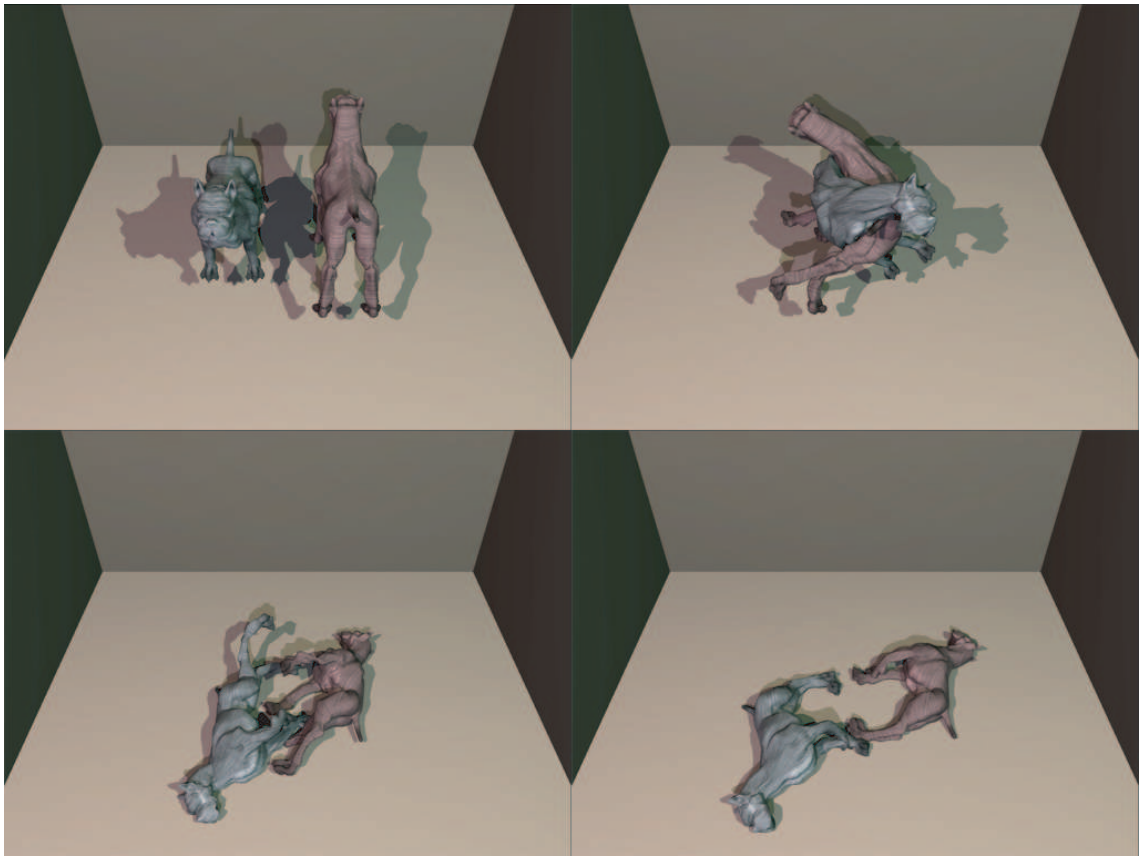
Section	Experiment	Objects	Vertices	Tetrahedrons	Self-collision
3.3.1	Boldie	97	1313	2062	no

The experiment starts with a single head model inside a box. The bottom of the model is fixed to the ground (top-left image). After a few seconds, dozens of deformable ball models are dropped from the sky (top-right image). They either collide with the head or the ground which results in a multitude of dynamic contacts between the objects. All models in the scene are elastically deformed as a result of these collisions (bottom-left image). The scene comes to a complete rest at the end of the experiment where multiple resting contacts ensures that the models do not penetrate each other (bottom-right image).

3.3.2 Pitbull Scene

The focus of this experiment lies on testing the performance of two high-resolution objects that experience an intense collision impact before separating again.

Section	Experiment	Objects	Vertices	Tetrahedrons	Self-collision
3.3.2	Pitbull	2	4962	19380	no

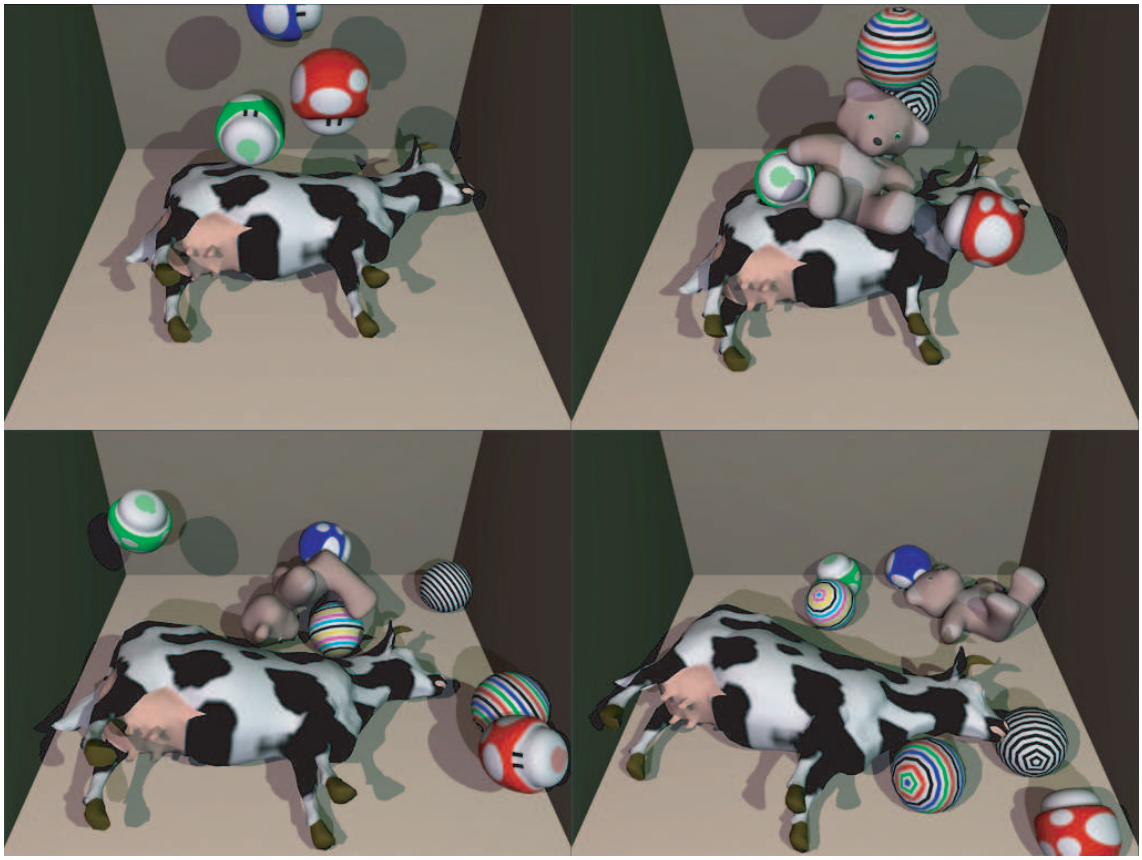


At the beginning of the experiment two opposing pitbull models approach each other with a high velocity (top-left image). An intense impact happens when they meet in midair and both models heavily bend in response to this (top-right image). Still in contact, the two models fall to the ground and start separating again (bottom-right image). In the end, they reach their final position and only a few number of resting contacts remain between their feet (bottom-right image).

3.3.3 Mixed Scene

The focus of this experiment lies on testing the performance of collision handling between several objects with different geometric or physical properties, such as small-large, convex-concave, light-heavy and soft-rigid models.

Section	Experiment	Objects	Vertices	Tetrahedrons	Self-collision
3.3.3	Mixed	8	746	1939	no

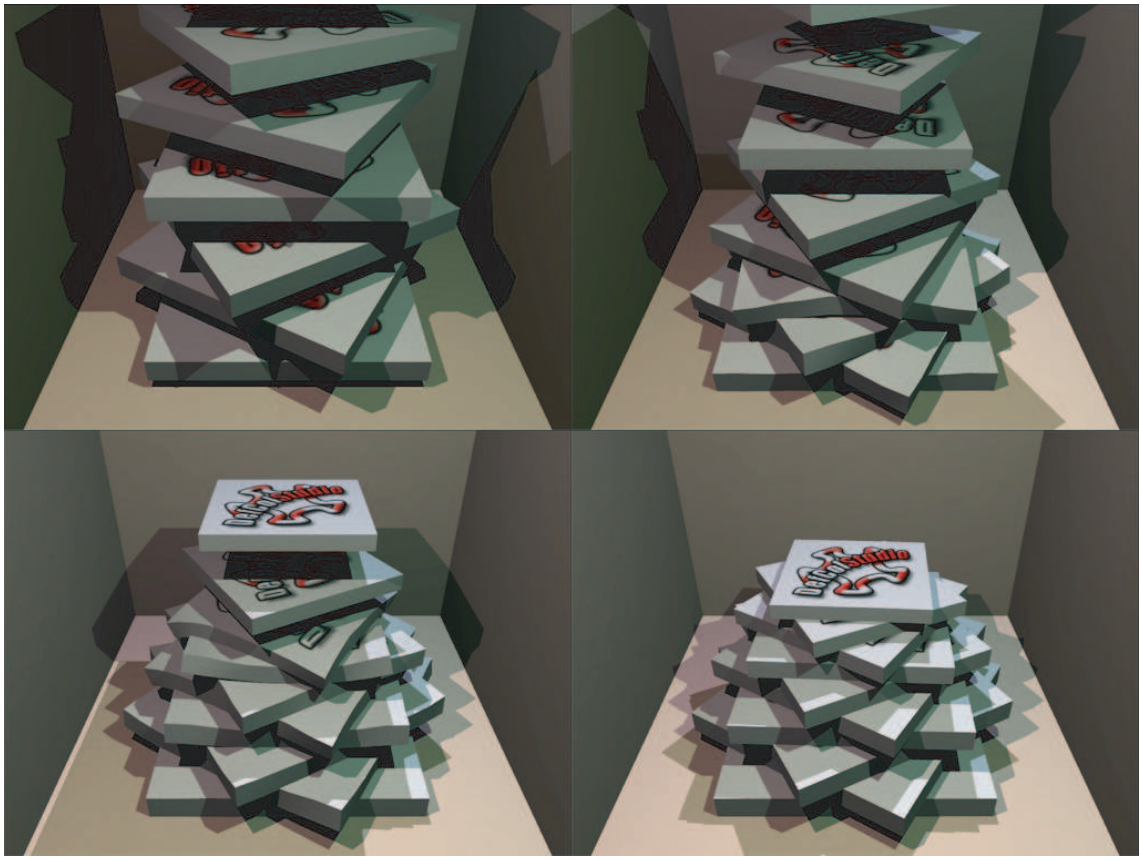


The models in this scene are dropped from a pre-defined height at the beginning of the experiment (top-left image). Multiple dynamic collisions start to happen when the lowest model reaches the ground (top-right image). The objects bounce away from each other due to the collision response and the elastic deformation (bottom-left image). After a while the scene comes to a rest when all models are completely separated again (bottom-right image).

3.3.4 Membrane Scene

The focus of this experiment lies on testing the collision handling performance and consistency of a stack of deformable objects that are influenced by a large number of resting contacts.

Section	Experiment	Objects	Vertices	Tetrahedrons	Self-collision
3.3.4	Membrane	10	2420	5000	no

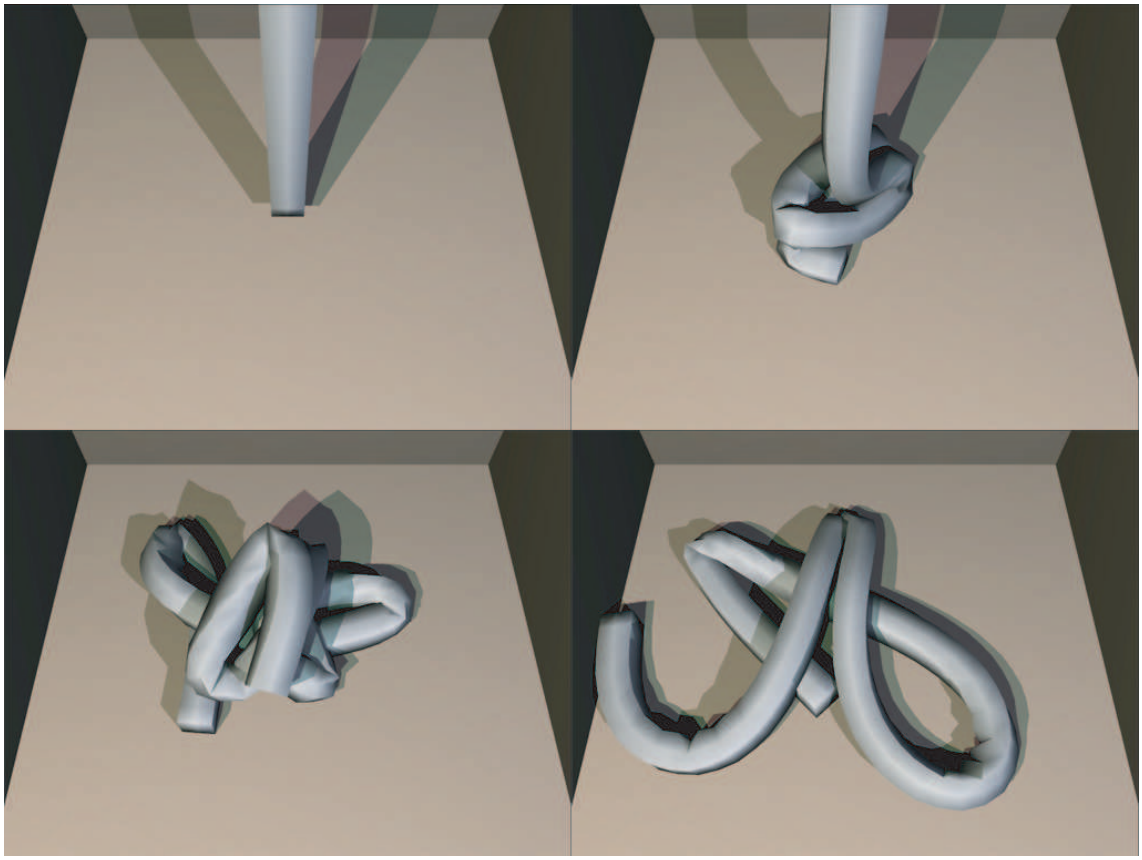


No collisions exist at the start of the experiment when the membrane models are dropped from a pre-defined height (top-left image). After the lowest membrane hits the ground the following objects start to eventually build a stack (top-right image). The collision response coupled with the elastic deformation leads to a separation of several membranes at the top (bottom-left image). The experiment ends when the stack of deformable models reaches an equilibrium due to the high number of resting contacts (bottom-right image).

3.3.5 Rod Scene

The focus of this experiment lies on testing the performance of a highly deformable model that experiences self-collisions.

Section	Experiment	Objects	Vertices	Tetrahedrons	Self-collision
3.3.5	Rod	1	918	2000	yes

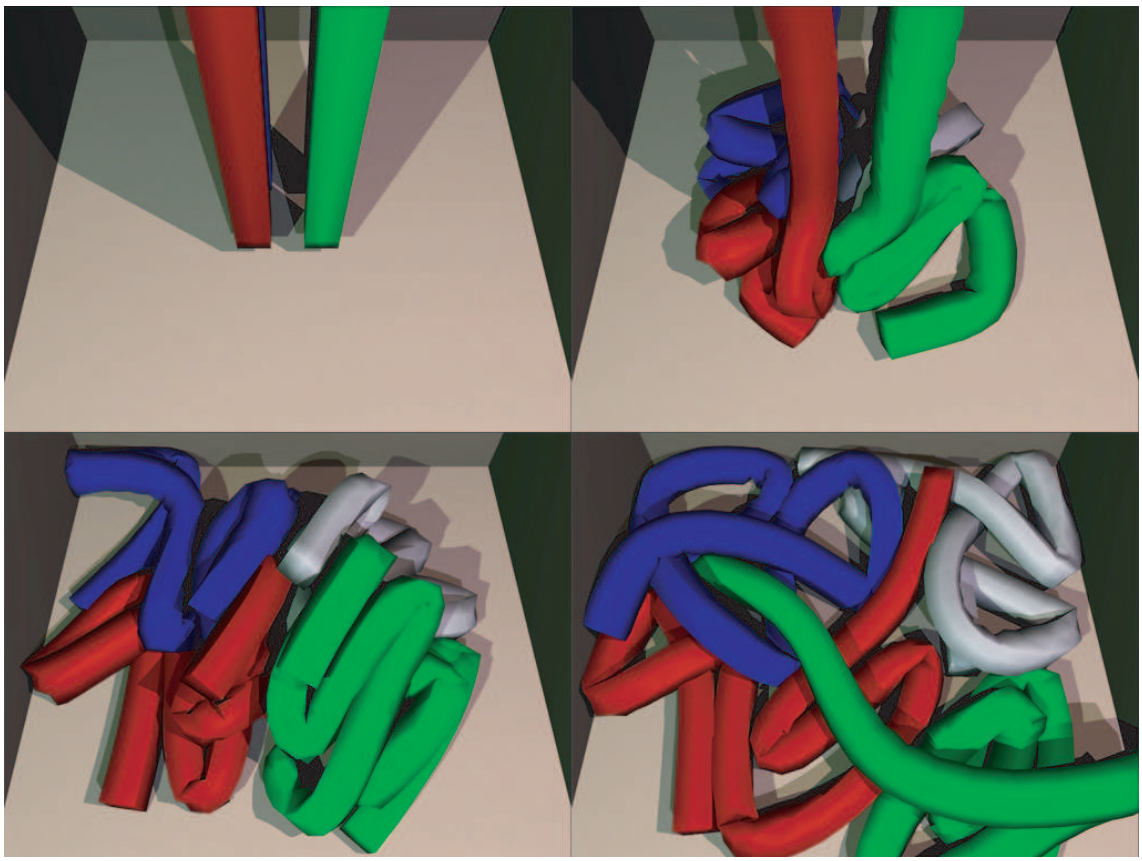


A long deformable rod falls to the ground at the beginning of the experiment (top-left image). After the initial contact with the ground the model deforms and self-collisions starts to occur (top-right image). Multiple dynamic self-collisions must be handled in the middle of the impact where the model is heavily deformed (bottom-left image). A few stable self-collisions still remain after the relaxation of the model when it reaches the the final rest state (bottom-right image).

3.3.6 MultiRod Scene

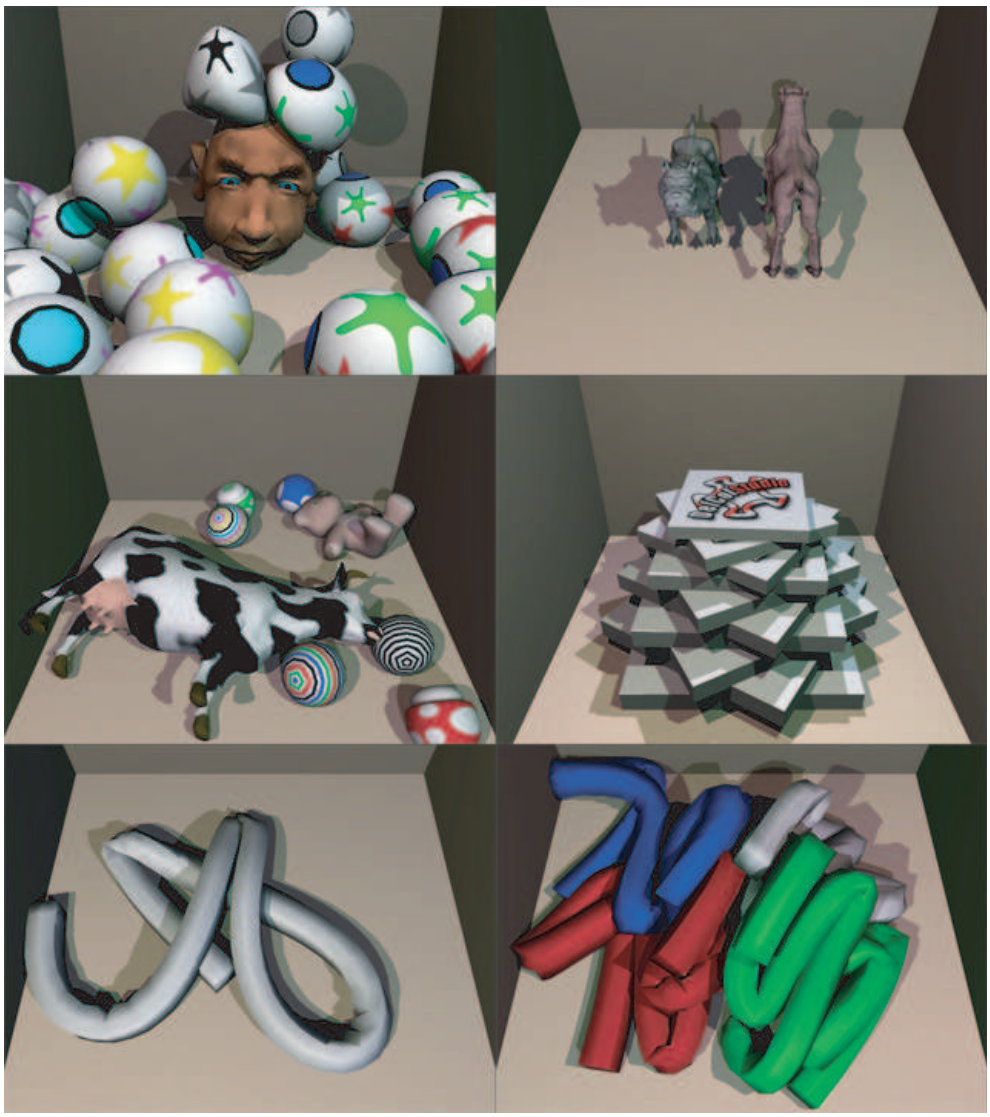
The focus of this experiment lies on testing the collision handling performance and consistency of highly deformable models that encounter collisions and self-collisions.

Section	Experiment	Objects	Vertices	Tetrahedrons	Self-collision
3.3.6	MultiRod	4	3672	8000	yes



Four long deformable rods are placed next to each other in midair at the start of the experiment (top-left image). The models are then dropped and start deforming and colliding when hitting the ground (top-right image). Many collisions and self-collisions emerge during the phase of the highest impact (bottom-left image). Many contacts remain when all objects in the scene reached their stable rest state (bottom-right image).

3.3.7 Scene Summary



Section	Experiment	Objects	Vertices	Tetrahedrons	Self-collision
3.3.1	Boldie	97	1313	2062	no
3.3.2	Pitbull	2	4962	19380	no
3.3.3	Mixed	8	746	1939	no
3.3.4	Membrane	10	2420	5000	no
3.3.5	Rod	1	918	2000	yes
3.3.6	MultiRod	4	3672	8000	yes

Table 3.2: Summary of all test scenes.

4 Optimized Spatial Partitioning

As evaluated in Sec. 2.5, spatial subdivision is the most promising class of collision detection methods in the context of this thesis. However, its main weakness is the inherent large memory requirement for the world cells and the difficulty to choose their optimal shape and size.

These issues motivated the optimized spatial hashing method presented in this chapter. It is also based on spatial subdivision but addresses the high memory consumption with the use of a hashing scheme. High performance is achieved by investigating all involved parameters and identifying their optimal setting. This leads to a highly efficient collision detection solution that is very well suited for the interactive applications presented in Chap. 7.

Sec. 4.1 describes the different stages of the algorithm and possible implementations for several object primitive types. The parameters that influence the performance of the method are presented and optimized in Sec. 4.2. Various experiments are carried out after a closer look at the time complexity of the algorithm in Sec. 4.3. A final discussion in Sec. 4.4 about the advantages and limitations of the method is followed by directions for future research that concludes this chapter.

4.1 Algorithm

The basic idea of the algorithm is to first subdivide space into uniform rectangular regions, called cells. Each cell of this grid maintains a list of object primitives that are fully or partially contained in the cell. This clustering of object primitives highly reduces the number of necessary collision tests since object primitives have only to be tested against other primitives sharing the same cell.

Spatial subdivision is a simple and effective method, but the infinite number of cells that are required to span the entire 3D space poses a significant implementation problem. One solution is to restrict the partitioning to a certain region in space. However, in a dynamic environment, this restricted region needs to be continuously updated in order to enclose all moving and deforming objects.

A more efficient way is to employ a hash function that maps cells to a finite number of hash table entries. This mapping is not unique and eventually leads to so called hash collisions. However, these hash collisions can easily be detected and resolved.

Algorithm Overview

The algorithm consists of the following two stages illustrated in Fig. 4.1:

- **Stage 1** discretizes all object primitives with respect to 3D cells and maps them to hash table entries.
- **Stage 2** performs exact intersection tests between object primitives sharing the same hash table entry.

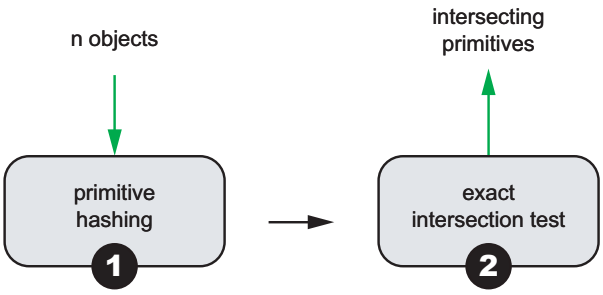


Figure 4.1: The two stages of the spatial partitioning algorithm: 1) Primitive hashing and 2) exact intersection test.

The versatility of spatial hashing allows for collision queries between many different types of object primitives from several objects at the same time. The only requirement is an appropriate discretization method and an intersection test suited for the specific object primitive types. The rest of this chapter investigates two of the most useful collision queries:

1. Collision and self-collision test between point and tetrahedron, which is necessary when checking solid against solid (i.e. tetrahedral meshes).
2. Collision and self-collision test between line-segment and triangle, which is necessary when checking surface against surface (i.e. triangle meshes).

4.1.1 Primitive Hashing

The first stage of the algorithm discretizes points, line-segments, triangles and tetrahedrons into cells in order to map them to entries in the spatial hash table. Given the cell coordinates (i, j, k) with respect to a chosen grid cell size and origin, the index h into the hash table is

$$h = \mathcal{H}(i, j, k) \tag{4.1}$$

where $\mathcal{H}$ is an appropriate hash function (see Sec. 4.2.1). For the rest of this chapter, without loosing generality, the origin of the grid is assumed to be at $(0, 0, 0)$.

The grid cell size, which is used during primitive hashing, influences the number of object primitives that are mapped to the same hash index. In case of larger cells, the number of primitives per hash index increases and the overall intersection test slows down. On the other hand, object primitives may also cover a larger number of cells if the cell size is significantly smaller than the object primitive. This leads to more collision tests in a larger number of hash entries. The performance thus highly depends on an appropriate size of the grid cells as well as the hash table. Both are investigated in more details later in this chapter (see Sec. 4.2.2 and 4.2.3).

Point Hashing

Spatial hashing of points is straightforward since points can only be contained in one single cell at a time (see Fig. 4.2). Given a point $\mathbf{p} = (x, y, z)$, the indices (i, j, k) of the corresponding cell are

$$(i, j, k) = (\lfloor \frac{x}{s_{grid}} \rfloor, \lfloor \frac{y}{s_{grid}} \rfloor, \lfloor \frac{z}{s_{grid}} \rfloor) \tag{4.2}$$

where s_{grid} is the grid cell size. The index h into the hash table can then be computed with 4.1.

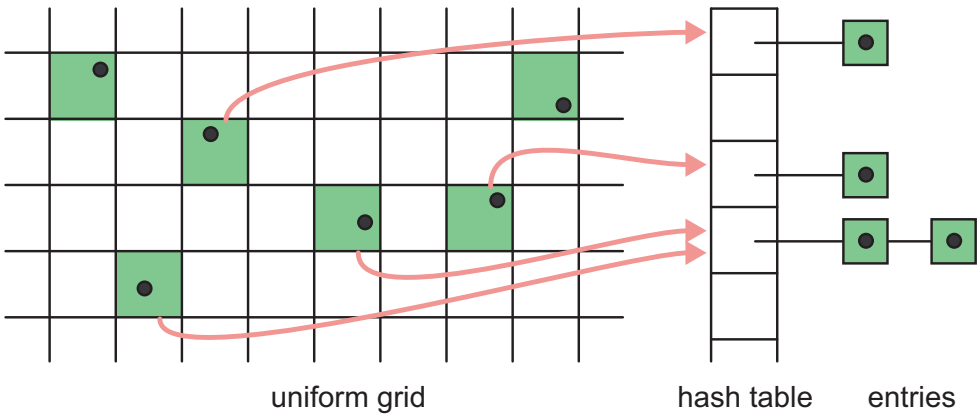


Figure 4.2: Points lie in only one grid cell and are hashed into a single hash table entry.

Line-Segment Hashing

In contrast to points, line-segments may intersect multiple cells (see Fig. 4.3). The following voxel traversal algorithm is used to discretize a line-segment into cells:

Algorithm *VoxelTraversal*
(∗ Voxel traversal technique [Ama87] ∗)
Input: Start position (x_s, y_s, z_s), End position (x_e, y_e, z_e)
Output: Voxel list *vlist*

```
1.  (x, y, z) ←(floor(x_s), floor(y_s), floor(z_s))
2.  (d_x, d_y, d_z) ←(x_e − x_s, y_e − y_s, z_e − z_s)
3.  (t_x, t_y, t_z) ←((x_s − x) ∗ d_x, (y_s − y) ∗ d_y, (z_s − z) ∗ d_z)
4.  vlist ←∅
5.  for n ← 0 to abs(floor(x_e)−x)+abs(floor(y_e)−y)+abs(floor(z_e)−z)
6.      do vlist ←vlist ∪ Voxel(x, y, z)
7.          if t_x < t_y
8.            then if t_x < t_z
9.                     then x = x + sign(d_x)
10.                         t_x = t_x + abs(1/d_x)
11.                    else z = z + sign(d_z)
12.                        t_z = t_z + abs(1/d_z)
13.           else if t_y < t_z
14.                    then y = y + sign(d_y)
15.                        t_y = t_y + abs(1/d_y)
16.                    else z = z + sign(d_z)
17.                        t_z = t_z + abs(1/d_z)
18.  return vlist
```

The traversal algorithm consists of two phases: Initialization and incremental traversal. The initialization phase begins by identifying the cell in which the line-segment starts. It then computes the first crossing of the line-segment and the cell borders with respect to each dimension of the grid. The second phase incrementally chooses the next cell based on the nearest crossing until the line-segment ends. After each step, the nearest crossings on all dimenstions are adjusted accordingly.

Given the start position $\mathbf{p_0} = (x_0, y_0, z_0)$ and the end position $\mathbf{p_1} = (x_1, y_1, z_1)$ of a line-segment, the normalized input parameters of the above algorithm *VoxelTraversal* are

$$(x_s, y_s, z_s) = (\lfloor \frac{x_0}{s_{grid}} \rfloor, \lfloor \frac{y_0}{s_{grid}} \rfloor, \lfloor \frac{z_0}{s_{grid}} \rfloor) \tag{4.3}$$

$$(x_e, y_e, z_e) = (\lfloor \frac{x_1}{s_{grid}} \rfloor, \lfloor \frac{y_1}{s_{grid}} \rfloor, \lfloor \frac{z_1}{s_{grid}} \rfloor) \tag{4.4}$$

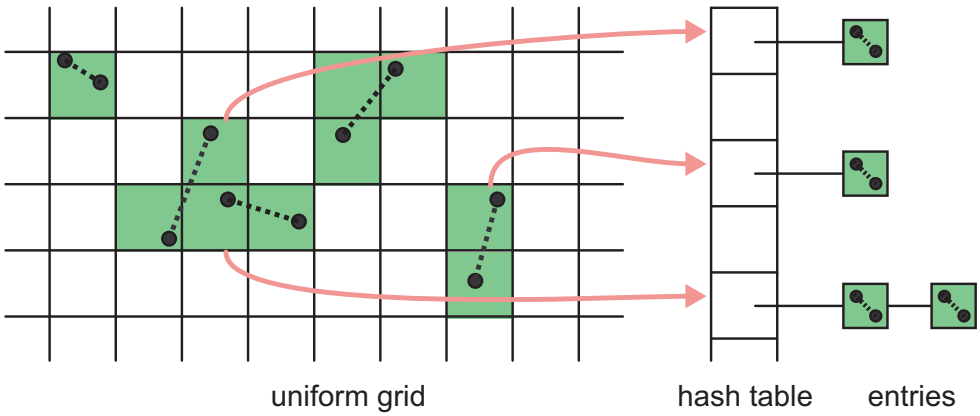


Figure 4.3: Line-segments may lie in several grid cells and are hashed into one or more hash table entries.

where s_{grid} is the grid cell size. The algorithm returns a list $vlist$ containing all voxels that are intersected by the line-segment. For each voxel in this list the corresponding hash table index h can be computed with 4.1.

Triangle and Tetrahedron Hashing

Similar to line-segments, triangles and tetrahedrons may intersect multiple cells (see Fig. 4.4). Voxelization methods for polygons and polytopes exist (see [Kau87]), but they tend to be inefficient for cases where the primitive is only contained in a few cells. Unfortunately, the hashing algorithm relies on a low number of intersected cells to achieve optimal performance. Therefore, a simpler but conservative hashing approach is used that performs better in such scenarios.

The hashing for triangles is done by first computing an AABB of the triangle. Then, for each cell that is intersected by the AABB, a test is performed that checks if the same cell is also intersected by the plane defined by the triangle. An efficient box-plane intersection test for this task can be found in [Gre94]. Finally, the triangle is hashed into all cells that pass the intersection test. Hashing for tetrahedrons is approximated even simpler: After computing the AABB of the tetrahedron, it is hashed into all cells which intersect its AABB.

The approximation of triangles by planes and tetrahedrons by boxes causes the primitives to be hashed into more cells than necessary. However, those false hash entries are discarded by the subsequent, exact intersection test between object primitives discussed in Sec. 4.1.2.

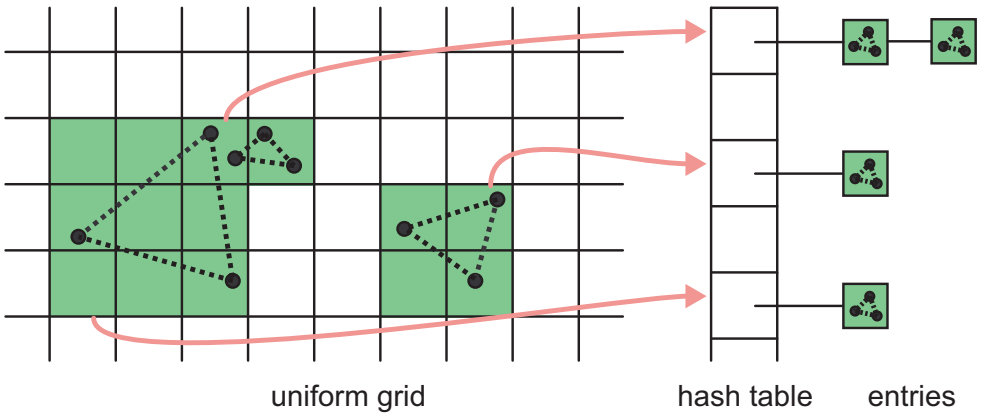


Figure 4.4: Triangles and tetrahedrons are hashed by discretizing a conservative approximation. This may lead to hashed cells that are not really intersected by the primitives themselves.

4.1.2 Intersection Test

In the first stage of the algorithm, object primitives are discretized into cells and then hashed into hash table entries. The second stage performs exact collision tests between all primitives within such a hash table entry. A collision is detected if an object primitive intersects another one. If both primitives belong to the same object, the test returns a self-collision.

Details about the two most useful tests, namely point against tetrahedron and line-segment against triangle, are given next.

Point - Tetrahedron Collisions

In order to reduce the number of necessary intersection tests and improve the performance, a point is first checked against the AABB of the tetrahedron. If the point is outside the AABB, the exact intersection test below can be skipped as no collision is possible.

The exact intersection test between a point $\mathbf{p} = (x, y, z)$ and a tetrahedron $\mathcal{T}$ spanned by the points

$$\begin{aligned} \mathbf{p_1} &= (x_1, y_1, z_1) \\ \mathbf{p_2} &= (x_2, y_2, z_2) \\ \mathbf{p_3} &= (x_3, y_3, z_3) \\ \mathbf{p_4} &= (x_4, y_4, z_4) \end{aligned}$$

is performed by calculating barycentric coordinates of $\mathbf{p}$ with respect to $\mathcal{T}$. These barycentric coordinates $\mathbf{b} = (b_1, b_2, b_3, b_4)$ are

$$(b_1, b_2, b_3, b_4) = (\frac{D_1}{D}, \frac{D_2}{D}, \frac{D_3}{D}, \frac{D_4}{D}) \tag{4.5}$$

where the determinant D is first computed as

$$\mathbf{D} = \begin{vmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{vmatrix}$$

If $D = 0$, the tetrahedron is degenerated (i.e. the points are coplanar) and the collision test is aborted. If $D <> 0$ then D_1, D_2, D_3, and D_4 are computed as

$$\mathbf{D_1} = \begin{vmatrix} x & y & z & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{vmatrix} \qquad \mathbf{D_3} = \begin{vmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x & y & z & 1 \\ x_4 & y_4 & z_4 & 1 \end{vmatrix}$$

$$\mathbf{D_2} = \begin{vmatrix} x_1 & y_1 & z_1 & 1 \\ x & y & z & 1 \\ x_3 & y_3 & z_3 & 1 \\ x_4 & y_4 & z_4 & 1 \end{vmatrix} \qquad \mathbf{D_4} = \begin{vmatrix} x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \\ x & y & z & 1 \end{vmatrix}$$

Finally, to be inside the tetrahedron $\mathcal{T}$, the barycentric coordinates $\mathbf{b} = (b_1, b_2, b_3, b_4)$ of a point $\mathbf{p}$ must satisfy

$$\begin{aligned} b_1 &\geq 0 \\ b_2 &\geq 0 \\ b_3 &\geq 0 \\ b_4 &\geq 0 \\ b_1 + b_2 + b_3 + b_4 &= 1 \end{aligned}$$

Line-Segment - Triangle Intersections

Barycentric coordinates can also be used to test if a triangle $\mathcal{A}$ spanned by the points

$$\begin{aligned} \mathbf{p_1} &= (x_1, y_1, z_1) \\ \mathbf{p_2} &= (x_2, y_2, z_2) \\ \mathbf{p_3} &= (x_3, y_3, z_3) \end{aligned}$$

intersects a line-segment $\mathcal{L} = (\mathbf{p}, \mathbf{p_{end}})$, where $\mathbf{p}$ is the start-point and $\mathbf{p_{end}}$ the end-point of the line-segment.

Denoting $\mathbf{e} = \mathbf{p} - \mathbf{p_1}$, $\mathbf{e_1} = \mathbf{p_2} - \mathbf{p_1}$, $\mathbf{e_2} = \mathbf{p_3} - \mathbf{p_1}$ and $\mathbf{d} = \mathbf{p_{end}} - \mathbf{p}$, the barycentric coordinates $\mathbf{b} = (b_1, b_2)$ of the intersection point between line-segment $\mathcal{L}$ and triangle $\mathcal{A}$ are

$$\begin{bmatrix} b_1 \\ b_2 \\ t \end{bmatrix} = \frac{1}{(\mathbf{d} \times \mathbf{e_2}) \cdot \mathbf{e_1}} \begin{bmatrix} (\mathbf{d} \times \mathbf{e_2}) \cdot \mathbf{e} \\ (\mathbf{e} \times \mathbf{e_1}) \cdot \mathbf{d} \\ (\mathbf{e} \times \mathbf{e_1}) \cdot \mathbf{e_2} \end{bmatrix} \tag{4.6}$$

where t is the distance from $\mathbf{p}$ to the intersection point (see [Moe97] for more details).

The barycentric coordinates $\mathbf{b} = (b_1, b_2)$ and distance t of the intersection point fulfill

$$\begin{aligned} b_1 &\geq 0 \\ b_2 &\geq 0 \\ b_1 + b_2 &\leq 1 \\ 0 \leq t &\leq 1 \end{aligned}$$

if the line-segment $\mathcal{L}$ intersects with the triangle $\mathcal{A}$.

4.2 Parameters

The characteristics of the hash function, the size of the hash table and the size of a grid cell for spatial subdivision influence the performance of the presented algorithm. This section investigates these important parameters and identifies possible optimizations.

4.2.1 Hash Function

In the first stage of the algorithm, hash values are computed for all discretized primitives. These hash values should be uniformly distributed to guarantee an adequate performance of the algorithm. The hash function has to work with primitives of the same object, that are close to each other, and with primitives of different objects, that are farther away.

A hash function takes data values as input and returns an integer in the range of available indices into the hash table. In general, there are four main characteristics of a good hash function:

1. The hash value is fully determined by the data being hashed. If something else besides the input data is used to determine the hash value, then the result is not as dependent upon the input data, thus allowing for a bad distribution of the hash values.

2. The hash function uses all the input data. If the hash function does not use all the input data, then slight variations to the input data would cause an inappropriate number of similar hash values resulting in too many hash collisions.

3. The hash function uniformly distributes the data across the entire set of possible hash values. If the hash function does not uniformly distribute the data, a large number of hash collisions will result, cutting down on the efficiency of the hash table.

4. The hash function generates very different hash values for similar input data. In real world applications, many data sets contain very similar data elements. These data elements should still be distributable over the whole hash table.

There are several ways for constructing such a hash function (see [McK90b]). However, not all hash functions are equally suited for spatial hashing. In addition to the above criteria, spatial hashing requires that the hash function performs best when called many times on small input data. In order to reach the highest possible computational efficiency, a good choice for the hash function $\mathcal{H}$ is

$$\mathcal{H}(i,j,k) = (ip_1 \otimes jp_2 \otimes kp_3) \textbf{ mod } n \tag{4.7}$$

where $\otimes$ denotes an XOR-operation, (i,j,k) are the cell coordinates, n the hash table size and p_1, p_2, p_3 large prime numbers.

Due to its simplicity, this function can be evaluated very efficiently. All involved arithmetic instructions are trivial and natively supported on current CPUs. The proposed function still fulfills all the above requirements of a good hash function and performed very well in all simulation scenarios presented throughout this thesis.

4.2.2 Hash Table Size

The size of the hash table significantly influences the performance of the collision detection algorithm. Larger hash tables reduce the risk of mapping different 3D positions to the same hash index. Therefore, the algorithm generally works faster with larger hash tables. On the other hand, the performance slightly decreases for larger hash tables due to memory and cache management overhead. It is also known,

that hash functions work most efficiently if the hash table size is a prime number [Cor90]. Fig. 4.5 shows the performance of the algorithm for a typical scenario with a varying hash table size. If the hash table is significantly larger than the number of object primitives, the risk of hash collisions is minimal. A setting that works very well in practice is to choose the hash table size as a small multiple of the number of object primitives.

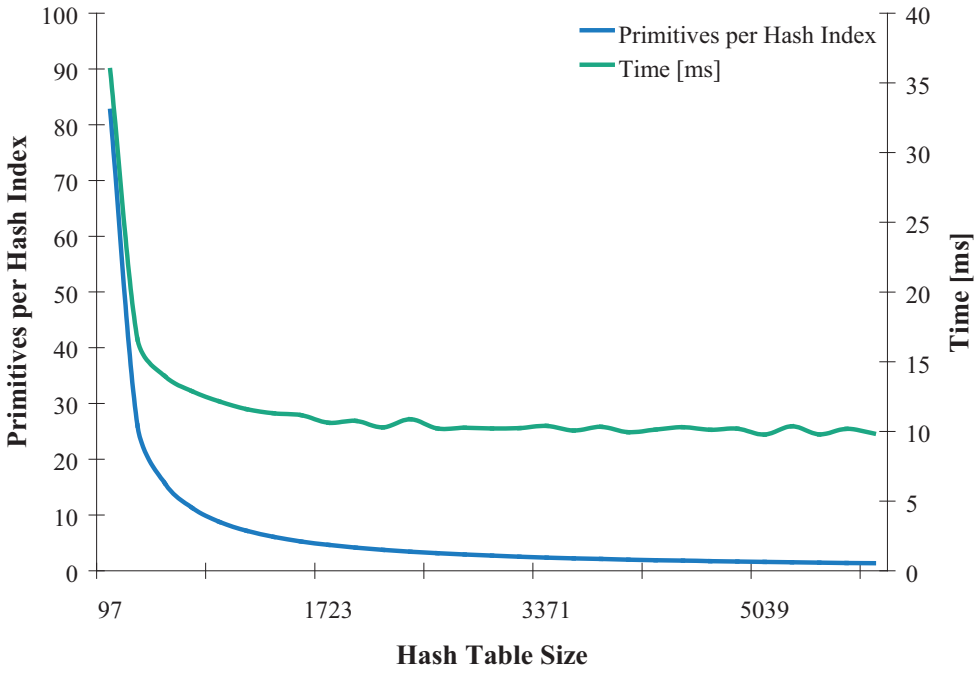


Figure 4.5: Influence of the hash table size on the performance.

The hash table size also affects the computational cost for initializing the underlying data structures. At the beginning of each simulation step, the hash table is cleared and each hash table index is initialized with an empty entry. Thus, the complexity of this process is linear in the number of hash table indices.

To avoid a full re-initialization of the hash table in each simulation step, the hash table indices are labeled with a unique time stamp. Furthermore, there exists a global time stamp that is incremented after each simulation step. Every time an object primitive is hashed to a hash table index, the corresponding time stamp is compared to the global time stamp. If they differ, the hash table index is first initialized before adding the new entry. This lazy initialization scheme can dramatically improve the performance of sparse scenes.

4.2.3 Grid Cell Size

The grid cell size affects the number of object primitives, that are mapped to the same hash table index. In case of larger grid cells, the number of primitives per hash table index increases and the intersection test slows down as more primitives have to be tested against each other. If the cell size is significantly smaller than the object primitives, they cover a larger number of cells and need to be checked against more hash table entries filled with object primitives.

A multitude of experiments have been carried out to find an optimal grid cell size. The test scenarios consist of several thousand tetrahedrons placed at random positions around the origin with varying density. Edge lengths of the tetrahedrons differ up to a factor of 10 to account for variable object discretizations. All experiments showed similar results and led to the same conclusion: The grid cell size should be equal to the average extent of all object primitives to achieve optimal performance. Fig. 4.6 illustrates a representative measurement of the influence of the grid cell size on the number of collision tests, the number of cells per object primitive and the collision detection time.

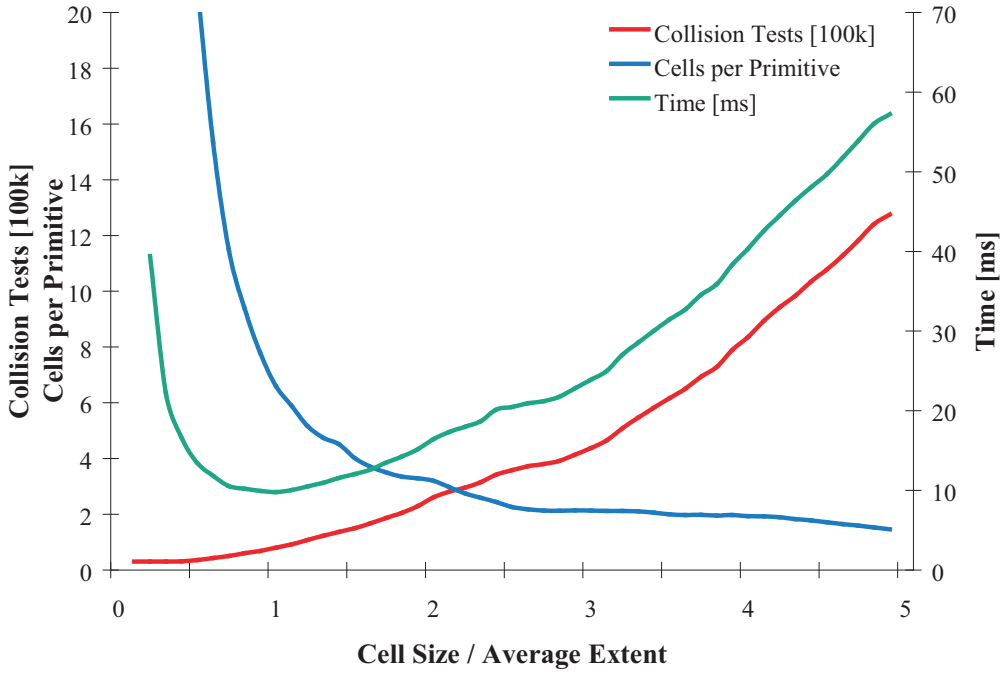


Figure 4.6: Influence of the grid cell size on the performance and the number of primitives per grid cell.

In practice, a good strategy is to initially set the grid cell size to the average extent of all existing objects in the scene. It should then be periodically updated

whenever objects are added to or removed from the scene. Changes in topology or geometric resolution of objects should also trigger an update.

A comparison of Fig. 4.5 and 4.6 illustrates, that the grid cell size has a more significant impact on the performance than the hash table size or hash function. The only exception is when the hash table size is much too small. Such configurations should be avoided as they will cause a huge amount of hash collisions and an excessive drop in performance.

4.3 Results

The first part of this section analyzes the theoretical time complexity of the optimized spatial hashing algorithm. The second part carries out the test suite from Chap. 3.

4.3.1 Time Complexity

Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be the two sets of object primitives that should be tested for collision against each other, and n_1 and n_2 the number of primitives in each set. To find all intersecting pairs, a naive approach would be to test all object primitives in $\mathcal{S}_1$ against all primitives in $\mathcal{S}_2$. This results in a time complexity in the order of $O(n_1 \cdot n_2)$. The goal of the spatial hashing approach is to reduce this complexity to $O(n_1 + n_2)$.

During the first pass, all primitives of $\mathcal{S}_1$ are inserted into the hash table. This takes $O(n_1 \cdot c)$ time where c is the average number of cells intersected by a primitive. The hash table can be lazy initialized (see Sec. 4.2.2), so the time complexity is independent of the hash table size.

In the second pass, the primitives of $\mathcal{S}_2$ are tested for collision against all primitives of $\mathcal{S}_1$ in the local neighborhood. The time complexity of this pass is in the order of $O(n_2 \cdot c \cdot p)$ where c is again the average number of cells intersected by a primitive and p the average number of primitives of $\mathcal{S}_1$ per cell.

If the cell size is chosen to be proportional to the average primitive extent then c is a constant. Furthermore, if the hash function does not produce hash collisions and the hash table size is chosen to be proportional to the number of primitives in the scene, the average number of primitives per cell p is constant too.

With both, c and p, being constant, the time complexity of the algorithm turns out to be linearly dependent on the number of primitives. Since deformation algorithms need to process all the primitives at each time step, linear time complexity for collision detection does not increase the overall time complexity of the simulation.

4.3.2 Experiments

The proposed method is part of an interactive simulation environment for deformable objects (see Sec. 7.1). Within this application, various experiments have been carried out to analyze the characteristics and the performance of the collision detection technique. A detailed description of the test scenarios can be found in Chap. 3, as they are referenced in several parts of this thesis. All timings presented in this section have been measured on a PC Pentium 4, 3.2 GHz. The hash table size is set to 49999, whereas the grid cell size is equal to the average edge length of all tetrahedrons in the corresponding test scene (as suggested in Sec. 4.2.2 and Sec. 4.2.3).

Boldie Scene

The first experiment consists of dropping dozens of balls on a head that is attached to the ground (see Sec. 3.3.1). The timing measurements in Fig. 4.7 show that no collisions exist at the beginning and during the first phase of the experiment. As the objects start to approach each other, the computational cost slightly increases. This is due to the increased number of nearby object primitives that are hashed into the same hash index. The second phase then shows a steep rise in the number of collisions when the balls hit the head and, in turn, other balls. The scene comes to a rest in the last phase while the number of collisions and the computational cost converge to an equilibrium. At the end of the experiment the collision detection needs 4.7 ms to find 782 resting contacts between the 97 objects in the scene. The maximum memory consumption for the hash table entries during the experiment is 10 KB. The average number of object primitives per non-empty grid cell is 9, the maximum is 15.

Pitbull Scene

This scene features two highly-detailed pitbulls which collide in midair (see Sec. 3.3.2). The heavy impact at the beginning of the experiment is clearly visible in the timing measurements (see Fig. 4.8). A total of 270 collisions are detected in 21.8 ms at the time of maximum penetration between the two objects. After a sharp decline, the number of collisions increases again when the two pitbulls fall on each others legs while hitting the ground. At the end of the experiment, 5 resting contacts remain which are detected in 16.8 ms. The maximum memory consumption for the hash table entries during the experiment is 39 KB. The average number of object primitives per non-empty grid cell is 18, the maximum is 35.

Mixed Scene

During this experiment, different types of models undergo several dynamic collisions before coming to a contact-free rest state on the ground (see Sec. 3.3.3). The collision detection needs between 1.75 and 2.05 ms to find all collisions per frame (see Fig. 4.9). Interestingly, the computational cost at the final rest state, when no collisions exist, is slightly higher than during some earlier phases with several collisions. The reason is that more object primitives are hashed into the same hash index when all objects lie on the ground due to the alignment and size of the hash grid cells. The maximum memory consumption for the hash table entries during the experiment is 6 KB. The average number of object primitives per non-empty grid cell is 19, the maximum is 29.

Membrane Scene

In this scene, ten membranes of different sizes are dropped to the ground where they eventually stack up (see Sec. 3.3.4). As expected, the computation cost sharply increases during the experiment as more and more collisions occur. Fig. 4.10 also shows several moments when the membranes shortly bounce away from each other, temporarily resulting in less collisions. Finally, 9.9 ms are required to find 1660 resting contacts between the 10 objects at the end of the experiment. The maximum memory consumption for the hash table entries during the experiment is 19 KB. The average number of object primitives per non-empty grid cell is 15, the maximum is 24.

Rod Scene

A highly deformable rod is tested for self-collisions in this experiment (see Sec. 3.3.5). During the phase of maximum impact the collision detection needs 3.5 ms to find the 283 self-collisions. At the end of the experiment, these numbers drop to 2.6 ms for 71 self-collisions due to the relaxation of the rod (see Fig. 4.11). An interesting artifact can be observed at the beginning of the timing measurements: The initial position and orientation of the rod, which is then exactly aligned to the hash grid cells, leads to a computational cost that is significantly higher than in later frames where the rod is already bent and better distributed to the hash indices. The maximum memory consumption for the hash table entries during the experiment is 7 KB. The average number of object primitives per non-empty grid cell is 8, the maximum is 17.

MultiRod Scene

Collisions and self-collisions of four rods are tested in this last experiment (see Sec. 3.3.6). The timing measurements in Fig. 4.12 show that a total of 1411 collisions and self-collisions are detected in 17.7 ms at the time of maximum impact. A total of 553 collisions and self-collisions remain when the objects finally find their rest states. This collision detection at the end of the experiment takes 13.4 ms per frame. The maximum memory consumption for the hash table entries during the experiment is 28 KB. The average number of object primitives per non-empty grid cell is 11, the maximum is 28.

4.4 Discussion

Spatial partitioning is a simple and fast technique to accelerate collision detection in case of moving and deforming objects. It is independent of topology changes and provides a straight-forward solution to self-collision detection. Spatial partitioning is not restricted to triangles, but naturally works with other primitive types if appropriate discretization algorithms and intersection tests are implemented.

The basic spatial partitioning can be extended by a hash function to optimize the memory footprint and overcome implementation problems, such as the need for potentially infinite number of grid cells. Instead of computing the global bounding box of all objects and explicitly performing a spatial partitioning, a hash function is used that maps 3D positions to hash table entries, thus realizing a very efficient, implicit spatial subdivision.

Various parameters influence the performance of the spatial hashing approach, such as the shape and size of a grid cells, the hash function and the number of the hash table entries. This chapter investigated these parameters and proposed several ways to optimize them.

A larger hash table decreases the chance of hash collisions which in turn increases the overall performance of the collision detection. The same is true if the hash function is carefully chosen so that it matches the characteristics of the spatial partitioning technique. Various experiments suggest that the optimal cell size is about the same as the extent of a single object primitive, which confirms a similar result that has been presented in [Ben77].

The presented method and its data structures are simple to implement, but turn out to be cache unfriendly due to the random memory access during the primitive hashing stage. This is most obvious when processing triangles and tetrahedrons which span multiple grid cells.

In contrast to other collision detection methods, spatial hashing does not show extreme performance fluctuations. This makes the computational cost more pre-

dictable which is important for interactive simulation environments. Unfortunately, there is a higher computational cost for trivial non-collision cases compared to other techniques. A simple solution to this problem is to use an efficient broad-phase based on SAP to find collision island and isolated objects. Spatial hashing is then performed for each collision island independently whereas isolated objects are completely ignored in later collision detection stages.

There exist numerous public domain collision detection libraries, such as RAPID [Got96], PQP [Lar99] and SWIFT [Ehm00]. A direct performance comparison of these methods with the optimized spatial partitioning is difficult, because they do not really address the same task. All above collision detection libraries are designed to work best with rigid bodies, whereas the spatial hashing has advantages when handling deformable objects. Therefore, it is not surprising that each category outperforms the other in their area of expertise.

The main advantage of the presented approach is its versatility. Despite the simple data structure, it allows to carry out n-body collision detection and self-collision detection at the same time and in one single pass. It is even applicable to related problems, such as proximity queries. The method supports deformable as well as rigid objects consisting of several primitive types. The collision information provided by the optimized spatial partitioning consists of pairs of intersecting object primitives. This output can directly be used to compute a suited collision response. In conclusion, the method is very efficient in physically-based simulation environments with dynamically changing spatial distributions of deformable objects.

Directions for future work

In very heterogeneous scenes where the objects are composed of different sized primitives, it is hard to find an optimal grid cell size. Unfortunately, the grid cell size is one of the parameters that directly influences the performance of the spatial partitioning method. Further investigations should lead to an approach that handles these scenarios in a more optimal manner. One possible way would be to apply hierarchical techniques to make the spatial hashing more adaptive to primitive sizes.

Spatial and temporal coherence is not exploited in the presented method. Such an extension could lead to better performance when many rigid, static or slow moving objects exist in the simulation. For example, static objects could permanently be stored in the hash table and do not need to be updated in every time step.

A perfect multidimensional hash function [Lef06] would help to minimize hash collisions and memory cache misses and thus increase the overall performance. Unfortunately, the construction of the perfect hash function is a nontrivial and costly operation.

Parallel architectures, such as multi-core processors and special purpose add-on

cards [Heg05], gain more and more popularity these days. The optimized spatial partitioning method could greatly benefit from such additional computing power. However, parallelizing the algorithm with its random memory access turns out to be difficult.

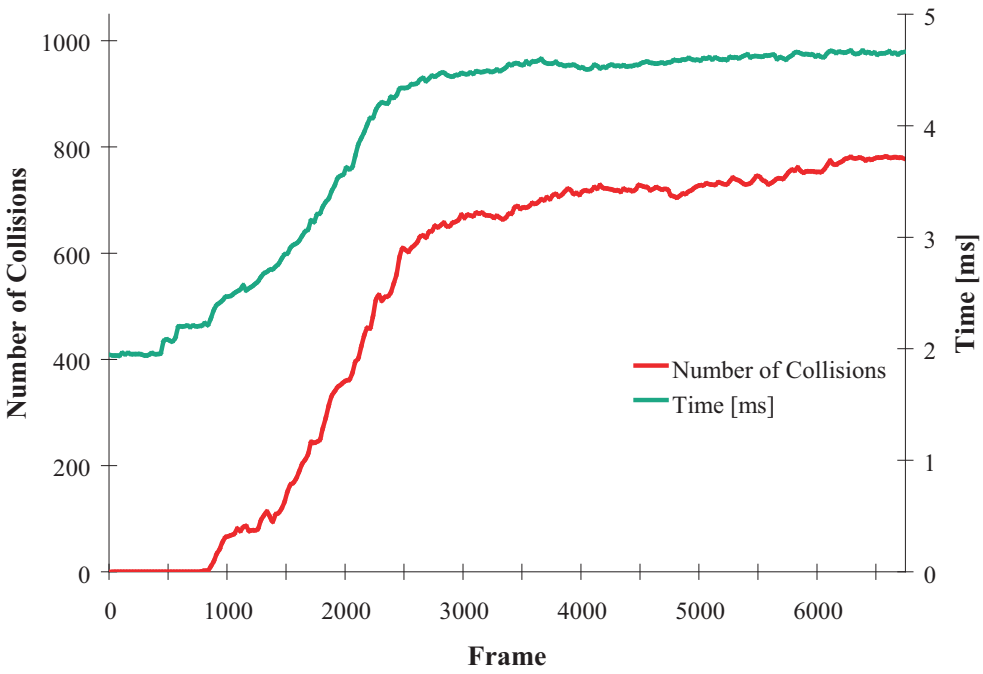


Figure 4.7: Experiment: Boldie scene.

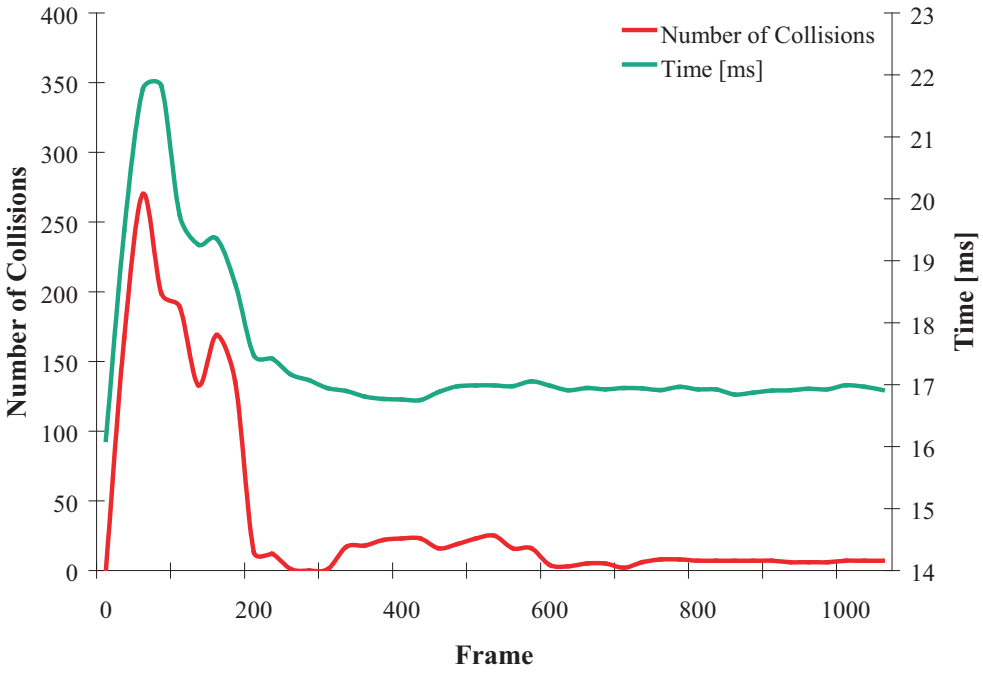


Figure 4.8: Experiment: Pitbull scene.

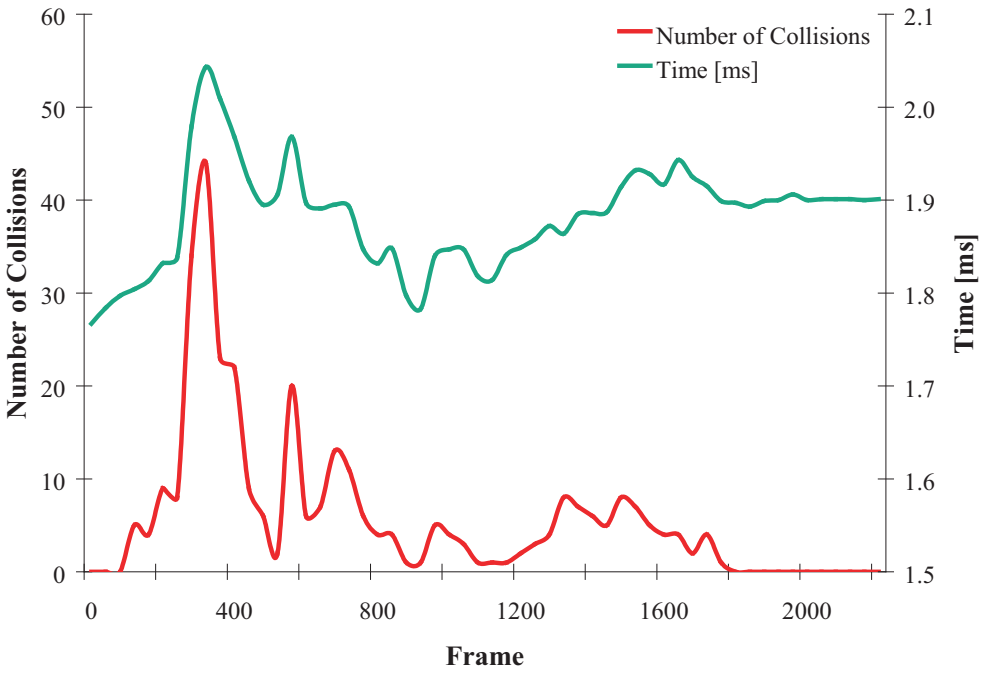


Figure 4.9: Experiment: Mixed scene.

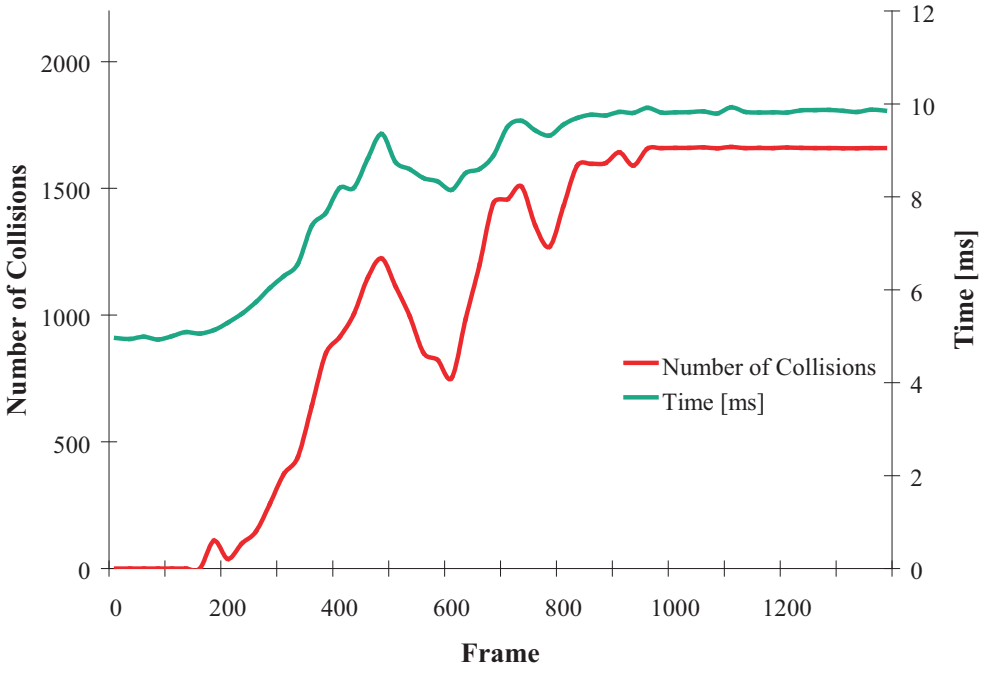


Figure 4.10: Experiment: Membrane scene.

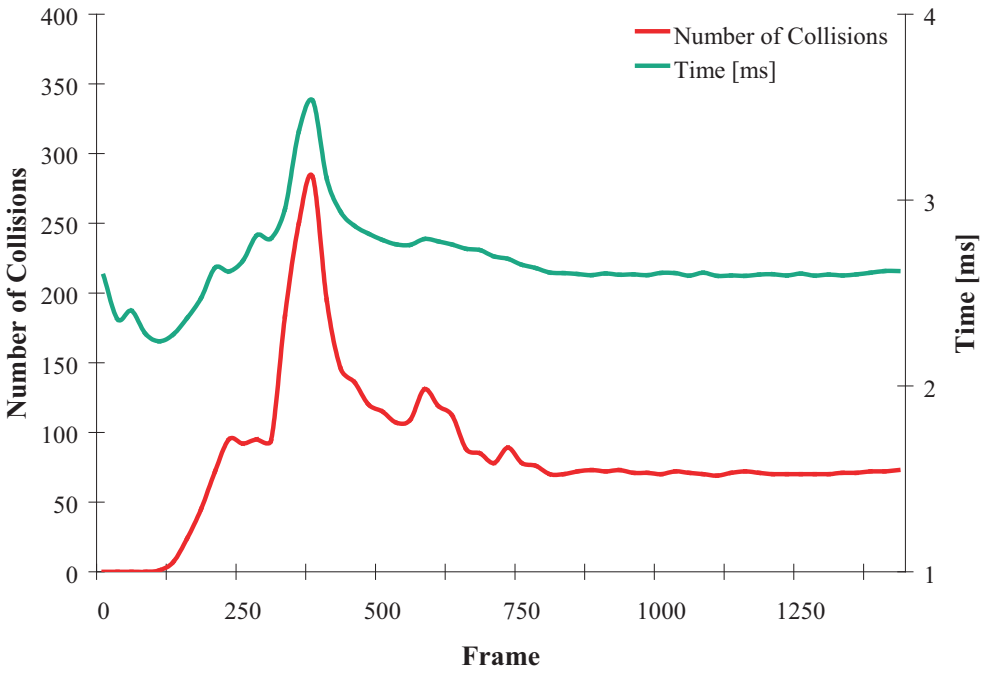


Figure 4.11: Experiment: Rod scene.

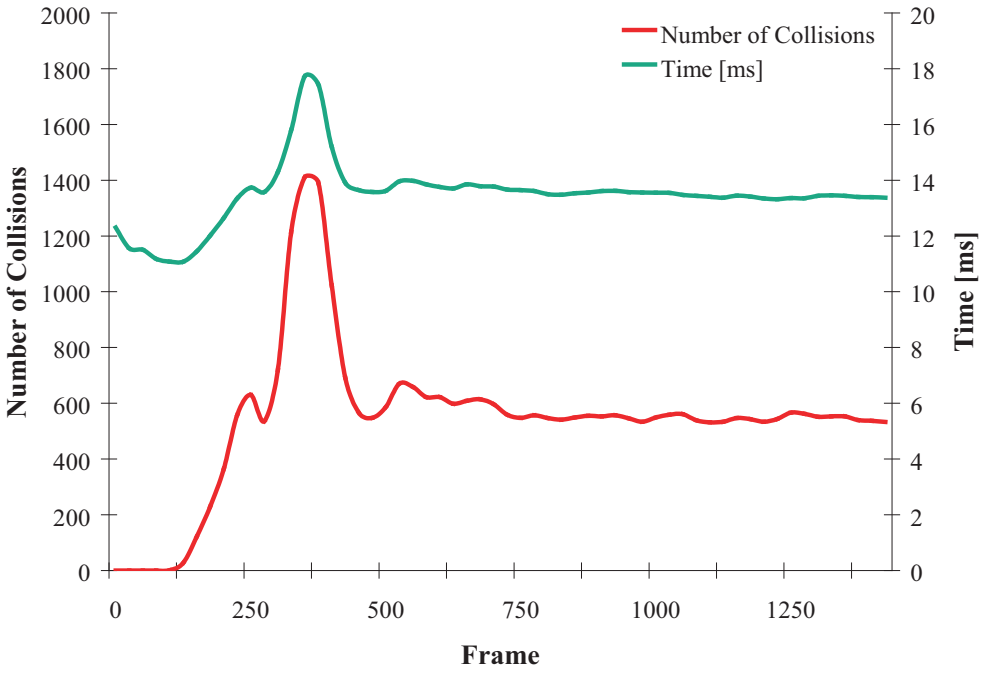


Figure 4.12: Experiment: MultiRod scene.

5 Image-Space Collision Detection

The optimized spatial hashing approach presented in the previous Chap. 4 is a highly efficient collision detection method for deformable objects. However, it does not map well onto graphics hardware to exploit the computational power of programmable GPUs.

Image-space techniques are much better suited for graphic hardware acceleration (see evaluation in Sec. 2.5). The method presented in this chapter is based on an approximate volumetric representation of the deformable objects and was motivated by several weaknesses, such as insufficient self-collision support, of other graphic hardware accelerated techniques.

Sec. 5.1 describes the different stages of the algorithm and explains the possible collision queries. Details of three different implementations of the algorithm can be found in Sec. 5.2. The performance of these implementations are then compared in Sec. 5.3 along with various other experiments. The chapter concludes with a discussion about the advantages and limitations of the method followed by directions for future research in Sec. 5.4.

5.1 Algorithm

The basic idea of the algorithm is to first compute an approximate volumetric representation of the involved objects which is then used for several variants of collision queries. The computation of the volumetric representation is done in image-space to take advantage of potential graphics hardware acceleration.

Algorithm Overview

The algorithm consists of the following three stages illustrated in Fig. 5.1:

- **Stage 1** computes the Volume-of-Interest (VoI). This VoI represents the volume where collision queries are actually performed.
- **Stage 2** generates a separate volumetric representation for each object inside the VoI. Note, that the generation is restricted to the VoI and all object primitives outside the VoI are discarded.
- **Stage 3** performs one (or more) of the following three collision queries:

a) Self-collision query of a single object.

b) Collision query between pairs of objects.

c) Collisions query between individual vertices and an object.

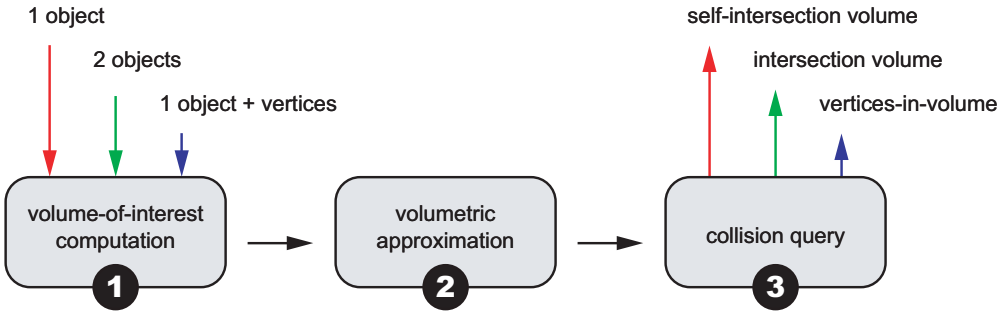


Figure 5.1: The three stages of the image-space algorithm: 1) Volume-of-Interest computation, 2) volumetric approximation and 3) collision queries.

5.1.1 Volume-of-Interest

The first stage of the algorithm computes a conservative region in space in which collisions may actually happen. This VoI is used as criterion for an early abort test and as domain for computations in the second and third stage of the algorithm.

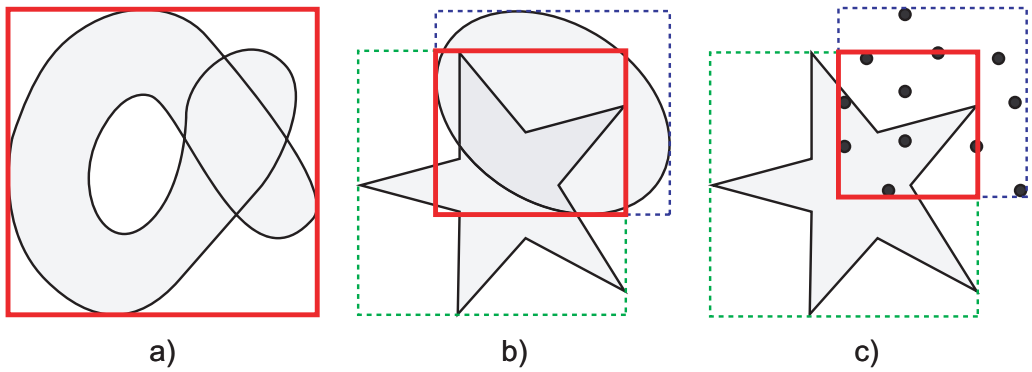


Figure 5.2: Volume-of-Interest for self-collision (a), collision between pairs of objects (2) and collision between an object and individual vertices (c).

When testing for self-collision, the VoI is given by the AABB of the entire object. For collision queries between pairs of objects or between individual vertices and an object, the VoI is computed as intersection of the two AABBs (see Fig. 5.2). If

the two AABBs do not overlap, the corresponding objects cannot interfere and the algorithm aborts. Otherwise, the intersection volume provides a VoI, which is considered for further processing. Although AABBs do not provide an optimal (i. e. smallest) bounding volume, they can be computed very efficiently. Furthermore, the intersection of two AABBs is again an AABB which keeps subsequent stages of the algorithm simple.

Alternative bounding volumes, such as oriented bounding boxes [Got96] or discrete-oriented polytopes [Klo96], provide tighter fitting object approximations. However, the computation of these representations is more involved and the resulting intersection volume can have a more complex shape making it much more difficult for further processing. In addition, such structures are impractical for deforming objects, which require a dynamic adaptation of the bounding volume.

The VoI consists of six faces defining the AABB. In order to simplify and accelerate further computations, the faces of the VoI have to meet the following two requirements:

1. To guarantee proper boundary conditions, there must exist at least one so-called *outside face* with respect to each object. A face of the VoI is an outside face if the object completely lies on one side of the plane defined by the face (see Fig. 5.3).

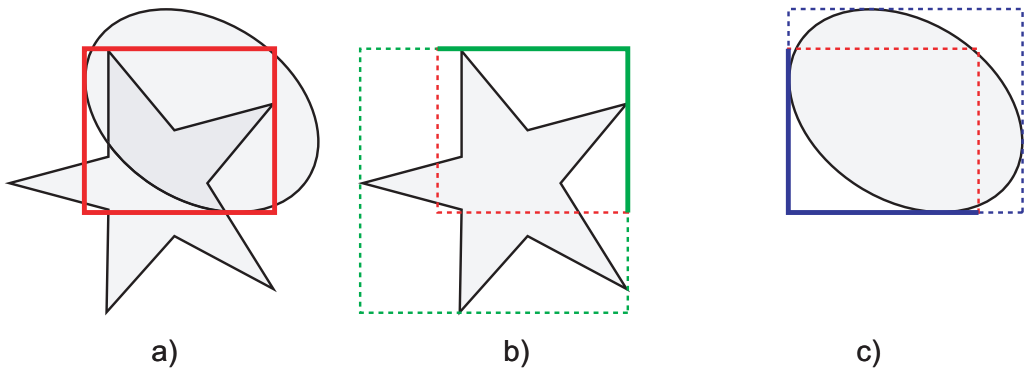


Figure 5.3: The VoI (a) defines outside faces for both involved objects (b and c).

2. Outside faces for pairs of objects should correspond to opposite sides of the VoI (see Fig. 5.4). This simplifies the combination of both volumetric representations in later stages.

In some cases, for instance when one box is entirely within another box, appropriate outside faces for both objects cannot be found. This problem is solved by extending the VoI. If the outside face of one object is fixed, the opposite face of the VoI can be scaled to touch the bounding box of the other object (see Fig. 5.5).

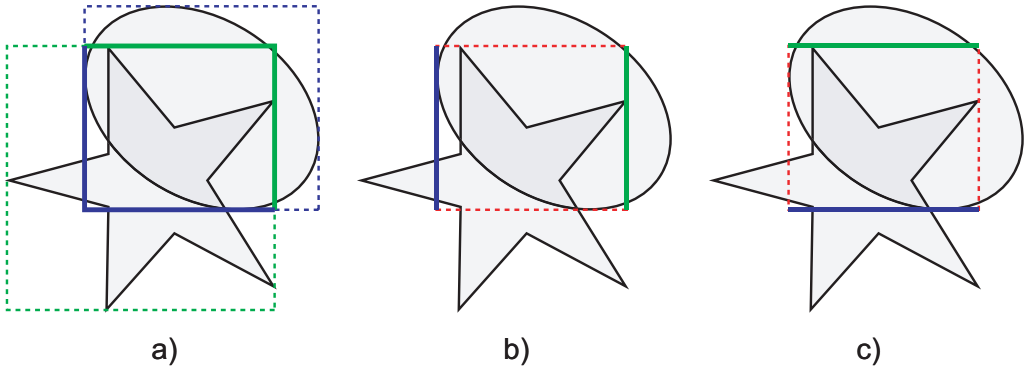


Figure 5.4: Outside faces (a) should correspond to opposite sides of the VoI. Two possibilities exist (b and c).

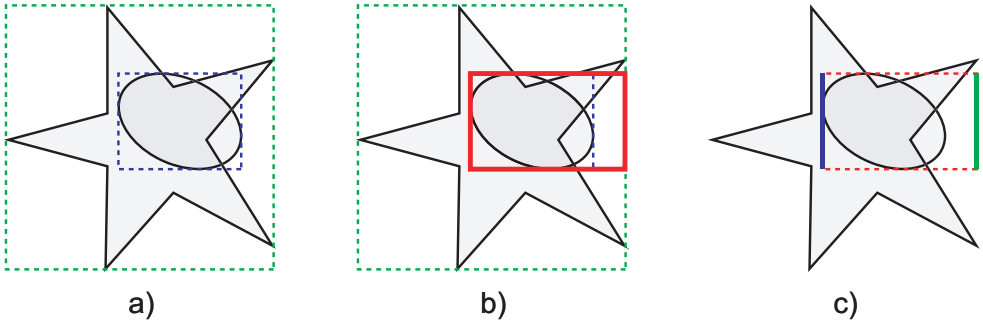


Figure 5.5: No opposite outside faces may be found (a). In this case, the VoI is extended (b) until a matching outside face pair is found (c).

5.1.2 LDI Generation

The second stage of the algorithm generates a separate *Layered Depth Image* (LDI) for each object. Discretized at a predefined resolution, these LDIs represent an explicit volumetric approximation of the corresponding object within the VoI.

An LDI basically consists of images or layers of depth values representing the object surface. The depth values of the LDI can be interpreted as intersections of parallel rays or 3D scan lines entering or leaving the object. Thus, an LDI classifies the VoI into inside and outside regions with respect to an object. The concept is similar in spirit to well-known scan-conversion algorithms that fill concave polygons [Fol90]. Intersections of a scan line with a polygon represent transitions between interior and exterior. The concept of using LDIs as a volumetric representation is exemplified in Fig. 5.6.

In order to generate an LDI, the object is rendered multiple times using an

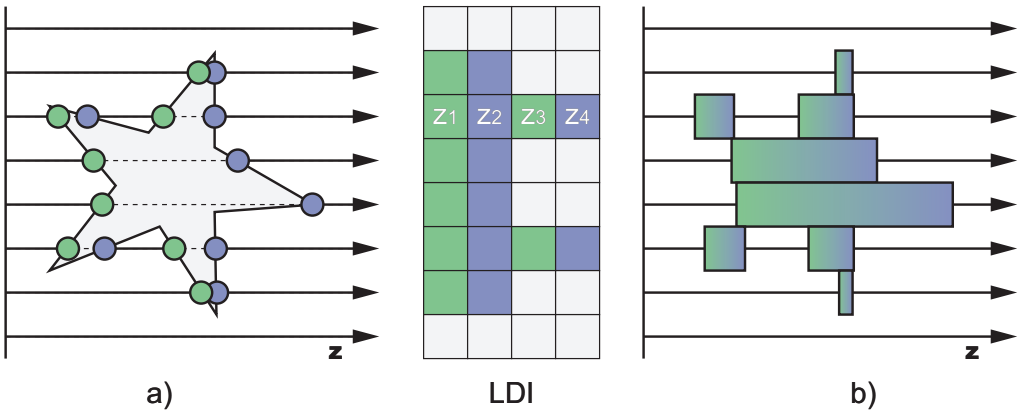


Figure 5.6: The intersections of parallel rays with object surface define an LDI (a) which approximates the volume of the object (b).

orthogonal projection. The viewing parameters for the rendering process are determined by the selected outside face of the VoI defining the near plane and the opposite face defining the far plane. The remaining faces define top, bottom, left and right of the rendering volume. If several eligible pairs of outside faces exist, the ones with the smallest distance between them are chosen. This criterion is a fast heuristic to optimize depth complexity assuming that depth complexity scales with the distance between the two outside faces.

Objects are rendered n_{max} times for LDI generation, where n_{max} denotes the depth complexity of the relevant part of the object within the VoI. Thus, the computational complexity of the LDI generation is $O(n_{max})$. The first rendering pass generates a single LDI layer and computes the value of n_{max}. If $n_{max} > 1$, additional rendering passes 2 to n_{max} generate the remaining layers (see Fig. 5.7). For details regarding the implementation variants, refer to Sec. 5.2.

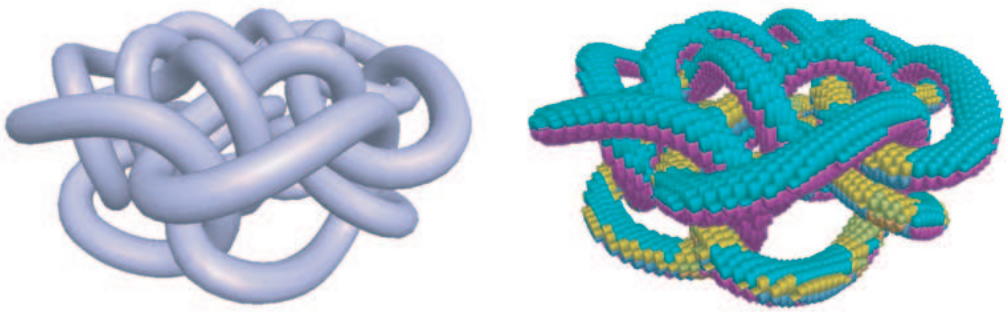


Figure 5.7: A knot model and its volumetric approximation. The colors indicate the layer within the LDI.

If an LDI is generated in the above way, entry points (front-faces) and exit points (back-faces) alternate with respect to their depth. For closed objects, the first layer is assumed to contain entry points, the second layer exit points and so on.

There are two parameters that influence the quality and performance of the algorithm, namely the LDI resolution (see Fig. 5.8) and the render direction (see Fig. 5.9).

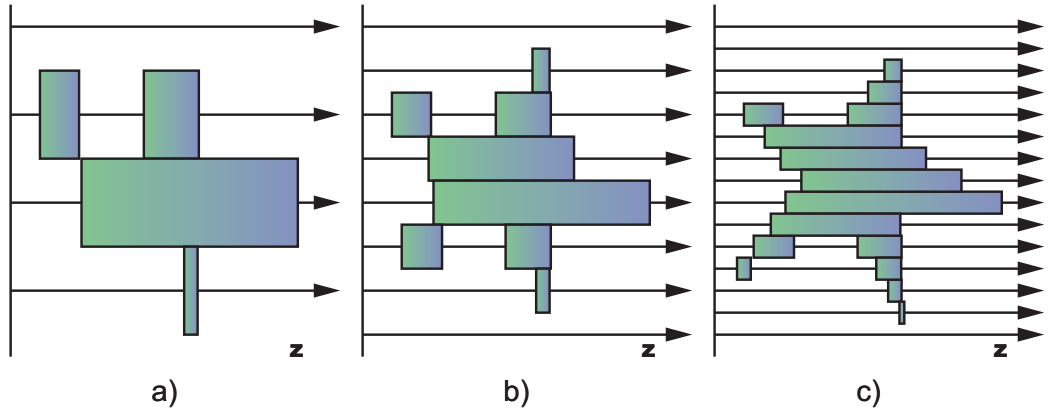


Figure 5.8: The LDI resolution influences the accuracy and the performance.

The accuracy of the method can be adjusted by the user. In (x, y)-direction, accuracy corresponds with the chosen LDI resolution. The accuracy in z-direction is given by the depth-buffer resolution which is dependent on the chosen implementation (see Sec. 5.2). Qualitative comparisons between different LDI resolutions and the impact on the performance are discussed in Sec. 5.3.

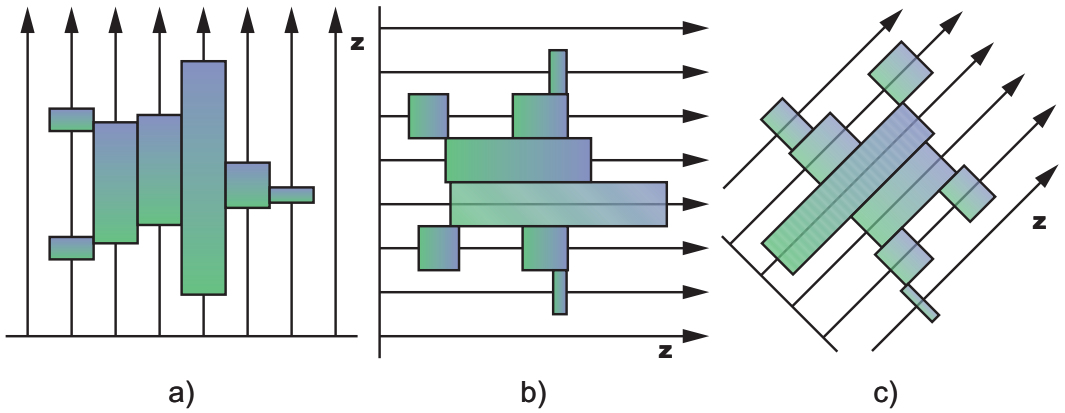


Figure 5.9: Changing the render direction leads to different volumetric approximations.

Compared to alternative data structures, such as bounding-volume hierarchies or

distance fields, the LDI representation is very memory efficient. Both the memory consumption and the performance of the LDI generation depend on the geometric complexity and the depth complexity of the objects. However, the results presented in Sec. 5.3 suggest, that the number of LDI layers does not exceed reasonable values even for complex objects. Furthermore, collision queries performed on the generated LDIs are very efficient even in the case of a large number of LDI layers.

At the end of the second stage, there exists an LDI for all objects within the VoI. Additional information, such as face orientation or normals, can also be generated and stored for each entry in the LDI. For example, the classification in entry points (front-faces) and exit points (back-faces) is required for the self-collision query described in the next stage.

5.1.3 Collision Query

The LDIs computed in the previous stage can be utilized to efficiently process a variety of collision queries in the third stage of the algorithm.

Self-collision query of a single object

Self-collisions are detected by analyzing the order of entry and exit points within the LDI. There exists no self-collision if they correctly alternate. If invalid sequences of entry and exit points are detected, the operation provides the explicit representation of the self-intersection volume (see Fig. 5.10).

Collision query between pairs of objects

Two LDIs can be combined to compute their intersection volume. The LDIs for two colliding objects are both discretized with the same resolution on corresponding sampling grids, but with opposite viewing directions. Hence, pixels in both LDIs represent corresponding volume spans. Therefore, intersection volumes can be computed by a pixel-wise intersection of the inside regions of both LDIs. A collision is detected if the resulting intersection is non-empty. The sum of all pixel-wise intersection regions constitutes a discrete representation of the intersection volume (see Fig. 5.11).

This collision query can also be considered as Boolean intersection operator on the two volumetric objects. Such operations on solids are commonly used in constructive solid geometry (CSG) [Hof89]. By combining multiple levels of CSG operators, very complex objects can be produced from simple primitives, such as spheres or boxes. There are three CSG operators available: Union, Intersection and Difference (see Fig. 5.12). Each operator acts upon two objects A and B and produces a single object C as result:

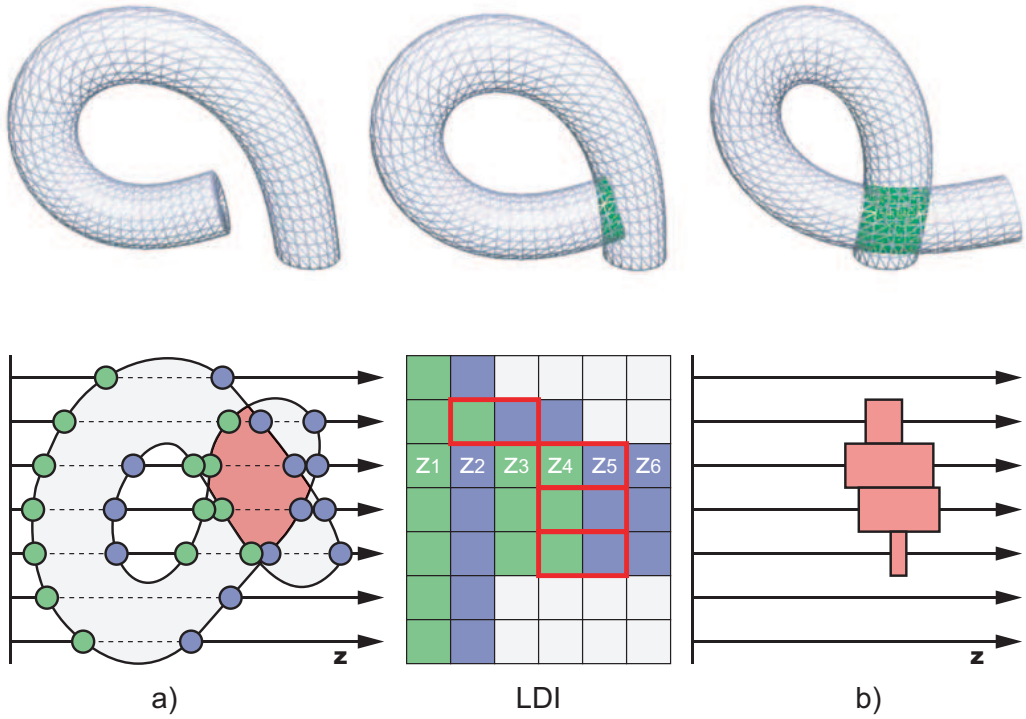


Figure 5.10: The self-collision query reports invalid ordering of entry and exit points (a) as self-intersection volume (b).

- $C = A \cup B$: The **Union** results in an object that encloses the combined space occupied by the two given objects.
- $C = A \cap B$: The **Intersection** results in an object that encloses the space where the two given objects overlap.
- $C = A - B$: The **Difference** is order dependent and results in the parent object minus the space where the child object intersected the parent object.

All three CSG operators can be applied to pairs of LDIs. However, only the Boolean intersection of two LDIs is useful in the context of collision detection:

$$C = A \cap B \begin{cases} = \emptyset & : \text{ no collision} \\ \neq \emptyset & : \text{ collision} \end{cases} \tag{5.1}$$

Collision query between individual vertices and an object

Individual vertices can also be tested against an LDI. To this end, the vertex is transformed into the local coordinate system of the LDI. A collision is detected if

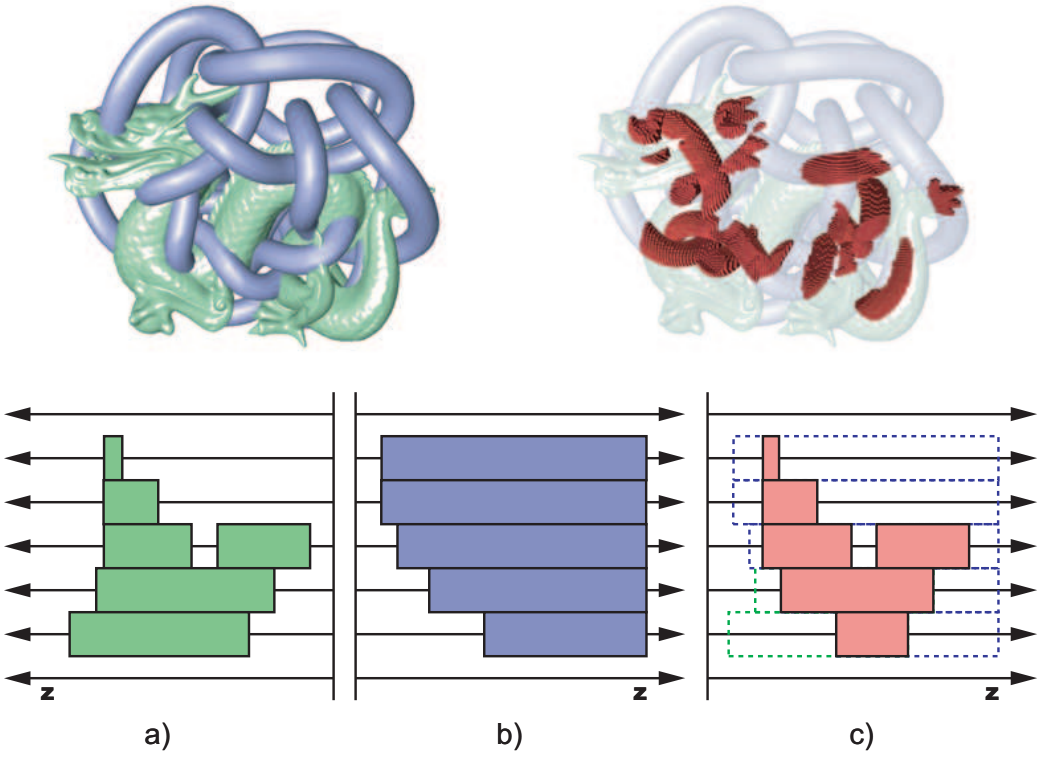


Figure 5.11: The collision query between pairs of objects reports the intersection volume (c) based on the LDIs of the objects within the VoI (a and b).

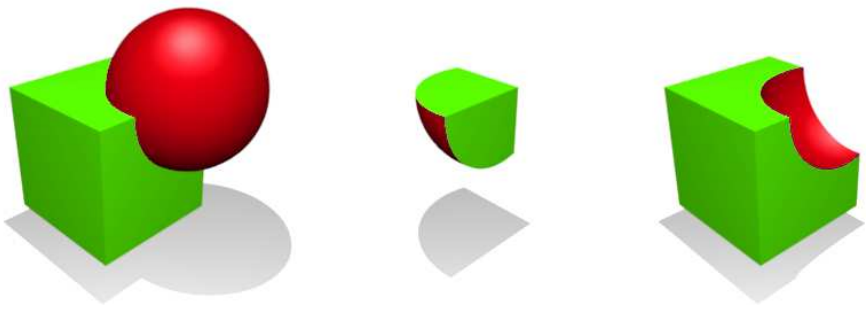


Figure 5.12: The CSG operators (from left to right): Union, Intersection and Difference.

the transformed vertex intersects with an inside region (see Fig. 5.13). This query can be used to compute collisions with atomic objects, such as particles or mass-spring systems often used in cloth simulation.

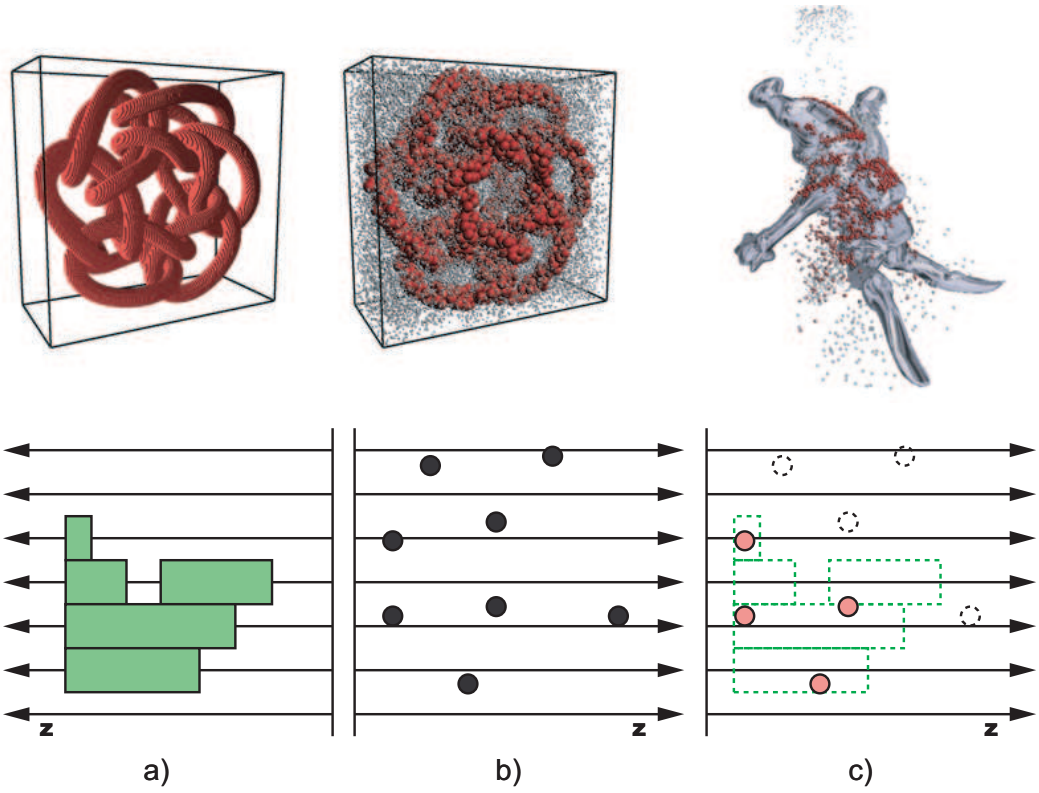


Figure 5.13: The collision query between an object (a) and individual vertices (b) reports a list of penetrating vertices (c). This is useful to test particle systems against objects in the scene (top-right image).

5.2 Implementations

The VoI computation in stage 1 and the collision queries in stage 3 of the algorithm do not significantly contribute to the overall computational cost (see Sec. 5.3). Since the LDI generation in stage 2 is comparatively expensive, a detailed description of three implementation variants is given in this section.

Two variants that exploit graphics hardware are addressed in Sec. 5.2.1 and Sec. 5.2.2. Using the GPU for this task is obviously useful, since stage 2 of the method basically rasterizes triangles. However, lack of flexibility in GPU implementations and the data read-back delay motivated a third implementation, which is completely processed on the CPU (see Sec. 5.2.3).

5.2.1 Ordered LDI

The first implementation generates LDIs using a generalized *depth peeling* approach on graphics hardware. As the name suggests, the technique peels away depth layer by depth layer resulting in an LDI whose depth values are correctly sorted per LDI

pixel. Fig. 5.14 illustrates the Ordered LDI generation.

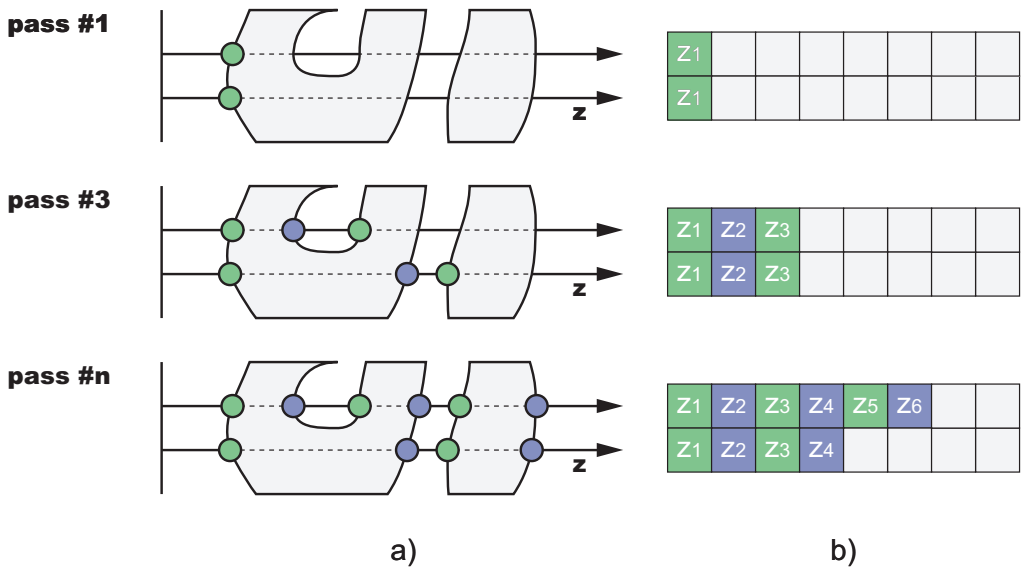


Figure 5.14: The Ordered LDI generation processes the entry and exit points according to their depth (a). This leads to an ordered LDI which does not need to be explicitly sorted (b).

Depth Peeling

Depth peeling was originally introduced for correct rendering of transparent surfaces in [Eve01]. The following modifications are made to make depth peeling suitable for LDI generation:

- A flexible abort criterion is added so that a dynamic number of LDI layers can be generated per object. This is in contrast to the original method that only allows to render a predefined, fixed number of depth layers.
- Color information is not considered for LDI generation and is therefore discarded.
- Back-faces and front-faces of an object are handled in two successive steps. This allows to label all LDI entries accordingly for self-collision queries. This separation also eliminates singularities of contour edges, which is a robustness problem of the original approach. If back-faces and front-faces coincide, the original algorithm produces a single depth value instead of two, thus falsifying the inside-outside classification (see Fig. 5.15).

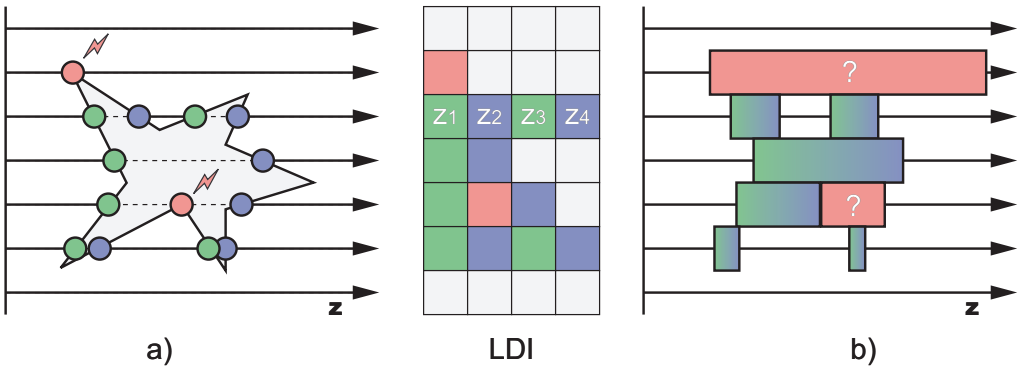


Figure 5.15: Successive steps for back- and front-faces eliminate potential singularities within the LDI (a) that lead to incorrect inside-outside classifications (b).

Multi-Pass Rendering

In order to solve the singularity problem and to generate an LDI with additional information on face orientation, the following multi-pass algorithm is performed twice. The first pass processes front-faces, followed by a second pass for back-faces. Two depth buffers with corresponding depth tests are used. One buffer is active, one is passive. Switching between active and passive buffer is necessary (see Alg. *OrderedLDI*).

Algorithm *OrderedLDI*
(∗ Ordered LDI Generation ∗)
Input: Viewing Parameters, Active and Passive Depth Buffer, Object
Output: Layered Depth Image LDI, Depth Complexity n

1. set $ViewingParameters$
2. $n \leftarrow 0$
3. clear $ActiveBuffer, LDI$
4. set $DepthTest_1$ to $Depth < Depth_{ActiveBuffer}$
5. set $DepthTest_2$ to $AlwaysPass$
6. render $Object$ into $ActiveBuffer$
7. **while** $ActiveBuffer$ has changed
8. **do** $LDI_n \leftarrow ActiveBuffer$
9. $n \leftarrow n + 1$
10. switch $ActiveBuffer$ and $PassiveBuffer$
11. clear $ActiveBuffer$
12. set $DepthTest_2$ to $Depth > Depth_{PassiveBuffer}$
13. render $Object$ into $ActiveBuffer$
14. **return** LDI, n

This method was implemented using OpenGL 1.4 functionality and a few widely used ARB extensions. The second depth test is provided by the *GL_ARB_shadow* and *GL_ARB_depth_texture* extensions and hardware-accelerated shadow-mapping functionality (see [Eve01]). By enabling *GL_ARB_occlusion_query*, the occlusion query mode, the number of written fragments is automatically accumulated by the GPU and queried after the rendering has completed. It is larger than zero if one or more fragments have been rendered, which means that another valid layer of the LDI has been produced and can be read back from the GPU. The multi-pass LDI generation is aborted if the occlusion query returns zero written fragments.

5.2.2 Unordered LDI

The second implementation generates unsorted LDI layers using graphics hardware. In contrast to the Ordered LDI approach, no second depth buffer and no depth tests are required. Fragments are discarded with the help of a stencil test. Fig. 5.16 illustrates the Unordered LDI generation.

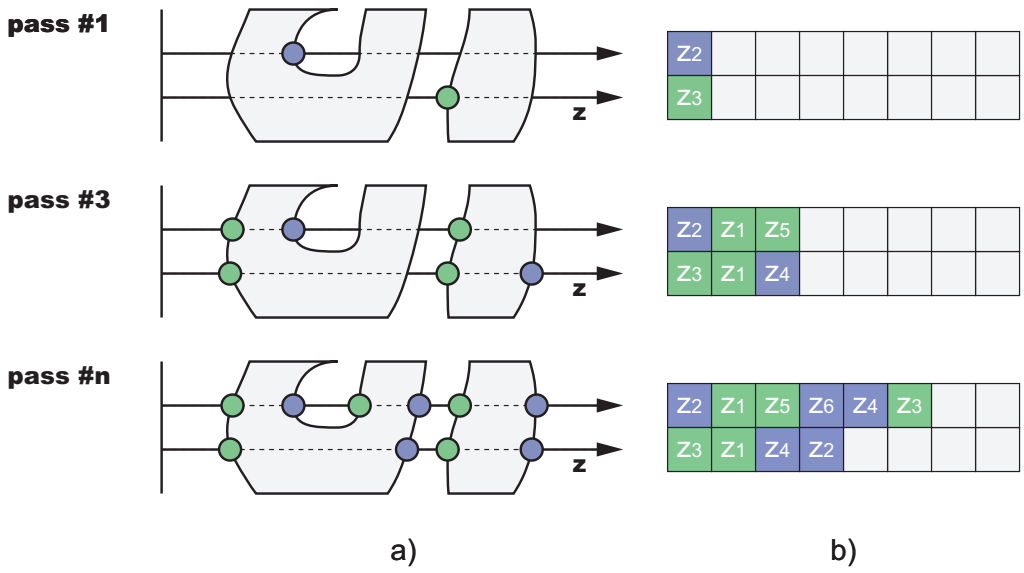


Figure 5.16: The Unordered LDI generation processes the entry and exit points in an arbitrary order (a). This leads to an LDI which does need to be sorted (b).

Stencil Setup

In the first rendering pass, the stencil test configuration allows the first fragment per pixel to pass, while the corresponding stencil value is incremented. Subsequent fragments are discarded by the stencil test, but still increment the stencil buffer.

Hence, after the first rendering pass the depth buffer contains the first object layer per pixel whereas the stencil buffer contains a map representing the depth complexity. Thus, n_{max} is found by searching the maximum stencil value. Note, that the $max()$ computation constitutes a very small computational burden, given the typical LDI resolution (see Sec. 5.3).

In consecutive passes, the rendering setup is similar to the first pass. However, during the n-th rendering pass, only the first n fragments per pixel pass the stencil test and, as a consequence, the resulting depth buffer contains the n-th LDI layer.

Since fragments are rendered in arbitrary order, the algorithm generates an unsorted LDI. However, the algorithm relies on consistency across the individual passes, which is provided by the GPU. For further processing, the LDI is sorted per pixel. Only the first n_p layers per pixel are considered, where n_p is the depth complexity of this pixel. n_p is taken from the stencil buffer as computed in the first rendering pass. If n_p is smaller than n_{max}, layers $n_p + 1$ to n_{max} do not contain valid LDI values for this pixel and are discarded.

Multi-Pass Rendering

Similar to the Ordered LDI implementation, the following multi-pass algorithm is performed once for front-faces and again for back-faces to provide information on face orientation. A single depth buffer is employed, but no depth test is needed. A stencil buffer with corresponding tests and operations is required (see Alg. *UnorderedLDI*).

Algorithm *UnorderedLDI*
(∗ Unordered LDI Generation ∗)
Input: Viewing Parameters, Depth Buffer, Stencil Buffer, Object
Output: Layered Depth Image LDI, Depth Complexity n

1. set $ViewingParameters$
2. $n \leftarrow 0$
3. clear $DepthBuffer, StencilBuffer, LDI$
4. set $StencilTest$ to $Stencil <= n$
5. set $StencilOp$ to $AlwaysIncrement$
6. render $Object$ into $DepthBuffer$
7. $n_{max} \leftarrow max(StencilBuffer)$
8. **while** $n < n_{max}$
9. **do** $LDI_n \leftarrow DepthBuffer$
10. $n \leftarrow n + 1$
11. clear $DepthBuffer$, $StencilBuffer$
12. set $StencilTest$ to $Stencil <= n$
13. render $Object$ into $DepthBuffer$
14. **return** LDI, n

This algorithm is implemented using core OpenGL 1.4 functionality. OpenGL extensions are not employed. Each rendering pass requires a buffer read-back. Reading back data from the GPU, however, can be expensive depending on the actual hardware. We propose two methods for optimizing the data transfer using OpenGL extensions. First, depth and stencil values of a pixel are usually stored in the same 32-bit word in the frame buffer. This allows to read-back both buffers in a single pass using an OpenGL extension, such as *GL_NV_packed_depth_stencil*. Second, all rendering passes can be performed independently from each other except for the first pass. We exploit this fact to render individual layers into different regions of the frame buffer. Once finished, the whole frame buffer is read back in a single pass. This optimization reduces the number of read-backs to a maximum of two, assuming that the frame buffer memory is sufficiently large. Thus, stalls in the rendering pipeline are reduced and the performance of the algorithm is significantly improved.

5.2.3 Software LDI

The third implementation for LDI generation is completely processed on the CPU. The investigation of this variant has been motivated by two drawbacks of GPU implementations. First, detailed timing measurements of the two graphics-hardware accelerated techniques have indicated that buffer read-backs are a main performance bottleneck (see Sec. 5.3). Second, graphics hardware requires multiple passes for LDI generation, since the output is restricted to one value per fragment in the frame buffer. In contrast, a software-renderer does not suffer from this restriction, as it can rasterize into multiple buffers at the same time. Therefore, a simplified software-renderer has been implemented, which is especially designed for LDI generation.

Software Renderer

This simplified renderer only features basic frustum culling, face clipping, orthogonal projection and rasterization of triangle meshes. The produced fragments are directly stored in the LDI structure. This is in contrast to hardware-accelerated methods which depend on an intermediate frame buffer that needs to be read back after each rendering pass.

First, the software-renderer culls object triangles against the VoI. If necessary, the remaining triangles are clipped against the VoI. This clipping might produce additional triangles, as shown in [Sut74]. Then, the software-renderer rasterizes all triangles that have passed the culling and clipping stages. The depth of each generated fragment is stored in the LDI structure along with the information on face orientation. Fragments rasterized at the same position in the LDI do not overwrite each other, but result in the generation of an additional LDI layer for this pixel. Fig. 5.17 illustrates the Software LDI generation.

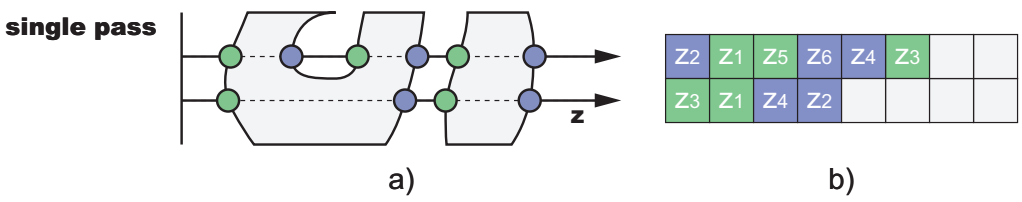


Figure 5.17: The Software LDI generation processes all entry and exit points in a single pass but arbitrary order (a). This leads to an LDI which does need to be sorted (b).

Single-Pass Rendering

The following single-pass algorithm is performed only once per object (see Alg. *SoftwareLDI*).

Algorithm *SoftwareLDI*
(∗ Software LDI Generation ∗)
Input: Viewing Parameters, Object
Output: Layered Depth Image LDI, Depth Complexity n
1. set $ViewingParameters$
2. clear LDI
3. **for** $t \leftarrow 1$ **to** $\#Triangles$ of $Object$
4. **do for** $f \leftarrow 1$ **to** $\#Fragments$ of $Triangle_t$
5. **do** $LDI_n(x_f, y_f) \leftarrow Depth_f$
6. **return** LDI, n

The software-renderer generates fragments in arbitrary order, similar to the Unordered LDI method. Therefore, per pixel sorting of the LDI is performed for further processing.

5.3 Results

The first part of this section presents performance comparisons between the implementation variants described in Sec. 5.2. Three test scenarios employing different collision queries are used for this. The second part carries out the test suite from Chap. 3. All timings in this section have been measured on a PC Pentium 4, 3.2 GHz with a GeForce FX Ultra 5800 GPU.

5.3.1 Comparisons

The first comparison scenario features a dynamically deforming hand and a phone, consisting of 4800 triangles and 900 triangles, respectively (see Fig. 5.18 and Fig. 5.19).

Volumetric collision detection is performed between the hand and the phone. Additionally, the animated hand is tested for self-collisions. The LDI resolution is 64x64. Tab. 5.1 shows timings for both collision queries.

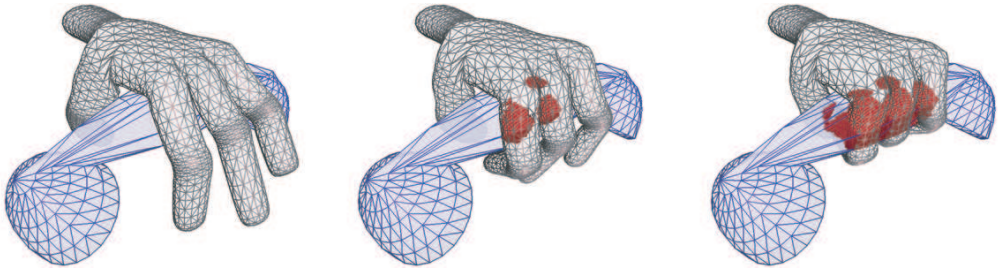


Figure 5.18: Dynamic animation of a hand grabbing a phone. Volumetric collisions (red) are detected. Refer to Tab. 5.1 for performance measurements.

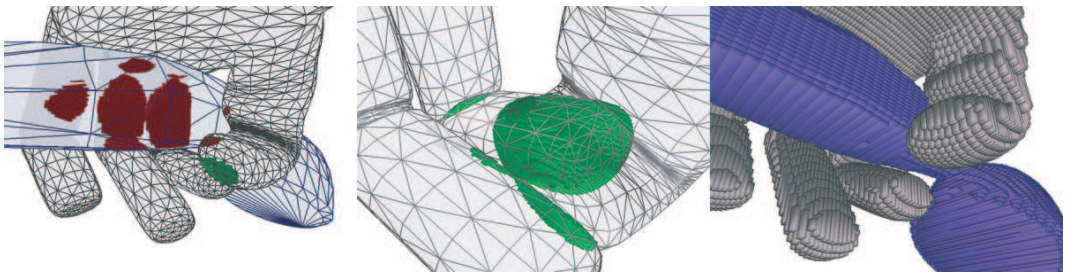


Figure 5.19: Left, Middle: Collisions (red) and self-collisions (green) of the hand are detected. Right: LDI representation with a resolution of 64x64. Refer to Tab. 5.1 for performance measurements.

Method	Collision		Self-collision		Overall	
	min [ms]	max [ms]	min [ms]	max [ms]	min [ms]	max [ms]
Ordered	26.1	34.3	36.1	49.8	62.2	84.1
Unordered	8.6	11.1	11.0	16.2	19.6	27.3
Software	2.5	3.6	4.4	6.4	6.9	10.0

Table 5.1: Dynamic test sequence with an animated hand (4800 triangles) and a phone (900 triangles). Minimum and maximum computational cost is given for collisions and self-collisions. The resolution of the LDI is 64x64. Fig. 5.18 and Fig. 5.19 illustrate the test.

The software LDI provides the best performance in this scenario, since data transfer from and to the GPU significantly reduces the performance of the Ordered

and Unordered LDI approach. Data read-back is independent from the model geometry. Therefore, rasterization performance of the graphics hardware is outweighed by data read-back for small geometry. Furthermore, the Unordered LDI approach is more efficient than the Ordered LDI approach. Although both methods require about the same number of rendering passes, the rendering setup for the Ordered LDI approach is more involved. All experiments indicate, that the Unordered LDI approach is about three times faster compared to the Ordered LDI approach.

In the second scenario, arbitrarily placed particles are tested against the volume of a dragon (see Fig. 5.20). The particles are randomly positioned within the AABB of the model. The LDI resolution is 64x64. Three different scene complexities have been tested with up to 500k triangles and 100k particles. Measurements are given in Tab. 5.2.

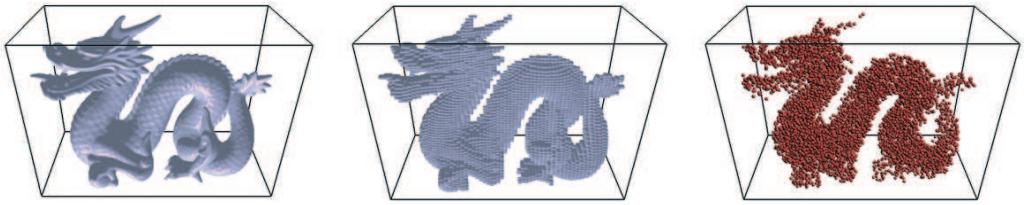


Figure 5.20: Left: Dragon with 500k faces. Middle: LDI representation with a resolution of 64x64. Right: Particles penetrating the volume of the dragon are detected. Performance measurements are given in Tab. 5.2.

Method	Model complexity		
	high [ms]	medium [ms]	low [ms]
Ordered	405.7	143.1	46.1
Unordered	202.9	67.1	21.8
Software	359.3	96.3	33.4

Table 5.2: Particles at arbitrary positions within the AABB of a dragon model are tested against the volume. Measurements are given for three model complexities: High (520k triangles, 100k particles), medium (150k triangles, 30k particles) and low (50k triangles, 10k particles). The LDI resolution is 64x64. Fig. 5.20 illustrates the test.

In this second experiment, the VoI encloses the entire model, i. e. all triangles of the dragon have to be rendered. In highly complex scenes, most time is spent for triangle rasterization and the read-back delay is less important. In this case, the GPU-based Unordered LDI approach outperforms the software-renderer and a rate

of 5Hz can be achieved for a highly complex scene with 500k triangles and 100k particles.

The third experiment consists of a hat and a mouse model with fixed geometric complexity (see Fig. 5.21). Collisions during the simulation are detected using varying LDI resolutions. Measurements are given in Tab. 5.3.

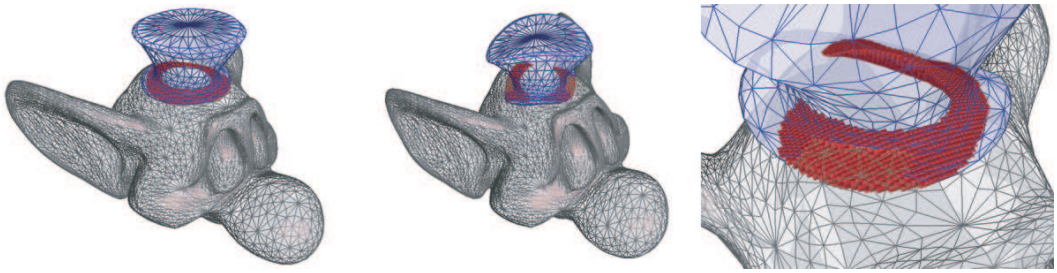


Figure 5.21: Left, middle: Dynamic animation of a mouse with 15000 faces and a hat with 1500 faces. The intersection volume (red) is shown. Right: Intersection volume with an LDI resolution of 64x64. Performance measurements are given in Tab. 5.3.

Method	LDI resolution		
	32x32 [ms]	64x64 [ms]	128x128 [ms]
Ordered	21.8	23.3	46.1
Unordered	7.2	7.9	15.5
Software	1.9	2.7	5.3

Table 5.3: Test with an animated hat (1500 triangles) and a mouse (15000 triangles). Collisions are detected with intersected LDIs. LDI resolution varies from 32x32 to 128x128. Fig. 5.21 illustrates the test.

Due to the moderate geometric complexity of the scene, the Software LDI shows the best performance in the third experiment. It can be seen, that the LDI resolution significantly influences the performance of our approach. In all methods, higher LDI resolution requires more fragments to be rendered. Additionally, data read-back slows down in case of higher LDI resolution for GPU-based approaches.

In general, all experiments show that the image-space approach to collision and self-collision detection can be accelerated with graphics hardware for large environments. However, in typical game environments, collisions are checked between less complex models comparable to the third experiment. In such applications, the CPU-based implementation provides the best performance.

5.3.2 Experiments

The software LDI method is part of an interactive simulation environment for deformable objects (see Sec. 7.1). Within this application, various experiments have been carried out to analyze the characteristics and the performance of the collision detection technique. A detailed description of the test scenarios can be found in Chap. 3, as they are referenced in several parts of this thesis.

Boldie Scene

This first scene features a head that is attached to the ground and dozens of balls that are dropped on it (see Sec. 3.3.1). Only AABB intersection tests on an object level are performed in the beginning of the simulation. This results in the same constant computational cost for all LDI resolutions until the first impact of a ball is detected. This point in time is clearly visible in the timing measurements in Fig. 5.22. After the first impact the number of collisions quickly increases and the performance drops as LDIs with more and more layers have to be generated. At the end of the experiment the collision detection needs 49.7 ms (32x32), 131.7 ms (64x64) or 439.5 ms (128x128) to compute all the intersection volumes between the 97 objects. The maximum LDI depth complexity measured during the experiment is 4 which leads to a maximum memory consumption of 7 KB (32x32), 30 KB (64x64) or 120 KB (128x128).

Pitbull Scene

Two highly-detailed pitbulls collide in midair in this experiment (see Sec. 3.3.2). The timing measurement in Fig. 5.23 show the heavy impact and the increased computational cost. A total of 1.7 ms (32x32), 1.9 ms (64x64) or 2.8 ms (128x128) is required to find the intersection volume of the two bodies at the time of maximum intersection. The computational cost suddenly drops to a lower level when the depth complexity of the involved LDIs decreases. The maximum LDI depth complexity measured during the experiment is 18 which leads to a maximum memory consumption of 72 KB (32x32), 288 KB (64x64) or 1152 KB (128x128).

Mixed Scene

Multiple dynamic collisions between different object types occur during this experiment (see Sec. 3.3.3). After an initial rise of the computational cost, the performance stays roughly at the same level, independent of the size of the intersection volume (see Fig. 5.24). A closer analysis reveals that there are no point collisions left at the end of the simulation but many AABBs of the objects still intersect and trigger the LDI generation. This way, the collision detection needs 1.1 ms (32x32), 2.1

ms (64x64) or 5.9 ms (128x128) to test for collisions when the objects reach the contact-free rest state. The maximum LDI depth complexity measured during the experiment is 10 which leads to a maximum memory consumption of 40 KB (32x32), 160 KB (64x64) or 640 KB (128x128).

Membrane Scene

A stack of ten membranes of different sizes is created in this experiment (see Sec. 3.3.4). Fig. 5.25 shows a sharp performance drop that follows the time of impact of the two lowest membranes. The computational cost increaes until it reaches 6.7 ms (32x32), 11.1 (64x64) or 25.8 ms (128x128) at the end of the simulation. The intersection volumes vanish at the end of the experiment when the membranes build a stable stack and only resting contacts remain. The maximum LDI depth complexity measured during the experiment is 4 which leads to a maximum memory consumption of 16 KB (32x32), 64 KB (64x64) or 256 KB (128x128).

Rod Scene

In this scene a highly deformable rod is tested for self-collisions (see Sec. 3.3.5). The collision detection needs 1.2 ms (32x32), 2.3 ms (64x64) or 6.1 ms (128x128) to compute the volume of self-intersection at the time of maximum impact (see Fig. 5.26). There is not much performance fluctuations during the experiment because a full LDI is needed to detect self-collisions in all the frames. The computational cost at the beginning of the simulation is higher than expected due to an unfavorable LDI render direction. The rod buckles at several locations when it hits the ground resulting in a high depth complexity of the LDI. The maximum LDI depth complexity measured during the experiment is 36 which leads to a maximum memory consumption of 144 KB (32x32), 576 KB (64x64) or 2304 KB (128x128).

MultiRod Scene

Four rods are tested for collisions and self-collisions in this experiment (see Sec. 3.3.6). When the objects finally find their rest state, a total of 17.0 ms (32x32), 28.7 ms (64x64) or 62.2 ms (128x128) is needed to compute the volumes of intersection and self-intersection (see Fig. 5.27). The slight performance drop over the course of the experiment is caused by the collisions between the objects. Similar to the previous scene, the performance fluctuations are minimal for self-collision detection. The maximum LDI depth complexity measured during the experiment is 42 which leads to a maximum memory consumption of 168 KB (32x32), 672 KB (64x64) or 2688 KB (128x128).

5.4 Discussion

Image-space methods for collision and self-collision detection are an interesting alternative to conventional techniques. Due to the fact that no pre-processing is required, these methods are especially useful in dynamic environments with deformable objects. In addition, their mode of operation allows for implementations that are accelerated by programmable graphics hardware.

Volumetric collision detection is enabled by using an LDI as the central data structure. It represents the involved objects by a volumetric approximation making it possible to execute different kinds of collision queries. Self-collisions are detected by generating a full LDI of the whole object and testing for well-defined inside and outside conditions. When testing two objects for collisions, two separate LDIs are generated in their VoI and queried for overlapping regions. Finally, arbitrary points can also be tested against the volume of an object by transforming their position to the LDI coordinate system.

Image-space techniques can be accelerated with graphics hardware. However, due to buffer read-back delays and the limited flexibility of programmable graphics hardware, it is not always guaranteed that implementations on graphics hardware are faster than software solutions. This chapter investigated and compared two GPU-based implementations and one CPU-based implementation of the proposed collision detection. Results suggest, that graphics hardware accelerates collision detection in geometrically complex environments, while the CPU-based implementation provides more flexibility and better performance in case of small environments.

LDI generation works with arbitrarily shaped objects, but is restricted to closed, watertight surfaces, since it relies on the notion of inside and outside. For this reason, the presented collision detection method requires that at least one object meets these requirements to work properly.

Since image-space techniques work with discretized representations of objects, they do not provide exact collision information. The accuracy of the collision detection depends on the discretization error. Thus, accuracy and performance can be balanced in a certain range by changing the resolution of the rendering process. This tradeoff is controllable by the user. To increase the precision of the approximation, the object can also be rendered from multiple directions resulting in a Layered Depth Cube (LDC) [Lis98].

The resolution of the LDI is not the only factor that influences the performance of the method. The depth complexity plays an important role for the graphic hardware accelerated implementations. A small change in the depth complexity can result in a sudden erratic performance drop as more LDI layers need to be generated and read back. Also, these fluctuations do not correlate with the size of the intersection volume which makes them unpredictable. In contrast to this, the computational

cost of the software implementation depends more on the geometric complexity of the objects.

The main advantages of the proposed method is its scalable nature and the possibility to accelerate it with graphics hardware. Unfortunately, these are also the limitations of the approach, due to approximation errors and read-back delays. The image-space method allows for collision and self-collision detection between rigid as well as deformable objects. The provided collision information is either an intersection volume for objects or an inside-outside information for points. In conclusion, the presented image-space technique is efficient for approximative collision detection of simple scenes when done in software or geometrically complex scenes when accelerated with graphics hardware.

Directions for future work

While the image-space technique efficiently detects collisions, it does not provide collision information that can directly be used by common collision response methods in physically-based simulation environments. Additional post-processing of the provided result is required to compute or to approximate information such as the penetration depth or closest surface point of colliding objects. Further investigations could lead to more sophisticated collision response approaches that directly work on the explicit representation of the intersection volume.

The performance and feature set of graphics hardware keep evolving in a very fast pace these days. New developments in this area could solve the problem of multiple rendering passes and the delay of the LDI read-back. Furthermore, it might be worthwhile to investigate solutions where the collision information does not need to be read back to the CPU at all, but stays on the GPU for further processing. Another option is to compute the full simulation cycle (deformation, collision detection and collision response) on the GPU.

Many separate LDIs for the same object are computed when it collides with multiple objects. This happens a lot in situations with many resting contacts between stacked objects. A clever method that combines all VoIs of an object into a single one would improve the performance here.

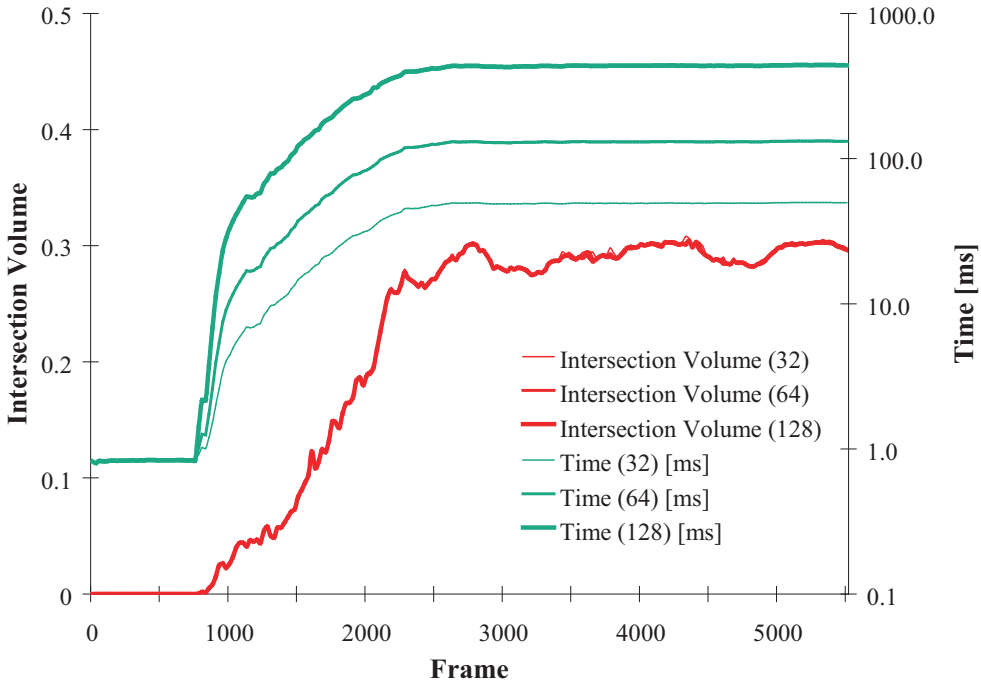


Figure 5.22: Experiment: Boldie scene.

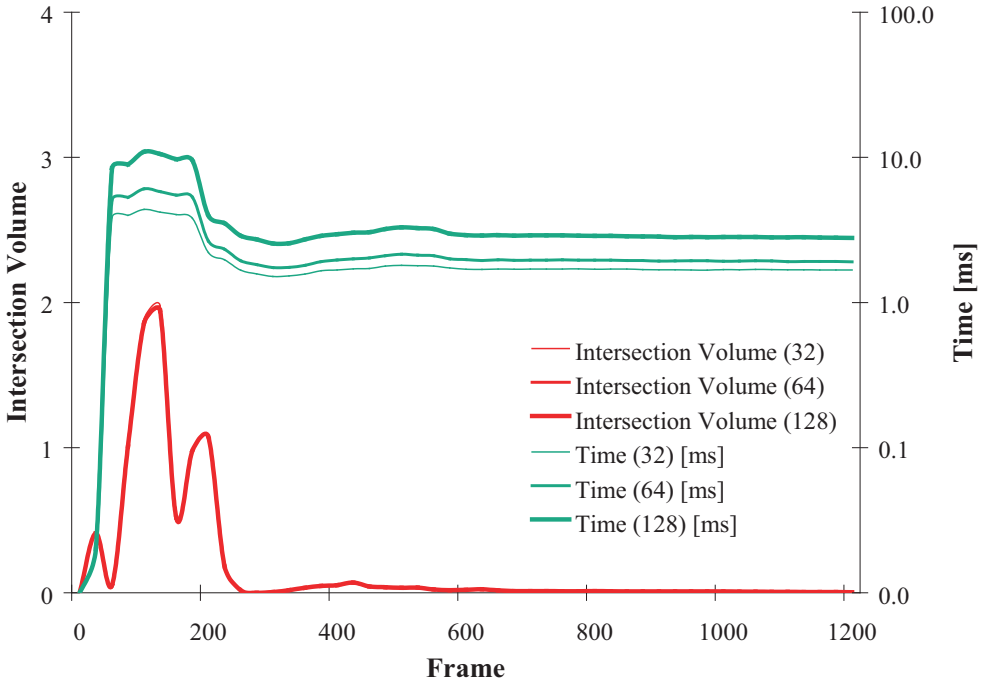


Figure 5.23: Experiment: Pitbull scene.

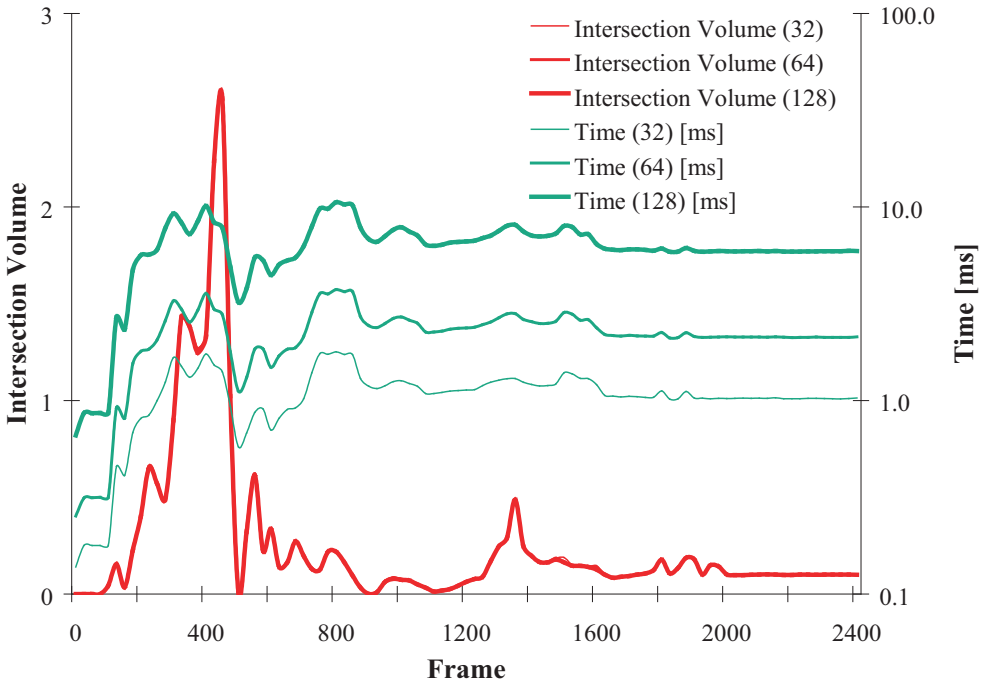


Figure 5.24: Experiment: Mixed scene.

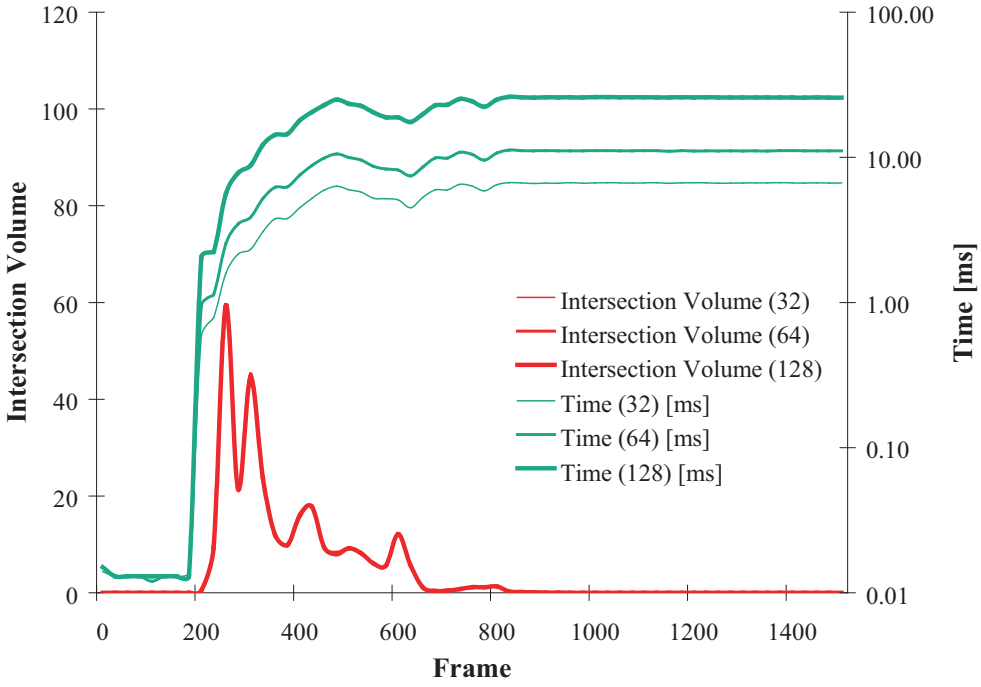


Figure 5.25: Experiment: Membrane scene.

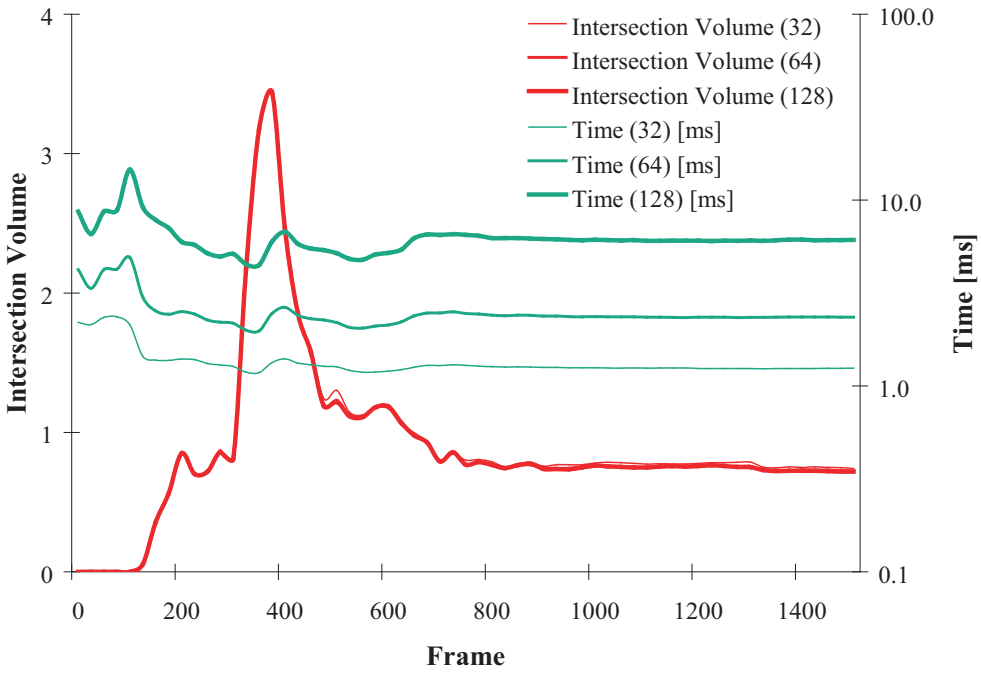


Figure 5.26: Experiment: Rod scene.

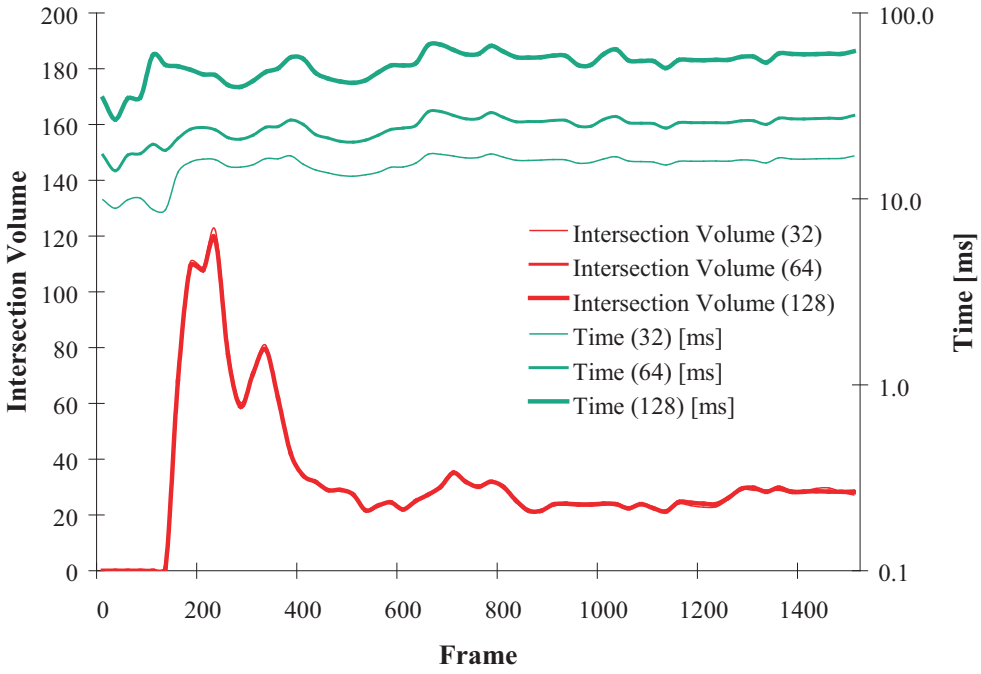


Figure 5.27: Experiment: MultiRod scene.

6 Consistent Penetration Depth Estimation

Penalty approaches can be used to efficiently resolve collisions of dynamically simulated rigid and deformable objects (see Sec. 2.3). These methods compute penalty forces based on the penetration depth and direction of intersecting objects and there exist many algorithms for estimating the exact penetration information. However, in discrete-time simulations, this information can cause non-plausible collision responses in case of large penetrations or due to the object discretization. Fig. 6.1 and Fig. 6.2 illustrate these problems.

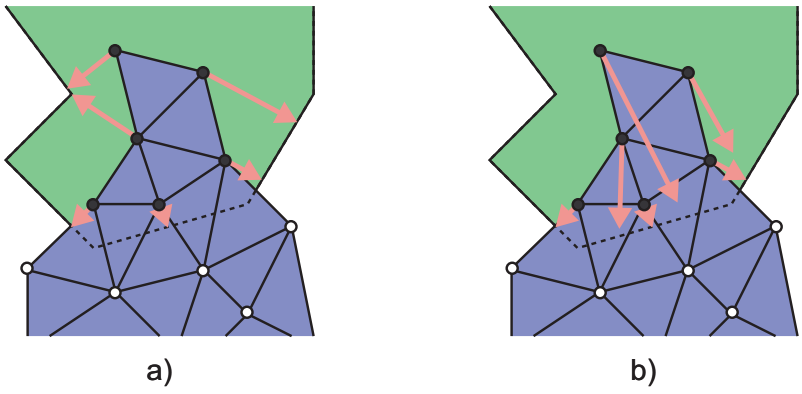


Figure 6.1: Collision response based on the minimal penetration depth can lead to non-plausible behavior in case of a large penetration (a). A novel penetration depth estimation results in a consistent collision response (b).

This chapter presents a method to compute consistent n-body penetration depth information in order to reduce collision response artifacts inherent to existing approaches. The method considers a set of close surface features to avoid discontinuous penetration depths. Furthermore, a propagation scheme is applied in case of large penetrations to avoid non-plausible, inconsistent depth information.

The different stages of the algorithm are presented in Sec. 6.1. The method is then compared to the nearest-surface approach and tested in various experiments in Sec. 6.2. Sec. 6.3 gives a final discussion about the advantages and limitations of the method followed by directions for future research.

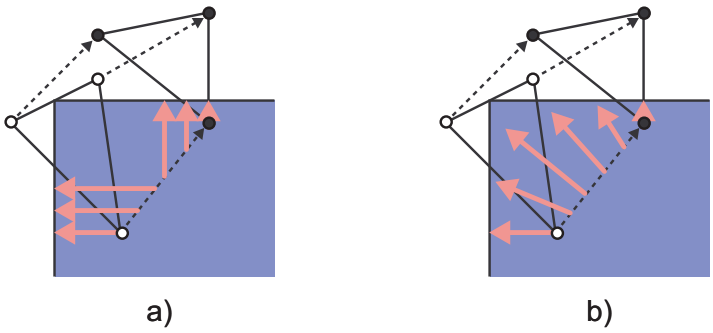


Figure 6.2: Abrupt changes of the minimal penetration direction cause severe discontinuities in the collision response (a). A smooth transition is achieved by the presented method to estimate penetration depth (b).

6.1 Algorithm

This section provides an overview of the proposed algorithm followed by a detailed description of its four stages.

The method takes a set of potentially colliding volumetric meshes as input and computes consistent n-body penetration depths and directions for all colliding mesh points. This collision information can then directly be used as input to any penalty-based collision response scheme.

Algorithm Overview

The algorithm consists of the following four stages illustrated in Fig. 6.3:

- **Stage 1** detects all colliding points in the scene based on a volumetric collision detection method (see Sec. 6.1.1).
- **Stage 2** identifies all colliding points adjacent to one or more non-colliding points as border points. Furthermore, it detects all intersecting edges that contain one non-colliding point and one border point. The exact intersection point and corresponding surface normal of the penetrated surface are computed for each intersection edge (see Sec. 6.1.2).
- **Stage 3** approximates the penetration depth and direction for each border point based on the adjacent intersection points and surface normals obtained from the second stage (see Sec. 6.1.3).
- **Stage 4** propagates the penetration depth and direction to all colliding points that are not border points (see Sec. 6.1.4).

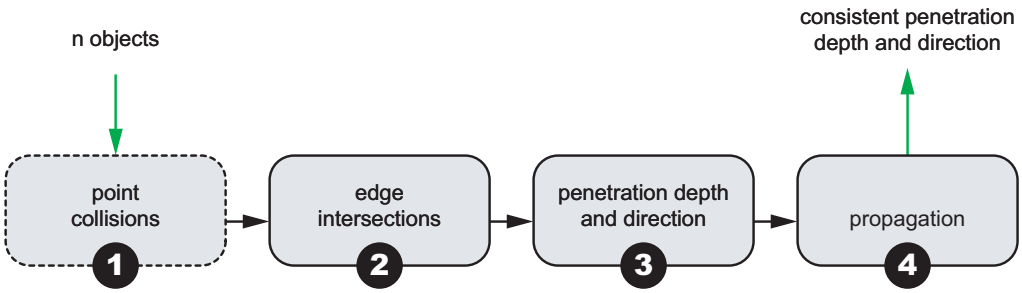


Figure 6.3: The four stages of the algorithm: 1) Collision detection, 2) border-point calculations, 3) penetration depth approximation and 4) penetration depth propagation.

6.1.1 Point Collisions

The first stage detects all object points that collide with any volumetric mesh in the scene. This volumetric collision detection is accomplished either by the spatial hashing approach presented in Chap. 4 or the image-space technique in Chap. 5.

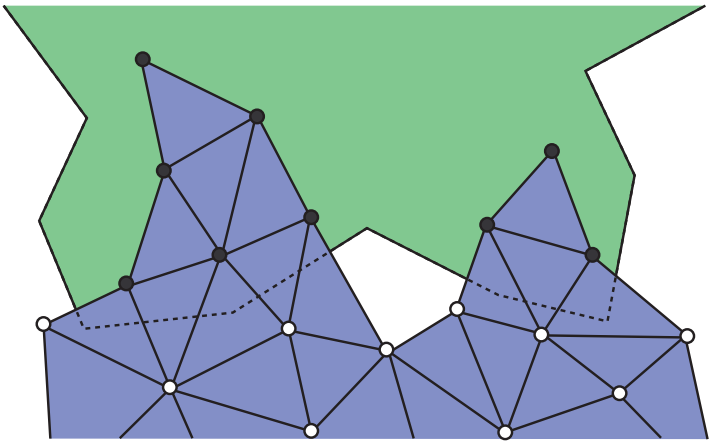


Figure 6.4: The first stage classifies all mesh points either as colliding points (black) or non-colliding points (white).

At the end of the first stage, all mesh points in the scene are either classified as colliding points or non-colliding points (see Fig. 6.4).

6.1.2 Edge Intersections

The second stage identifies all colliding points with at least one adjacent non-colliding point as border points. The underlying idea is to classify colliding points with respect to their penetration depth. Based on this information, the second stage

finds all intersecting edges that contain one non-colliding point and one border point. Moreover, the exact intersection point of each of those edges with the surface along with the corresponding surface normal of the penetrated mesh is computed. In order to efficiently compute this information, the original spatial hashing approach has been extended to handle collisions between edges and surfaces.

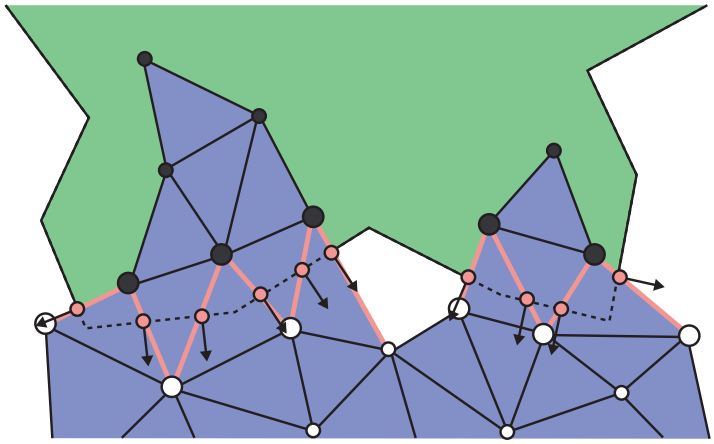


Figure 6.5: The second stage finds all intersecting edges (red) of the tetrahedral meshes that contain one non-colliding point (white) and one border point (black). Furthermore, the exact intersection point and the corresponding surface normal are computed for each intersection edge.

In addition, the barycentric coordinates can also be used to interpolate a smooth surface normal based on the three vertex normals of the face. This results in a smooth approximation of the penetration direction (see Sec. 6.1.3).

Each edge can possibly intersect with more than one mesh face. Therefore, only the intersection point nearest to the non-colliding point of the edge is considered in further stages.

At the end of the second stage, each border point is adjacent to one or more intersection edges. In addition, all intersecting edges have an exact intersection point and a corresponding surface normal (see Fig. 6.5).

6.1.3 Penetration Depth and Direction

The third stage approximates the penetration depth and direction for all border points based on the adjacent intersection points and surface normals computed in the second stage.

First, the influence on a border point is computed for all adjacent intersection points. This influence is dependent on the distance between an intersection and a border point. The respective weighting function has to be positive for all non-zero

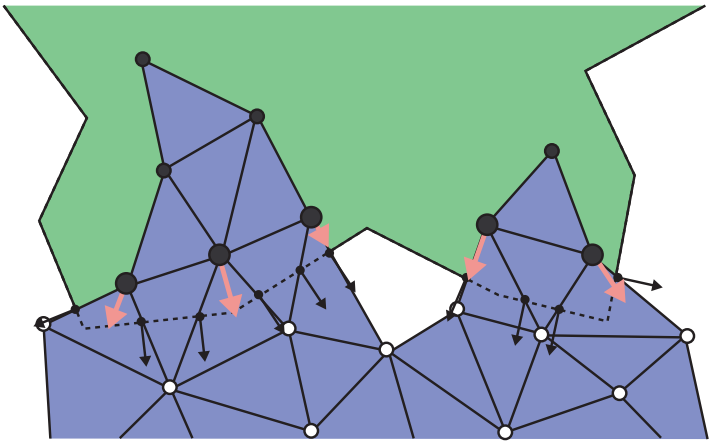


Figure 6.6: The third stage approximates the penetration depth and direction for all border points based on the adjacent intersection points and surface normals.

distances and increasing for decreasing distances. Furthermore, it has to ensure convergence to the penetration depth information with respect to a intersection point $\mathbf{x}_i$ if a colliding point $\mathbf{p}$ approaches $\mathbf{x}_i$. This leads to the following weighting function for the influence $\omega(\mathbf{x}_i, \mathbf{p})$:

$$\omega(\mathbf{x}_i, \mathbf{p}) = \frac{1}{\|\mathbf{x}_i - \mathbf{p}\|^2} \tag{6.1}$$

with $\mathbf{x}_i$ denoting an intersection point and $\mathbf{p}$ denoting the border point. The weighting function does not have to be normalized, since this would not avoid any normalization steps in further processing. The weight is undefined for coinciding points. However, the first stage ensures that there is no collision detected in this case. The penetration depth $d(\mathbf{p})$ of a border point $\mathbf{p}$ is now computed based on the influences resulting from (6.1):

$$d(\mathbf{p}) = \frac{\sum_{i=1}^{k}(\omega(\mathbf{x}_i, \mathbf{p}) \cdot (\mathbf{x}_i - \mathbf{p}) \cdot \mathbf{n}_i)}{\sum_{i=1}^{k} \omega(\mathbf{x}_i, \mathbf{p})} \tag{6.2}$$

with $\mathbf{n}_i$ denoting the unit surface normal of the penetrated object surface at the intersection point. The number of intersection points adjacent to the border point $\mathbf{p}$ is given by k. Finally, the penetration direction $\hat{\mathbf{r}}(\mathbf{p})$ of a border point is computed as a weighted average of the surface normals

$$\hat{\mathbf{r}}(\mathbf{p}) = \frac{\sum_{i=1}^{k}(\omega(\mathbf{x}_i, \mathbf{p}) \cdot \mathbf{n}_i)}{\sum_{i=1}^{k} \omega(\mathbf{x}_i, \mathbf{p})} \tag{6.3}$$

and the normalized penetration direction $\mathbf{r}(\mathbf{p})$ is obtained as

$$\mathbf{r}(\mathbf{p}) = \frac{\hat{\mathbf{r}}(\mathbf{p})}{\|\hat{\mathbf{r}}(\mathbf{p})\|}. \tag{6.4}$$

At the end of the third stage, consistent penetration depths and directions have been computed for all border points (see Fig. 6.6). In contrast to existing penetration depth approaches that consider only one distance, the weighted averaging of distances and directions provides a continuous behavior of the penetration depth function for small displacements of colliding points and for colliding points that are adjacent to each other. Non-plausible penetration directions due to the surface discretization of the penetrated object are avoided.

6.1.4 Propagation

Based on the computed penetration depth information for border points, the fourth stage propagates the information to all other colliding points that are not border points (see Fig. 6.7). This is in contrast to existing penetration depth approaches that compute the penetration depth for all points independently. The idea of the propagation scheme is to avoid non-plausible penetration depths in case of large penetrations.

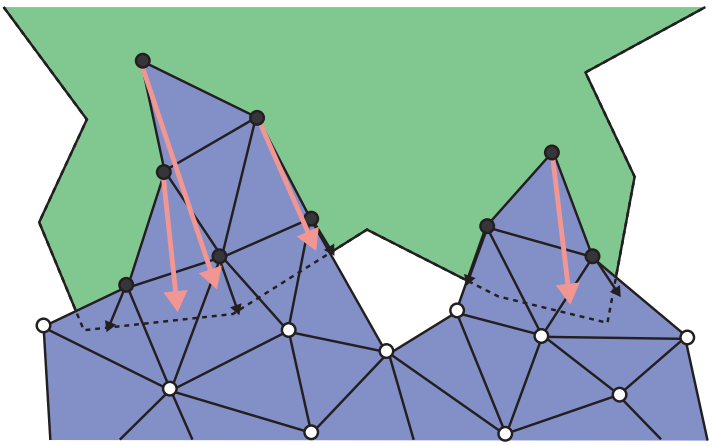


Figure 6.7: Stage 4 propagates the penetration depth and direction to all colliding points that are not border points.

The propagation is an iterative process that consists of the following two steps: First, the current border points are marked as processed points. Second, a new set of border points is identified from all colliding points that are adjacent to one or more processed points. The iteration is aborted, if no new border points are found. Otherwise, the penetration depth and direction for the new border points is computed based on the information available from all adjacent processed points.

Similar to the method described in Sec. 6.1.3, a weighting function is used to compute the influence $\mu(\mathbf{p}_j, \mathbf{p})$ of an adjacent processed point $\mathbf{p}_j$ on a border point $\mathbf{p}$ as

$$\mu(\mathbf{p}_j, \mathbf{p}) = \frac{1}{\|\mathbf{p}_j - \mathbf{p}\|^2}. \tag{6.5}$$

Based on the influences $\mu(\mathbf{p}_j, \mathbf{p})$, the penetration depth $d(\mathbf{p})$ of a border point $\mathbf{p}$ is computed as

$$d(\mathbf{p}) = \frac{\sum_{j=1}^{l}(\mu(\mathbf{p}_j, \mathbf{p}) \cdot ((\mathbf{p}_j - \mathbf{p}) \cdot \mathbf{r}(\mathbf{p}_j) + d(\mathbf{p}_j)))}{\sum_{j=1}^{l} \mu(\mathbf{p}_j, \mathbf{p})} \tag{6.6}$$

with $\mathbf{r}(\mathbf{p}_j)$ denoting the normalized penetration direction of the processed point $\mathbf{p}_j$ and $d(\mathbf{p}_j)$ denoting its penetration depth. The number of processed points adjacent to the border point $\mathbf{p}$ is given by l.

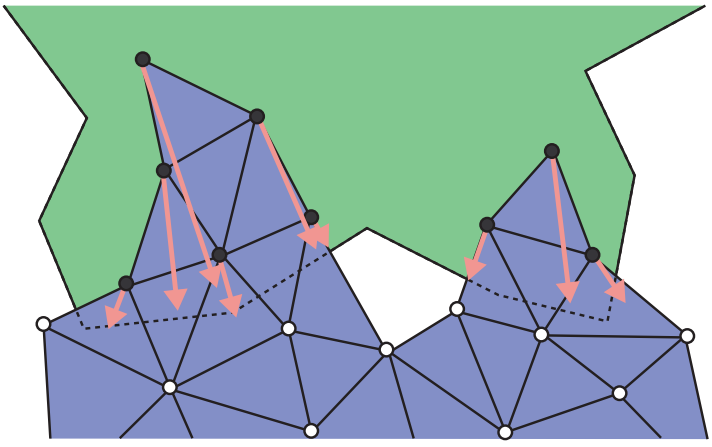


Figure 6.8: The algorithm computes consistent penetration depths and directions for all colliding points.

Finally, the penetration direction $\hat{\mathbf{r}}(\mathbf{p})$ is computed as a weighted average of the penetration direction of the processed points adjacent to the border point as

$$\hat{\mathbf{r}}(\mathbf{p}) = \frac{\sum_{j=1}^{l} \mu_j \mathbf{r}_j}{\sum_{j=1}^{l} \mu_j}. \tag{6.7}$$

and normalized with

$$\mathbf{r}(\mathbf{p}) = \frac{\hat{\mathbf{r}}(\mathbf{p})}{\|\hat{\mathbf{r}}(\mathbf{p})\|}. \tag{6.8}$$

At the end of the fourth stage, all colliding points have a consistent penetration depth and direction assigned (see Fig. 6.8).

6.2 Results

The first part of this section compares the quality and performance of the proposed method with the standard closest-feature approach. Measurements for two test scenarios employing the different methods are presented. The second part carries out the test suite from Chap. 3. All timings in this section have been measured on a PC Pentium 4, 3.2 GHz.

6.2.1 Comparisons

In a first test, two deformable cubes consisting of 1250 tetrahedrons are simulated. Large penetrations between the objects occur due to the high relative velocity and the discrete-time simulation. As illustrated in Fig. 6.9, the standard approach fails to compute a consistent penetration depth. This results in a non-plausible collision response. Employing the presented penetration depth approach to the same scenario results in consistent, plausible penetration depth information.

The second scenario simulates 120 deformable spheres consisting of 2400 tetrahedrons. Starting from a random position, they build a stack of spheres. Computing the penetration depth with the standard approach leads to heavy artifacts (see Fig. 6.10). The spheres tend to stick together due to inconsistent handling of penetrated object points. In this case, inconsistent penetration depths and response forces cause non-plausible equilibrium states. By applying the presented penetration depth approach, these response artifacts are avoided.

The presented approach scales linearly with the number of colliding points. In all experiments presented in this section, an average time of 32 μs is needed for resolving a colliding point. Most time is spent for detecting the edge intersections required by the second stage of the method (see Sec. 6.1.2). Similar computational costs are experienced to calculate the closest feature in the standard approach.

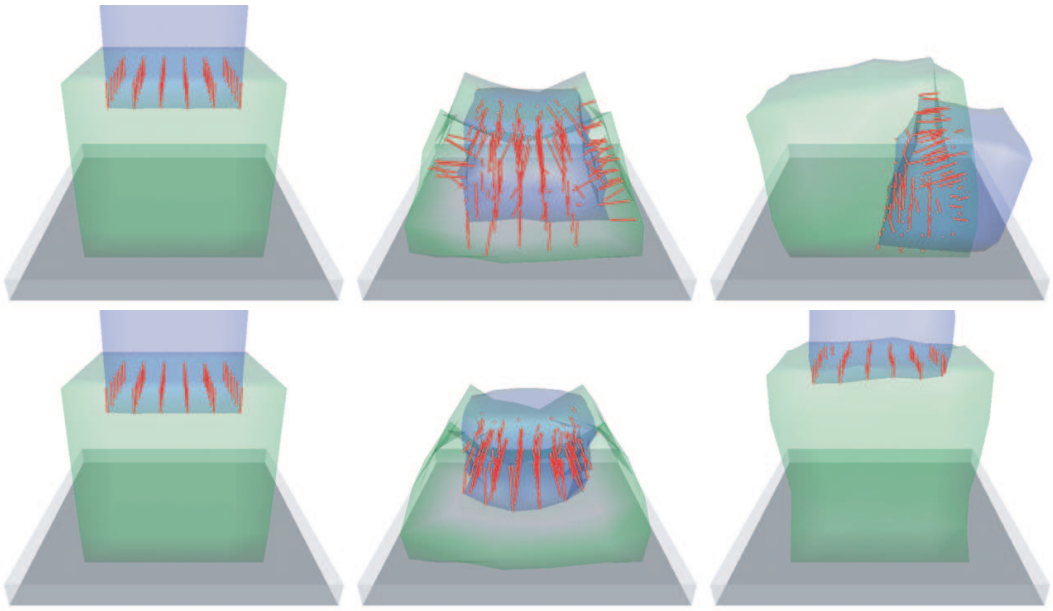


Figure 6.9: In this example, the standard approach fails to compute a plausible collision response (top row) whereas the presented method based on consistent penetration depth estimation succeeds (bottom row).

6.2.2 Experiments

The proposed method is part of an interactive simulation environment for deformable objects (see Sec. 7.1). Within this application, various experiments have been carried out to analyze the characteristics and the performance of the penetration depth technique. A detailed description of the test scenarios can be found in Chap. 3, as they are referenced in several parts of this thesis.

Boldie Scene

Dozens of balls are dropped on a head that is attach to the ground in the first experiment (see Sec. 3.3.1). No time is spent for penetration depth computation in beginning of the simulation. The computational cost starts to raise as soon as the first colliding point is detected (see Fig. 6.11). The number of edge intersections quickly increases as more and more balls collide with the head and other balls. At the end of the experiment the method considers 3840 edge intersections and needs 19.5 ms to find consistent penetration depths and directions for 782 resting contacts between the 97 objects in the scene.

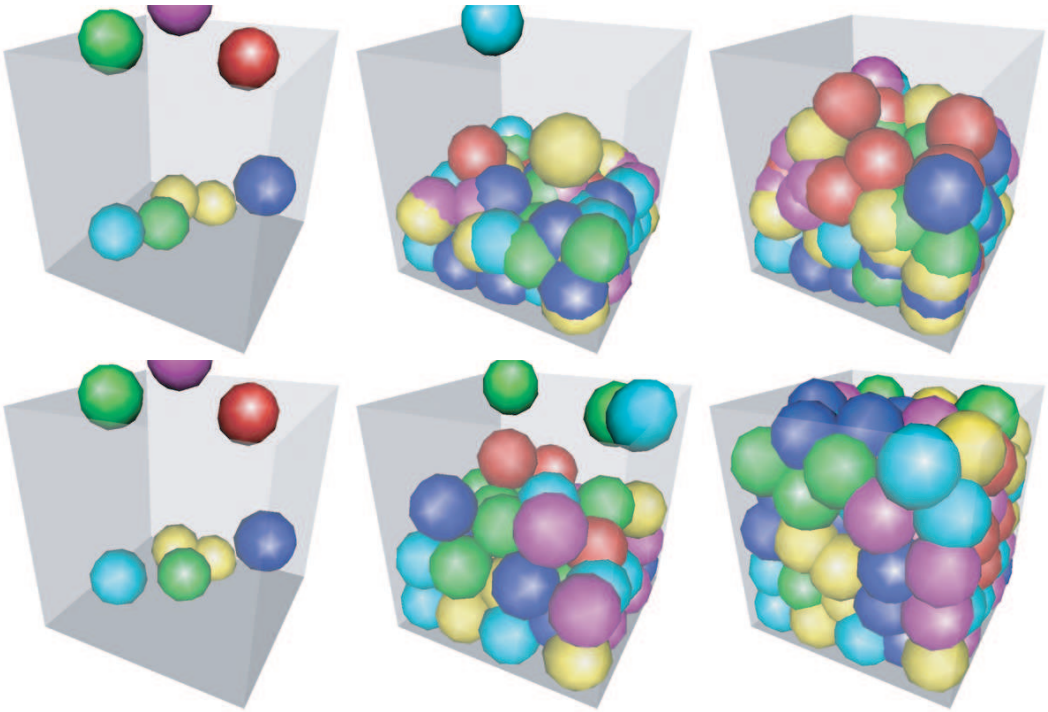


Figure 6.10: Inconsistent handling of penetrated points leads to spheres that stick together (top row). The presented approach avoids these artifacts due to consistent penetration depth estimation (bottom row).

Pitbull Scene

A heavy impact of two highly-detailed pitbulls occurs in this scene (see Sec.3.3.2). As expected, the performance drop is most noticeable at the time of maximum impact when consistent penetration depth and direction for 270 point collisions is computed. The presented method provides this information based on 1062 edge intersections found in 8.6 ms (see Fig. 6.12).

Mixed Scene

In this scene multiple collisions occur between different types of models (see Sec.3.3.3). The dynamic nature of this experiment as well as the effect on the performance are clearly visible in the timing measurements in Fig. 6.13. The method needs up to 1.4 ms to compute consistent penetration depth and direction for 54 colliding points based on 223 edge intersections. When the objects in the scene reach the contact-free rest state this computational cost drops to a negligible level.

Membrane Scene

Ten different sized membranes fall to the ground before building a stack in this experiment (see Sec. 3.3.4). The timing measurements in Fig. 6.14 show several bounces of the membranes before they approach a stable rest state at the end of the simulation. The number of edge intersections perfectly correlates with the number of collisions and the computational cost to compute the consistent penetration depth and direction. At the end of the simulation the presented methods needs 34.2 ms for a total of 5582 edge intersections caused by 1660 collisions.

Rod Scene

Self-collisions of a highly deformable rod are detected in this experiment (see Sec. 3.3.5). At the time of maximum impact the method needs 10.3 ms to find consistent penetration depths and directions for 283 self-collisions. A total of 815 edge intersections are found while doing this (see Fig. 6.15). When the rod starts to relax the number of self-collisions decreases as does the computational cost. Finally, 2.6 ms are required to find 284 edge intersections and compute penetration information for 72 resting contacts at the end of the simulation.

MultiRod Scene

The last experiment tests four rods for collisions and self-collisions (see Sec. 3.3.6). During the frames of maximum impact the presented method computes consistent penetration information for 1411 colliding points in 58.1 ms and considers 4382 edge intersections for this. The timing measurements in Fig. 6.16 show that the computational cost drops to 21.2 ms at the end of the simulation when 549 collisions and 2265 edge intersections remain.

6.3 Discussion

The presented method to estimate consistent penetration information eliminates many collision response artifacts inherent to existing penetration depth approaches for discrete-time simulations. Instead of computing only the closest surface feature for colliding points, a set of consistent surface features is considered to avoid dynamic discontinuities of the penetration depth function. The penetration depth is only computed for colliding points close to the surface, whereas consistent information is propagated to colliding points with larger penetrations.

While the presented approach eliminates many collision response artifacts, there still exist configurations where a plausible collision response can not be computed.

If a colliding object is entirely enclosed by the penetrated object, the presented algorithm does not compute any penetration depth, since there are no border points. The response scheme would not generate any forces until at least one object point leaves the penetrated object. In contrast, standard approaches would compute penetration depth information for all object points and probably resolve the collision in an arbitrary direction. However, if at least one object point of a colliding object is outside the penetrated object, the presented approach is likely to compute more plausible and consistent penetration depth information for all colliding points. Furthermore, there exist cases of objects crossing each other due to high relative velocities, where neither the existing nor the proposed approach are able to compute useful penetration depth information.

The presented approach does not compute the penetration depth according to its definition. Instead of computing the shortest distance to the surface of the penetrated object, the approach approximates the penetration depth only for points close to the surface. For all colliding non-border points, the depth is propagated from border points without considering the penetrated object. This supports consistency and is desired for plausible simulations but leads to results that can differ significantly from the actual definition of penetration depth. However, this disregard of the definition eliminates many artifacts in the respective collision response scheme. If colliding points converge to the surface of a penetrated object, the computed penetration depth converges to the exact penetration depth.

The main advantage of the presented approach is its consistency. Despite the simple idea it enhances discrete-time simulations a lot by computing plausible penetration depth and direction for points of deformable objects. It requires information about adjacent points and works best when the involved objects are discretized not only on the surface but also inside. The algorithm is generally faster than standard penetration depth approaches due to the efficient propagation process. In conclusion, the method is very efficient and dramatically enhances the robustness of physically-based simulation with deformable objects.

Directions for future work

Further investigation is necessary if the simulation requires edge-edge collision handling for a more precise interaction between models. In such a case the approach needs to be extended so that it does not only compute consistent penetration depths and directions for points but also for edges. This involves detecting edge-edge intersections and a method to propagate the penetration information to deeper edges and points.

The method could benefit from optimizations in situations with many resting contacts. As example, at the end of the experiment with the stacked membranes

(see Fig. 3.3.4), a lot of point collisions are detected which themselves trigger many edge intersection tests. Unfortunately, the penetration depth of all these points is so small that it is much more efficient and equally consistent to just take the nearest surface.

Tighter coupling of the presented approach with other techniques could also lead to promising insight. For example, combining the consistent penetration depth approach with continuous collision detection could further eliminate collision handling artifacts and increase the robustness of simulation environments.

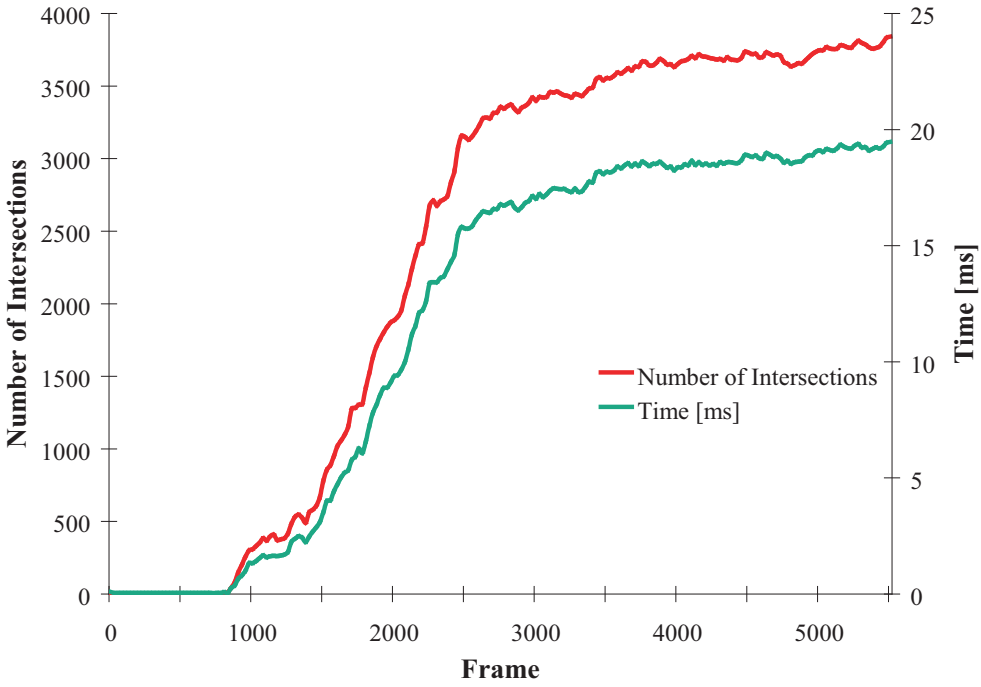


Figure 6.11: Experiment: Boldie scene.

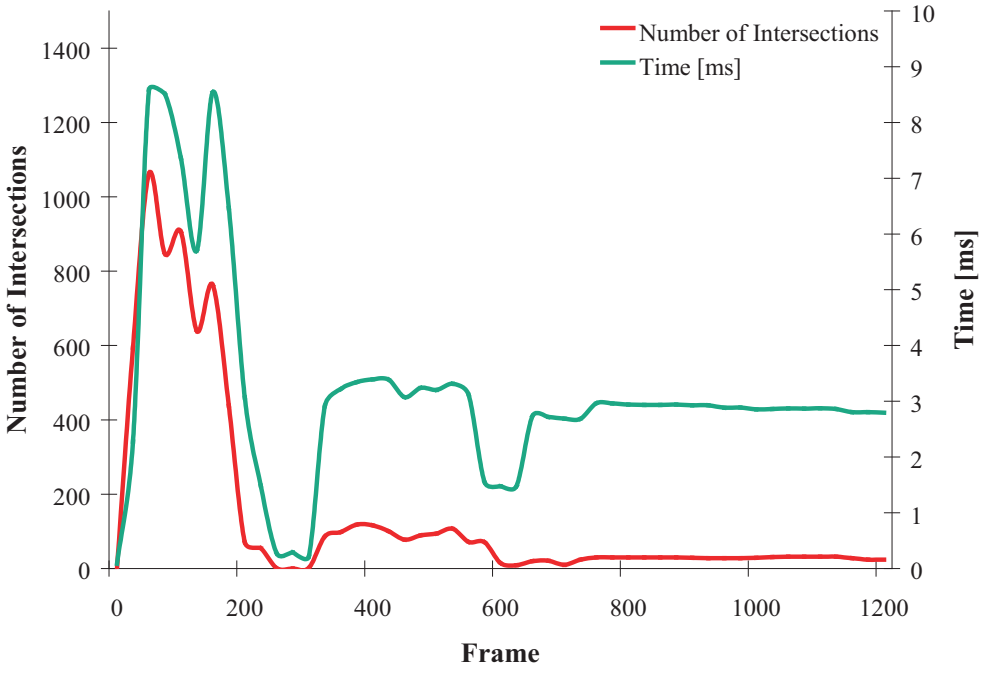


Figure 6.12: Experiment: Pitbull scene.

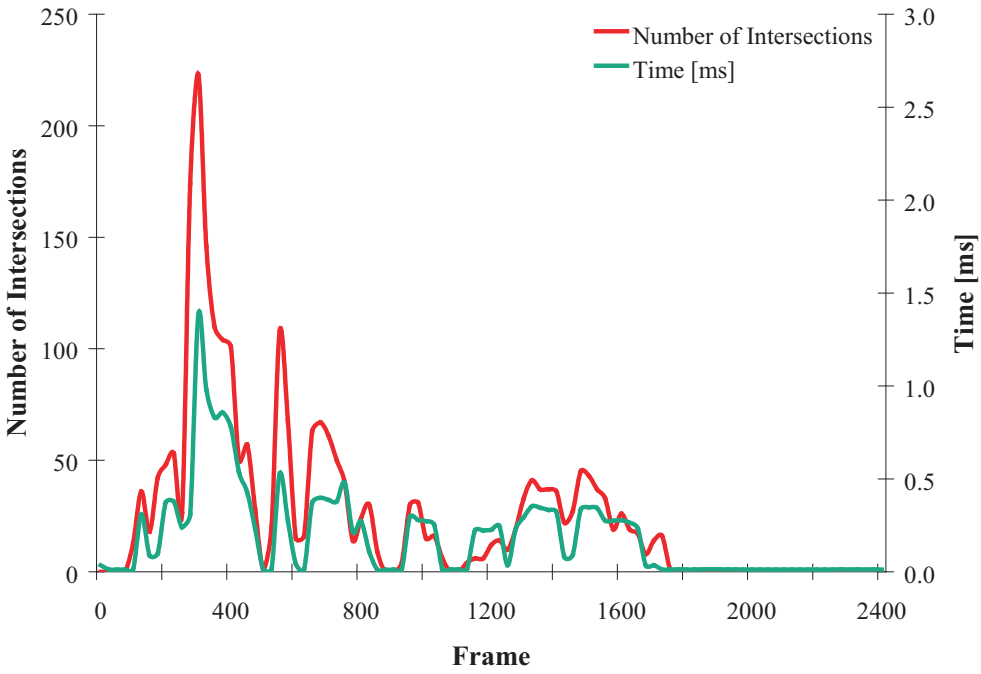


Figure 6.13: Experiment: Mixed scene.

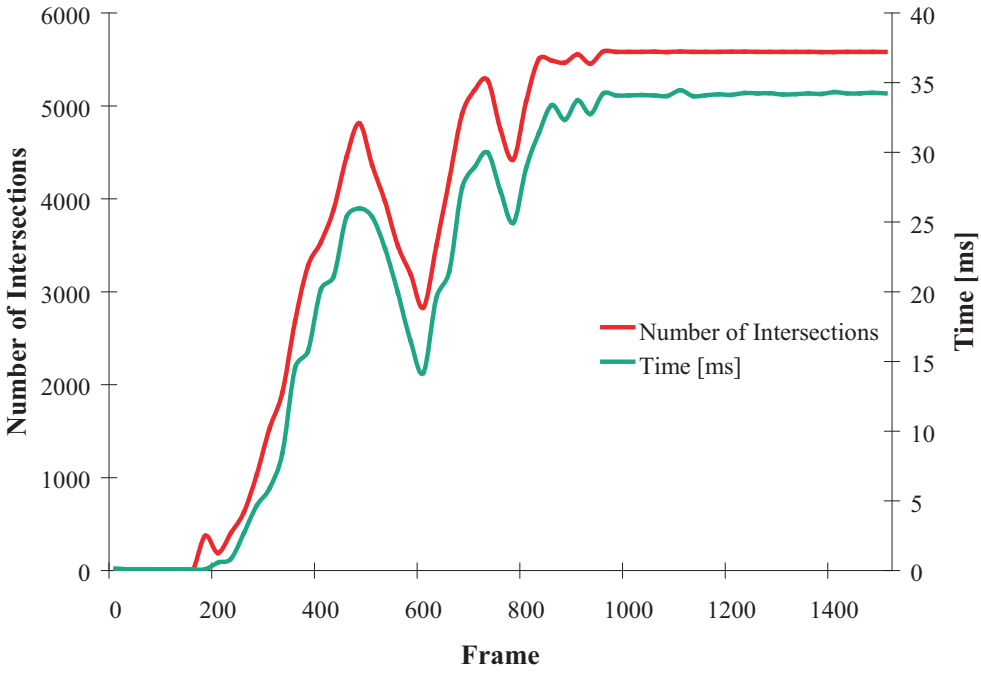


Figure 6.14: Experiment: Membrane scene.

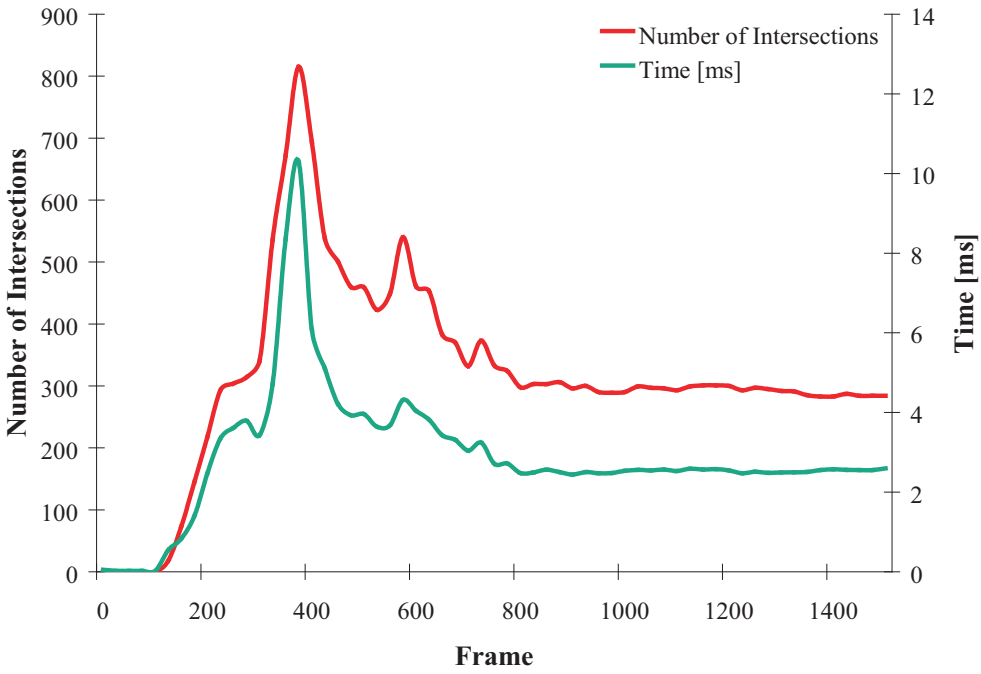


Figure 6.15: Experiment: Rod scene.

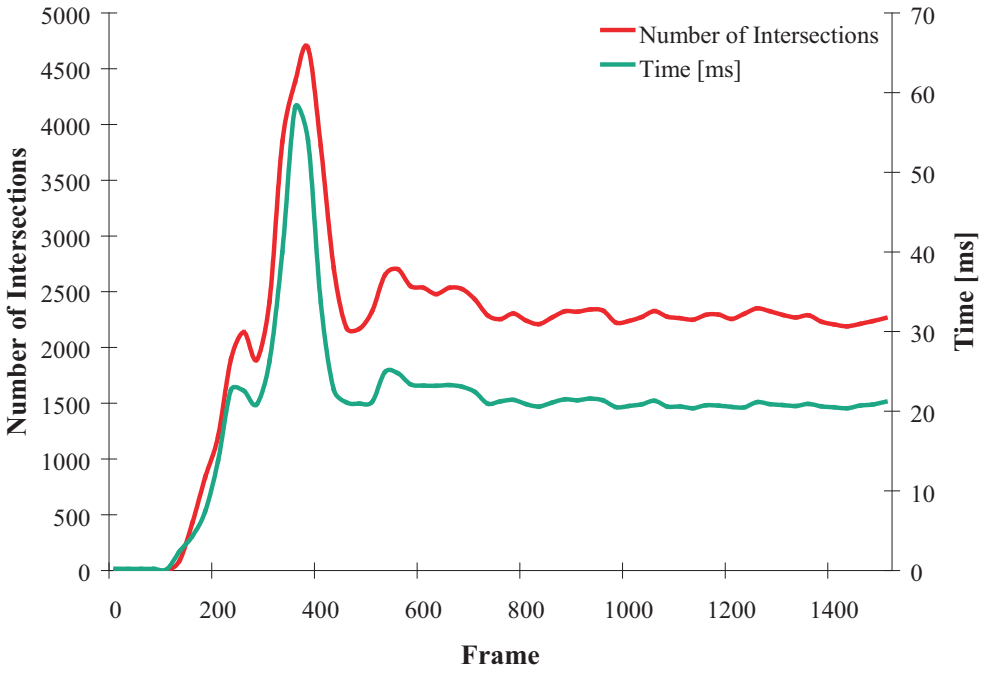


Figure 6.16: Experiment: MultiRod scene.

7 Applications

The collision handling methods described in this thesis are integrated into several applications. This section presents two of them: An interactive simulation framework (see Sec. 7.1) and a hysteroscopy simulator (see Sec. 7.2). Other projects, such as a stent placement simulation [Hei04b] or a fluid simulation [Mue04] further demonstrate the capability of the collision handling components.

7.1 DefCol Studio

DefCol Studio [Hei05] is a simulation framework that implements various methods for deformable modeling and collision handling. It consists of a low-level simulation development kit (DefCol SDK), a high-level simulation environment (DefCol Studio) and a visualization module that is based on OGRE [Ogr05] and CEGUI [Ceg05].

DefCol SDK

The DefCol SDK implements a feature-complete simulation engine for deformable objects. In addition to numerous integration methods it features all methods presented in this thesis and published in the following publications:

- Chap. 4: Optimized Spatial Partitioning
  Optimized spatial hashing for collision detection of deformable objects [Tes03].
- Chap. 5: Image-Space Collision Detection
  Real-time volumetric intersections of deforming objects [Hei03b].
  Detection of collisions and self-collisions using image-space techniques [Hei04].
- Chap. 6: Consistent Penetration Depth Estimation
  Consistent penetration depth estimation for deformable collision response [Hei04c].

The framework also includes the following components:

- A versatile and robust model for geometrically complex deformable solids [Tes04].
- Meshless deformations based on shape matching [Mue05].
- Contact surface computation for coarsely sampled deformable objects [Spi05].

The platform-independent SDK is written in C++ and exposes the functionality of the different algorithms through a clean, unified API. The user first creates a simulation instance and adds one or more models to it. So called draggers can be used to link parts of different models together or attach them to user-defined positions in space. Force fields that influence the models can be simulated by adding magnets to the simulation instance. After defining the desired physical properties of all the above entities, the simulation can be started and regularly queried for updates. Most of the simulation parameters, such as collision handling method, deformation model, integration scheme, physical model properties and dragger targets, can be modified while the simulation is running. Additional features such as configurable timing measurements, support for single/double-precision floating-point computations and platform encapsulation complement the SDK package.

DefCol Studio

The DefCol Studio is a high-level simulation environment that is built upon the DefCol SDK and the visualization module. It features an easy-to-use graphical user interface (GUI) based on CEGUI [Ceg05] with tool menus, property windows, mouse and keyboard shortcuts. DefCol Studio comes with scene management features, such as creation, loading, cloning and destroying of not only models, but also cameras, lights and draggers. A console window exposes the scripting functionality that allows access to most of the simulation properties. Models in the scene can be selected, dragged and moved directly with the mouse. All these interactions are recorded and can be played back later if desired. To this end, a flexible parser system processes all direct user input, scripts and data files and routes them to the responsible component.

The high-quality rendering is based on OGRE, a powerful open-source rendering engine [Ogr05]. It was extended to fully support deformable models and LDIs. The visual appearance of the objects is improved by adding a high-resolution surface mesh to them which deforms based on the underlying, coarser simulation geometry. This surface mesh can be textured and enhanced by advanced shaders. Multiple light sources with corresponding shadow volumes create realistically looking scenes simulated and rendered at interactive rates. An export toolbox enables capturing of screenshots, movies and POV-Ray scenes [Pov05] while the simulation is running.

Figure 7.1: The default scene of DefCol Studio with an open console window.

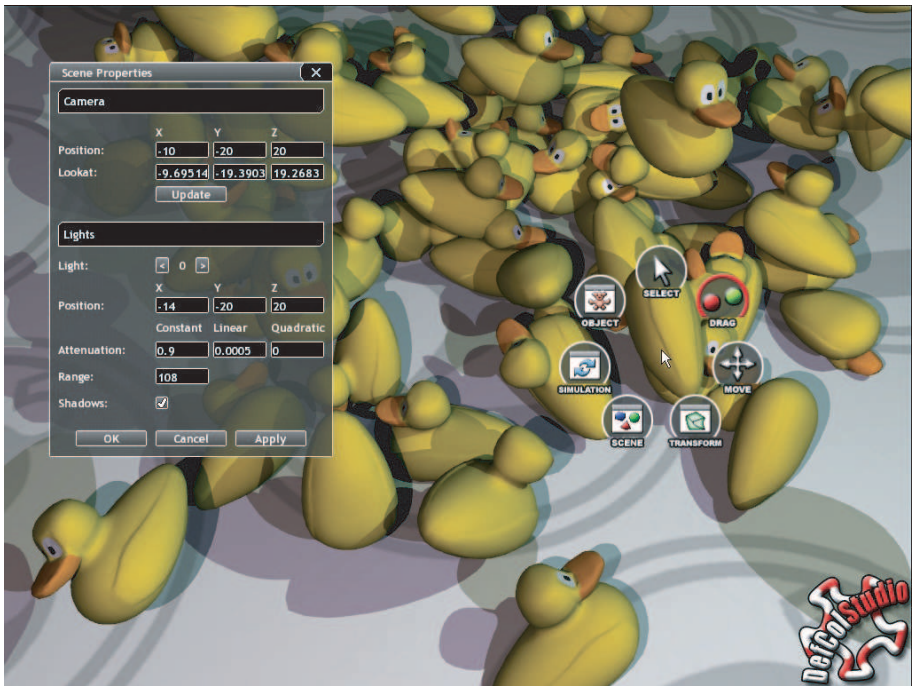


Figure 7.2: Scene properties window and the tool menu.

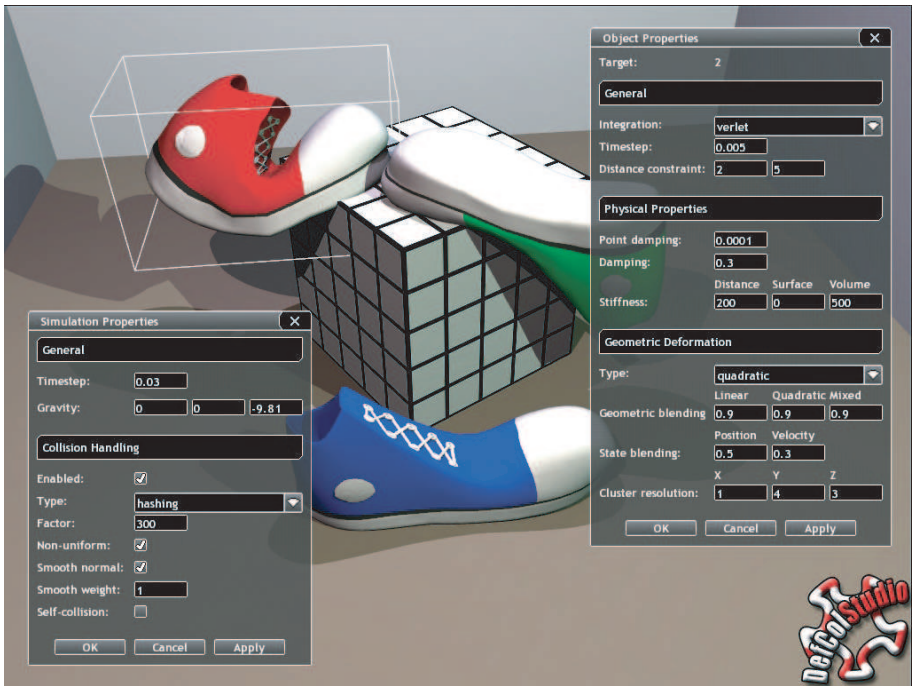


Figure 7.3: Simulation and object properties windows.

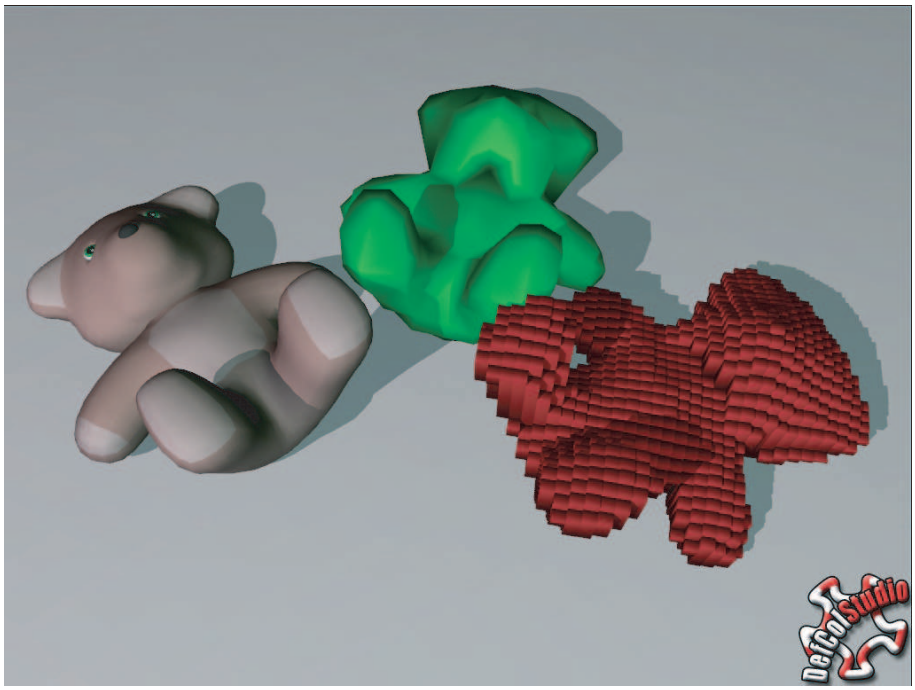


Figure 7.4: Different visualization methods: High-resolution, textured triangle mesh (left), simulated tetrahedral mesh (middle) and the LDI representation (right).

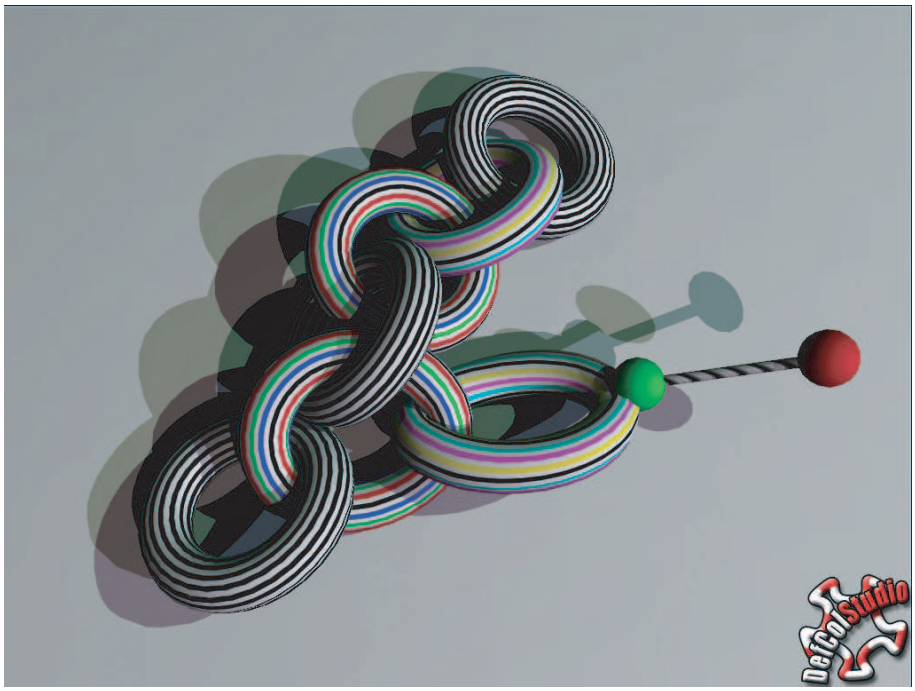


Figure 7.5: Employing the dragging tool on a deformable model.

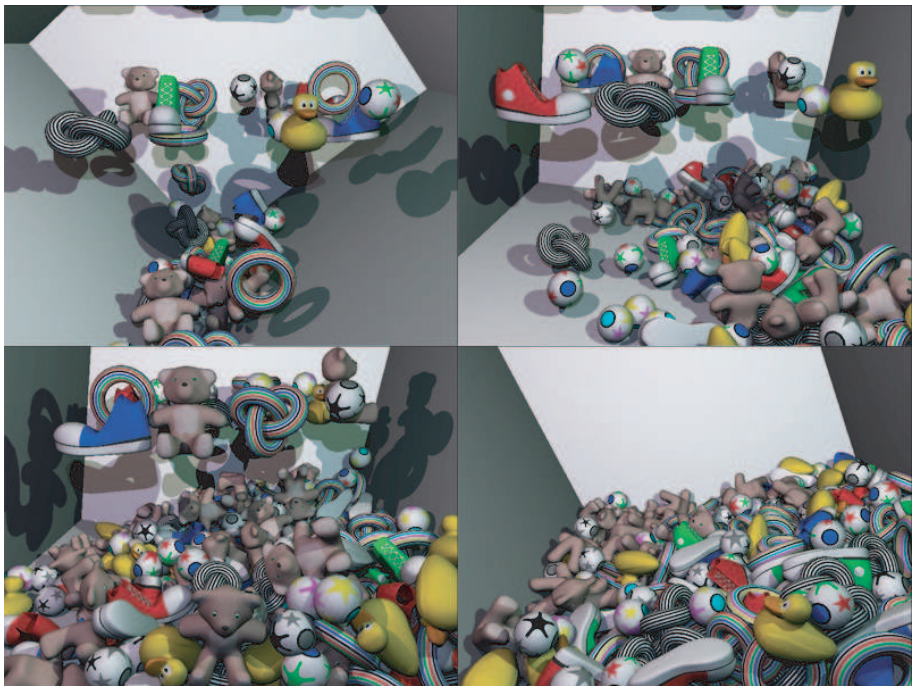


Figure 7.6: Complex scene simulated within DefCol Studio.

7.2 Hysteroscopy Simulator

Recent advances in interactive physically-based simulation have a significant influence on medical education, training and practice. They allow individuals to be immersed in dynamic computer-generated, three-dimensional environments and can provide realistic simulations of surgical procedures. Simulators for hysteroscopy and other procedures offer the promise of improving training in a low risk environment. The performance of surgeons can be quantified by presenting them different anatomical variations of pathologies.

Project members of the Swiss National Center of Competence in Research on Computer Aided and Image Guided Medical Interventions (NCCR Co-Me) [Com01] developed such a surgical simulator. The system provides an interactive simulation environment for the most common techniques in diagnostic and operative hysteroscopy, namely cervical dilation, endometrial resection and ablation and lesion excision.

The simulator is composed of multiple modules that are integrated into a generic simulator platform. These components provide simulation of soft-tissue deformation, collision handling, cutting and realistic rendering. In addition, a computational fluid dynamics (CFD) module has been integrated for blood flow simulation and a hysteroscopy tool serves as haptic input device to the simulator. Moreover, an operating room (OR) was replicated in the lab and provides a standard hysteroscopic environment for user interaction (see Fig. 7.7). In this setting, the training starts as soon as the trainee enters the OR and it ends, when he leaves the room.

A typical simulation scene consists of one uterus model and one or more pathology models (either myomas or polyps) as shown in Fig. 7.9 and 7.10. For each of these deformable objects there exist a tetrahedral mesh for the soft-tissue simulation and collision handling, as well as a high resolution triangle mesh for visualization. A particle system is employed to simulate air bubbles that appear during the surgical procedure. All mentioned models so far are influenced by other models and the hydrometra and flow simulation inside the cavity of the uterus. The user can further interact with the scene by directly controlling the exchangeable tool through the haptic device attached to the simulator hardware. This tool is treated as rigid body by the simulation and allows pushing, pulling and cutting of soft-tissue (see Fig. 7.8).

Collision handling of deformable soft-tissue is one of the greatest challenges of the simulator. As example, the insertion of a dilator into the simulated cervix is a worst-case scenario since there is nearly complete overlap of one object by the other. The hysteroscopy simulator relies on the collision handling components presented in Chap. 4 and 6 to achieve consistent and robust results even in such difficult scenarios.

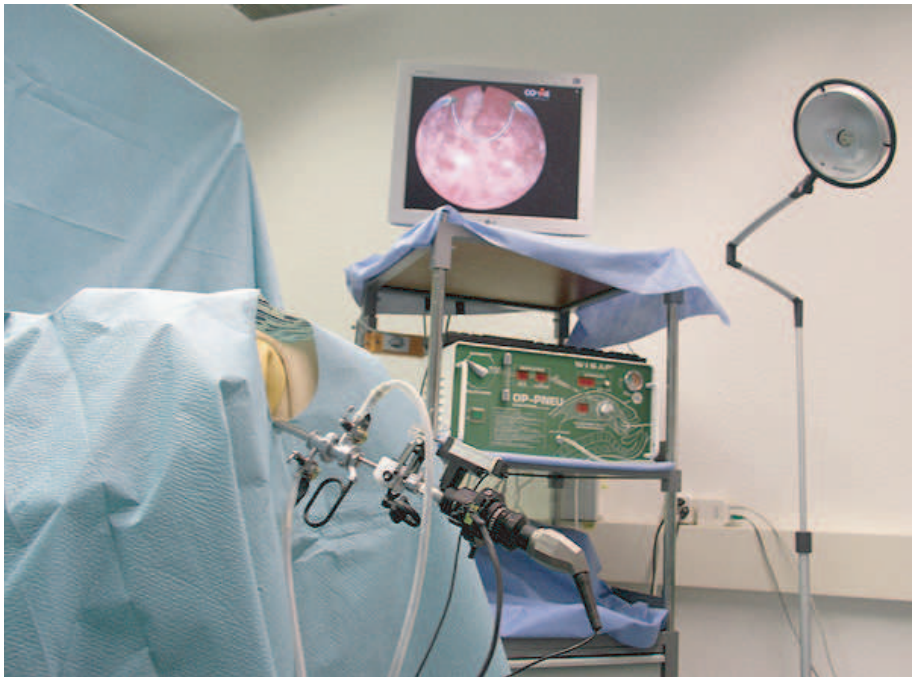


Figure 7.7: The setup of the hysteroscopy simulator with haptic device, control panel and display.

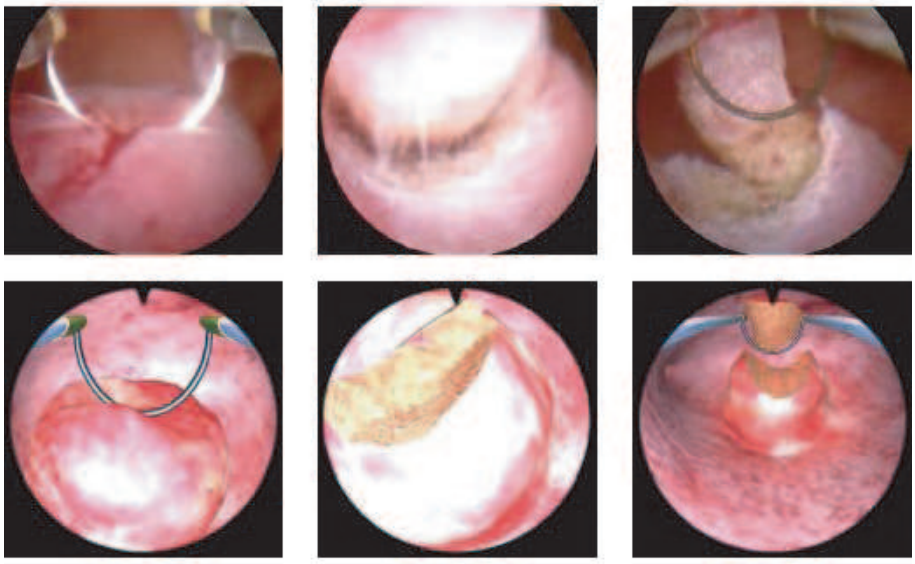


Figure 7.8: Comparison of a real surgical procedure (top row) with a simulated procedure (bottom row).

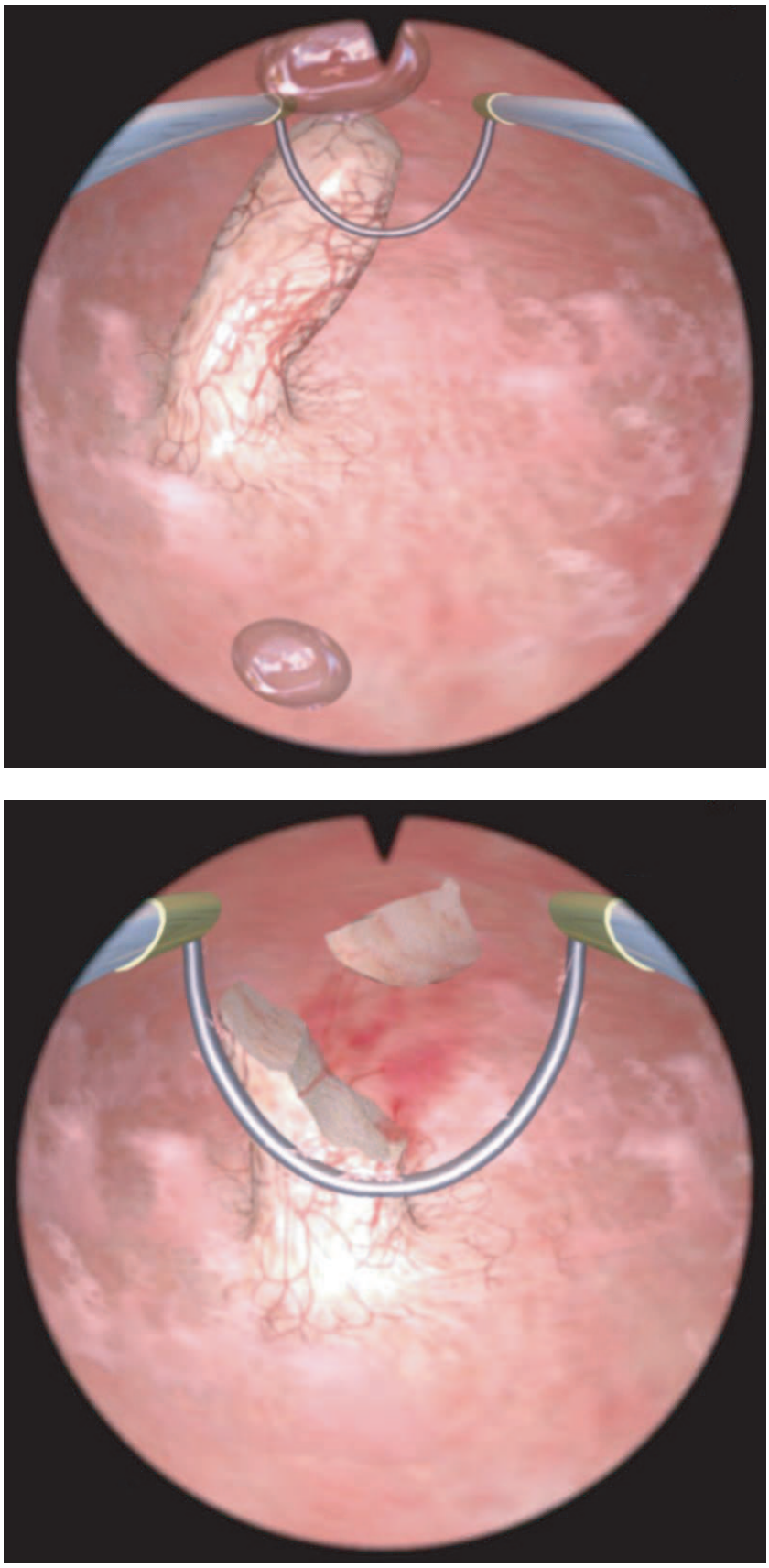


Figure 7.9: Ablation of a polyp. Deformable objects, such as the pathology and the uterus, are tested for collisions with the surgical tool in order to cut the soft-tissue. Simulation of air bubbles and bleeding enhances the visual appearance of the scene.

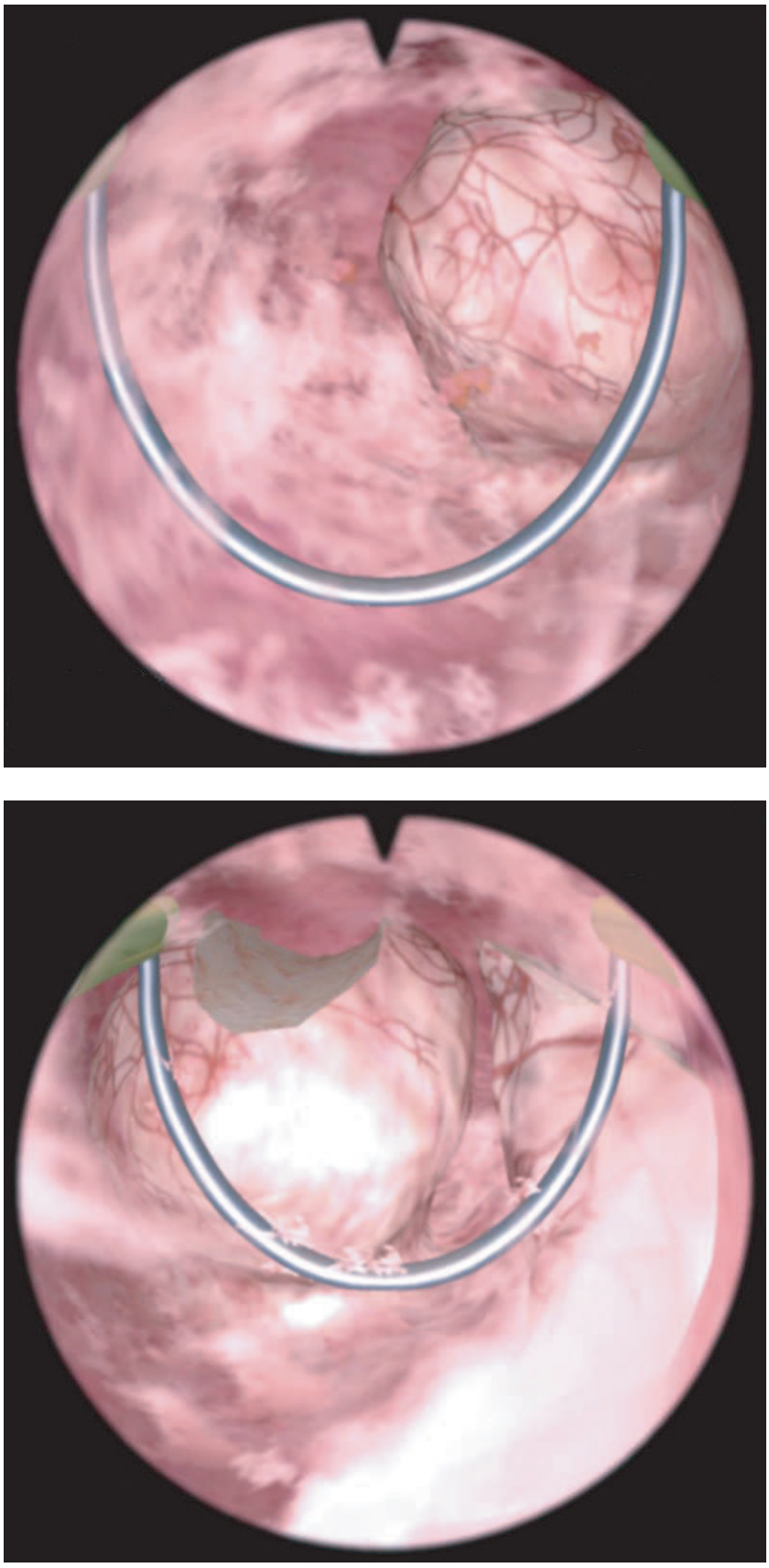


Figure 7.10: Ablation of a myoma. The fluid simulation inside the cavity influences pieces cut from the pathology, other tissue and air bubbles.

8 Conclusions

This chapter summarizes the results achieved in this thesis and gives an outlook on further research directions and possible extensions.

8.1 Summary

Interactive environments with dynamically deforming objects require efficient and robust collision handling. This thesis presented several methods that address the specific issues of this task. They robustly handle volumetric collisions, self-collisions and resting contacts based on versatile data structures that do not require preprocessing. A collection of test scenarios is used throughout the whole thesis to analyze the characteristics and the performance of each collision handling technique. The results show that they provide interactive performance for typical simulation environment complexities. While especially suited for deformable objects, the methods can also be applied to rigid bodies.

Optimized Spatial Partitioning

A spatial partitioning approach was presented that can be used for the detection of collisions between deformable objects. It is based on an optimized spatial hashing technique that efficiently finds intersections between various types of object primitives. It is independent of topology changes and provides a straight-forward solution to self-collision detection.

The main advantage of the presented approach is its versatility. Despite the simple data structure it allows to carry out n-body collision detection and self-collision detection at the same time and in one single pass. Various parameters influence the performance of the spatial hashing approach, such as the shape and size of a grid cells, the hash function and the number of the hash table entries. These parameters were first analyzed, then optimized and finally discussed.

Image-Space Collision Detection

A method was introduced which detects collisions and self-collisions of deformable objects based on an LDI decomposition of the intersection volume. This technique

allows for three volumetric collision queries: An explicit representation of the intersection volume, a vertex-in-volume check and a self-collisions test. Three possible implementations are discussed. The first two variants are accelerated by graphics hardware whereas the third one is a software-only approach.

The main advantages of the proposed method is its scalable nature and the possibility to accelerate it with graphics hardware. Unfortunately, these are also the limitations of the approach, due to approximation errors and read-back delays. Accuracy and performance can be balanced in a certain range by changing the resolution of the LDI structure. This tradeoff is controllable by the user.

Consistent Penetration Depth Estimation

A novel method to estimate consistent penetration information eliminates many collision response artifacts inherent to existing penetration depth approaches. Furthermore, the method addresses the issue of large time-steps and the corresponding collision handling artifacts. Instead of computing only the closest surface feature for colliding points, a set of consistent surface features is considered to avoid dynamic discontinuities of the penetration depth function. The penetration depth is only computed for colliding points close to the surface, whereas consistent information is propagated to colliding points with larger penetrations.

The main advantage of the presented approach is its consistency. Despite the simple idea, it enhances discrete-time simulations a lot by computing plausible penetration depth and direction for points of deformable objects. The algorithm is generally faster than standard penetration depth approaches due to the efficient propagation process. The method is very efficient and dramatically enhances the robustness of physically-based simulation with deformable objects.

DefCol Studio and Hysteroscopy Simulator

All above collision handling methods were integrated into DefCol Studio, a complete framework for interactive simulation of dynamically deforming objects. It features all necessary components for deformation, collision detection and collision response. It consists of a low-level simulation development kit, a high-level simulation environment and a visualization module. This framework was used for a thorough analysis and validation of all presented methods in this thesis.

The methods are also an essential part of a hysteroscopy simulator prototype developed by project members of the Swiss National Center of Competence in Research on Computer Aided and Image Guided Medical Interventions (NCCR Co-Me) [Com01]. This prototype provides an interactive surgical simulation environment with deformable soft tissue, surgical instruments, haptic feedback and realistic rendering.

8.2 Outlook

The presented methods are designed to work best with volumetric models. Further research is necessary to apply them to deformable models such as cloth or thin shells that do not exhibit a volume. A new definition of inside and outside needs to be found for this. Even more difficult is the support of so called meshless models or fluids that only consist of a set of points.

Discrete-time simulations inherently suffer from artifacts caused by high relative velocities and large time steps, such as missed collisions or large penetrations. The method for consistent estimation of penetration depth addresses these issues but can not solve all of them. Adding continuous techniques to the presented methods could further enhance their robustness.

Another interesting research area is the parallelization of collision handling methods. New multi-core processors, special purpose add-on cards and the increased feature set of graphics hardware promise additional computing power but require more sophisticated algorithms. Synchronization of multiple threads, concurrent memory access and data dependencies need to be investigated. Bundling several methods or even the full simulation cycle (deformation, collision detection and collision response) together on special hardware might also be worthwhile in this context.

Bibliography

[Abd02] K. Abdel-Malek, D. Blackmore, K. Joy. "Swept volumes: Foundations, perspectives, and applications." *Journal of Shape Modeling*, submitted, 2002.

[Aga00] P. K. Agarwal, J. Basch, L. J. Guibas, J. Hershberger, L. Zhang. "Deformable free space tiling for kinetic collision detection." *Workshop on Algorithmic Foundations of Robotics*, 2000.

[Aga02] P. K. Agarwal, M. T. de Berg, J. G. Gudmundsson, M. Hammar, H. J. Haverkort. "Box-Trees and R-Trees with Near-Optimal Query Time." *Journal of Discrete and Computational Geometry*, vol 28:3, pp. 291–312, 2002.

[Ama87] J. Amanatides, A. Woo. "A Fast Voxel Traversal Algorithm for Ray Tracing." *Proceedings of Eurographics*, pp. 3–9, 1987.

[Bac99] G. Baciu, W. Wong, H. Sun. "RECODE: an image-based collision detection algorithm." *Journal of Visualization and Computer Animation*, vol. 10, pp. 181–192, 1999.

[Bac02] G. Baciu, W. Wong. "Hardware-assisted self-collision for deformable surfaces." *Proceedings of Symposium on Virtual Reality Software and Technology*, pp. 129–136, 2002.

[Ban95] S. Bandi, D. Thalmann. "An adaptive spatial subdivision of the object space for fast collision detection of animating rigid bodies." *Proceedings of Eurographics*, pp. 259–270, 1995.

[Bar89] D. Baraff. "Analytical Methods for Dynamic Simulation of Non-penetrating Rigid Bodies." *Proceedings of SIGGRAPH*, pp. 223–232, 1989.

[Bar91] D. Baraff. "Coping with Friction for Non-Penetrating Rigid Body Simulation." *Journal on Computer Graphics*, vol. 25:4, pp. 31–40, 1991.

[Bar93] D. Baraff. "Issues in Computing Contact Forces for Non-Penetrating Rigid Bodies." *Algorithmica*, vol. 10, pp. 292–352, 1993.

[Bar94] D. Baraff. "Fast Contact Force Computation for Non-penetrating Rigid Bodies." *Proceedings of SIGGRAPH*, pp. 23–34, 1994.

[Bar98] D. Baraff, W. Witkin. "Large steps in cloth simulation." *Proceedings of SIGGRAPH*, pp. 43–54, 1998.

[Bar03] D. Baraff, W. Witkin, M. Kass. "Untangling cloth." *Proceedings of SIGGRAPH*, pp. 862–870, 2003.

[Bar96] R. Barzel, J. Hughes, D. N. Wood. "Plausible motion simulation for computer graphics animation." *Proceedings of Computer Animation and Simulation*, pp. 183–197, 1996.

[Ber97] G. van den Bergen. "Efficient collision detection of complex deformable models using AABB trees." *Journal of Graphics Tools*, vol. 2:4. pp. 1–13, 1997.

[Ber01] G. van den Bergen. "Proximity Queries and Penetration Depth Computation on 3D Game Objects." *Proceedings of Game Developers Conference*, 2001.

[Ben77] J. Bentley, D. Stanat, E. Williams. "The complexity of fixed-radius near neighbor searching." *Information Processing Letters*, vol. 6:6, pp. 209–212, 1977.

[Bri02] R. Bridson, R. Fedkiw, J. Anderson. "Robust treatment of collisions, contact and friction for cloth animation." *Proceedings of SIGGRAPH*, pp. 594–603, 2002.

[Bri03] R. Bridson, S. Marino, R. Fedkiw. "Simulation of clothing with folds and wrinkles." *Proceedings of Symposium on Computer Animation*, pp. 28–36, 2003.

[Cam86] S. A. Cameron, R. K. Culley. "Determining the Minimum Translational Distance Between Two Convex Polyhedra." *Proceedings of Robotics and Automation*, pp. 591–596, 1986.

[Cam97] S. Cameron. "Enhancing GJK: Computing Minimum and Penetration Distance Between Convex Polyhedra." *Proceedings of Robotics and Automation*, pp. 3112–3117, 1997.

[Can86] J. F. Canny. "Collision detection for moving polyhedra." *Transactions on Pattern Analysis and Machine Intelligence*, vol. 8:22, pp. 200–209, 1986.

[Cap02] S. Capell, S. Green, B. Curless, T. Duchamp, Z. Popovic. "Interactive skeleton-driven dynamic deformations." *Proceedings of SIGGRAPG*, pp. 586-593, 2002.

[Ceg05] *Crazy Eddie's GUI System*, http://www.cegui.org.uk.

[Coh95] J. D. Cohen, M. C. Lin, D. Manocha, M. K. Ponamgi. "I-COLLIDE: An interactive and exact collision detection system for largescale environments." *Proceedings of Interactive 3D Graphics*, pp. 189-196, 1995.

[Com01] *Swiss National Center of Competence in Research on Computer Aided and Image Guided Medical Interventions (NCCR Co-Me)*, Swiss National Science Foundation (SNSF), http://www.co-me.ch.

[Com05] D. S. Coming, O. G. Staadt. "Kinetic Sweep and Prune for Collision Detection." *Proceedings of Virtual Reality Interactions and Physical Simulations*, pp. 81-90, 2005.

[Cor90] T. Cormen, C. Leiserson, R. Rivest. *Introduction to Algorithms*, The MIT Press, 1990.

[Deb01] G. Debunne, M. Desbrun, M. P. Cani, A. H. Barr. "Dynamic Real-Time Deformations Using Space and Time Adaptive Sampling." *Proceedings of SIGGRAPH*, pp. 31–36, 2001.

[Des95] M. Desbrun, M. P. Cani. "Animating soft substances with implicit surfaces." *Proceedings of SIGGRAPH*, pp. 287–290, 1995.

[Des96] M. Desbrun, M. P. Cani. "Smoothed particles: A new paradigm for animating highly deformable bodies." *Proceedings of SIGGRAPH*, pp. 61–76, 1996.

[Des99] M. Desbrun, P. Schröder, A. Barr. "Interactive Animation of Structured Deformable Objects." *Proceedings of Graphics Interface*, pp. 1–8, 1999.

[Dob93] D. Dobkin, J. Hershberger, D. Kirkpatrick, S. Suri. "Computing the Intersection Depth of Polyhedra." *Algorithmica*, vol. 9, pp. 518–533, 1993.

[Ehm00] S. Ehmann, M. Lin. "SWIFT: Accelerated proximity queries between convex polyhedra by multi-level voronoi marching." *Technical Report No. TR00-026*, University of North Carolina at Chapel Hill, 2000.

[Ehm01] S. Ehmann, M. C. Lin. "Accurate and Fast Proximity Queries Between Polyhedra Using Convex Surface Decomposition." *Computer Graphics Forum*, vol. 20:3, pp. 500–510, 2001.

[Eve01] C. Everitt. "Interactive order-independent transparency." *Technical Report*, NVIDIA Corporation, 2001.

[Fau96] F. Faure. "An Energy-Based Method for Contact Force Computation." *Proceedings of Eurographics*, pp. 357–366, 1996.

[Fis01] S. Fisher, M. C. Lin. "Deformed Distance Fields for Simulation of Non-Penetrating Flexible Bodies." *Proceedings of Computer Animation and Simulation*, pp. 99–111, 2001.

[Fol90] J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes. *Computer Graphics: Principles and Practics*, Addison-Wesley Publishing Company, 1990.

[Fri00] S. F. Fischer, R. N. Perry, A. P. Rockwood, T. R. Jones. "Adaptively sampled distance fields: A general representation of shape for computer graphics." *Proceedings of SIGGRAPH*, pp. 249-254, 2000.

[Fuc80] H. Fuchs, Z. M. Kedem, B. F. Naylor. "On Visible Surface Generation by A Priori Tree Structures." *Proceedings of SIGGRAPH*, pp. 124–133, 1980.

[Gan00] F. Ganovelli, J. Dingliana, C. O'Sullivan. "BucketTree: Improving collision detection between deformable objects." *Proceedings of Spring Conference on Computer Graphics*, pp. 156–163, 2000.

[Gas93] M. P. Gascuel. "An Implicit Formulation for Precise Contact Modeling Between Flexible Solids." *Proceedings of SIGGRAPH*, pp. 313–320, 1993.

[Gil88] E. G. Gilbert, D. W. Johnson, S. S. Keerthi. "A Fast Procedure for Computing the Distance Between Objects in Three-Dimensional Space." *Journal on Robotics and Automation*, vol. 4, pp. 193–203, 1988.

[Gol87] J. Goldsmith, J. Salmon. "Automatic Creation of Object Hierarchies for Ray Tracing." *Computer Graphics and Applications*, vol. 7:5, pp. 14–20, 1987.

[Got96] S. Gottschalk, M. C. Lin, D. Manocha. "OBB-Tree: A Hierarchical Structure for Rapid Interference Detection." *Proceedings of SIGGRAPH*, pp. 171–180, 1996.

[Gov03] N. Govindaraju, S. Redon, M. Lin, D. Manocha. "CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware." *Proceedings of Graphics Hardware*, pp. 25–32, 2003.

[Gov05] N. Govindaraju, M. Lin, D. Manocha. "Quick-CULLIDE: Efficient Inter- and Intra-Object Collision Culling using Graphics Hardware." *Proceedings of Virtual Reality*, pp. 59-66, 2005.

[Gre93] N. Greene, M. Kass, G. Miller. "Hierarchical Z-buffer Visibility.", pp. 231–238, 1993.

[Gre94] N. Greene. "Detecting Intersection of a Rectangular Solid and a Convex Polyhedron." *Graphics Gems IV*, pp. 74–82, 1994.

[Gre99] A. Gregory, M. Lin, S. Gottschalk, R. Taylor. "H-COLLIDE: A Framework for Fast and Accurate Collision Detection for Haptic Interaction." *Proceedings of Virtual Reality*, pp. 38–45, 1999.

[Gri02] E. Grinspun, P. Krysl, P. Schroeder. "Charms: A simple framework for adaptive simulation." *Proceedings of SIGGRAPH*, pp. 281-290, 2002.

[Gui86] L. Guibas, R. Seidel. "Computing Convolutions by Reciprocal Search." *Proceedings of Symposium on Computational Geometry*, pp. 90–99, 1986.

[Hah88] J. K. Hahn. "Realistic Animation of Rigid Bodies." *Proceedings of SIGGRAPH*, pp. 299–308, 1988.

[He97] T. He, A. Kaufman. "Collision detection for volumetric objects." *Proceedings of Visualization*, pp. 27–34, 1997.

[Heg05] M. Hegde. "Physics, Gameplay and the Physics Processing Unit." *White Paper*, AGEIA Technologies Corporation, 2005.

[Hei03] B. Heidelberger, M. Teschner, M. Gross. "Volumetric Collision Detection for Deformable Objects." *Technical Report No. 395*, Institute of Scientific Computing, ETH Zurich, Switzerland, 2003.

[Hei03b] B. Heidelberger, M. Teschner, M. Gross. "Real-time volumetric intersections of deforming objects." *Proceedings of Vision, Modeling, Visualization*, pp. 461–468, 2003.

[Hei04] B. Heidelberger, M. Teschner, M. Gross. "Detection of Collisions and Self-collisions Using Image-space Techniques." *Proceedings of Computer Graphics, Visualization and Computer Vision*, pp. 145–152, 2004.

[Hei04b] B. Heidelberger, M. Teschner, T. Frauenfelder, M. Gross. "Collision Handling of Deformable Anatomical Models for Real-Time Surgery Simulation." *Journal of Technology and Health Care*, IOS Press, pp. 235-243, vol. 12:3, 2004.

[Hei04c] B. Heidelberger, M. Teschner, R. Keiser, M. Müller, M. Gross. "Consistent Penetration Depth Estimation for Deformable Collision Response." *Proceedings of Vision, Modeling, Visualization*, pp. 339–346, 2004.

[Hei05] B. Heidelberger, M. Teschner, J. Spillmann, M. Müller. *DefColStudio - Interactive Deformable Modeling Framework*, http://www.graphics.ethz.ch/∼brunoh/defcolstudio.html.

[Hof89] C. M. Hoffmann. "Geometric and solid modeling: an introduction." *Morgan Kaufmann Publishers Inc.*, ISBN 1-55860-067-1, 1989.

[Hof01] K. E. Hoff, A. Zaferakis, M. C. Lin, D. Manocha. "Fast and simple 2D geometric proximity queries using graphics hardware." *Proceedings of Symposium on Interactive 3D Graphics*, pp. 145–148, 2001.

[Hof02] K. Hoff, A. Zaferakis, M. Lin, D. Manocha. "Fast 3D Geometric Proximity Queries Between Rigid and Deformable Models Using Graphics Hardware Acceleration." *Technical Report No. TR02-004*, University of North Carolina and Chapel Hill, 2002.

[Hub95] P. M. Hubbard. "Collision detection for interactive graphics applications." *Transactions on Visualization and Computer Graphics*, vol. 1:3, pp. 218-230, 1995.

[Hub96] P. M. Hubbard. "Approximating polyhedra with spheres for time-critical collision detection." *Transactions of Graphics*, volume 15:3, pp. 179–210, 1996.

[Jam99] D. L. James, D. K. Pai. "ArtDefo: accurate real time deformable objects." *Proceedings of SIGGRAPH*, pp. 65-72, 1999.

[Jam02] D. L. James, D. K. Pai. "Dyrt: Dynamic response textures for realtime deformation simulation with graphics hardware." *Proceedings of SIGGRAPH*, pp. 582-585, 2002.

[Jam04] D. L. James, D. K. Pai. "BD-Tree: Output-Sensitive Collision Detection for Reduced Deformable Models." *Proceedings of SIGGRAPH*, pp. 393–398, 2004.

[Kau87] A. Kaufman, E. Shimony. "3D scan-conversion algorithms for voxel-based graphics." *Proceedings of Interactive 3D Graphics*, pp. 45–75, 1987.

[Kei04] R. Keiser, M. Müller, B. Heidelberger, M. Teschner, M. Gross. "Contact Handling for Deformable Point-Based Objects." *Proceedings of Vision, Modeling, Visualization*, pp. 315–322, 2004.

[Kim02] Y. Kim, M. Otaduy, M. Lin, D. Manocha. "Fast Penetration Depth Computation for Physically-based Animation." *Proceedings of Symposium on Computer Animation*, pp. 23–31, 2002.

[Kim02b] Y. J. Kim, K. E. Hoff, M. C. Lin, D. Manocha. "Closest point query among the union of convex polytopes using rasterization hardware." *Journal of Graphics Tools*, vol. 7:4, pp. 43–51, 2002.

[Kim03] B. Kim, J. Rossignac. "Collision prediction for polyhedra under screw motion." *Proceedings of Solid Modeling and Applications*, pp. 4–10, 2003.

[Kim04] Y. J. Kim, M. C. Lin, D. Manocha. "Incremental Penetration Depth Estimation Between Convex Polytopes Using Dual-Space Expansion." *Transactions on Visualization and Computer Graphics*, vol. 10:2, pp. 152–163, 2004.

[Kim04b] S. Kimmerle, M. Nesme, F. Faure. "Hierarchy Accelerated Stochastic Collision Detection." *Proceedings of Vision, Modeling, Visualization*, pp. 307–314, 2004.

[Kle03] J. Klein, G. Zachmann. "ADB-trees: Controlling the error of time-critical collision detection." *Proceedings of Vision, Modeling, Visualization*, pp. 37–45, 2003.

[Klo96] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, K. Zikan. "Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs." *Proceedings of SIGGRAPH*, pp. 171–180, 1996.

[Kno03] D. Knott, D. Pai. "CinDeR: Collision and interference detection in real-time using graphics hardware." *Proceedings of Graphics Interface*, pp. 73–80, 2003.

[Kri98] S. Krishnan, M. Gopi, M. C. Lin, D. Manocha, A. Pattekar. "Rapid and Accurate Contact Determination between Spline Models using Shell-Trees." *Computer Graphics Forum*, vol. 17:3, pp. 315–326, 1998.

[Lar99] E. Larsen, S. Gottschalk, M. Lin, D. Manocha. "Fast proximity queries with swept sphere volumes." *Technical Report No. TR99-018*, University of North Carolina at Chapel Hill, 1999.

[Lar01] T. Larsson, T. Akenine-Möller. "Collision Detection for Continuously Deforming Bodies." *Proceedings of Eurographics*, Short Presentation, pp. 325–333, 2001.

[Lar03] T. Larsson, T. Akenine-Möller. "Efficient Collision Detection for Models Deformed by Morphing." *The Visual Computer*, vol. 19:2, pp. 164–174, 2003.

[Lef06] S. Lefebvre, H. Hoppe. "Perfect spatial hashing." *Proceedings of SIGGRAPH*, pp. 579–588 , 2006.

[Lev66] C. Levinthal. "Molecular model-building by computer." *Scientific American*, vol 214, pp. 42–52, 1966.

[Lin91] M. C. Lin, J. F. Canny. "A fast algorithm for incremental distance calculation." *Proceedings of Robotics and Automation*, pp. 1008–1014, 1991.

[Lis98] D. Lischinski, A. Rappoport. "Image-Based Rendering for Non-Diffuse Synthetic Scenes." *Proceedings of Workshop on Rendering*, pp. 301–314, 1998.

[Lom99] J. C. Lombardo, M.-P. Cani, F. Neyret. "Real-time Collision Detection for Virtual Surgery." *Proceedings of Symposium on Computer Animation*, pp. 82–91, 1999.

[McK90] M. McKenna, D. Zeltzer. "Dynamic Simulation of Autonomous Legged Locomotion." *Proceedings of SIGGRAPH*, pp. 29–38, 1990.

[McK90b] B. J. McKenzie, R. Harries, T. Bell. "Selecting a hash algorithm." *Software-Practice & Experience*, vol. 20:2, pp. 209–224, 1990.

[Met92] D. Metaxas, D. Terzopoulos. "Dynamic deformation of solid primitives with constraints." *Proceedings of SIGGRAPH*, pp. 309-312, 1992.

[Mel00] S. Melax. "Dynamic plane shifting BSP traversal." *Proceedings of Graphics Interface*, pp. 213–220, 2000.

[Mez03] J. Mezger, S. Kimmerle, O. Etzmuß. "Hierarchical Techniques in Collision Detection for Cloth Animation." *Proceedings of Computer Graphics, Visualization and Computer Vision*, pp. 322–329, 2003.

[Mir97] B. Mirtich. "Efficient algorithms for two-phase collision detection." *Technical Report No. TR-97-23*, Mitsubishi Electric Research Laboratory, 1997.

[Moe97] T. Möller, B. Trumbore. "Fast, minimum storage ray-triangle intersection." *Journal of Graphics Tools*, vol 2:1, pp. 21–28, 1997.

[Moo88] M. Moore, J. Wilhelms. "Collision Detection and Response for Computer Animation." *Proceedings of SIGGRAPH*, pp. 289–298, 1988.

[Mue02] M. Müller, J. Dorsey, L. McMillan, R. Jagnow, B. Cutler. "Stable Real-Time Deformations." *Proceedings of Symposium on Computer Animation*, pp. 49–54, 2002.

[Mue04] M. Müller, S. Schirm, M. Teschner, B. Heidelberger, M. Gross. "Interaction of Fluids with Deformable Solids." *Proceedings of Symposium on Computer Animation & Social Agents*, pp. 159-171, 2004.

[Mue04b] M. Müller, M. Gross. "Interactive virtual materials." *Proceedings of Graphics Interface*, pp. 239–246, 2004.

[Mue04c] M. Müller, R. Keiser, A. Nealen, M. Pauly, M. Gross, M. Alexa. "Point based animation of elastic, plastic and melting objects." *Proceedings of Computer Animation*, pp. 141-151, 2004.

[Mue05] M. Müller, B. Heidelberger, M. Teschner, M. Gross. "Meshless Deformations Based on Shape Matching." *Proceedings of SIGGRAPH*, pp. 471–478, 2005.

[Mue06] M. Müller, B. Heidelberger, M. Hennix, J. Ratcliff. "Position Based Dynamics." *Proceedings of VRIPhys*, pp. 71–80, 2006.

[Mys95] K. Myszkowski, O. Okunev, T. Kunii. "Fast collision detection between complex solids using rasterizing graphics hardware." *The Visual Computer*, vol. 11:9, pp. 497–512, 1995.

[Nea05] A. Nealen, M. Mller, R. Keiser, E. Boxerman, M. Carlson. "Physically Based Deformable Models in Computer Graphics." *Proceedings of Eurographics (State of the Art Reports)*, pp. 71–94, 2005.

[Ogr05] *OGRE - Object-Oriented Graphics Rendering Engine*, http://www.ogre3d.org.

[Sul03] C. O'Sullivan, J. Dingliana, T. Giang, M. K. Kaiser. "Evaluating the Visual Fidelity of Physically Based Animations." *Proceedings of SIGGRAPH*, pp. 527–536, 2003.

[Pal95] I. J. Palmer, R. L. Grimsdale. "Collision Detection for Animation using Sphere-Trees." *Computer Graphics Forum*, vol. 14:2, pp. 105–116, 1995.

[Pau04] M. Pauly, D. K. Pai, L. J. Guibas. "Quasi-Rigid Objects in Contact." *Proceedings of Symposium on Computer Animation*, pp 109–119, 2004.

[Pen89] A. Pentland, J. Williams. "Good vibrations: Modal dynamics for graphics and animation." *Proceedings of SIGGRAPH*, pp. 215-222, 1989.

[Pla88] J. C. Platt, A. H. Barr. "Constraint Methods for Flexible Models." *Proceedings of SIGGRAPH*, pp. 279–288, 1988.

[Pov05] *POV-Ray - Persistence of Vision Raytracer*, http://www.povray.org.

[Pro97] X. Provot. "Collision and Self-Collision Handling in Cloth Model Dedicated to Design Garments." *Proceedings of Graphics Interface*, pp. 177–189, 1997.

[Rab76] M. O. Rabin. "Probabilistic algorithms." *Algorithms and complexity: new directions and recent results*, Academic Press, pp. 21–39, 1976.

[Rag04] L. Raghupathi, L. Grisoni, F. Faure, D. Marchal, M. P. Cani, C. Chaillou. "An intestine surgery simulator: Real-time collision processing and visualization." *Transactions on Visualization and Computer Graphics*, vol. 10:6, pp. 708–718, 2004.

[Red00] S. Redon, A. Kheddar, S. Coquillart. "An algebraic solution to the problem of collision detection for rigid polyhedral objects." *Proceedings of Robotics and Automation*, pp. 3733–3738, 2000.

[Red02] S. Redon, A. Kheddar, S. Coquillart. "Fast continuous collision detection between rigid bodies." *Proceedings of Eurographics*, pp. 279–288, 2002.

[Red04] S. Redon, Y. J. Kim, M. C. Lin, D. Manocha. "Fast Continuous Collision Detection for Articulated Models." *Proceedings of Symposium Solid Modeling and Applications*, pp. 126-137, 2004.

[Red04b] S. Redon, Y. J. Kim, M. C. Lin, D. Manocha. "Interactive and continuous collision detection for avatars in virtual environments." *Proceedings of Virtual Reality*, pp. 117–124, 2004.

[Rou85] N. Roussopoulos, D. Leifker. "Direct spatial search on pictorial databases using packed R-trees." *Proceedings of SIGMOD*, pp. 17–31, 1985.

[Sch02] F. Schwarzer, M. Saha, J. C. Latombe. "Exact collision checking of robot paths." *Workshop on Algorithmic Foundations of Robotics*, 2002.

[She02] C. Shen, K. Hauser, C. Gatchalian, J. O'Brien. "Model analysis for real-time viscoelastic deformation." *Proceedings of SIGGRAPH (Conference Abstracts and Applications)*, 2002.

[Shi91] M. Shinya, M. Forgue. "Interference detection through rasterization." *Journal of Visualization and Computer Animation*, vol. 2, pp. 132-134, 1991.

[Sig03] C. Sigg, R. Peikert, M. Gross. "Signed distance transform using graphics hardware." *Proceedings of Visualization*, pp. 83–90, 2003.

[Spi05] J. Spillmann, M. Teschner. "Contact Surface Computation for Coarsely Sampled Deformable Objects." *Proceedings of Vision, Modeling, Visualization*, pp. 289–296, 2005.

[Sud04] A. Sud, M. A. Otaduy, D. Manocha. "DiFi: Fast 3D Distance Field Computation Using Graphics Hardware." *Proceedings of Eurographics*, pp. 557–566, 2004.

[Sut74] I. E. Sutherland, G. W. Hodgman. "Reentrant polygon clipping." *Communications of the ACM*, vol. 17:1, pp. 32–42, 1974.

[Tel91] S. J. Teller, C. H. Sequin. "Visibility preprocessing for interactive walkthroughs." *Proceedings of SIGGRAPH*, pp. 61–71, 1991.

[Ter03] J. Teran, S. Blemker, V. N. T. Hing, R. Fedkiw. "Finite volume methods for the simulation of skeletal muscle." *Proceedings of Symposium on Computer Animation*, pp. 68–74, 2003.

[Ter87] D. Terzopoulos, J. C. Platt, A. H. Barr. "Elastically Deformable Models." *Proceedings of SIGGRAPH*, pp. 205–214, 1987.

[Tes03] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranets, M. Gross. "Optimized spatial hashing for collision detection of deformable objects." *Proceedings of Vision, Modeling, Visualization*, pp. 47–54, 2003.

[Tes04] M. Teschner, B. Heidelberger, M. Müller, M. Gross. "A Versatile and Robust Model for Geometrically Complex Deformable Solids." *Proceedings of Computer Graphics International*, pp. 312–319, 2004.

[Tes05] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, P. Volino. "Collision Detection for Deformable Objects." *Computer Graphics Forum*, pp. 61–81, vol. 24:1, 2005.

[Tha00] U. Thatcher. "Loose octrees." *Game Programming Gems*, Charles River Media, pp. 444–453, 2000.

[Ton98] D. Tonnesen. "Dynamically Coupled Particle Systems for Geometric Modeling, Reconstruction, and Animation." *PhD thesis*, University of Toronto, 1998.

[Tur90] G. Turk. "Interactive collision detection for molecular graphics." *Technical Report No. TR90-014*, University of North Carolina at Chapel Hill, 1990.

[Uno97] S. Uno, M. Slater. "The sensitivity of presence to collision response." *Proceedings of Virtual Reality Annual International Symposium*, pp. 95-103, 1997.

[Vas01] T. Vassilev, B. Spanlang, Y. Chrysanthou. "Fast Cloth Animation on Walking Avatars." *Proceedings of Eurographics*, pp. 260–267, 2001.

[Vol94] P. Volino, N. Magnenat-Thalmann. "Efficient Self-Collision Detection on Smoothly Discretized Surface Animations using Geometrical Shape Regularity." *Computer Graphics Forum*, vol. 13:3, pp. 155–166, 1994.

[Vol95] P. Volino, M. Courshesnes, N. Magnenat-Thalmann. "Versatile and Efficient Techniques for Simulating Cloth and Other Deformable Objects." *Proceedings of SIGGRAPH*, pp. 137–144, 1995.

[Wes99] R. Westermann, L. Kobbelt, T. Ertl. "Real-time exploration of regular volume data by adaptive reconstruction of isosurfaces." *The Visual Computer*, vol 15:2, pp. 100–111, 1999.

[Wu03] J. Wu, L. Kobbelt. "Piecewise linear approximation of signed distance fields." *Proceedings of Vision, Modeling, Visualization*, pp. 513-520, 2003.

[Zac01] G. Zachmann. "Optimizing the collision detection pipeline." *Proceedings of Game Technology Conference*, 2001.

[Zac02] G. Zachmann. "Minimal Hierarchical Collision Detection." *Proceedings of Virtual Reality Software and Technology*, pp. 121–128, 2002.

[Zha00] D. Zhang, M. Yuen. "Collision detection for clothed human animation." *Proceedings of Pacific Graphics*, pp. 328–337, 2000.

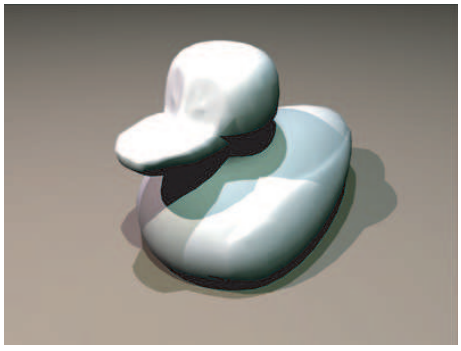
Copyrights

3D Cafe

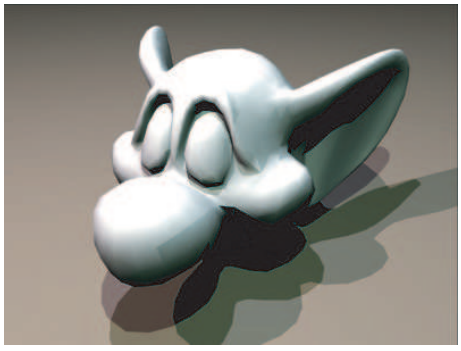
3D Cafe

Viewpoint Animation Engineering Inc.

Cyberware Inc.



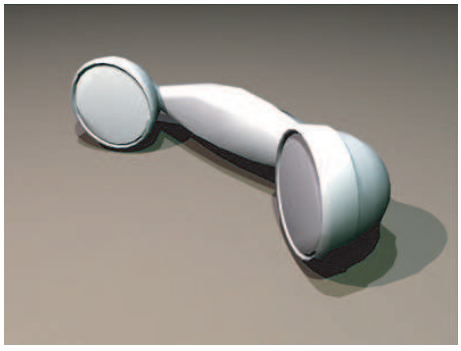
Marco Heidelberger



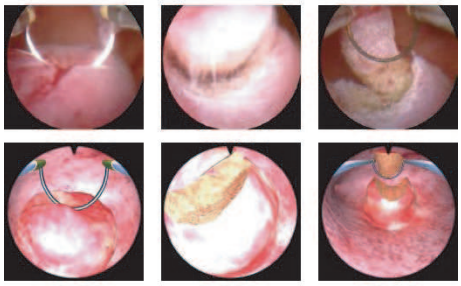
Turbosquid

Turbosquid

Turbosquid



Turbosquid



NCCR Co-Me

Curriculum Vitae

Personal Data

Name:	Bruno Heidelberger
Date of Birth:	November 17, 1972
Place of Birth:	Zurich, Switzerland

Education

2002 – 2005	Research and Teaching Assistent Computer Graphics Laboratory Swiss Federal Institute of Technology, Zurich, Switzerland
1997 – 2002	Studies in Computer Science Swiss Federal Institute of Technology, Zurich, Switzerland
1992 – 1997	Studies in Economics and Business Administration University of Zurich, Switzerland
1988 – 1992	Mathematical & Scientific High School MNG Rämibühl, Zurich, Switzerland

Publications

[Mue06] M. Müller, B. Heidelberger, M. Hennix, J. Ratcliff. "Position Based Dynamics." *Proceedings of VRIPhys*, pp. 71-80, 2006.

[Mue05] M. Müller, B. Heidelberger, M. Teschner, M. Gross. "Meshless Deformations Based on Shape Matching." *Proceedings of SIGGRAPH*, pp. 471–478, 2005.

[Tes05] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, P. Volino. "Collision Detection for Deformable Objects." *Computer Graphics Forum*, pp. 61–81, vol. 24:1, 2005.

[Hei04c] B. Heidelberger, M. Teschner, R. Keiser, M. Müller, M. Gross. "Consistent Penetration Depth Estimation for Deformable Collision Response." *Proceedings of Vision, Modeling, Visualization*, pp. 339–346, 2004.

[Kei04] R. Keiser, M. Müller, B. Heidelberger, M. Teschner, M. Gross. "Contact Handling for Deformable Point-Based Objects." *Proceedings of Vision, Modeling, Visualization*, pp. 315–322, 2004.

[Mue04] M. Müller, S. Schirm, M. Teschner, B. Heidelberger, M. Gross. "Interaction of Fluids with Deformable Solids." *Proceedings of Computer Animation & Social Agents*, pp. 159-171, 2004.

[Hei04] B. Heidelberger, M. Teschner, M. Gross. "Detection of Collisions and Self-collisions Using Image-space Techniques." *Proceedings of Computer Graphics, Visualization and Computer Vision*, pp. 145–152, 2004.

[Hei04b] B. Heidelberger, M. Teschner, T. Frauenfelder, M. Gross. "Collision Handling of Deformable Anatomical Models for Real-Time Surgery Simulation." *Journal of Technology and Health Care*, IOS Press, pp. 235-243, vol. 12:3, 2004.

[Tes04] M. Teschner, B. Heidelberger, M. Müller, M. Gross. "A Versatile and Robust Model for Geometrically Complex Deformable Solids." *Proceedings of Computer Graphics International*, pp. 312–319, 2004.

[Tes03] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranets, M. Gross. "Optimized spatial hashing for collision detection of deformable objects." *Proceedings of Vision, Modeling, Visualization*, pp. 47–54, 2003.

[Hei03b] B. Heidelberger, M. Teschner, M. Gross. "Real-time volumetric intersections of deforming objects." *Proceedings of Vision, Modeling, Visualization*, pp. 461–468, 2003.

[Hei03] B. Heidelberger, M. Teschner, M. Gross. "Volumetric Collision Detection for Deformable Objects." *Technical Report No. 395*, Institute of Scientific Computing, ETH Zurich, Switzerland, 2003.