Efficient Contact Detection for Game Engines and Robotics

A non-polygonal approach with smooth convex objects

Artur Alves Reis Leite Gonçalves

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor: Doctor Daniel Simões Lopes

Doctor Joaquim Armando Pires Jorge

Doctor Alexandre José Malheiro Bernardino

Examination Committee

Chairperson: Doctor João Fernando Cardoso Silva Sequeira

Supervisor: Doctor Joaquim Armando Pires Jorge

Member of the Committee: Rodrigo Martins de Matos Ventura

December 2015

Acknowledgments

I was fortunate to have the assistance and support of several people, who helped me not only in successfully completing this thesis, but also in boosting my education and skills. I would like to acknowledge and thank them in this section.

To my supervisors, Prof. Joaquim Jorge and Prof. Alexandre Bernardino, for sharing their knowledge and insights on the application themes of this dissertation. To Dr. Daniel Simões Lopes, who not only introduced me to the interesting world of Contact Detection, but also took the time to constructively criticize and closely oversee and review my work. His exceptional effort allowed me to significantly enhance the quality of this thesis, and to hone my scientific skills. For that I am deeply grateful.

I also acknowledge the financial support supported by national funds through the Portuguese Foundation for Science and Technology with reference TECTON-3D PTDC/EEI-SII/3154/2012.

To my mother and father, who have always encouraged and supported my education. To my grandmother, with whom I lived during my higher education studies. To my sister, for the always-welcome company and distraction from work. To my cousin Mariana, for being an amazing friend, inspiring and motivating me on our shared dream of working in the video game industry.

Thank you!

Abstract

Physics-based simulations are used in numerous applications, such as videogames and robotic action planning, to recreate the behavior of mechanical and non-mechanical systems, employing collision detection algorithms. As demand for geometric precision increases, traditional mesh-based representations and bounding volume hierarchies become inefficient and inadequate. This thesis studies the feasibility of using smooth convex surfaces, namely superellipsoids and superovoids, to model 3D bodies and perform accurate and efficient collision detection, using implicit and parametric surface representations.

In order to model smooth convex contact geometries and associated mechanical systems, an interactive sketch-based application was designed to expeditiously model articulated bodies, or multibody systems, using various superellipsoids attached to links of rigid bodies. Furthermore, the potential use of superellipsoids to model deformable contact geometries, such as terrain scenes or cloths, was also explored.

To test the efficiency and accuracy of the contact detection algorithms, a robotics case-study is considered: modeling the iCub robot's hands with superellipsoids and superovoids, and comparing the proposed algorithms with mesh-based equivalents. Superovoids are tapered transformations of superellipsoids, and adequately represent various man-made and organic objects. The proposed superovoid algorithms are based on the common normal concept and the equations are numerically solved using the Newton-Raphson method. Benchmarks show the developed algorithms are faster, and orders of magnitude more accurate than mesh-based methods, while requiring less memory to store the contact geometries. As the precision and processing speed of physics simulations depend greatly on employed collision detection algorithms, this thesis can contribute significantly to numerous areas.

Keywords: collision detection, ovoidal shape, multibody modeling, iCub robot, grasp modeling

Resumo

São aplicadas simulações físicas em vários ramos, como videojogos e controlo robótico, para recriar o comportamento de sistemas mecânicos e não-mecânicos, utilizando algoritmos de deteção de colisões. As tradicionais representações por malhas poligonais e hierarquias de volumes envolventes tornam-se ineficientes e desadequadas para requisitos de precisão geométrica altos. Esta tese estuda a viabilidade da utilização de superfícies suaves e convexas, em particular superelipsóides e superovóides, para a modelação de corpos 3D e execução de deteção de colisões eficiente e precisa, empregando representações implícitas e paramétricas.

Foi desenvolvida uma aplicação *sketch-based* interativa para modelação de sistemas mecânicos e respetivas geometrias de contacto, em que é possível modelar corpos articulados e sistemas multicorpo de maneira expedita, acoplando superelipsóides aos constituintes dos corpos. A modelação de geometrias deformáveis utilizando superovóides é também explorada.

A eficiência e precisão dos algoritmos de deteção de contacto foram testadas e avaliadas numa aplicação robótica: a modelação das mãos do robot iCub com superelipsóides e superovóides. O superovóide é uma transformação fusiforme do superelipsóide, e permite modelar vários objetos manufaturados e orgânicos. Os algoritmos para superovóides propostos baseiam-se no conceito da normal comum, e as respetivas equações são resolvidas numericamente com o método Newton-Raphson. Os resultados indicam que os algoritmos desenvolvidos são mais rápidos e precisos que algoritmos utilizando malhas, e consomem menos memória para definir as geometrias de contacto. Uma vez que a precisão e velocidade de processamento de simulações físicas depende fortemente dos algoritmos de deteção de colisões empregues, esta tese contribui significativamente para várias áreas.

Palavras-chave: deteção de colisões, forma ovóide, modelação de sistemas multicorpo, robot iCub, modelação de *grasp* robótico

Table of Contents

List of Figures

List of Tables

List of Symbols

Convention

a, A, α	Scalar
$\mathbf{a}$	Column vector
$\mathbf{A}$	Square matrix

Superscripts and subscripts

a^T, A^T	Transpose of a vector or matrix
$_i, \ _j$	Relative to surface i or j

Acronyms

2D	Two-dimensional
3D	Three-dimensional
AABB	Axis-aligned bounding box
BVH	Bounding volume hierarchy
CAD	Computer-aided design
CCD	Continuous collision detection
CPU	Central processing unit
CQM	Composite quadric model
FCL	Flexible Collision Library
FPS	Frames per second
GPU	Graphics processing unit
K-DOP	Discrete oriented polytope (K planes)
K-IOS	Intersection of spheres (K spheres)
OBB	Oriented bounding box
SCS	Smooth convex surface
SSCH	Sphere swept convex hull
WIMP	Window, icon, menu, pointer

1. Introduction

1.1. Motivation

Mechanics is the science which studies the movement of physical bodies under the action of forces and the consequent effects of this movement in their environment. A rather powerful formalism used in Mechanics is the set of equations known as Newton's laws of motion. These laws are used to mathematically describe a mechanical system and allow an analysis of the behavior of the system. For some systems with relatively simple interactions, this analysis may be carried out in an analytical, closed-form, and exact manner. For more complex systems, an analytical solution may be hard to derive, or even impossible to achieve. Yet, solutions can be attained by resorting to Computational Engineering, a discipline which calls upon the use of advanced numerical methods to solve complex engineering problems. Through careful development of these methods, physical parameters and quantities of a mechanical system can be estimated with great accuracy. To apply a computational model efficiently, the problem should first be simplified, such that only the significant behavior of the system is simulated: this may include removal of non-relevant features and linearization of non-linearities (Stein et al. 2004).

One of the most noteworthy interactions between bodies in a mechanical system is collision, i.e., when two bodies physically intersect at one point or region, exerting a force in one another (Pfeiffer & Glocker 1996; Johnson 1985). Because collisions greatly influence the behavior of a mechanical system, they must be accurately modelled and accounted.

Collision detection depends on the geometry of the considered objects. Therefore, the representation of objects' geometry should be chosen wisely, so that the simulated system is convincing when compared to its real counterpart. A popular approach is to approximate the objects by polyhedrons, also called as meshes (van den Bergen 1997). Since meshes can only represent straight edges and faces, any smooth curves must be discretized, thus, approximated using a large number of vertices to achieve a faithful representation of the surface geometry. Alternatively, objects can be defined using analytical curves and surfaces, such as spheres, ellipsoids and superellipsoids (Barr 1981). Since many real-life objects can be modelled with these simple primitives, this representation becomes interesting from a computational point of view. In addition, there exist efficient algorithms to detect collisions between these surfaces (Wellmann et al. 2008; Lopes et al. 2010).

However, polygonal meshes are the gold standard to represent contact geometries in games and interactive applications. Usually, low-resolution meshes are adopted in games (Valve Developer Community 2012) to model the geometric loci of candidate locations for contact interaction. Whenever polygonal meshes are not used, very simplified configurations of boxes, spheres, and capsules are then considered to represent an object's contact geometry (SmashWiki 2014; Valve Corporation 2015). Despite their eventual round shape, the elements of these configurations are traditionally called hitboxes. Some games, such as first-person-shooter ones, employ more detailed collision schemes than hack-and-slash genre games, for example (Figure 1.2 and Figure 1.4) as the reactions to user input

may be more perceptible in the first. Still, unavoidably, the user experiences awkward collision behaviors as accurate spatial resolution is not accounted for (TVTropes 2013).

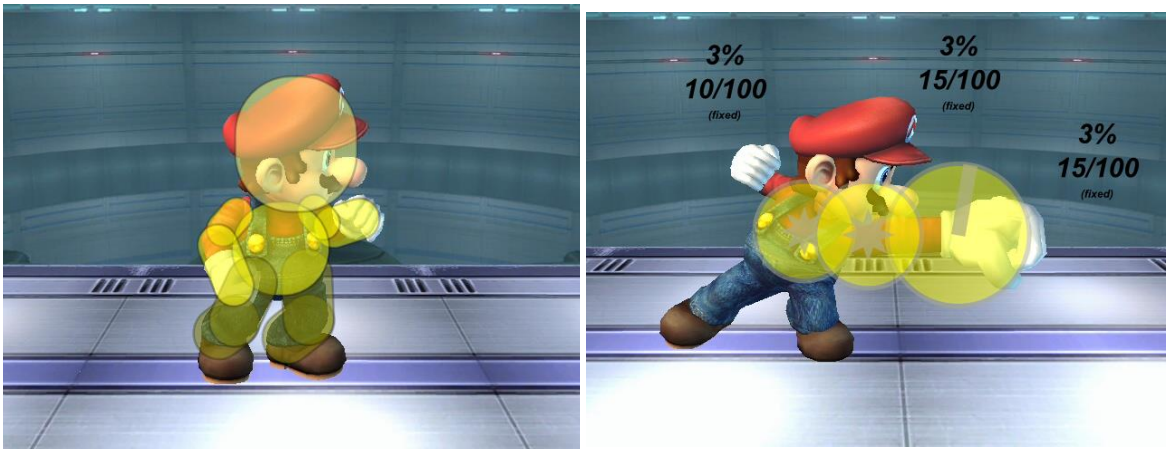


Figure 1.1 – Mario from Super Smash Bros. Brawl (Nintendo Co., Ltd. 2008) and his collision representation (hitboxes), composed of spheres and capsules. Image from (SmashWiki 2013).

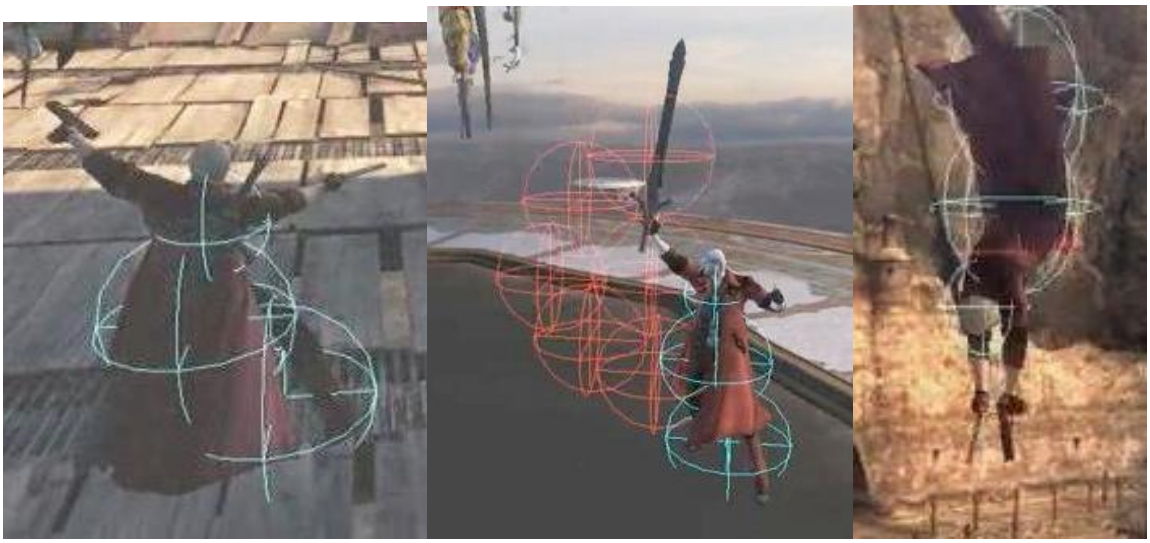


Figure 1.2 – Dante from Devil May Cry 4 (Capcom Co., Ltd. 2008) is a complex video game character but its associated collision representation consists of only three spheres mounted on top of each other (the hit spheres are outlined in light blue). His arms inevitably leave the inside of the spheres, and are not registered when contact with enemies or the scene occurs. The collision representation of the sword in the center figure uses spheres as well (outlined in red), despite being an elongated object.

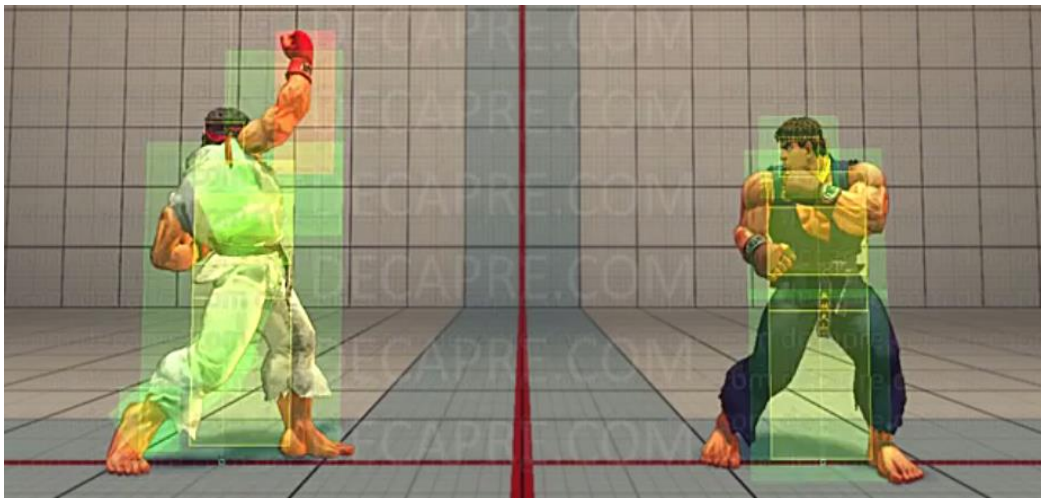


Figure 1.3 – Ryu from Ultra Street Fighter IV (Capcom Co., Ltd. 2014). Note that the tridimensional characters are loosely represented with 2D hitboxes in this fighting game. (Decapre 2014).

Figure 1.4 – Characters from Counter Strike: Global Offensive (Valve Corporation 2012), (Counter-Strike Wiki 2014). In first person shooters such as this one, gun shots in different parts of the body have varying effects on the other players' vitality, which is why collision representations are more complex compared to, for example, Devil May Cry 4 (Figure 1.2). However, they were still box-shaped (left-side images) until September 2015, when an update replaced them with capsule hitboxes (right-side images) (Valve Corporation 2015), which fit the characters' bodies more faithfully.

A huge number of commercial video games depend on physical simulation to create interactive and engaging scenarios. Many video game development tools implement physical engines, using both primitive geometric shapes and polygonal meshes (Unreal Engine, Unity3D). As mentioned before, the mesh option is typically used with low-polygon meshes. For more realistic environments, simulations with a great number of non-polygonal particles have been used, but the computational demands are so high that this simulation is outsourced to GPUs, which have to manage the workload of graphical rendering and physical calculations (Macklin et al. 2014).

Even in more serious applications, such as computational engineering simulations that require great geometric precision and accuracy to properly emulate contact phenomena, meshes are still commonly used (Stein et al. 2004). In this case, high resolution meshes are considered, which imply a heavy computational burden in terms of memory and performance (Razzaq et al. 2011).

Nevertheless, the possibility of simulating and analyzing mechanical systems, namely the contact events, is valued, and has numerous applications, of both academic and industrial natures. In pharmaceutical manufacturing, coating processes are simulated with thousands of medicine tablets tumbling and interacting with each other, so that the coating variability and effectiveness can be studied (Kodam et al. 2012). In robotics, complex machinery is simulated to accurately plan its movements, while avoiding collision between the machine and its surroundings (Tang 2014). Simulations of this kind have also been used to plan stable movements of hands for robotic grasping (Xue et al. 2009). In the animated movie industry, computational simulations of objects such as cloths are used to achieve very realistic scenes, including complex folds and wrinkles (Selle et al. 2008).

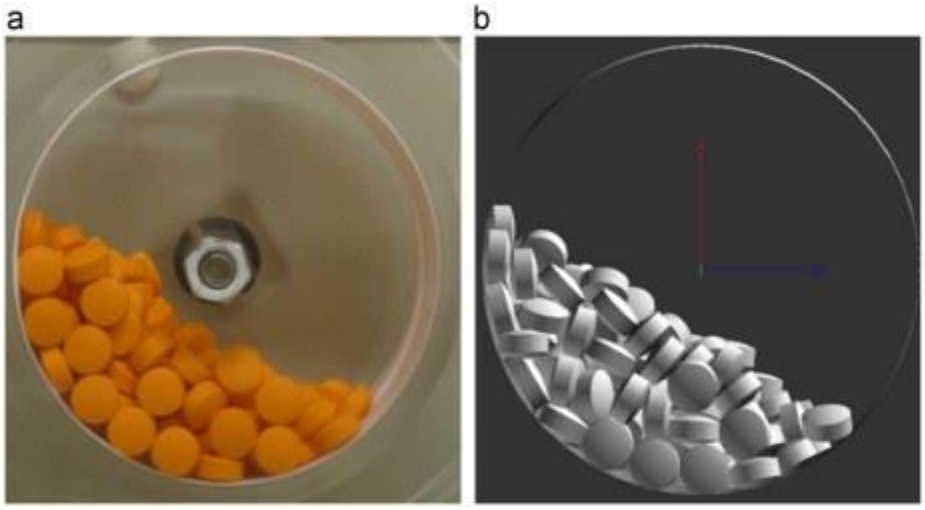


Figure 1.5 – Pharmaceutical tablets in a rotating drum, in a real-life experiment (a), and represented with the intersection of three spheres (b). {Copied from (Kodam et al. 2012) with Elsevier permission.}

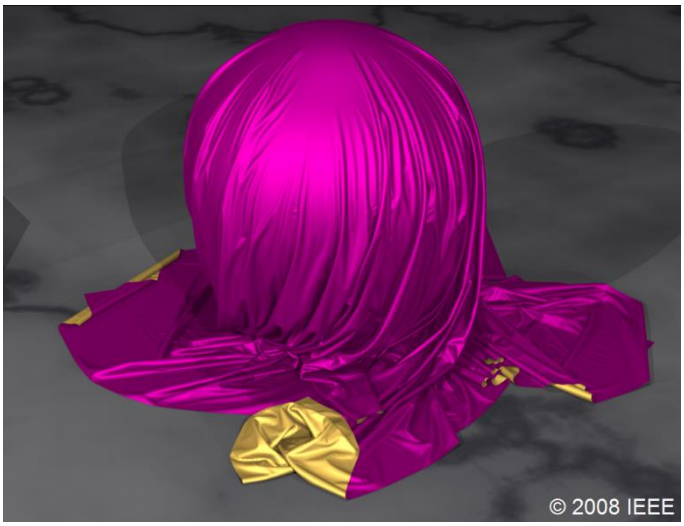


Figure 1.6 – A realistic simulation of a piece of cloth covering a suspended sphere. The cloth shows detailed bends and wrinkles, as well as sections which are folded onto themselves. {Copied from (Selle et al. 2008) with IEEE permission.}

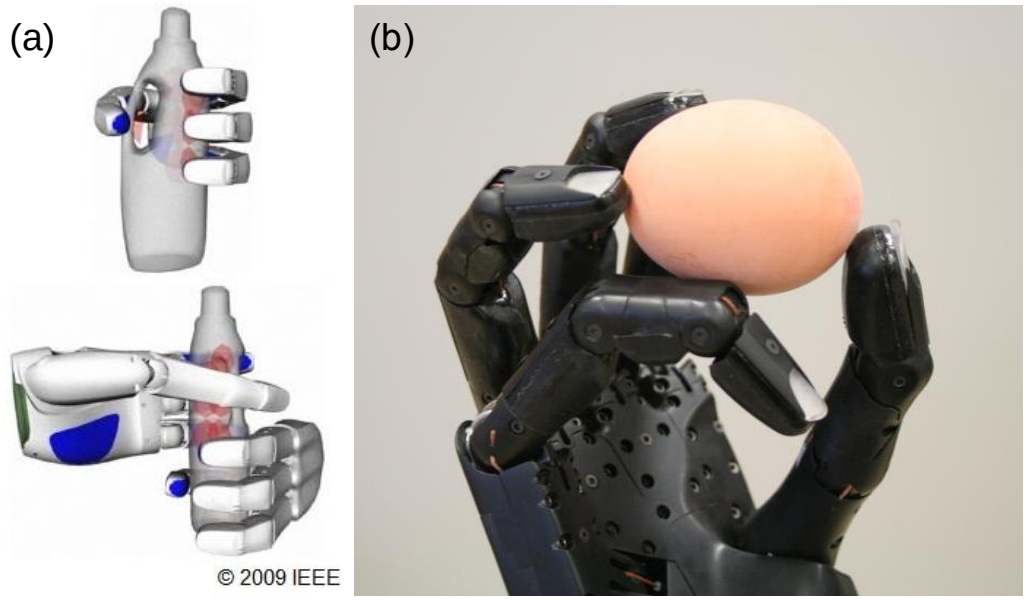


Figure 1.7 – In (a), a simulation of robotic hands grasping an object for a pick-and-place operation. In (b), the Shadow Dexterous Hand (Shadow Robot Company 2013) carefully holds an egg (Handle project 2009). {(a) adapted from (Xue et al. 2009) with IEEE permission.}

The main motivation of this thesis consists of exploring the potential of non-conventional smooth convex surfaces (SCS), in particular, superellipsoids and superovoids to provide more accurate collision detections with, at least, the same order of magnitude in terms of computational performance as the one provided by low-level polygonal contact geometries. In other words, to demonstrate that smooth convex surfaces are suitable for real-time applications with accuracy-controlled geometric resolution, thus, providing a greater realism and similar performance compared to polygonal contact geometries.

1.2. Problem statement

In a broad sense, this thesis aims to study the viability of using SCS for collision detection. This can be formulated as the following geometric problem: given two arbitrary SCS in a non-conformal configuration (arbitrary dimensions, position and orientation), what is the minimum distance between them? Based on this distance, it is possible to decide whether the two surfaces are in contact or not. In particular, this thesis will focus on superellipsoids and on superovoids (Barr 1981).

The solution to this problem should be computationally efficient, in other words, be computed in real-time to be used in interactive applications and video game engines. Prevalently, performance of video games is evaluated through the number of time steps that are rendered every second – frames per second (FPS) – as higher frame rates lead to a smoother visualization of the game world, and a better user experience for the player. Frame rates of 30 to 60 FPS are very common, which imply a time span of 30 to 16 ms for all the computations of a time step to be performed. As video games and real-time applications have to spend time for rendering graphics, the remaining time to execute collision detection algorithms is even shorter.

In addition to the study of collision detection algorithms, case studies for the developed methodologies will be considered. For a simulation to be performed, the simulated objects must first be

modelled, and their collision representations must be defined. Usually this is a time-consuming task executed in professional, complex software suites such as Maya (Autodesk, Inc. 2015) employing simple SCS such as spheres, or low-resolution meshes. The first case study, the Multibody Sketcher application, addresses this problem. With this tool, the user can easily assemble an articulated skeleton held together by joints, and flesh out each link with SCS shapes such as ellipsoids and superellipsoids, which act as collision volumes. The resulting model will be ready to be used in physical simulations, using the developed collision algorithms proposed in this thesis.

The second case study is the Terrain Cloth, an interactive application. It consists in a piece of cloth that falls on top of various smooth convex objects, specifically superellipsoids. This case-study explores the usage of SCS for modeling, as these objects are creatively placed by the user on a surface, such that the cloth gains the shape of a terrain when it settles.

The third case study involves accelerating the process of grasp planning for robotic hands. Through computer-aided planning, the path to reach out and grasp objects will be computed using the proposed collision detection algorithms, with the aim of efficiently estimating the quality of given grasping motions and configurations, with a good degree of accuracy. The different parts composing the robotic hand will be represented with the considered SCS.

1.3. Literature review

The following literature review presents basic collision detection methodologies, and subsequently focuses on algorithms which do not use meshes to represent the simulated objects. One section is dedicated to the collision detection features built into game engines. For the particularities of the three case-studies, short reviews are presented on the following themes: 3D body modeling (i), cloth modelling and simulation (ii), and collision detection applied to grasp planning (iii).

1.3.1. Discrete and continuous collision detection

There are two approaches to represent the passing of time for collisions in computational simulations. In discrete-time collision detection, time is discretized in steps, and the positions of every object are atomically changed at the end of each time step. A collision is detected when, at the end of one of these time steps, two objects are intersecting or satisfy a minimum distance tolerance. While this method is relatively easy to implement, it may lead to object interpenetration, or even miss collisions of very fast-moving objects if the time step is too coarse. Alternatively, continuous simulations identify collisions based on the precise trajectories of each object: the detection is made before the objects actually move, and the time of collision can be calculated. This is more complex, but leads to a more correct simulation, and is referred to as Continuous Collision Detection (CCD) (Ericson 2004).

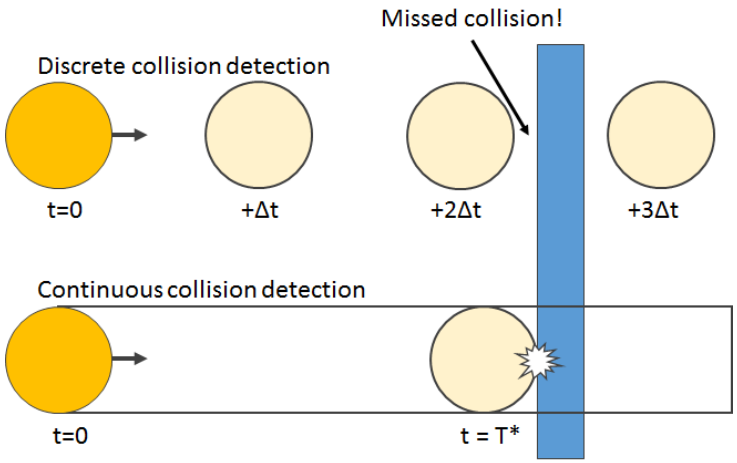


Figure 1.8 – Illustration of a missed collision when discrete collision detection is employed. In the discrete case, the speed (and thus, the displacement) of the simulated sphere is so high that it passes through the blue obstacle between the second and third time-steps. In the continuous case, the collision is reported correctly.

(Wang 2014) presents modifications applicable to existing CCD systems to make them failure-proof; that is, to completely avoid missing an existing collision, while also minimizing the false-positive collision detection. Their algorithms apply to mesh-based objects, and take into account both numerical and rounding errors. Collision detection is separated into four cases: Vertex-vertex, vertex-edge, vertex-triangle and edge-edge. The author's benchmarks to this algorithm indicated an average time of 355 ns per vertex-triangle or edge-edge collision detection, which makes this algorithm viable for real-time applications.

The works of (Jia et al. 2011) and (Choi et al. 2014) make use of continuous collision detection, and are referred to in the following sections. Also, for robotic grasp planning, (Xue et al. 2009), and the libraries proposed by (Taeubig & Frese 2012) and (Pan et al. 2012) report practical usage of CCD.

1.3.2. Broad and narrow collision detection

Collision detection can be a computationally expensive task if the problem being simulated involves a high number of bodies. Therefore, any naïve approach of testing the collision between every possible pair of bodies can become infeasible. A solution is to discretize the simulated space in regions, such as grid cells, and considering collisions only between objects which are inside the same cell, or in adjacent cells. Because objects in non-adjacent cells are too far apart to be colliding, the corresponding tests are skipped, and a gain in efficiency is achieved. An even more efficient approach is sub-dividing each space cell into smaller sub-cells, in a hierarchical fashion, in an effort to minimize the number of objects inside each cell. This method is traditionally called Space Partitioning (Ericson 2004).

A similar problem occurs if each considered body is very complex, and each proximity query becomes time-consuming. These proximity queries may not always result in useful computational work, as the bodies might not collide during a significant part of the simulation. For this reason, it may be efficient to approximate each complex object by simpler geometric primitives which fully enclose the object. This is also a form of broad collision detection, and the geometric primitives are called bounding volumes. The objective is to first execute the collision detection between these bounding volumes, which are faster

because of the simplified geometry of the bounding volumes. If these do not indicate a collision between the bounding volumes, it is certain that the two objects are not colliding. Therefore, no more processing is needed regarding those two objects. Similarly to the space partitioning methods, bounding volume hierarchies can be employed to boost efficiency: the levels of the hierarchy are composed of an increasing number of bounding volumes, fitting the considered object more and more tightly.

Conversely, the algorithms that consider two elemental objects in the simulation and test for collision between them are called narrow collision detection algorithms. These are only executed if the broad collision detection cannot exclude the possibility of contact between the two objects.

There are many well-studied and commonly-used bounding volumes (van den Bergen 1997), for which there are simple and efficient collision tests. A selection is depicted in Figure 1.9 (a), and the shapes are presented in the following bullet points:

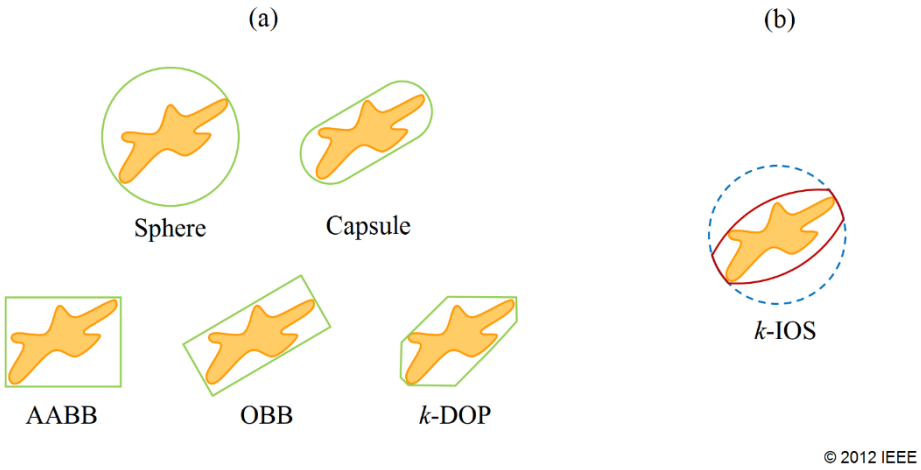
- Spheres are widely used as collision bounding volumes, as the problem of finding the minimum distance between two spheres has a simple closed-form solution: two spheres are intersecting if the Euclidean distance between their centers is lower than the sum of the radii of the spheres.

- Capsules are the union of a cylinder with two spheres on top, all with the same radius. They can also be seen as the swept volume taken by a sphere as it moves in a straight line segment. Collision between capsules in arbitrary poses can always be calculated using the distance between their central axis segment, and comparing it to the sum of their radii.

- Axis-aligned bounding boxes (AABB) are boxes whose edges are all collinear with one of the Cartesian coordinate axes, X, Y and Z. Collision detection is trivial with these boxes, by interpreting each dimension as the interval of coordinates in which the box exists. However, if the enclosed object rotates, the AABB must be recalculated to avoid part of the object from leaking through the box.

- Oriented bounding boxes (OBB) are a generalization of AABB that can be rotated with the object as it moves, providing a tighter fit and avoiding the constant recalculation of the box. However, the collision detection is computationally heavier.

- Discrete oriented polytopes (DOP) are another generalization of the AABB: they are composed of K faces (K-DOP), and are built by considering a number of appropriately-oriented planes far away from the object, which are progressively advanced until they collide with the object. The bounding volume is the intersection of the half-spaces bounded by these planes, and is always a convex polytope. With 6 planes perpendicular to each axis, the 6-DOP degenerates into a simple AABB. With 18 planes, all the edges of an AABB can be beveled, creating a tighter fit around the object. With 26 planes, all the corners can also be beveled. K-DOPs can be seen as a particularization of a convex bounding mesh, and the collision detection is increasingly more expensive to compute with higher number of planes.



© 2012 IEEE

Figure 1.9 – Examples of broad-phase bounding volumes, fully enclosing a freeform shape. (a) commonly-used shapes found in many applications and collision detection libraries; (b) the novel Intersection of Spheres bounding volume proposed in (Zhang & Kim 2012). {Adapted from (Zhang & Kim 2012) with IEEE permission.}

The usage of the intersection of a number of spheres (k-IOS) as a bounding volume has been proposed in (Zhang & Kim 2012). The k-IOS is strictly convex, and provides a tight fit around objects, while maintaining computational efficiency. The volume is defined as the intersection of spheres with different radii; hence, it is not smooth at the intersection zones (see Figure 1.9 (b)). Two k-IOSs are said to be not colliding if any pair of spheres from the first and the second k-IOSs do not intersect. The distance between two k-IOSs is estimated as the maximum distance between every pair of spheres from the first and the second volumes. A C++ implementation of the algorithms yielded results of about 0.5 ms per collision detection, which is relatively high for a bounding volume algorithm with possible false positives.

For collision detection with a great number of similarly-sized objects, it is usual to divide the simulation space in a grid of cells, using the Linked-Cell method. (Ogarko & Luding 2012) present an alternative algorithm, efficient with objects of any size, based on a hierarchical cell grid. The space is divided using grids of different arbitrary cell sizes, contrariwise to other methods in which each level has cells with half the size of the previous level, and to Linked-Cell, which uses only one level. Each particle is mapped to the layer with the smallest cell in which the particle fits completely. The authors propose methods to choose the cell sizes optimally for a given distribution of object sizes, which are then experimentally demonstrated. Presented tests show that the new algorithm is considerably more efficient than Linked-Cell for a high number of objects with heterogeneously sizes and distributions. This algorithm could become even more interesting if objects other than spheres were used.

1.3.3. Collision detection algorithms for smooth convex surfaces

A considerable amount of work is focused on the usage of ellipsoids for collision detection. These shapes are smooth and convex, and are slightly more flexible than spheres, meaning that they can model more varied objects. They are, however, not as flexible as superellipsoids. They can be used as representations of ellipsoidal objects, or as bounding volumes of similarly shaped bodies.

(Chen et al. 2012) present a method to compute the minimum and maximum distance from a point to an affine ellipse or ellipsoid. The two axes of an affine ellipse may be non-perpendicular to each other, which allows for a tighter fit around objects. The proposed method involves finding the root of one quartic equation to obtain the closest point to an ellipse, and a sequence of quartic equations for an ellipsoid. An iterative application of the method to find the minimum distance between two ellipsoids is presented, and achieves an error less than 10^{-6} in 11 iterations, but the method was not benchmarked or compared with other algorithms.

(Pazouki et al. 2012) propose an efficient and parallelizable method based on ellipsoids for simulating large scale multibody systems, instead of the traditional spheres. The algorithm is divided in three levels. On the lowermost level, an unconstrained optimization is solved to find the closest points between two ellipsoids, based on the common normal concept. The intermediate level is a broad collision detection step, and detects if two ellipsoids are in contact or not, using sign of the roots of the characteristic equation of the pair of ellipsoids. The third level is a space partitioning algorithm. The presented benchmarks consist in handling 10^5 to 10^6 collisions, taking 1 to 10 minutes to run on a CPU and about 3 seconds on a massively-parallel GPU. This may be too slow for real-time applications, in which each time step takes a few milliseconds and the GPU may be busy rendering graphics.

A collision detection problem can involve simply deciding whether two objects are in contact or not, without calculating any further information. An algebraic algorithm of this kind has been proposed by (Jia et al. 2011), which characterizes two ellipsoids into three distinct states: completely separated, externally touching, and overlapping. The algorithm involves computing the number of positive roots of a quartic equation, which is related to the state of the two ellipsoids. Instead of explicitly computing the roots, only five formulae are calculated, which take 28 multiplications and 12 additions. The authors then present a CCD algorithm using the mentioned function, and compare it to existing methods. The proposed algorithm is comparatively fast for degree 2 rigid-body and affine movements of the ellipsoids, but is an order of magnitude slower for movements represented by degree 4 transformations. It is, however, faster than the compared method if only the first time instant of collision is computed. Considering the experimental times for each collision detection – 0.1 to 0.5 ms for degree 2 movements – this algorithm may be adequate for real-time applications. It depends, however, on the usage of other algorithms to compute the exact point of collision between the ellipsoids, once they are asserted to be in contact.

An algorithm is proposed by (Gilitschenski & Hanebeck 2012) for detecting overlaps between two ellipsoids, in an exact manner, by calculating the roots of a convex function. The algorithm is formulated for dimensions higher than 3. As the method only detects collision, and not the point of contact, it could be used for elliptical bounding volumes, with a particularization for 3 dimensions, but other methods would have to be employed to find the exact points of minimum distance.

Besides ellipsoids, examples in literature cover the usage of other smooth convex surfaces for the representation of objects for collision detection. Contact between a superquadric and a 3D mesh object can be detected by calculating the inside-outside function of the superquadric, at the relative position of each point of the mesh (Moustakas et al. 2005). To avoid implementing edge-superquadric collision

detection, and to improve contact resolution, additional points are inserted in the edges of each triangle of the mesh. The authors mention speed-ups of 20 to 30, comparing to regular mesh-mesh collision detections. Collision detection can also be performed based on distance fields: these are pre-calculated 3D arrays overlapped onto the considered mesh geometry, and store the distance from each cell to the mesh's surface. A more efficient variation proposed in (Moustakas et al. 2007) computes a bounding superellipsoid around the object's mesh and stores the distances to points in the surface of this superellipsoid instead. This reduces the distance field into a 2D array, saving a considerable amount of memory, but does not completely forgo the usage of polygonal meshes.

(Boon et al. 2013) define an algorithm for collision detection between convex, non-spherical particles, as a Discrete Elements method. The particles are defined by an inside-outside function composed of an intersection of planes, mixed with a spherical component. Contact detection is implemented as a constrained optimization problem. The presented examples show that the algorithm behaves with physical accuracy, in relation to the flow of particles through an orifice of a box. Using a MOSEK implementation, running the algorithm for one pair of colliding objects took between 40 and 400 μs, depending on optimization parameters. These results suggest this algorithm could work in real time with simulations of a few hundred particles, though the number of parameters needed to represent particles is high compared to representing (super)ellipsoids and similar shapes.

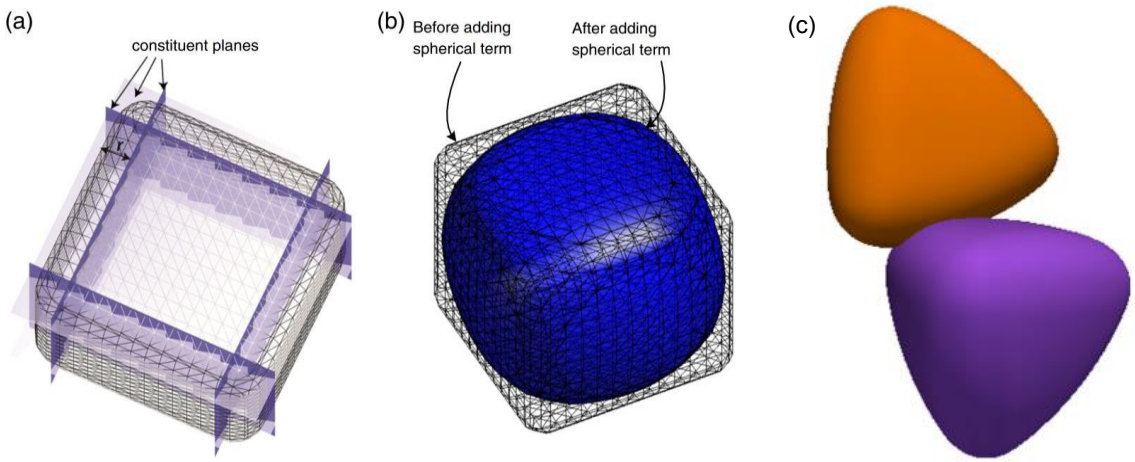


Figure 1.10 – Representation for non-spherical particles proposed in (Boon et al. 2013). The shapes are defined with planes (a) and then smoothed out with an adjustable spherical term (b). A pair of rounded tetrahedral particles is shown in (c). {Adapted from (Boon et al. 2013) with Elsevier permission.}

(Macklin et al. 2014) report a framework capable of simulating rigid and deformable bodies, cloth, liquids and gases, in a unified way, representing the objects with particles and constraints between them. Using particles avoids having specialized collision detection algorithms for many different shapes, and allows implementation on massively-parallel architectures such as GPUs. In this framework, rigid objects may be non-convex, and their particles are simulated as if they were independent and not-connected. The rigid shape is recovered at the end of each time iteration. A GPU-accelerated implementation of the method could handle tens of thousands of particles at 10 ms per frame, a suitable period for real-time applications. Currently, this proposed method only supports fixed particle sizes, which limits the size ratio of the simulated objects; to simulate very large and very small objects at the

same time, a great number of particles must be used. Also, the approximation of objects using particles leads to non-exact simulations.

While cylinders are not completely smooth because of the circular outline on their tops and bottoms, they are piecewise-smooth and convex shapes, and there is interest in using them as collision bounding volumes, as they represent faithfully some objects used in video games and real-time applications. For cylinders whose axes are parallel, closed form collision detection algorithms can be deduced, as the cylinders' projections on planes parallel and orthogonal to the axes become rectangles and circles, respectively.

An algorithm to detect collisions and calculate the minimum distance between two arbitrarily-posed cylinders has been reported in (Srivatsan & Bandyopadhyay 2013). This problem is decomposed into five sub-problems involving different parts of the cylinders, of which four have an analytical (closed-form) solution. The exception, the shortest distance between two circles, involves solving an 8^{th}-order one-variable polynomial numerically. While no experimental benchmarks are provided, a nearly-entirely closed-form algorithm is favorable for real-time applications.

Alternatively, (Chittawadigi & Saha 2013) note that the symmetrical structure of cylinders could be exploited. They describe a solution using Dual Number algebra and represent the position and orientation of cylinders with Denavit-Hartenberg parameters. The used benchmark simulation ran 360 collision detections in 0.05 - 0.12 ms, which yields about 0.3 µs for a collision detection per frame, greatly reinforcing the viability of the cylinder as a collision primitive, though the high number of collision cases (edge-face, edge-edge, etc.), stemming from the fact cylinders are not SCS, complicates the algorithm.

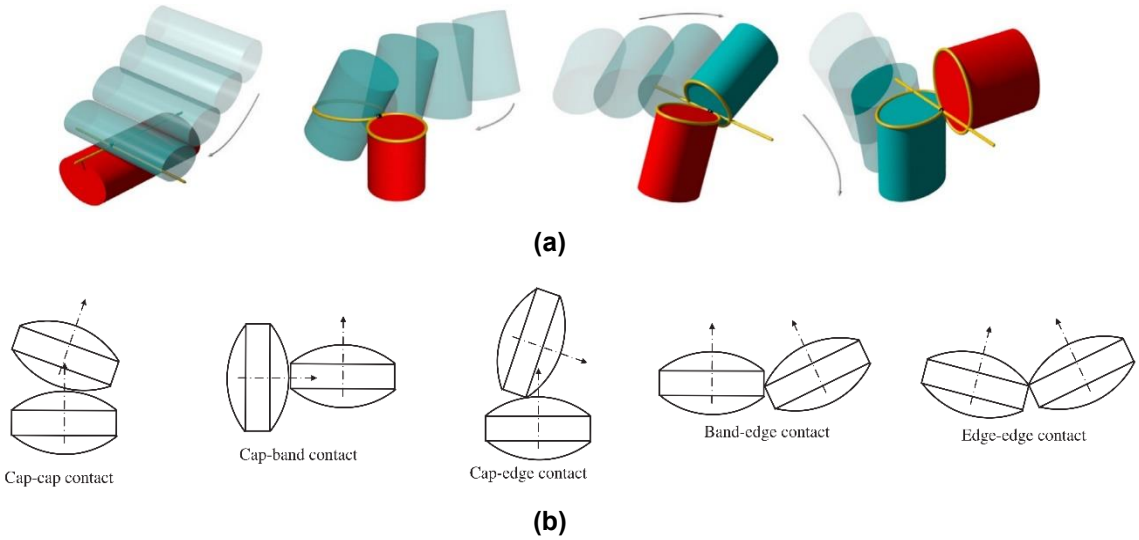


Figure 1.11 – For complex shapes composed of intersections between geometric primitives, such as the ones proposed in (Choi et al. 2014) (a) and (Kodam et al. 2012) (b), collision detection is decomposed into queries for each primitive geometry pair, for example: circle-circle, circle-sphere cylinder-cylinder. {(a) Adapted from (Choi et al. 2014) with Elsevier permission; (b) Adapted from (Kodam et al. 2012) with Elsevier permission.}

(Choi et al. 2014) put forward a framework for continuous collision detection for composite quadric models (CQM). These are objects built through the union and intersection of simple primitives, in this case, quadric and linear surfaces such as cones, cylinders and planes. The framework applies only to specific CQMs whose surface intersections are just straight lines and conic curve segments. Because

of this, the complete surfaces are only piecewise-smooth (see Figure 1.11). As these particular CQMs are commonly used for modeling in computer graphics and robotics, this is a relevant contribution. Benchmarks in a Maple implementation took 0.12 seconds to complete for a pair of capped elliptic cylinders, and 5 seconds for two objects of 45 and 13 boundary elements each. In a precision-oriented implementation, this method could be useful for accurate offline robotic path/grasp planning, but challenging to apply to real-time applications or video games.

(Kodam et al. 2012) put forward a solution to a particular problem: the modeling and simulation of interactions of a great number of bi-convex pharmaceutical tablets, using the Discrete Elements method. The tablets are modeled as the intersection of two spheres for each side and one cylinder for the central band. Ten configurations of collisions are analyzed and implemented, representing the interactions between the different faces and edges of the tablets, and of tablets and a flat surface. Calculations are presented in each case for detecting a collision, the penetration between tablets, and the point and normal direction of collision. The proposed method is an alternative to other tablet models, such as glued clusters of spheres, ellipsoids, superellipsoids, etc. Experiments demonstrate that the proposed method is more accurate and effective in simulating the collision and friction interactions between the tablets. It is also claimed to be more computationally efficient, though no run-time benchmark is provided. The need of subdividing the algorithm into pairs of geometric primitives also adds complexity to the overall process.

Superellipsoids were extensively studied in (Lopes et al. 2010; Lopes et al. 2015a; Lopes et al. 2015b) in the context of collision detection. The authors propose the usage of implicit SCS, namely superellipsoids, to model shapes of kinematic biostructures (Lopes et al. 2015b), instead of using polygonal meshes or freeform surfaces. Associated to this representation, a mathematical framework is presented to compute the minimum distance between two SCS, based on the common normal concept (Lopes et al. 2010). The Householder transformation is investigated for efficient and robust calculation of sets of orthogonal 3D vectors, and is then applied in said framework (Lopes et al. 2013). Finally, a numerical algorithm for the problem of finding the minimum distance between two superellipsoids is presented, as well as an analytical solution for the distance between a superellipsoid and a plane (Lopes et al. 2015a). The developed work is applied in a case study related to simulation and analysis of human gait motion. Although the contact detection algorithm was formulated for any pair of smooth convex implicit surfaces, it has been specifically implemented for the (super)ellipsoid-(super)ellipsoid contact pair and the numerical results show that it has second-order convergence. However, this study lacks a comparative analysis between other contact geometry representations such as parametric surfaces and meshes.

Also in the chemical engineering field, (Lu et al. 2012) propose discretizing superellipsoids into points (nodes) and perform collision detection by evaluating a surface's inside-outside function on the points defining another surface. The points are sorted according to their evaluation value between time-steps, so that they are not exhaustively tested every time-step. Two discretizations approaches are studied which generate equally-spaced and curvature-adaptive points, respectively. This method is compared to a continuous contact detection algorithm based on the minimization of the sum of inside-outside

functions of two surfaces, solved numerically. Performed experiments show that, while these methods are unsurprisingly slower than the usage of simple spheres, the behavior of the simulated system is significantly different. This suggests that simplification of non-spherical particles with spheres is not adequate for precision-critical simulations. Further comparisons with other contact geometry representations in (Lu et al. 2015) note that the discrete method is numerically more stable, but less precise and more memory-demanding, while the continuous inside-outside-based method is easier to implement but does not model the mechanical definition of contact as well as the common normal concept employed in (Wellmann et al. 2008; Lopes et al. 2010).

As exposed in this section, the state-of-the-art for SCS-based collision detection focuses on ellipsoids, and exploits ellipsoid-specific proximity queries to accelerate the process (Pazouki et al. 2012; Jia et al. 2011; Gilitschenski & Hanebeck 2012), but otherwise using methodologies which can be applied to other SCS. When more complex SCS are considered, methods also partially depend on other representations, such as distance fields for meshes (Moustakas et al. 2007) or collections of planes (Boon et al. 2013), increasing memory requirements for the definition of objects. For quasi-SCS objects, such as cylinders and intersection of primitive geometries, each topological component of the objects (edges, straight and rounded faces, etc.) has to be tested against the other components using different procedures, which adds complexity to the algorithms (Kodam et al. 2012; Srivatsan & Bandyopadhyay 2013; Chittawadigi & Saha 2013; Choi et al. 2014). The algorithms presented in this subsection are organized and categorized in Table 1.

In part, this thesis presents itself as an extension to the work of (Lopes et al. 2010; Lopes et al. 2015a; Lopes et al. 2015b), basing itself on the usage of smooth convex surfaces, which are mathematically elegant and can be represented with few parameters. Parametrical representations for the SCS will be studied, in addition to the already-reported implicit approach. Besides superellipsoids, this thesis will propose the usage of tapered superellipsoids, or superovoids, which are more geometrically flexible and can represent a greater variety of objects.

Table 1 – Survey of collision detection algorithms and methods regarding SCS. Symbols appearing in this table are: PQ (proximity query), MD (minimum distance), A (analytical), N (numerical), 2D (two-dimensional) and 3D (three-dimensional).

Reference	Proximity query / Minimum distance	Analytical / Numerical	2D / 3D	Smooth	Convex	Polygonal
(Boon et al. 2013)	MD	N	3D	✓	✓	
(Chen et al. 2012)	MD	N	2D, 3D	✓	✓	
(Chittawadigi & Saha 2013)	MD	A	3D		✓	
(Choi et al. 2014)	PQ	N	3D		✓	
(Gilitschenski & Hanebeck 2012)	PQ	A	2D, 3D	✓	✓	
(Jia et al. 2011)	PQ	A	3D	✓	✓	
(Kodam et al. 2012)	MD	A	3D		✓	
(Lopes et al. 2010)	MD	N	3D	✓	✓	
(Lopes et al. 2015a)	MD	A	3D	✓	✓	
(Lu et al. 2012) continuous	MD	N	3D	✓	✓	
(Lu et al. 2012) discrete	MD	N	3D		✓	
(Macklin et al. 2014)	PQ	N	3D	✓	✓	
(Moustakas et al. 2005)	PQ	A	3D	✓	✓	✓
(Ogarko & Luding 2012)	PQ	A	3D	✓	✓	
(Pazouki et al. 2012)	MD	N	3D	✓	✓	
(Srivatsan & Bandyopadhyay 2013)	MD	A, N	3D		✓	
(Wang 2014)	MD	N	3D			✓
(Zhang & Kim 2012)	PQ	A	3D	✓	✓	

1.3.4. Collision detection in video game engines

Because collision detection is of such paramount importance for many computer games, several game engines have built-in support for collision detection, presenting developers with primitive contact detection shapes and algorithms, and freeing them from implementing complex physics simulations from scratch. This support in game engines is often delegated to dedicated physics engines, such as PhysX (Nvidia Corporation 2014) and Havok (Havok 2011). Many game engines use PhysX, of which Unity3D (Unity Technologies 2014), Unreal Engine (Epic Games, Inc 2014) and Gamebryo (Gamebase USA & Gamebase Co., Ltd. 2012) are examples.

Unity3D provides boxes and a couple of SCS, namely spheres and capsules, as collision detection primitives. In addition, a mesh collider is available, which uses a triangular 3D mesh as a collision shape, with some limitations. Unity3D documentation warns that mesh colliders should use low-resolution meshes, for performance reasons. By default, collisions between two mesh colliders are not registered, unless they are converted to polygonal convex hulls. In this case, they will also be limited to 255 faces.

Unity3D is highly customizable, and maintains an Asset Store, where developers can buy and sell content and tools that add features to the engine. One of such extensions is the uPhysicsPro (SharpDevelopment 2014), which aims to be a full-featured alternative to Unity3D's default PhysX implementation. The package provides extra collision shapes such as k-DOPs, and is multi-thread-ready, aimed for computationally heavy video games such as MMORPGs[1], in which hundreds of players may interact simultaneously on the same spatial zone of the game.

Besides the box, sphere and capsule primitives, Unreal Engine offers the option to generate k-DOPs around a given mesh object for collision detection. The DOPs can be customized to use 6, 10, 18 or 26 faces, for an increased fit around the original mesh. A polygonal convex hull can also be automatically generated, and its level of detail can be adjusted (Epic Games, Inc 2014).

While some engines presented in this section do feature SCS for collision detection, only the simplest SCS such as spheres and capsules are used. Video game developers tend to prioritize the real-time performance and ease of development of games, sacrificing precision and presenting only visually convincing physical simulations and scenarios. In addition, much of the available computational power is invested in graphical rendering. However, it is worth noting that efficient SCS-based algorithms could present a viable alternative to the simple hit-volumes used, and this thesis intends to demonstrate so.

1.3.5. Simulation of cloth materials

Collision detection methodologies are commonly formulated for rigid bodies, which maintain their shape, and thus, can preserve their contact geometry representation throughout the simulation, and this simplifies the algorithms. However, deformable bodies entail different, more dynamic strategies of simulation and collision detection, as their shape does change with time, and phenomena such as self-collision can occur. For this end, a variety of tailor-made methods exist for simulating deformable objects.

[1] Massively Multiplayer Online Role Playing Game

This section focuses on the modeling and collision detection of cloth and textile-like objects, and also on the user interaction of applications using said objects.

To simulate complex physical phenomena, (Bender et al. 2014) report a method capable of representing lateral contraction, bending, anisotropy and elastoplasticity of cloths and deformable objects. The system uses a position-based model with objects represented as particles. In each iteration of the physical simulation, the new locations of particles are estimated based on their velocity, and the system of equations corresponding to the position constraints is numerically solved using only a few iterations. This intentional inaccuracy introduces an elasticity effect on the particles. Finally, the calculated positions are corrected and the new velocities are calculated. A parallel implementation of the algorithm using a 12-core CPU was able to simulate a realistic cloth, taking 7 ms per frame, and 100 falling objects, with 1.5 ms per frame per object. Using a GPU implementation could allow real-time applications to use more objects in one simulation. A consequence of using a position-based method is that the simulation is not physically accurate, but only visually convincing. The intensity of the effects also depends on the time step period used on the simulation. The particle-based unified framework from (Macklin et al. 2014) referred before is also capable of simulating cloth materials in real time, as well as their interaction with bodies on different physical states: rigid bodies, liquids and gases.

The work of (Selle et al. 2008) focuses on efficiently simulating very high-quality, realistic cloths, achieving lifelike phenomena such as intricate folds and wrinkles. Because very large meshes are used (millions of triangles), history-based self-repulsions are applied to avoid computing self-collisions on every time step. Since the focus of the authors is on realism, this proposal is not real-time-friendly, instead being used in pre-rendered movies.

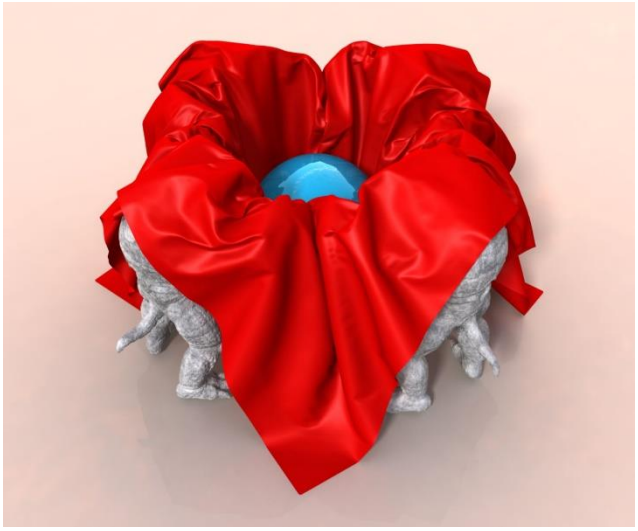


Figure 1.12 – A simulated cloth suspended onto statues is pushed down by a sphere, creating intricate wrinkles on the cloth's surface. {Adapted from (Bender et al. 2014) with Elsevier permission.}

Video games avoid simulating cloth-like materials in general, unless they are absolutely necessary for a particular plot or mechanic. This sometimes leads to awkward inconsistencies, like having game characters sleeping on top of bedsheets, which are uncomfortable-looking rigid objects with textures applied (TVTropes 2015). The Unity3D game engine has a built-in cloth object, which can be made to realistically interact with other collision primitives (in version 4.6), and allows external forces to be

applied via scripting (Unity Technologies 2014). Not much information is publicly available about this feature's inner workings, other than it is implemented with PhysX (Nvidia Corporation 2014) and uses meshes to represent and simulate the cloth. In Unity 5, the cloth primitive was simplified, and no longer collides with arbitrary colliders, only with spheres and capsules (Unity Technologies 2015).

Besides the correct and sufficiently authentic simulation of clothing objects, the way the users interact with the simulated world is noteworthy. Regular mouse and keyboard interfaces may become awkward and hard to use for intuitively manipulating cloths, which are flexible objects. With the advent of multi-touch-sensitive surfaces and accurate game-oriented physics engines, even more intuitive and life-like interactions can be achieved, as reported in (Wilson et al. 2008). In this system, the touch-based inputs of the user, and even shape input, are injected into a physics system as forces, in order to manipulate virtual objects such as boxes, pieces of paper and cloths. A few demonstrations are shown: stacking object with drag-drop motions; stopping in-movement balls by touching them and holding the fingers in place, applying virtual friction, or pressing the side of the hand against the surface, making a barrier; and folding pieces of paper into themselves, or even flipping them over, using dragging and holding gestures with multiple fingers.

On the theme of tangible interfaces, (Leithinger & Ishii 2010) present a table-top system composed of motorized pins which deform a Lycra surface, and which the user can pull and push to intuitively shape a terrain. The pins can also be controlled programmatically to virtualize and animate terrains. The proposed implementation can be scaled to platforms of different sizes with addition of pins. A virtual emulator of this kind of system could allow features hard to implement physically, such as terrains with stacked layers.

Unity's cloth implementation is appealing to this thesis's terrain-cloth modelling case-study, as it is readily available and can be built upon without much effort. Specifically, it allows the developer to push the cloth with forces generated from the contact with some object, for example, a smooth convex surface. The touch-screen-based interactions exposed in (Wilson et al. 2008) are also interesting, and served as reference for the Terrain Cloth application.

1.3.6. 3D body sketching and rigging

As discussed in the previous sections, the 3D representation of objects in physics-based simulations is crucial for their efficiency and accuracy. As such, it becomes important to study the means of creating these 3D representations, in particular, the design and rigging of contact geometries. Different geometric entities, like polygonal meshes and smooth convex surfaces, may better lend themselves to different interactive or automatic techniques of modelling. Many modeling applications use the traditional WIMP (Window, Icon, Menu, Pointer) interaction style, which is sometimes suited for precision work. However, professional modeling software packages such as Maya (Autodesk, Inc. 2015) are complicated to use, and involve a steep learning curve for effective usage. The traditional rigging process involves creating bones and joints overlapped onto an existing rigid 3D model, and then binding the two, so that the model is deformed and follows the bone structure when the latter is animated. The bones, joints and their range of movements are commonly symbolized with widgets (Campos & Passerin 2015).

In alternative, sketching interfaces and applications are being researched. These rely on intuition of the users, and present simpler methods of input, like hand-drawn sketches; the software imposes constraints which are expected to guide the user, and attempts to automate repetitive or predictable tasks. With these, the production of a draft-like 3D model can be prepared without much effort, and the resulting model can be used, for example, for visual communication between teams (Olsen et al. 2009).

A considerably popular model sketching system is Teddy from (Igarashi et al. 1999). This application creates freeform 3D shapes using interactive 2D cursor strokes. The shapes are convincingly filled in real time, and intuitive editions such as extrusions and deformations are possible, by drawing on top of the 3D shapes. Because the input consists only of 2D strokes, the system can be easily used with touch-screens or pen tablets instead of a computer mouse and keyboard.

Because traditional skeleton creation for 3D models is complicated, (Zheng et al. 2010) created a method to generate a skeleton structure for a pre-existing model, based on user sketches through touch-screen input. The skeleton can then be used to digitally animate the model. Users draw freeform lines on top of the mesh, representing the bones of its structure; the result is similar to classic stick-characters used on storyboards. Single strokes may be separated into many segments if the stroke is heavily curved or bent. Bones belonging to the same region (e.g. arm, leg) are clustered together, taking into account the sketched strokes and the topology of the original mesh. The automatic skeleton generation is completed by joining the bones with joints. The algorithm runs into problems when the provided mesh shows limbs folded and merged with the rest of the body, for example, the classic Stanford Bunny model.

To further simplify the task of animating a 3D model, (Yang & Wunsche 2009) suggest completely automatizing the generation of a skeleton for a sketched 3D shape. The algorithm analyses the contours of the shape to produce a hierarchical set of bones and joints, which are then rigged to the 3D shape to allow animation. Because the algorithm is completely automatic and has no knowledge of the nature of the shape it is applied to, or its physical context, some joints may be mistakenly configured. The process attempts to create merely plausible skeletons. Experimental tests resulted in the generation of a skeleton for a humanoid shape in half a second, while the subsequent animation could run in real time. Examples of designed shapes can be seen in Figure 1.13.

The Virtual Human Sketcher (Mao et al. 2009) aims to be a comprehensive tool which allows the user to model and animate 3D human figures with an intrinsic skeleton using intuitive sketches (see Figure 1.13). The sketch starts with a stickman figure, which is interpreted as a 3D pose. This stickman is then fleshed-out with a freehand silhouette, which is interpreted using body fat distribution models to extract a natural shape. This shape can be enhanced with local outlines drawn on top of it, or on the 2D silhouette. Lastly, the 3D model mesh can be animated using the skeleton structure from the first phase (stickman). Tests on the learning and modeling time were performed to compare the proposed application with other modeling software packages, such as Teddy (Igarashi et al. 1999), 3DS MAX and Poser, which were all more time-consuming than Virtual Human Sketcher.

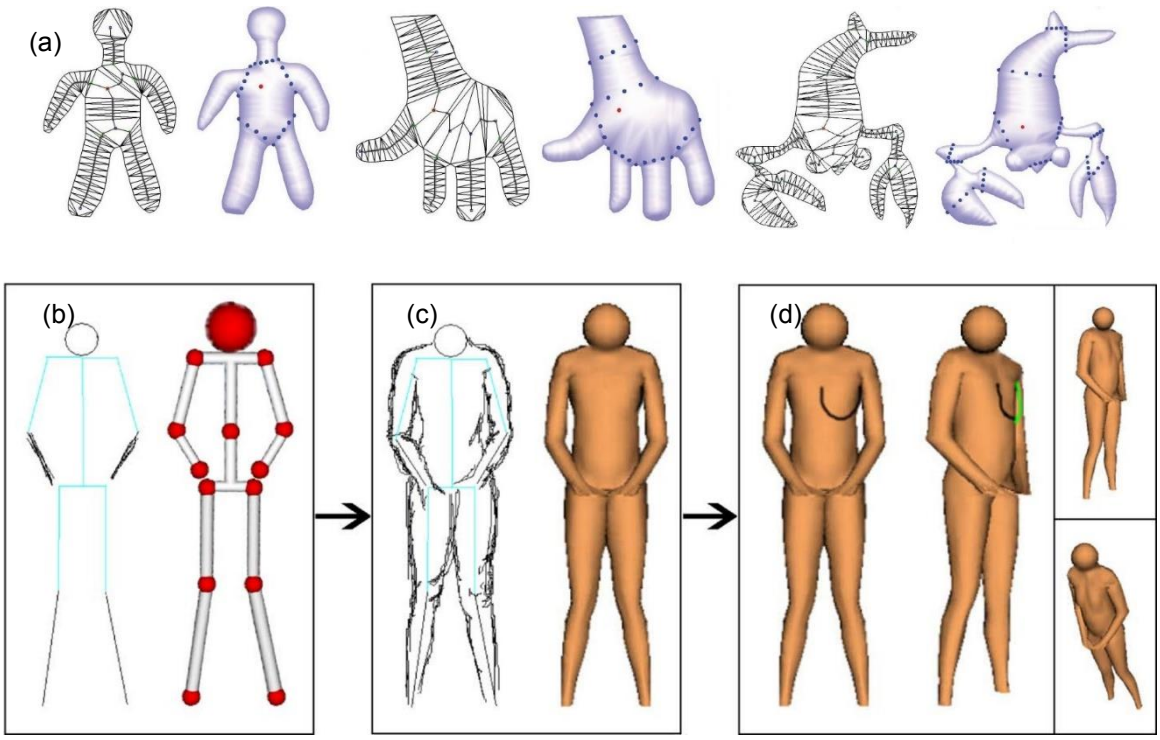


Figure 1.13 – Different methodologies for modeling rigged, animation-ready bodies. In (Yang & Wunsche 2009)'s general approach (a), the user's 2D sketches are automatically converted to 3D models and joints are added for animation. On the other hand, the sketch-based method proposed in (Mao et al. 2009) is tailored for human bodies (b, c), and allows the resulting model to be enhanced with 2D inputs (d). {(a) Adapted from (Yang & Wunsche 2009) with IEEE permission; (b, c, d) Adapted from (Mao et al. 2009) with Elsevier permission.}

While 3D mesh models may be quite complex, it is desirable that the collision detection representations are simple, to save computational time. Furthermore, simplification can aid in eliminating noise from the meshes, in cases they have been obtained from noisy sources such as 3D scanners or quick sketches. Procedures to decompose a 2D shape into ellipses have been presented in (Kimoto & Yasuda 1997): one method approximates shapes by extracting the shape's skeleton and assigning an ellipse to each bone, aligned with its longest axis; another aims to represent the shape exactly using hierarchies of unions of ellipses and morphological operations. More recently, an abstraction technique is proposed by (Yumer & Kara 2012). The method uses a collection of similar 3D meshes, and extracts meaningful abstractions for that collection as a whole; hence the used term co-abstraction. Each mesh is simplified, while maintaining defining characteristics in relation to the other members of the collection. Many levels of progressively more detailed simplifications are generated. The meshes are approximated with volumetric primitives, such as prisms, cylinders and truncated cones, which are then beautified into polynomial surfaces. Primitives which are close enough are merged into one contiguous body.

For fast collision detection for articulated bodies, (Steinbach et al. 2006) developed an algorithm to automatically generate a sphere tree hierarchy for a 3D polygonal mesh. The spheres' radii and positions are placed as tightly around the mesh as possible, and in a way that allows for detection of self-collision and collision with obstacles (see Figure 1.14). Using this hierarchy, it is possible to calculate both collision detection and approximated minimum distances. In this work, the algorithm is applied to a humanoid articulated robot. The sphere tree is calculated not only for a particular configuration of the

robot's arms and links (its pose) but also the range of motion of those links – that is, the swept volume of the robot's articulated parts, if they moved along all their possible positions. This can be done only once, offline. Collision detection using this range of motion hull will not be as accurate, as it does not represent a particular configuration of the moving parts, but all of them. Nevertheless, it allows to quickly detect possible impacts with obstacles during online operation.

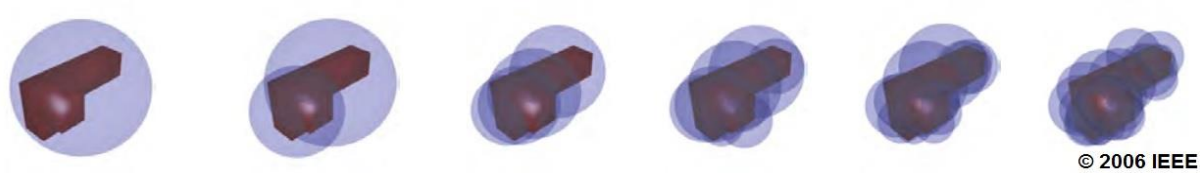


Figure 1.14 – Different levels of the bounding sphere hierarchy proposed in (Steinbach et al. 2006), for an L-shaped object. {Adapted from (Steinbach et al. 2006) with IEEE permission.}

Posing and animation of 3D articulated characters requires a good understanding of their spatial positioning, which is not always easy to convey and manipulate with 2D interfaces. Mouse and keyboard devices offer a successful, but limited means of interaction with the 3D models, and professional software packages that resort to these devices are often complicated to use and require substantial training. To ease this task, a tangible input device has been developed (Jacobson et al. 2014), which consists of interchangeable hot-pluggable joints and extensions which can be assembled in the shape of a skeleton suited for the particular model to be animated or posed. Sensors in these articulated parts report the orientations and poses of each component of the skeleton in real time, such that the user can manipulate the skeleton interactively and see the 3D model being animated on a computer. Because the device is something the users hold in their hands, full visibility of the device is not required; this contrasts with traditional 2D interfaces in which markers and handles are rendered on top of the character to be animated, contributing to a cluttered interface. The system works with either a 3D mesh, or a complete rig (mesh and skeleton): if only the mesh is available, the system allows the user to semi-automatically create a skeleton for it, by mimicking the rest-pose of the model using the physical pieces, and then matching its orientation to the mesh through the computer. If a skeleton rig already exists for the model, it can be mapped into the tangible device in a similar way.

There has also been research aimed at improving the posing of 3D characters without extra hardware: a technique inspired by the line of action, a simple yet intuitive and expressive tool used by illustrators and cartoonists to sketch out a general pose of a character (Guay et al. 2013). The authors' system takes one single stroke (the line of action) from the user, drawn on top of a 3D model, associates it to a chain of bones of the model based on the position and tangent direction of the line, and then adjusts the model so that the selected chain of bones matches the line of action, from the considered point of view. It is reported this novel method of posturing leads to very quick production of sequences of poses: non-experts in animation were faster using the proposed line of action method than experts using Autodesk Maya producing the same animations.

1.3.7. Collision detection on grasp planning

For robots to adequately interact with their environment, their actions should be planned with a sufficient degree of accuracy. The simulations for this planning involve, among other things, queries for the distance between objects and parts of the robot, the penetration depth between intersecting objects and robot parts, and the collision time instant between moving objects. As such, contact geometries employed to represent the robotic parts must be suitably chosen. Grasp planning techniques very often rely on the computation of grasp quality criteria, such as force closure (Murray et al. 1994), that require the position of the contact points and corresponding surface normals. Hence, the accurate computation of these quantities is highly important for the success of grasp planning algorithms. By representing the robotic contact models using SCS, this thesis aims to provide efficient and accurate means to compute contact points and surface normals, which can then be used for grasp planning.

Many libraries exist to compute a series of collision-related queries, of which RAPID, SOLID and PQP are examples. These are specialized in particular queries, and are designed to use specific tools such as AABBs or OBBs, being hard to extend. In alternative, the Flexible Collision Library, or FCL (Pan et al. 2012), has been developed to perform a vast array of proximity queries, using different models such as rigid/deformable bodies and articulated models. It supports alternative representations like convex and non-convex polytopes (meshes) and point clouds, and also implements Bounding Volume Hierarchy (BVH) algorithms. FCL was developed with flexibility in mind, allowing for addition of new collision detection and minimum distance algorithms, and different representations of objects. Out of the box, FCL supports spheres, boxes, cones, cylinders, capsules, planes and meshes. FCL is used in MoveIt! (Sucan & Chitta 2015), a robotic motion planning, navigation and control library.

The Kinematic Continuous Collision Detection Library (Taeubig & Frese 2012) has been developed with focus on real-time continuous collision detection for greater accuracy and safety, in the context of very fast robotic movement. Robot parts are represented with Sphere Swept Convex Hulls (SSCH), avoiding the usage of mesh-based volumes. These SSCH are composed of a convex hull of a set of points, plus a region of a certain radius around the points: this allows representing both sharp-edged and rounded robotic parts with few points. The library is able to detect collision, and compute minimum distances between a pair or all pairs of robotic parts.

For higher dexterity and human familiarity, there is interest in creating humanoid robotic hands with several fingers. This complicates the grasping process, since it adds more parts to be actuated, more collision/touching points to manage in the grasping process, and therefore requires more computational power and efficient solutions (Xue et al. 2009; Shi & Koonjul 2014).

In (Xue et al. 2009)'s methodology, the grasping process is divided in two phases: moving the hand near the object to be picked up, and closing the fingers around it. For the second phase, the quality of the planned grasping is evaluated taking into account the contact points between the object and the hand fingers, and the contact forces at each of these points. The authors use oriented bounding boxes to represent the robotic apparatus, and sweep these boxes during finger movement for broad-phase continuous collision detection. Basic shapes and superquadrics are also used to represent the object to

be grasped, but are only employed to generate possible grasping directions, not to simulate the grasping attempts with narrow-phase collision detection. This is performed with polygonal meshes instead (Xue et al. 2007), which do not provide precise methods for computing contact normals.

(Shi & Koonjul 2014) focus on the positioning of the palm of the robotic hand near the object, so that the grasping can occur afterwards. For this purpose, the volume where all the fingers can be moved is calculated, and roughly represented with cylinders. This “grasp zone” volume represents the “preferred” position of the hand, and is used for the planning: the objective is to detect intersections between this volume and the object to be picked up, which implies the hand is close enough to the object to begin the second phase of the process, the grasping itself.

For a robot to properly grasp and hold an object, the grasping pose should provide a secure grip and exploit the shape of the object. For example, concave or constricted sections of objects such as handles can be used as favored grasping sectors. With this in mind, (Tsuji et al. 2014) propose an automatic method for approximation of objects to be grasped using quadric surfaces: ellipsoids, elliptic cylinders and one-sheet hyperboloids. They also report exploiting concave crevices resulting between two adjacent quadric surfaces, to explore more stable grasping poses. The presented benchmark indicates the optimization employed to fit the robot fingers around the quadric surfaces is not very efficient, taking approximately 5 seconds.

While the reviewed literature suggests that there is interest in using SCS for robotic action planning, the reported implementations do not exploit the analytical representations of these surfaces, opting for the usage of polygonal meshes (Xue et al. 2007) or computationally expensive optimization procedures (Tsuji et al. 2014). There exist, however, tools that can be easily extended to introduce analytically-represented SCS to the robotic scene (Pan et al. 2012). This thesis thus proposes to implement the studied superellipsoid and superovoid algorithms in FCL, and compare them against mesh-based algorithms, which are commonly employed in robotics.

1.4. Scopes and objectives

The main objective of this thesis is to explore geometric properties of superellipsoids and superovoids for developing real-time and accuracy-controlled collision detection algorithms. If analytical surfaces prove to be more efficient, then SCS can be considered an appealing alternative to polygonal meshes.

1.4.1. Contact Detection

As a core contribution, this thesis will present studied and developed efficient collision detection modules with minimum distance methods for analytical (non-polygonal) smooth convex contact geometries in non-conformal configuration: This goal is detailed in the following bullet points:

- Produce general-purpose implicit and parametric surface collision detection modules that consider a modified Newton-Raphson method with analytical Jacobian and, in case of failure, vector-valued multivariate bisection or *regula falsi* methods to provide a proper initial approximation for the minimum distance problem;

- Develop specific collision detection module for superellipsoid-plane;
- Benchmark polygonal and non-polygonal (implicit and parametric) approaches. The comparison criteria will rely on computational efficiency, accuracy and robustness. Also, the pros and cons of each method will be listed in order for the developer to make an informed decision on which method is most suitable for the application under consideration.

1.4.2. Contact Geometry of Articulated Objects

Modeling articulated systems with contact geometries can be a tedious task. Therefore, and as a secondary goal, this thesis also addresses a modeling problematic:

- Present an interactive sketch-based framework for building articulated objects in an expeditious fashion;
- Provide a swift way of representing articulated bodies' contact geometries as smooth convex objects (spheres, ellipsoids and superellipsoids), which can represent shapes using a lesser amount of elements when compared to triangular meshes.

1.4.3. Terrain Cloth

- Improve the interactive experience of a cloth simulator by implementing a superellipsoid-vertex contact module;
- Explore the effectiveness of the superellipsoid-vertex contact detection in cloth simulation.

1.4.4. Robot grasping

- Accelerate the process of grasp movement on robots with humanoid hands;
- Propose the use of superellipsoids and superovoids for the fitting and modeling of robotic hands used for grasp movements;
- Submit a conference paper reporting the results and conclusions of this case-study to the IEEE International Conference on Robotics and Automation (ICRA) 2016.

1.5. Contributions

The contributions of this thesis are of various natures: scientific content has been produced and reported in articles, and technical content was materialized into two prototypes of interactive applications, as well as additions to an existing collision detection library and the production of a simple video game. These contributions are categorized in the following bullet points:

Contact detection:

- Propose a more efficient contact algorithm for smooth convex surfaces, in particular, for superellipsoids;

- Benchmark the projected methods against existing alternatives: implicit vs parametric representations; polygonal vs non-polygonal;
- Produce a simple video game inspired in Pong, featuring an analytical collision detection algorithm between a superellipse and a line segment.

Contact Geometry of Articulated Objects:

- “Multibody Sketcher”: interactive CAD project to build articulated systems (armatures or kinematic structures) capable of rigidly attaching 2D and 3D geometric primitives, i.e., analytical surface contact elements (e.g., smooth convex surfaces): skeleton modeling via sketching;
- 3D modeling module to design armatures/skeletons/rigs or, in other words, multibody systems with contact surfaces.

Terrain Cloth Simulator:

- Based on the “blanket fort” metaphor (CustomTerrain 2012; Fire and Fury Games 2014), this work aims to develop an interactive application that makes use of a game engine simulator to collide a cloth with superellipsoids.

Robot Grasping:

- Implement contact modules (superellipsoid-superellipsoid, superellipsoid-superovoid and superovoid-superovoid) for computational pre-planning of robotic grasping, in FCL (Pan et al. 2012), as free and open-source code;
- Model the iCub robot’s hands (Metta et al. 2010) with superellipsoid and superovoids, and benchmark the developed minimum distance algorithms in this application.

List of publications:

- Lopes, DS, Neptune, RR, **Gonçalves, AA**, Ambrósio, JA and Silva, MT (2015), 'Shape Analysis of the Femoral Head: A Comparative Study Between Spherical, (Super)Ellipsoidal, and (Super)Ovoidal Shapes', Journal of Biomechanical Engineering, vol 137, no. 11, pp. 114504-114504-8;
- **Gonçalves, AA**, Lopes, DS, Jorge, JA and Bernardino, A (2016), 'Accurate Contact Geometries for the iCub Hand', IEEE International Conference on Robotics and Automation 2016 (submitted).

2. Fundamentals of computational geometry

This section explains fundamental geometric concepts which will be used throughout the rest of the thesis. Different representations of 3D objects are exposed in section 2.1. The properties that define smooth convex surfaces are presented in section 2.2. Section 2.3 defines the specific surfaces for which collision detection algorithms were studied. Section 2.4 explains how to transform 3D points between local and global coordinate systems.

2.1. Geometric representations

The different representations of geometric shapes studied in this thesis will be presented in this section. Each representation has both advantages and disadvantages which make them suitable for certain applications, and these benefits are also summarily described.

The polygonal mesh representation, already mentioned in previous sections, is a collection of polygons, stored as the list of vertices which compose said polygons. Typically, meshes are composed of triangles, because 3 vertices forming a triangle are guaranteed to be coplanar, and any polygonal shape can be decomposed into triangles. Triangular meshes are the gold standard for 3D modeling and visualization, as they can approximate almost any kind of object, including rigid and deformable bodies. However, meshes can only faithfully represent faceted objects; smooth surfaces must be approximated by triangulation, and more accuracy is only achieved by using a greater number of vertices and triangles, leading to a higher memory usage, and slower collision detection algorithms. Furthermore, mesh edges are only C^0-continuous, which means that the surface normal direction is not defined in these regions.

Geometric shapes may be represented implicitly, through the means of an implicit function, written as

$$F(x, y, z) = 0 \tag{1}$$

where x, y and z are the Cartesian coordinates of a 3D point located on the surface. The implicit function also defines two distinct regions of space besides the surface itself, conventioned here as follows: points for which the result of the function is lower than 0 are inside the surface, while those that yield a result greater than 0 are outside the surface. Because of this property, F is also called an inside-outside function. The surface normal direction can be readily computed by taking the gradient of F, which is a useful property for collision detection. On the other hand, implicit representations do not provide an explicit way to obtain a set of points on the surface, which is convenient for surface triangulation and visualization.

Parametric representation is based on a mapping between the surface in 3D space and a 2D parameter domain, written in the form

$$\mathbf{s}(u,v)=\begin{bmatrix} x=x(u,v) \\ y=y(u,v) \\ z=z(u,v) \end{bmatrix}, \quad \begin{matrix} u_1<u<u_2 \\ v_1<v<v_2 \end{matrix} \tag{2}$$

where the parameters u and v generate the whole set of points on the surface, when swept over the domain $[u_1, v_1] \times [u_2, v_2] \in \mathbb{R}^2$. Because of this, generating polygonal meshes from parametric surfaces for visualization is straightforward, and this representation is frequently used in computational geometry and design applications (Farin 2002; Duncan 2005). By changing the parametric coordinates, the same shape can be defined by multiple parametric representations.

2.2. Smoothness and convexity

Smoothness and convexity, visually depicted in Figure 2.1, are two important geometrical properties for the surfaces studied in this thesis. A mathematical real function of a number of variables is said to be smooth if it can be differentiated up to a desired order, over its entire domain. Thus, smooth functions do not present pointy edges or spikes, and because they are differentiable at any point, the surface tangents and normals are always defined and can be computed.

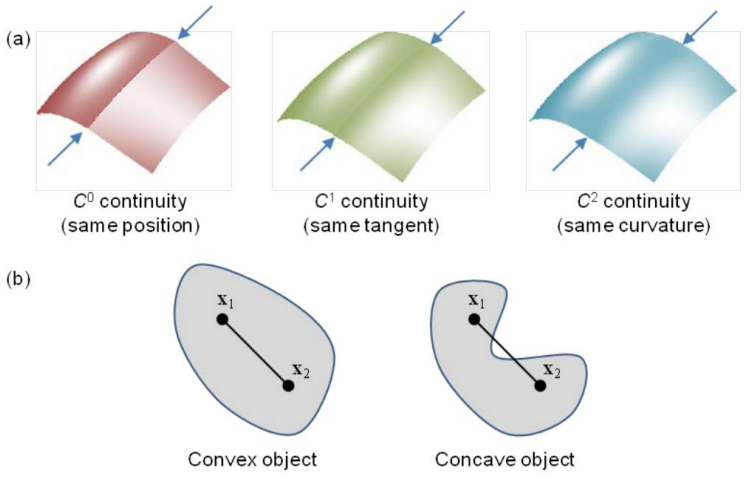


Figure 2.1 – (a) Smoothness or continuity on surface patches. (b) Convexity of objects. {Copied from (Lopes 2013) with author permission.}

A surface is said to be convex if the straight line connecting any pair of points of the surface also lies inside the surface. That is, for every pair of points $\mathbf{x}_1, \mathbf{x}_2$ in the surface $\boldsymbol{\Omega}$,

$$\mathbf{x}_3=(1-\alpha)\mathbf{x}_1+\alpha\mathbf{x}_2 \in \boldsymbol{\Omega},\ 0<\alpha<1 \tag{3}$$

where $\mathbf{x}_3$, a linear combination of the two points on the surface, travels along a straight line as α increases from 0 to 1. Planes, cubes, spheres and ellipsoids are all examples of convex surfaces.

2.3. Contact detection geometries

2.3.1. Plane

The plane is the simplest smooth convex surface, represented with a linear polynomial implicit function of the form

$$F_P(x, y, z) = ax + by + cz - d \tag{4}$$

where a, b and c are the x, y and z components of a vector normal to the plane's surface, and d is the inner product of the normal and a known point belonging to the surface of the plane.

2.3.2. Superellipsoid

The superellipsoid is a generalization of the ellipsoid proposed in (Barr 1981). Figure 2.2 depicts examples of this shape. It is defined by its inside-outside function

$$F_{SE}(x, y, z) = \left(\left| \frac{x}{a_1} \right|^{\frac{2}{\epsilon_1}} + \left| \frac{y}{a_2} \right|^{\frac{2}{\epsilon_1}} \right)^{\frac{\epsilon_1}{\epsilon_2}} + \left| \frac{z}{a_3} \right|^{\frac{2}{\epsilon_2}} \tag{5}$$

where $a_1, a_2, a_3 > 0$ are the lengths of the shape in the x, y and z axes, and $\epsilon_1, \epsilon_2 \in]0,2[$ are the squareness factors for the $x0y$ plane and z axis, respectively. The range between 0 and 2 of the squareness exponents, $\epsilon_k, k \in \{1,2\}$, ensures that the resulting shapes are convex. Thus, the implicit function in the canonical form for the superellipsoid is

$$F_{SE}(x, y, z) - 1 = 0 \tag{6}$$

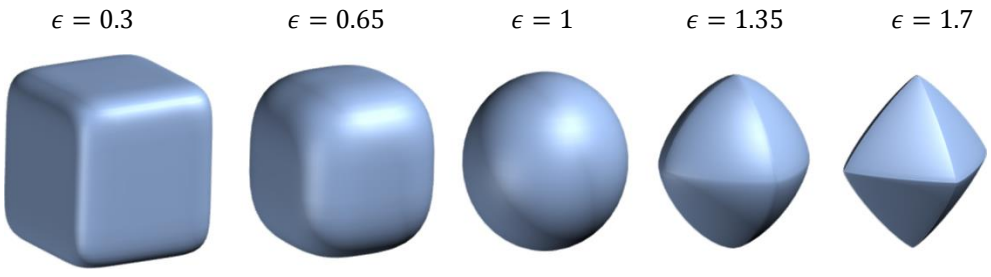


Figure 2.2 – Superellipsoids for varying exponent values ($\epsilon_1 = \epsilon_2 = \epsilon$).

Superellipsoids can be described using an angle-center parametrization (Wellmann et al. 2008):

$$\mathbf{s}_{SE}(\varphi_1,\varphi_2)=\begin{bmatrix}x\\y\\z\end{bmatrix}=\begin{bmatrix}a_1\,\mathrm{sign}(\cos\varphi_1)\left|\cos\varphi_1\right|^{\epsilon_1}\,\mathrm{sign}(\cos\varphi_2)\left|\cos\varphi_2\right|^{\epsilon_2}\\ a_2\,\mathrm{sign}(\sin\varphi_1)\left|\sin\varphi_1\right|^{\epsilon_1}\,\mathrm{sign}(\cos\varphi_2)\left|\cos\varphi_2\right|^{\epsilon_2}\\ a_3\,\mathrm{sign}(\sin\varphi_2)\left|\sin\varphi_2\right|^{\epsilon_2}\end{bmatrix},\quad \begin{matrix}-\pi<\varphi_1<\pi\\ -\frac{\pi}{2}<\varphi_2<\frac{\pi}{2}\end{matrix} \tag{7}$$

and the expressions for the surface normal can be deduced via differential calculus, leading to

$$\mathbf{n}_{SE}(\varphi_1,\varphi_2)=\begin{bmatrix}x\\y\\z\end{bmatrix}=\begin{bmatrix}\frac{1}{a_1}\mathrm{sign}(\cos\varphi_1)\left|\cos\varphi_1\right|^{2-\epsilon_1}\,\mathrm{sign}(\cos\varphi_2)\left|\cos\varphi_2\right|^{2-\epsilon_2}\\ \frac{1}{a_2}\mathrm{sign}(\sin\varphi_1)\left|\sin\varphi_1\right|^{2-\epsilon_1}\,\mathrm{sign}(\cos\varphi_2)\left|\cos\varphi_2\right|^{2-\epsilon_2}\\ \frac{1}{a_3}\mathrm{sign}(\sin\varphi_2)\left|\sin\varphi_2\right|^{2-\epsilon_2}\end{bmatrix},\quad \begin{matrix}-\pi<\varphi_1<\pi\\ -\frac{\pi}{2}<\varphi_2<\frac{\pi}{2}\end{matrix} \tag{8}$$

where φ_1 and φ_2 are the azimuth and zenith parameters, respectively. These expressions can be manipulated in order to obtain the parameters of the surface corresponding to a given 3D normal vector

$$\begin{aligned}\varphi_1 &= \mathrm{atan}\,2\left(s_1\left|a_1n_1\right|^{\delta_1},s_2\left|a_2n_2\right|^{\delta_1}\right)\\ \varphi_2 &= \begin{cases}\mathrm{atan}\,2\left(\left|a_1n_1\right|^{\delta_2},s_3\left|\left(a_3n_3\right)\left|\cos\varphi_1\right|^{2-\epsilon_1}\right|^{\delta_2}\right), & \text{if } \left|a_1n_1\right|>\left|a_2n_2\right|\\ \mathrm{atan}\,2\left(\left|a_2n_2\right|^{\delta_2},s_3\left|\left(a_3n_3\right)\left|\sin\varphi_1\right|^{2-\epsilon_1}\right|^{\delta_2}\right), & \text{else}\end{cases}\\ \delta_i &= \frac{1}{2-\epsilon_i},\qquad s_i=\mathrm{sign}(n_i)\end{aligned} \tag{9}$$

in which $\mathbf{n}$ is the given normal, and $\mathrm{atan2}(x,y)$ is the arc tangent of y/x, adjusted according to which quadrant the point (x,y) is in.

2.3.3. Superovoid

The superovoid is a smooth convex surface based on the Barr tapered superellipsoid (Barr 1981). Examples of superovoids can be seen in Figure 2.3. The shape is defined by its inside-outside function

$$F_{SO}\left(x,y,z\right)=\left(\left|\frac{x}{\left(T_x\frac{z}{a_3}+1\right)a_1}\right|^{\frac{2}{\epsilon_1}}+\left|\frac{y}{\left(T_y\frac{z}{a_3}+1\right)a_2}\right|^{\frac{2}{\epsilon_1}}\right)^{\frac{\epsilon_1}{\epsilon_2}}+\left|\frac{z}{a_3}\right|^{\frac{2}{\epsilon_2}} \tag{10}$$

where $a_1, a_2, a_3, \epsilon_1, \epsilon_2$ have the same meaning as for superellipsoids, and $T_x, T_y \in [-0.5, 0.5]$ are the tapering (egg-shape) factors in x and y, respectively. Similarly to the superellipsoid, the implicit function for the superovoid in the canonical form is

$$F_{SO}(x, y, z) - 1 = 0 \tag{11}$$

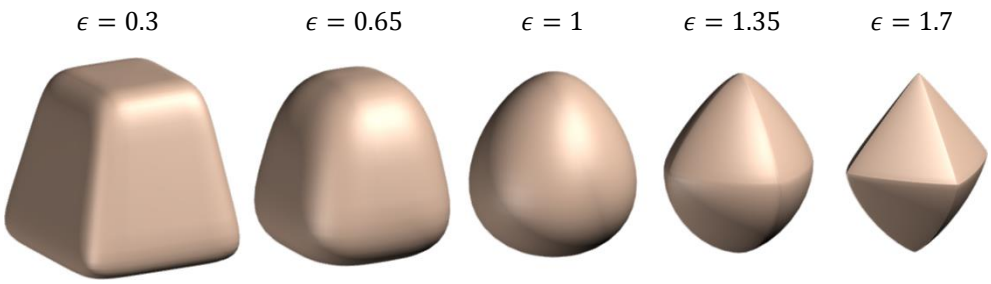


Figure 2.3 – Superovoids for varying exponent values ($\epsilon_1 = \epsilon_2 = \epsilon$) and $T_x = T_y = -0.25$.

When $T_x = T_y = 0$, the superovoid is symmetrical in relation to its equatorial plane (it does not have an ovoidal shape anymore), becoming a superellipsoid (Barr 1981).

Superovoids can also be described using an angle-center parametrization

$$\mathbf{s}_{SO}(\varphi_1, \varphi_2) = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_1 \left(T_x \dfrac{z}{a_3} + 1 \right) \text{sign}(\cos\varphi_1) |\cos\varphi_1|^{\epsilon_1} \text{sign}(\cos\varphi_2) |\cos\varphi_2|^{\epsilon_2} \\ a_2 \left(T_y \dfrac{z}{a_3} + 1 \right) \text{sign}(\sin\varphi_1) |\sin\varphi_1|^{\epsilon_1} \text{sign}(\cos\varphi_2) |\cos\varphi_2|^{\epsilon_2} \\ a_3 \, \text{sign}(\sin\varphi_2) |\sin\varphi_2|^{\epsilon_2} \end{bmatrix}, \tag{12}$$

$$-\pi < \varphi_1 < \pi$$

$$-\frac{\pi}{2} < \varphi_2 < \frac{\pi}{2}$$

Note that in the computational implementation, the z coordinate in (12) must be calculated first as the x and y coordinates depend on its value.

2.4. Local and global coordinate transformations

The representations described in section 2.3 are formulated in relation to the surfaces' local coordinate system. Transforming 3D points from one coordinate system to another is an essential operation for many geometric algorithms, including collision detection. A transformation is applied to a point in global coordinates $\mathbf{x}_g$ to obtain the same point in local coordinates $\mathbf{x}_l$. This transformation is composed of a translation vector $\mathbf{t}$, a rotation matrix $\mathbf{R}$, and a scale matrix $\mathbf{D}$, as

$$\mathbf{x}_l = \begin{bmatrix} x_l & y_l & z_l & 1 \end{bmatrix}^T = \left(\begin{bmatrix} \mathbf{RD} & \mathbf{t} \\ 0_{1\times 3} & 1 \end{bmatrix} \right)^{-1} \mathbf{x}_g \tag{13}$$

in which $\mathbf{x}_l$ and $\mathbf{x}_g$ are written in homogeneous coordinates. The rotation matrix is obtained, for example, through "yaw-pitch-roll" angles using the product of $\mathbf{R}_x(\varphi)\mathbf{R}_y(\theta)\mathbf{R}_z(\psi)$, where φ, θ, ψ are the angles of rotation around each coordinate axis (Nikravesh 1988). Matrix $\mathbf{D}$ is a diagonal matrix composed of the scaling factors along each local reference direction, x_l, y_l and z_l. Equation (18) can be rewritten to convert points from local to global coordinates as

$$\mathbf{x}_g = \begin{bmatrix} x_g & y_g & z_g & 1 \end{bmatrix}^T = \left(\begin{bmatrix} \mathbf{RD} & \mathbf{t} \\ 0_{1\times 3} & 1 \end{bmatrix} \right) \mathbf{x}_l \tag{14}$$

The formulas presented in this thesis assume a right-handed axes system with the z axis pointing up. Modifications and additional transformations must be applied to implement these formulas in other coordinate systems. For example, Unity3D employs a left-handed referential with y pointing up.

3. Contact detection between smooth convex surfaces

3.1. Common normal concept

A central problem in the theme of this thesis can be formulated as follows: given two smooth convex surfaces in 3D space, what is the minimum distance between them? Specifically, what is the pair of points that defines said minimum distance? Using the distance between these points, it is possible to assert whether the surfaces are in collision or not. The common-normal concept (Johnson 1985) states an important condition for the solution of this problem. Let P and Q be two points on two C^1 continuous surfaces (i) and (j), respectively, which are in contact on those points, or have a minimum distance expressed by those points. The common-normal concept states that the normal direction to each surface on the respective points, $\mathbf{n}_{OP}$ and $\mathbf{n}_{OQ}$, must be the same, and equal to the direction of the distance vector $\mathbf{d}_{PQ}$. As a consequence, the tangent and binormal vector directions on both surface points define two planes parallel to each other. A 2D case is illustrated in Figure 3.1.

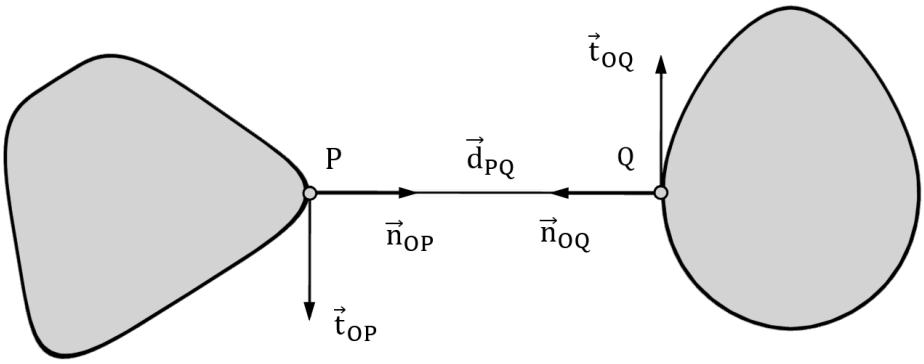


Figure 3.1 – Orthogonal and collinear vector relationships that define the common normal concept among the surface normals, the distance vector, and the tangent vectors. Surfaces are represented in 2D, thus, the binormal vectors are not shown. By convention, surface normal vectors point outwards.

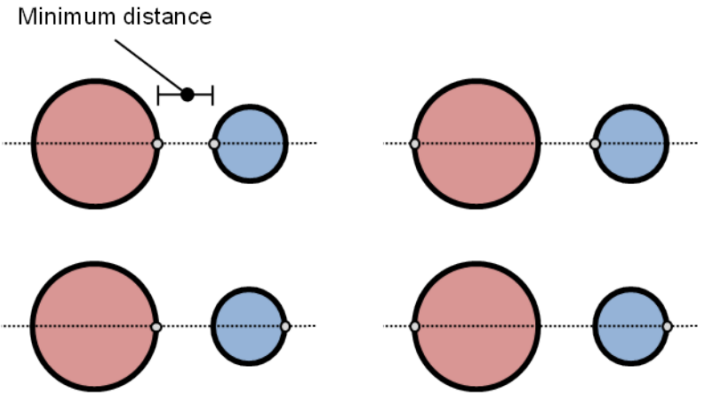


Figure 3.2 – A pair of spheres illustrating that the common normal concept is only a necessary condition for determining the minimum distance between smooth convex surfaces. {Copied from (Lopes 2013) with author's permission.}

The common normal concept outlines a necessary condition for finding the minimum distance between two surfaces. However, it is not a sufficient condition: the fact that two points in the surfaces share the same normal does not imply that they define the minimum distance between the surfaces. The example with spheres in Figure 3.2 illustrates this: while there are 4 potential pairs of points with shared normal direction, only one pair corresponds to the minimum distance.

3.2. Superellipsoid-plane

A Unity3D module for the calculation of the minimum distance between an infinite plane and a superellipsoid was implemented, following the algorithm described in (Lopes et al. 2015a). Its operation can be verified in Figure 3.3. An analogous script for the 2D case was developed for the minimum distance between a superellipse and a finite line segment, and is described in Appendix B.

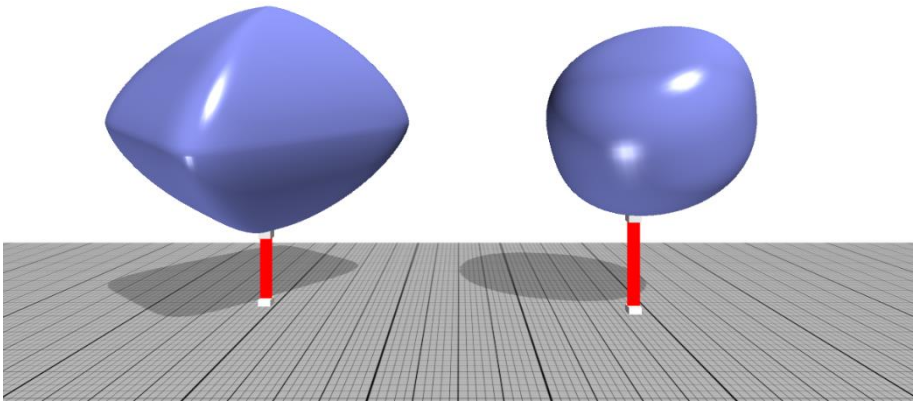


Figure 3.3 – Minimum distance between a superellipsoid and a plane (2 examples), computed analytically.

This algorithm is based on the common-normal concept, and its efficiency hinges on two properties of the considered shapes. Firstly, a 3D plane has one unique normal throughout its surface. Secondly, given a superellipsoid, it is possible to analytically compute the points in the superellipsoidal surface that present a normal direction parallel to a given vector. Considering these facts, the direction of the minimum distance between a superellipsoid and a point is trivial to obtain (it must be the plane's normal direction) and the set of minimum distance points can be computed analytically.

Given a superellipsoid (*j*) and a plane (*i*) whose unit normal in global coordinates is $\mathbf{n}_{OP}$, the plane's normal can be expressed in the superellipsoid's local coordinate system as

$$\mathbf{n}_{\pi} = \mathbf{A}_{Oj}^{-1}\mathbf{n}_{OP} \tag{15}$$

where $\mathbf{A}_{Oj}$ is the rotation transform between the superellipsoid's and the global coordinate systems. This normal $\mathbf{n}_{\pi}$ can be used in (9) to obtain the parametric coordinates φ_1, φ_2 of the point $\mathbf{r}_{\pi}$ in the superellipsoid's surface: the point which presents that normal direction. The 3D local coordinates of this point can be obtained via (7), and then converted to global coordinates to obtain

$$\mathbf{r}_{OQ} = \mathbf{r}_{Oj} + \mathbf{A}_{Oj}\mathbf{r}_{\pi} \tag{16}$$

where $\mathbf{r}_{Oj}$ is the origin of the plane's local coordinate system. The corresponding minimum distance point in the plane $\mathbf{r}_{OP}$ is obtained by projecting $\mathbf{r}_{OQ}$ onto the plane, as described in (Weisstein 2015)

$$\begin{aligned} \mathbf{w} &= \mathbf{r}_{OQ} - \mathbf{r}_{Oj} \\ d &= \mathbf{n}_{OP} \cdot \mathbf{w} \\ \mathbf{r}_{OP} &= \mathbf{r}_{OQ} - \|d\|\mathbf{n}_{OP} \end{aligned} \tag{17}$$

in which d is the signed distance between $\mathbf{r}_{OP}$ and $\mathbf{r}_{OQ}$. Positive, zero or negative values of d indicate that the superellipsoid and the plane are not intersecting, intersecting at a point, or penetrating each other, respectively. The pseudo-code for this algorithm is presented in Table 4.

3.3. Superellipsoid-cloth

The proposed collision detection algorithm between superellipsoidal surfaces and mesh-based cloth objects is similar to the one reported in (Moustakas et al. 2005). The implicit function of the superellipsoid (5) is evaluated at each vertex of the cloth's mesh, and a collision is reported if any of those vertices evaluates to less than or equal to 1. To avoid performing this test for every vertex of the mesh, its vertices are first grouped in AABBs. If, at a given time-step, the superellipsoid's AABB is intersecting one of the mesh's AABBs, then the implicit function is evaluated for the vertices inside that AABB. Otherwise, a collision is immediately ruled out for that particular zone of the cloth. An example of this AABB-based procedure is pictured in Figure 3.4.

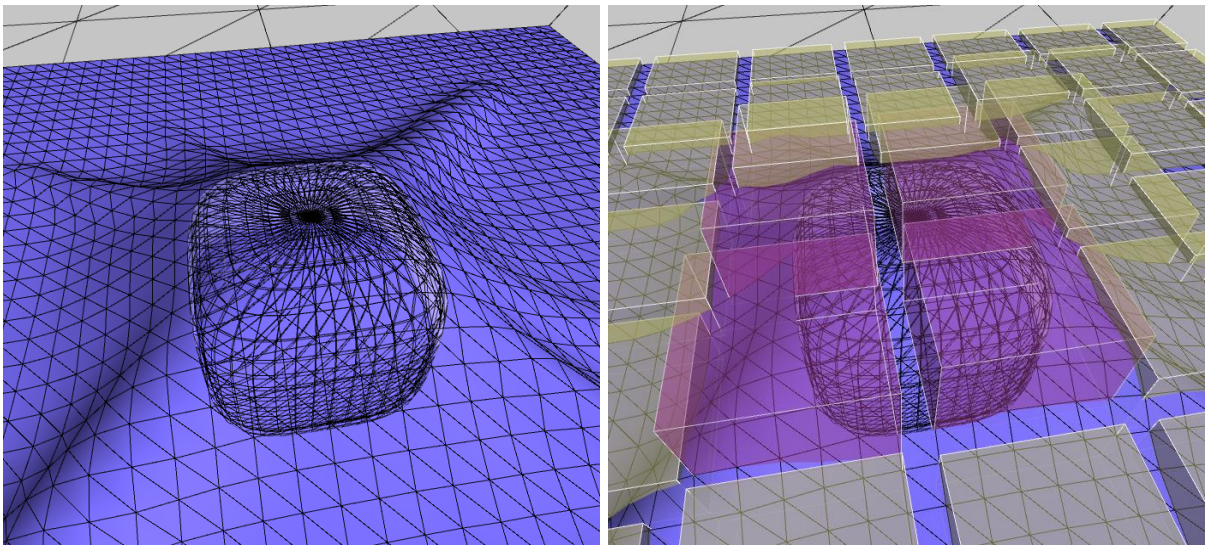


Figure 3.4 – Visualization of the broad phase of the collision detection between a superellipsoid and a mesh-based cloth. The $41 \times 41 = 1681$ cloth vertices are grouped and bounded into AABBs, of $6 \times 6 = 36$ vertices each, shown in yellow. In this case, only four AABBs shown in purple intersected the superellipsoid's AABB (not pictured). Thus, the implicit function value for the superellipsoid will only be computed for the 144 vertices inside the purple AABBs.

If a mesh vertex is found to be inside the superellipsoid, the contact normal direction is computed analytically using (8), and the radial Euclidean distance from the point to the surface is obtained analytically as described in (Jaklič et al. 2000). These quantities are used to generate a physical contact force to push the cloth and keep it out of the superellipsoid. While the radial distance is not the true penetration depth, it is much faster to compute, as the minimum distance would have to be calculated numerically, and the radial distance yields visually convincing results.

3.4. Superellipsoid and superovoid combinations

The proposed algorithms are based on (Lopes et al. 2010) and can compute the minimum distance between pairs of superellipsoid and superovoid combinations, using a numerical iterative approach to solve the equations provided by the common normal concept. Because superellipsoids can be represented as superovoids with $T_x = T_y = 0$, this section refers to the considered surfaces as superovoids in general. The original algorithm is formulated for implicitly-represented surfaces, while this thesis also proposes an adaptation using parametric representations. Furthermore, two novel methods of obtaining initial guesses for the iterative algorithm are presented.

For expressing the common normal condition mathematically, the normal and tangent directions to the surfaces must be computed. The expression for the normal direction to the superovoid surface, at a given point, can be obtained from the gradient of its implicit function (11). For the tangent and binormal directions, the Householder transformation is employed, as it reveals to be an efficient vector orthogonalization technique whose analytical nature provides a well-defined formula for the tangent and binormal vector spaces of implicit surfaces, as explained in (Lopes et al. 2013). This transformation is expressed as

$$\mathbf{H} = \mathbf{I} - 2\frac{\mathbf{h}\mathbf{h}^{\boldsymbol{T}}}{\mathbf{h}^{\boldsymbol{T}}\mathbf{h}} = \begin{bmatrix} 1-2\dfrac{h_1^2}{h^2} & -2\dfrac{h_1h_2}{h^2} & -2\dfrac{h_1h_3}{h^2} \\ -2\dfrac{h_1h_2}{h^2} & 1-2\dfrac{2h_2^2}{h^2} & -2\dfrac{h_2h_3}{h^2} \\ -2\dfrac{h_1h_3}{h^2} & -2\dfrac{h_2h_3}{h^2} & 1-2\dfrac{h_3^2}{h^2} \end{bmatrix} \tag{18}$$

where $\mathbf{n}$ is the given normal and

$$\mathbf{h} \equiv \mathbf{h}(\mathbf{n}) = \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix}^T = \begin{bmatrix} \max\left\{ n_x - \|\mathbf{n}\|_2 , n_x + \|\mathbf{n}\|_2 \right\} & n_y & n_z \end{bmatrix}^T \tag{19}$$

with

$$h \equiv \|\mathbf{h}\|_2 \tag{20}$$

Matrix $\mathbf{H}$ is symmetric and orthogonal, and its rows (or columns) form an orthogonal vector basis. Its first column is collinear to $\mathbf{n}$. Its second and third columns are perpendicular to $\mathbf{n}$, and can thus be considered as tangent $\mathbf{t}(\mathbf{n})$ and binormal $\mathbf{b}(\mathbf{n})$ directions associated with the normal at a given surface point.

For implicit surfaces, the common normal concept can be mathematically expressed as the following system of non-linear equations applied to the problem of finding the minimum distance between two surfaces:

$$\mathbf{\Phi}(\mathbf{q}) = 0 \Leftrightarrow \begin{bmatrix} \mathbf{n}_{OP}^T \cdot \mathbf{t}_{OQ} \\ \mathbf{n}_{OP}^T \cdot \mathbf{b}_{OQ} \\ \mathbf{d}_{PQ}^T \cdot \mathbf{t}_{OQ} \\ \mathbf{d}_{PQ}^T \cdot \mathbf{b}_{OQ} \\ F_i \\ F_j \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{21}$$

in which

$$\mathbf{q} \equiv \mathbf{q}_{implicit} = \begin{bmatrix} x_i & y_i & z_i & x_j & y_j & z_j \end{bmatrix}^T \tag{22}$$

is the vector of unknowns, containing the coordinates of the potential minimum distance points, in the local coordinate system of each implicit surface. The last two geometric constraints in (21) are geometric loci constraints, and ensure the points are on the surface of the superovoids.

For parametric surfaces, the two geometric loci conditions are not considered, as the first four equations are sufficient to find the four unknowns, and the parametric representation guarantees that the considered points lie on the surface. Thus, for the parametric variant of the algorithm introduced in this thesis, the vector of unknowns becomes

$$\mathbf{q} \equiv \mathbf{q}_{parametric} = \begin{bmatrix} \varphi_{1i} & \varphi_{2i} & \varphi_{1j} & \varphi_{2j} \end{bmatrix}^T \tag{23}$$

where $\varphi_{1,2\,i,j}$ are the parametric coordinates of two potential solution points for the minimum distance problem, on surfaces i and j.

For the case of superellipsoids and superovoids, the vector of geometric constraints $\mathbf{\Phi}$ is composed of C^2 continuous functions, which makes the function twice-differentiable. As such, (21) can be solved using the iterative Newton-Raphson method. An improved approximation to the solution (22) or (23) $\mathbf{q}_{k+1}$ is computed based on a potential candidate $\mathbf{q}_k$:

$$\mathbf{q}_{k+1} = \mathbf{q}_k - \left(\mathbf{\Phi}_{\mathbf{q}}(\mathbf{q}_k)\right)^{-1} \mathbf{\Phi}(\mathbf{q}_k) \tag{24}$$

where $\mathbf{\Phi_q}$ is the Jacobian matrix of the geometric constraints function (21) and k is the Newton-Raphson iteration index. This Jacobian matrix is computed numerically using finite differences. For performance reasons, the Jacobian matrix inversion is not explicitly computed; instead, the linear system $\mathbf{\Phi_q}(\mathbf{q}_k)x = \mathbf{\Phi}(\mathbf{q}_k)$ is solved in order to x, and this x is then subtracted from $\mathbf{q}_k$ to obtain $\mathbf{q}_{k+1}$. The iterative procedure stops once the difference between $\mathbf{q}_{k+1}$ and $\mathbf{q}_k$ is less than a certain tolerance, which can be easily adjusted to compute more accurate results, or accelerate the algorithm at the cost of precision.

3.4.1. Initial iteration

The presented algorithm is of numerical nature, and as such, an initial guess is needed to iterate over and reach an approximation of the solution. Due to the non-linearity of the superovoid expressions and the locality of the Newton-Raphson convergence, the proximity of the initial candidate points to the solution is critical; a poor guess will make the algorithm diverge and not reach a solution, or perform many iterations and run slowly.

Two methods were explored to find acceptable initial guesses. The first one consists of generating an octree approximation of the superovoids around their surfaces, and compute the minimum distance between them. The resulting pair of points are used as an initial guess for the Newton-Raphson algorithm. An example is visible in Figure 3.5. FCL already implements octree minimum distance algorithms through OctoMap (Hornung et al. 2013). This method is used when implicit surfaces are considered.

The second method is based on the parametric representation of superovoids. The two-dimensional parametric space defined in (12) is split into a quadtree for each superovoid, and the center of each cell of the quadtree is considered. For each level of the quadtree, every pair of center points of each superovoid is evaluated (16 pairs): the equivalent 3D points are computed using (12) and the Euclidean distance between them is calculated. The algorithm selects the two cells for which the computed distance is the smallest, and repeats the process inside the two selected cells for a given number of iterations. The two parametric coordinates found in the final iteration are used as the initial estimation for the parametric Newton-Raphson procedure. Experimentation suggests that 6 iterations yield acceptable estimations and allow the numerical procedure to converge. An example of execution of this algorithm is shown in Figure 3.6.

Both these methods converge globally to the solution, albeit at a slower rate than the Newton-Raphson algorithm. This means that, should the Newton-Raphson procedure not converge to the solution, the initial guess can always be recomputed with increased precision, either by considering a more finely divided octrees in the implicit method, or further subdividing the quadtrees in the parametric method. The improved guesses can then be used to restart the Newton-Raphson algorithm.

Collision detection algorithms can greatly benefit from exploiting temporal coherence, as the relative position of two simulated objects is likely not to change significantly during short time-steps. Thus, for the proposed algorithm, the minimum distance points computed in one collision query are used as the initial guess of the following query using the same surfaces.

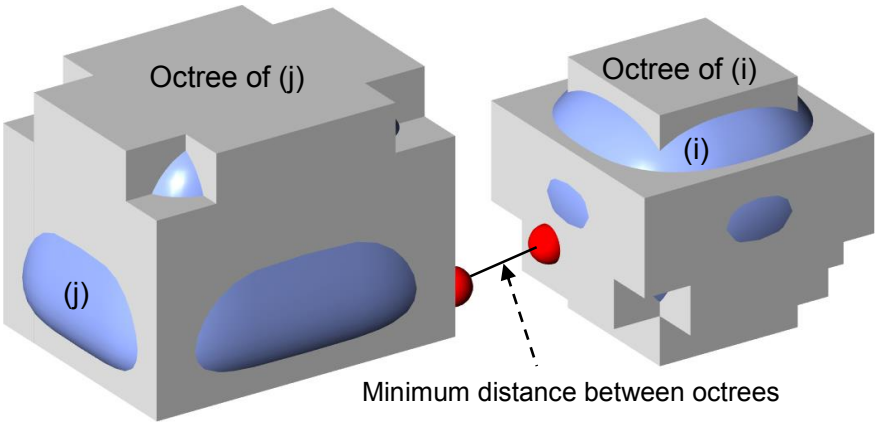


Figure 3.5 – Example of execution of the octree initial guess method (5x5x5 octrees), with the octrees in light grey, overlaid onto the respective superovoids. In red, the minimum distance points between the two octrees.

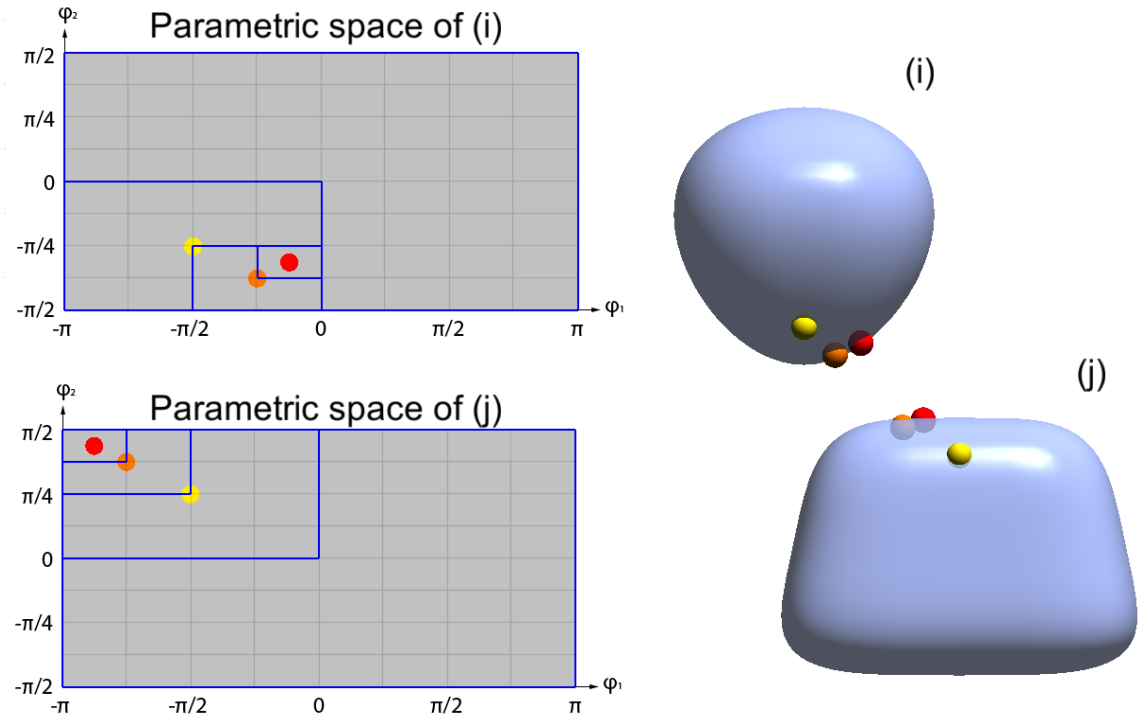


Figure 3.6 – Example of execution of the parametric quadtree initial guess method (3 iterations). On the left, the parametric spaces of each surface, along with the quadtree division and the iterations selected by the algorithm. Yellow is the first iteration, red is the last. On the right, a 3D representation of the surfaces, their relative position, and the corresponding 3D points of the iterations, in matching colors.

3.4.2. Implementation

The superovoid shape and the described algorithm were implemented in FCL, in C++. The matrix operations necessary for the Newton-Raphson method use the LAPACK library (Anderson et al. 1999). The Newton-Raphson procedure stops once the tolerance of 10^{-6} is reached, or when 30 iterations are executed, as a safeguard. Slight modifications were introduced in ROS (Quigley et al. 2009) and MoveIt! (Sucan & Chitta 2015), in particular to the URDF parsing modules, to support this new shape. The code has been published as a fork of FCL's repository (https://github.com/artur-ag/fcl), free and open source.

4. 3D modeling with smooth convex contact geometries

4.1. Multibody Sketcher

Traditional programs for 3D body modeling are complex and not very user-friendly, making them inappropriate for expeditiously creating sketches of articulated bodies. Multibody Sketcher was developed to tackle these problems, allowing the design of articulated structures associated with SCS contact geometries. This tool was designed to be used on touch-screen devices such as tablets and table-top screens. The output of this program is a skeleton rigged with superellipsoid-shaped contact geometries (hit volumes), which can be used in physical simulations. A working prototype of Multibody Sketcher was developed using Unity3D, and its interface is shown in Figure 4.1. All edition tools were implemented, but the planned module to export created models to COLLADA format was not, due to inconsistencies in how different programs interpreted the exported files. As this functionality was not critical for this prototype, it was left unimplemented.

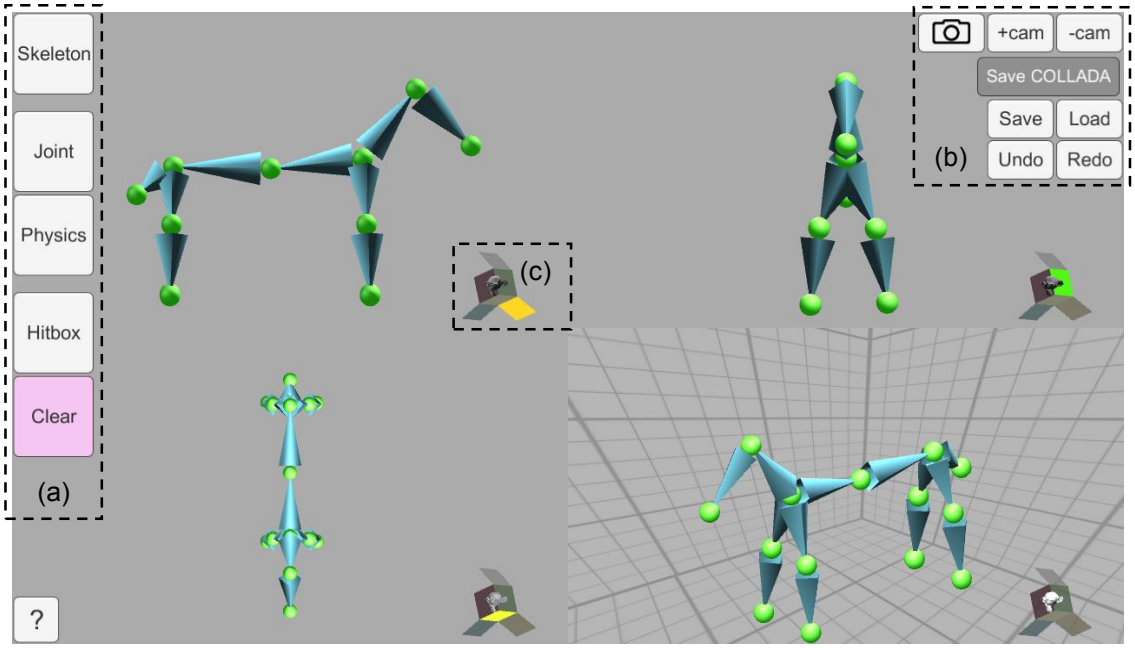


Figure 4.1 – Main interface of the developed Multibody Sketcher prototype, showing 4 perspectives of the sketched articulated body example, a horse. (a) mode selection, (b) addition or removal of perspectives, and general options, (c) open-box camera selection widget, one for each perspective.

Multibody Sketcher's workflow is divided into 3 modes. In "Skeleton" mode, the bones composing the body are laid out; in "Joint", the degree of freedom of each joint is configured, and this set-up can be verified by the user in "Physics" mode; in "Hitbox", the user creates the SCS which are attached to each bone. Specifics of each mode are explained in the following paragraphs.

In Multibody Sketcher, a body is composed of bones, attached together with joints. In "Skeleton" mode, the user creates bones with touch-based strokes, and two joints are created at the ends of this

stroke, visually represented as spheres. If the stroke is not a straight line, it is automatically segmented into smaller bones, creating a bone chain (see Figure 4.2). This is achieved by sampling the user's stroke at almost-regular intervals and computing the angle between each sampled segment. If this angle is above a threshold (35º), a new bone is created, unless this would result in a very short bone. An existing bone can also be split into two, by touching a point in the bone along its length. Created joints snap together and weld two or more bones if they are created close enough to each other: this allows the user to branch out of an existing bone, even with coarse strokes. To change an existing bone, one of the joints should be dragged after a touch-and-hold gesture.

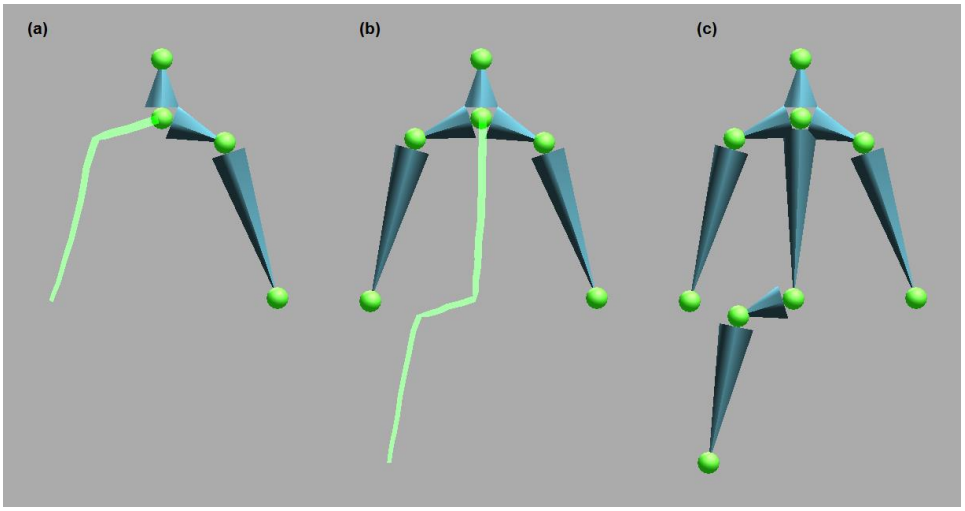


Figure 4.2 – Simple humanoid skeleton created in Multibody Sketcher, in a sequence of screenshots. Blue segments are bones, green spheres are joints. In (a) and (b), the light green line shows the paths drawn by the user: when the user releases their finger, the line becomes a chain of bones and joints.

Each joint can be configured individually, to allow or restrict the relative movement of the bones. In "Joint" mode, touching a joint will invoke a panel with three sliders that control its angular degrees of freedom, and the range of movement for each rotation. The ranges of movement are also symbolized in the 3D model as pie-charts aligned with the joint, displaying the possible rotations of the bones in an intuitive manner (see Figure 4.3). In addition, there are pre-built configurations for joints such as ball-and-socket and hinge joints, accessible through buttons on the joint panel. The joints are implemented as Unity `ConfigurableJoint` components. The created body can be physically tested in the "Physics" mode, which allows the user to confirm the bones and joints behave as expected when pulled or twisted. Manipulations in this mode are temporary, and unmade once the user enters another mode.

Each bone can be associated with one or more contact geometries, in the shape of superellipsoids, which become rigidly attached to the bone. The user creates these 3D geometries by drawing ellipses on top of bones on the screen, which are recognized using the CALI library (Fonseca et al. 2002). The initial size of the superellipsoid is based on the lengths of the sketched ellipse's axes (see Figure 4.4). These shapes can then be manipulated with two-finger touch gestures: circular motions will rotate the shape, pinch motions affect its scale, and dragging both fingers in the same direction moves the shape relative to the bone. To adjust the superellipsoid-specific parameters, the user invokes the widget described in section 4.3, by touching-and-holding on a superellipsoid.

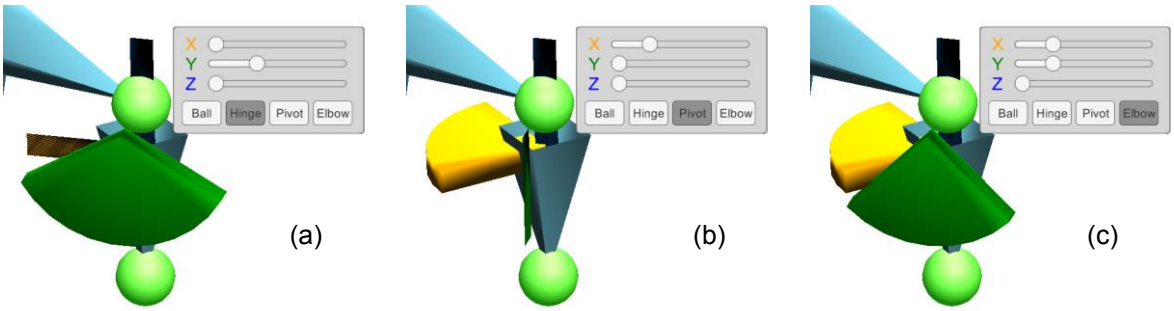


Figure 4.3 – Examples of usage of the joint edition panel, along with the 3D pie-chart models depicting the range of movement in each axis of rotation. These examples are the pre-configured joints available through buttons in the panel: hinge (a), pivot (b) and elbow (c) joints.

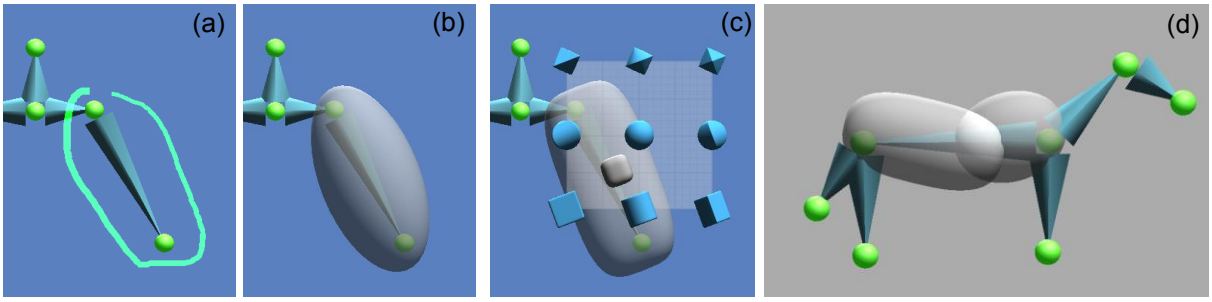


Figure 4.4 – Creation of contact geometries in Multibody Sketcher, via calligraphic sketching. The user's strokes (a) are interpreted to create a 3D ellipsoid (b), which can then be deformed as a superellipsoid (c), as well as rotated, translated and scaled. More than one shape can be attached to the same bone (d).

4.2. Open-box camera widget

Modeling 3D objects in a 2D surface imposes the projection of the objects into orthographic planes. For this reason, Multibody Sketcher offers 6 perspectives, all featuring orthographic projection: front, back, top, bottom, left and right. The views are picked via an open-box widget in the corner of the screen. This cube-shaped box represents the space around the 3D object, and recognizes touches in each of the 6 cube faces, which become brightly colored when selected. Each face of the box corresponds to a projection of the body. The back, bottom and left faces are folded outwards, so that the inside of the box is visible and clickable, hence the term open-box. The face-view correspondence is detailed in Figure 4.5.

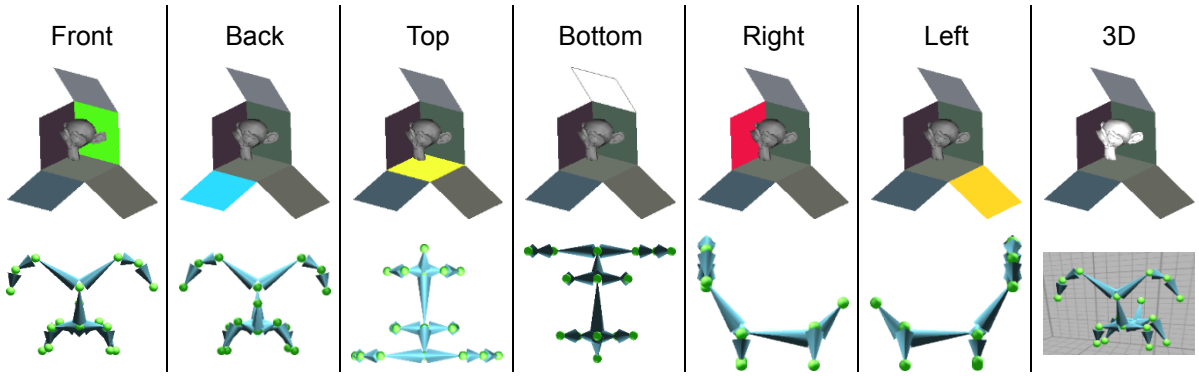


Figure 4.5 – Different selections of the open-box camera widget, and examples of respective views of a six-legged skeleton.

A seventh view is provided, for visualization only, in which the user can freely orbit around the created object. The object is surrounded by a grid, so that the exact movement of the camera in relation to the body is perceptible. The model cannot be modified in this view. The 3D view is selected by touching on the white chimpanzee head icon inside the open-box, which is a placeholder model from Blender (Blender Online Community 2015).

4.3. Superellipsoid parameter widget

The squareness of the superellipsoid shape is controlled by the ϵ_1 and ϵ_2 exponents. A touch-based widget was designed to allow users to easily modify these parameters. This widget is composed of a grid, with values of ϵ_1 and ϵ_2 $]0,2[$ mapped to the horizontal and vertical axes, respectively. Overlaid on the grid's edges are small icons that allow the user to preview what a superellipsoid looks like when its parameters have the values mapped to the icons' positions. The user invokes the widget by touching and holding over a superellipsoid, and without releasing the finger, sliding to the desired shape. Screenshots can be seen in Figure 4.6.

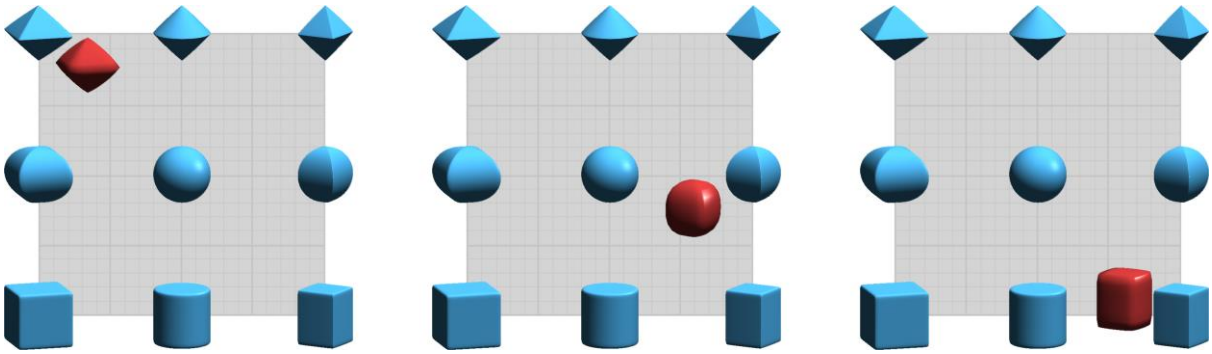


Figure 4.6 – Three screenshots of the superellipsoid parameter widget. In blue, the icon previews mapped to particular coordinates on the grid. In red, the cursor that the user moves around the grid. The red cursor changes shape in real time as the user drags it.

The preview icons are rotated according to the orientation of the superellipsoid being edited, so they are more visually suggestive of the final shape to the user.

4.4. Terrain Cloth

The Terrain Cloth is a terrain modeling application inspired on the blanket fort metaphor. Blanket forts, also known as pillow forts, are constructions made of blankets, bed sheets, pillows or sofa cushions, which young children build for fun. In Terrain Cloth, objects in the shape of superellipsoids are placed on a planar surface, and a blanket is spread on top of them. As the blanket settles down, it gains the shape of mountains, hills and valleys. The user creatively places and moves the objects around to change the shape of the terrain, also being able to push and pull the cloth, or even clasp parts of the cloth with virtual clothespins, suspended in the air. Very intricate representations of terrains can be modelled with a relatively small and quick input from the user, as the computational simulation of cloth is exploited to make it naturally form wrinkles and folds around objects. In particular, it is possible to create overhangs, unlike with traditional height-map-based terrain tools (Unity Technologies 2015). The

application was designed to run in tablets and table-top multi-touch screens, but also supports regular mouse controls.

A working prototype of Terrain Cloth was developed in Unity 4.6, and employs its physics engine to simulate the cloth. The contact detection between superellipsoids and the cloth was implemented separately, and is explained in detail in section 3.3. The computed contact forces are passed on to Unity's physics engine, which handles the remainder of the simulation.

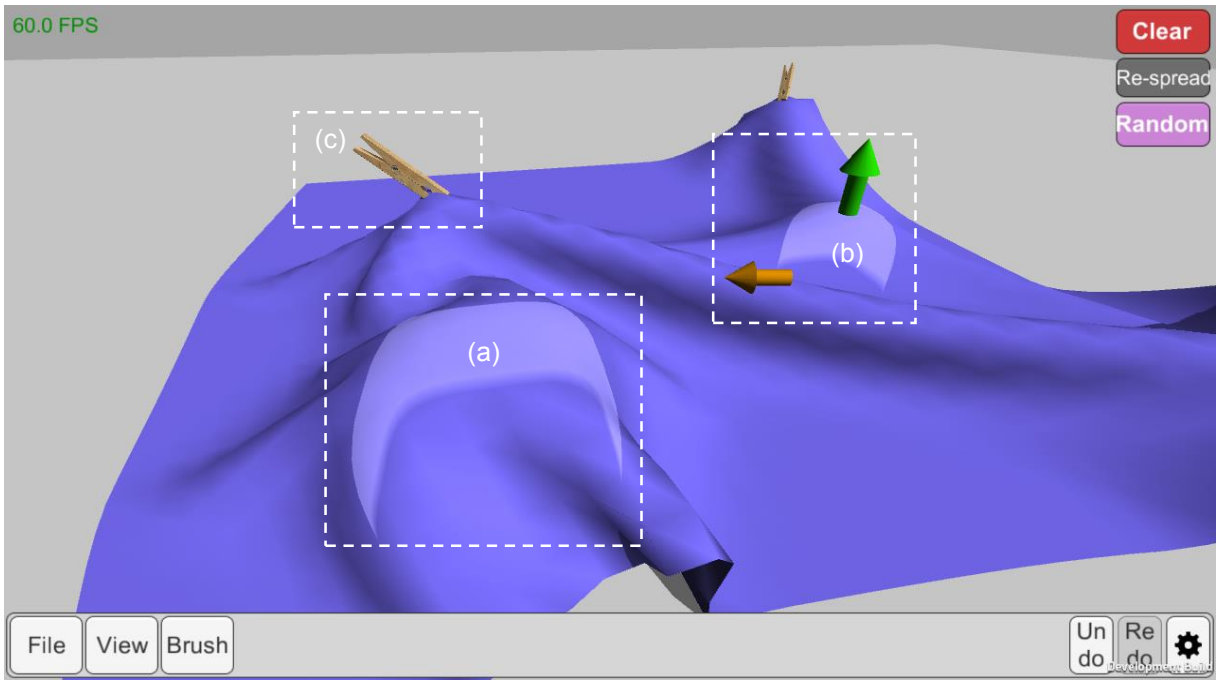


Figure 4.7 – Main interface of the Terrain Cloth app, showcasing the silhouette of objects under the cloth (a), the object scaling handles (b), and optional clothespins that suspend the cloth (c).

To instantiate objects, the user taps on the surface with 2 fingers: the object rises from below the cloth and can then be translated through the surface with 1 finger. The objects are visible as a transparent silhouette through the cloth, so that the user is aware of their position and shape without having to hide the cloth itself. After being selected, two handles appear next to the object, which control its height and width. To regulate the objects' superellipsoid squareness parameters, the widget presented in 4.3 is used.

If the user touches on a zone where no objects are present, they will grab onto the cloth instead. They can then push or pull the cloth around, to adjust its position and creases. If the user releases their finger right away, the cloth will naturally fall down; however, if the finger is held for a certain time (around half a second), a virtual clothespin will appear, indicating that the current pull force will be made permanent once the finger is released. These clothespins will stay in place, and can prevent the cloth from moving, suspending it in the air, as shown in Figure 4.7 (c). Clothespins can be detached and reattached in another zone of the cloth with a touch-and-hold action. To orbit the camera around the cloth, a 2-finger swipe is used, and a 2-finger pinch gesture will zoom the camera in or out.

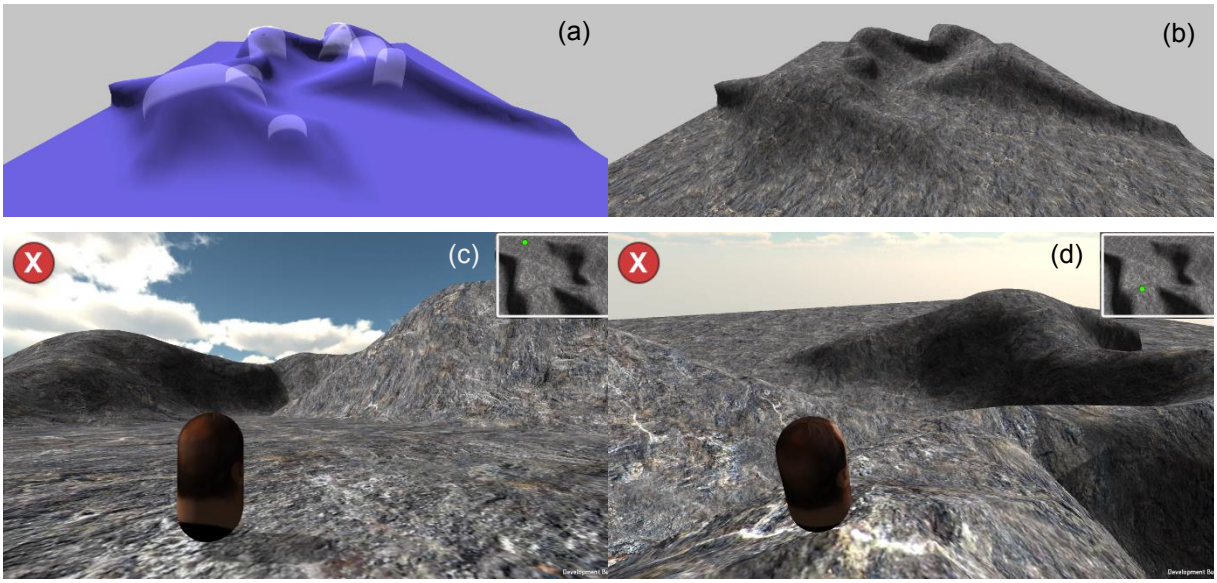


Figure 4.8 – Example of a mountainous landscape created in Terrain Cloth. The superellipsoid objects are visible during edition (a), but can be hidden for a visualization where the cloth has a rocky texture applied (b). In (c) and (d), the user explores their creation in “Game” mode, a third-person perspective free roam.

The appearance of the cloth can be changed to a mountain-like texture in real-time, allowing the user to confirm the terrain is gaining the intended shape. Underlying superellipsoidal objects are hidden in this mode, but can still be manipulated, along with the cloth itself. In “Game” mode, the terrain is scaled up and converted to a rigid mesh, so that the user can control a simple video game character and walk around the environment, evaluating their creation. For this mode, inputs on the left side of the touch screen move the character, and touches on the right side orbit the camera around the character, in a third person perspective. Screenshots can be seen in Figure 4.8.

5. Robotic grasping

This thesis proposes the usage of novel analytical SCS, in particular the superovoid, as contact geometries to be used in physical simulations for robotic grasp motion planning. To test this proposal, the iCub robot's hand (see Figure 5.1) (Metta et al. 2010) was represented with superovoids and superellipsoids, and comparisons were made with an existing mesh-based model in terms of accuracy and computation time of the minimum distance algorithms. The algorithm described in section 3.4 was implemented in FCL, and has been published as a fork of its online repository, free and open-source.

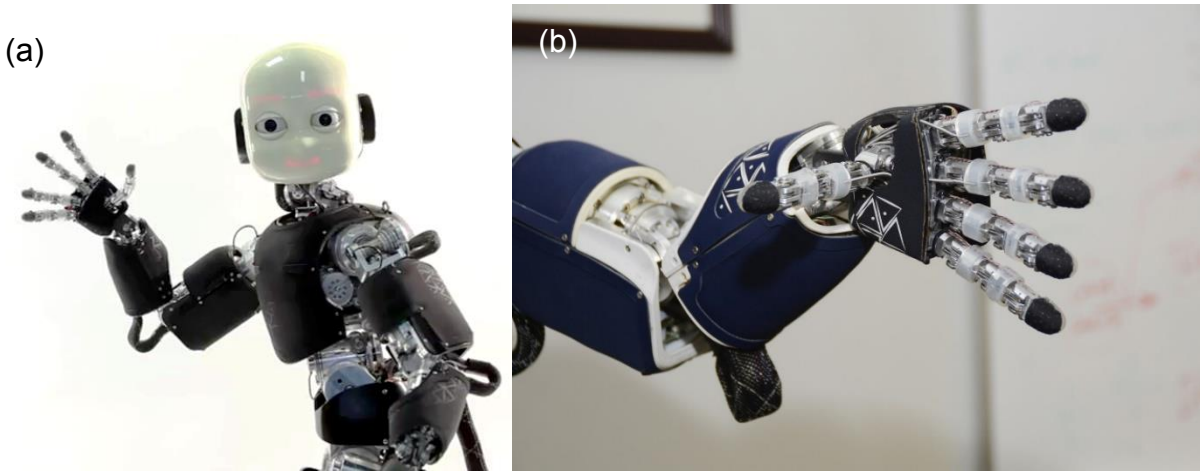


Figure 5.1 – Photos of the iCub robot and its hand. {(a) Adapted from (33rd Square 2015); (b) Adapted from IST's academic image repository (Técnico Lisboa 2015)}

The iCub robot's hands were manually approximated with superovoids and superellipsoids, using the original CAD drawings as reference (RobotCub 2014). The Multibody Sketcher modeling tool proposed in section 4.1 was used to produce a sketch version of the design, which was then improved with Unity's editor interface and converted to URDF. An approximation using polygonal meshes, available at (VisLab, ISR Lisboa 2014), was used for comparison, in which each finger is divided in 4 segments, or links. The meshes are composed of 300, 440, 910 and 770 vertices, in order from the base of the finger to the fingertip links, which means each finger link is represented by 605 vertices on average.

Superovoid and superellipsoid versions are implemented as modifications of the mentioned URDF file. The 5 fingers are represented with a total of 21 surfaces, and each individual link is around 16 mm long. The different representations are shown in Figure 5.2. Special attention was given to the underside and tips of the iCub's fingers, as these are the part of the surfaces that come in contact with objects during grasping interactions. As such, parts of the proposed geometries cover empty space on top of the fingers, but this is not problematic for the application in mind.

The resulting URDF descriptions were finally imported into the iCub's toolset (VisLab, ISR Lisboa 2014), allowing them to be used in the MoveIt! motion planner (Sucan & Chitta 2015). To support the new superovoid and superellipsoid shapes, some 3D model and URDF parsing modules in MoveIt! and

ROS (Quigley et al. 2009) had to be slightly modified, and so these tools were recompiled. The results and benchmarks of this case-study are presented in section 6.2.

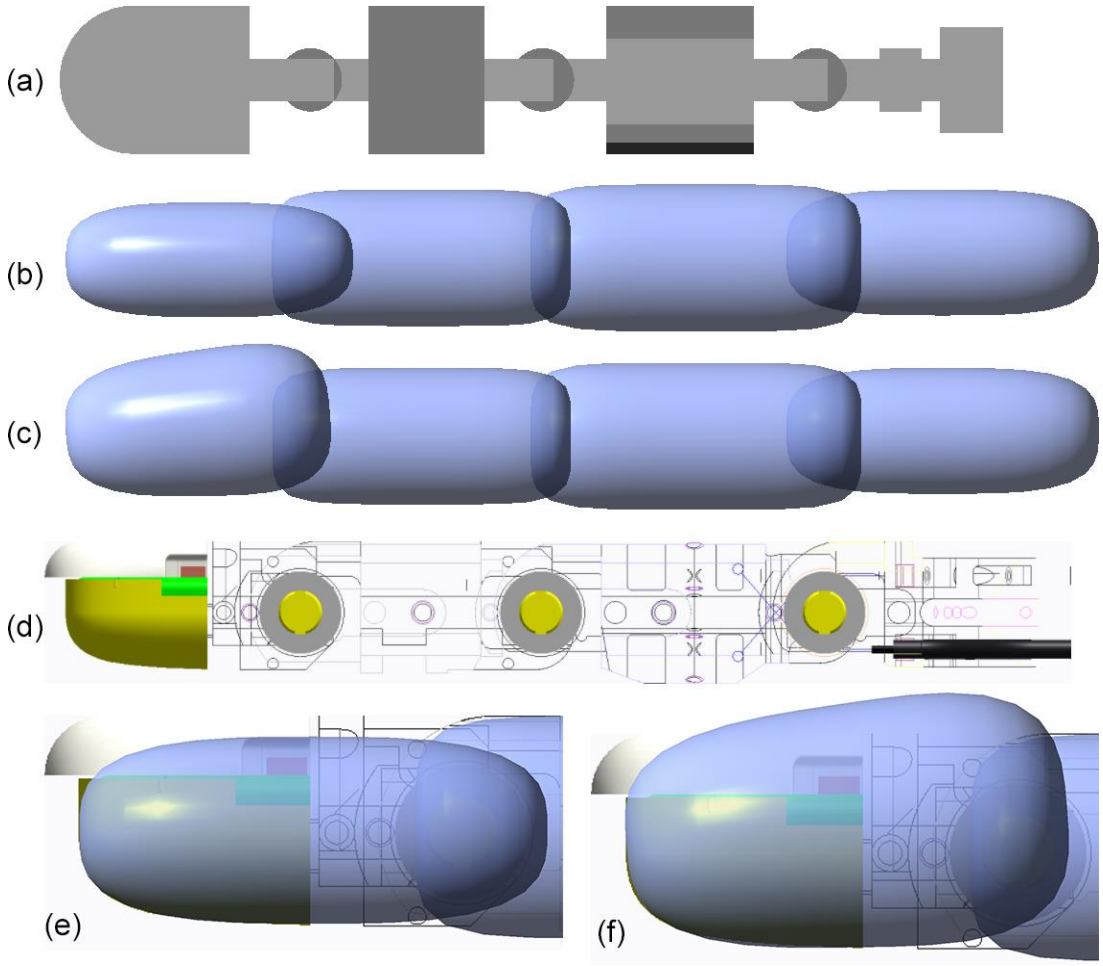


Figure 5.2 – Approximation of the iCub's right index finger using different representations. (a) simplified sphere and cylinder meshes from (VisLab, ISR Lisboa 2014); (b) proposed superellipsoid representation; (c) proposed superovoid representation; (d) original CAD illustration from (RobotCub 2014); (e) close-up of the fingertip in (b) superimposed on the CAD image; (f) close-up of the fingertip in (c) superimposed on the CAD image.

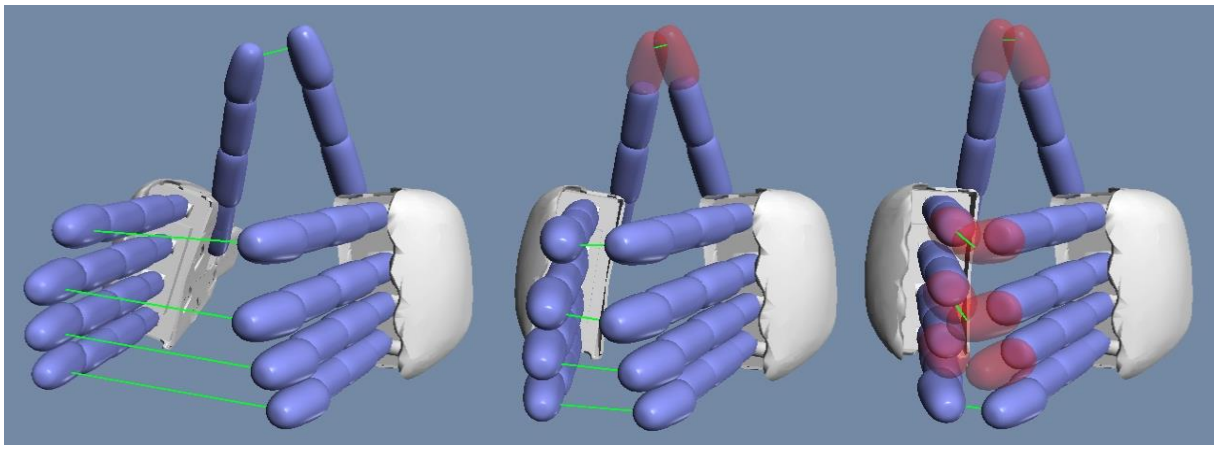


Figure 5.3 – Superovoid representation of the iCub hand performing a test movement. In green, the minimum distance between each fingertip. In red, links which are intersecting one another.

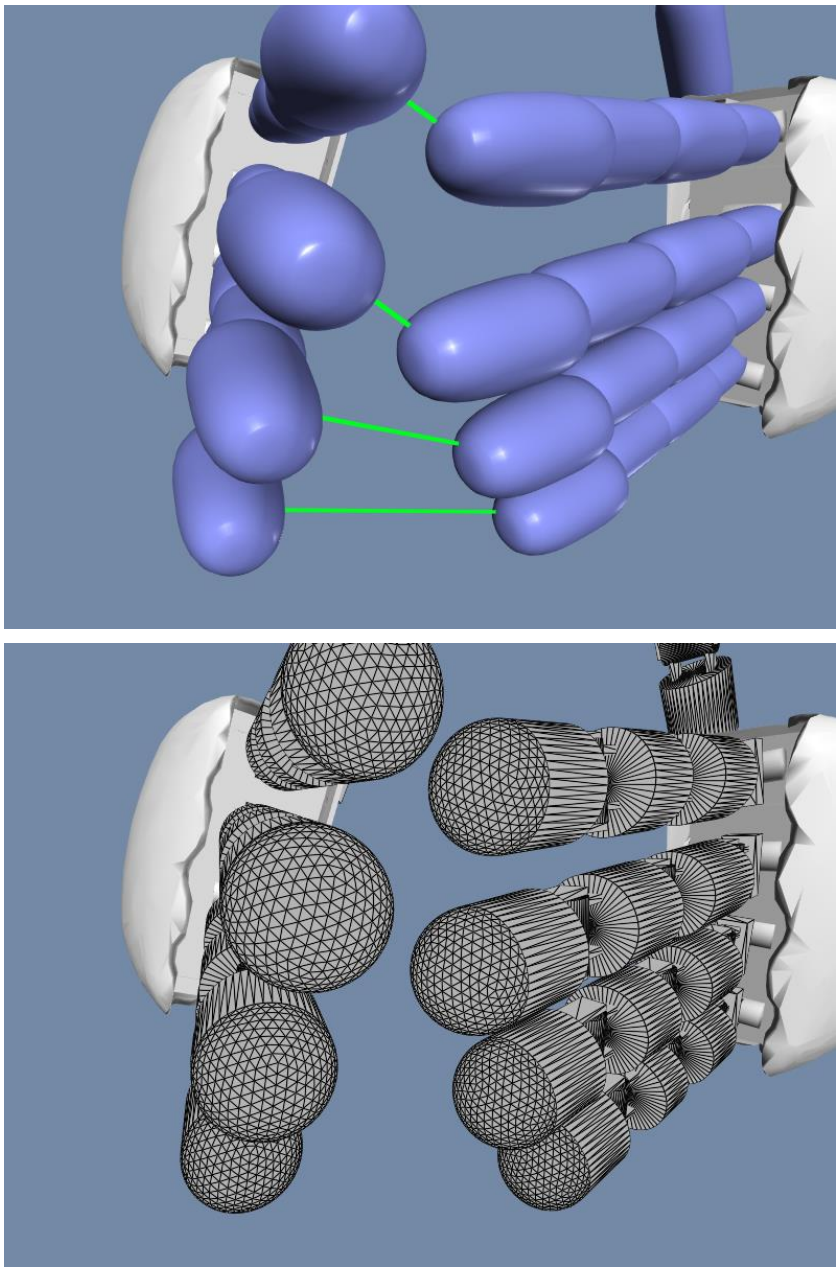


Figure 5.4 – Comparison between the proposed superovoid representation of the iCub fingers (a) and the mesh-based version (b) from (VisLab, ISR Lisboa 2014). In green, the minimum distance between the superovoid-shaped fingertips.

6. Results and discussion

6.1. Terrain Cloth

The computational performance of the Terrain Cloth application was evaluated through the frame rate metric, for different numbers of superellipsoids colliding against a cloth, modeled with a 1681-vertex mesh. The superellipsoids were laid out in a squared grid, completely covered by the cloth, and the cloth was left to fall over them. Once the cloth was stable, the application's frame rate was measured and averaged for 5 seconds. The minimum frame rate during this time was also registered, and the results can be seen in Figure 6.1. These tests were performed under a Windows 8.1 x64 system with the following specs: Intel® Core™ i7-4700MQ @ 2.40 GHz, NVIDIA® GeForce® GT 740M and 4 GB of RAM, and a rendering resolution of 430x318.

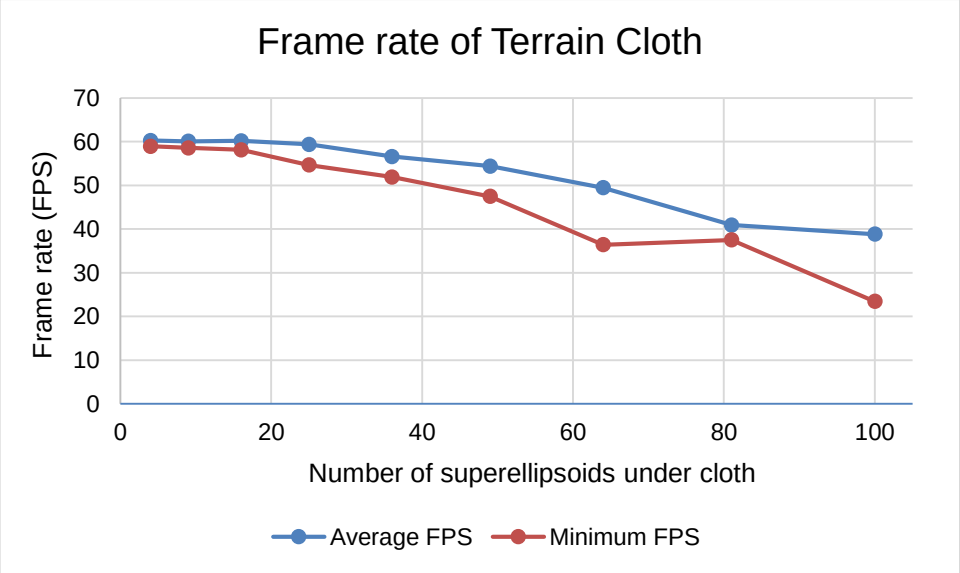


Figure 6.1 – Average and minimum frame rate of the Terrain Cloth application, with respect to the number of superellipsoidal objects under the cloth.

As the presented results englobe all the Unity game loop (script updates, physics, rendering, etc.), they are an indicator of the overall performance perceived by the users of the application: for 25 superellipsoid objects or less, the application can render 60 frames per second, the gold standard for interactive software. As the number of objects increases to 80 and more, frame rate approaches the considered minimum acceptable 30 FPS. Nevertheless, Terrain Cloth users are not likely to use those many objects in a single terrain scene. Even though this benchmark does not focus exclusively on the collision detection algorithm, its results suggest that the SCS-based algorithm exposed in section 3.3 is adequate for real-time interactive applications.

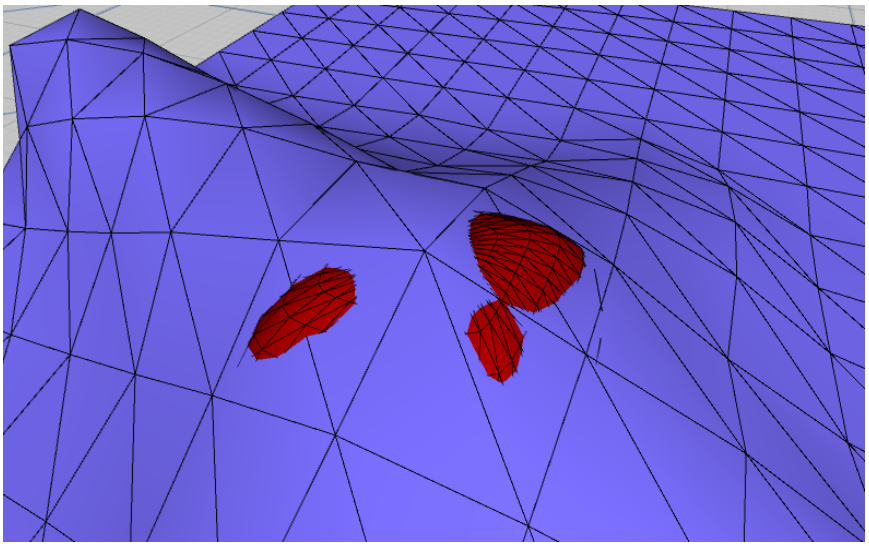


Figure 6.2 – Interpenetration between a superellipsoid and the cloth, shown in red, occurring when a low-resolution mesh is used to represent the cloth (441 vertices).

Some visual interpenetration problems occur when a low-resolution mesh is used to represent the cloth, because of the dependence of the collision detection algorithm on the vertices of said mesh. If the distance between vertices is too large compared to the size of the superellipsoid objects underneath the cloth, then parts of the triangles composing the mesh will not be acted by any contact force, and will penetrate the superellipsoids. This does not happen with the default resolution of the cloth mesh (1681 vertices), and this higher-resolution mesh did not threaten the real-time performance of the application, as seen in the previous experiment, so it is considered a minor problem. It does, however, place restrictions in the relative size of the simulated superellipsoid objects compared to the cloth: they cannot be very small, or this problem will occur frequently.

6.2. Robotic grasping

The proposed implicit and parametric versions of the algorithm exposed in section 3.4 were timed and compared with FCL's mesh minimum distance query. The simulation was performed in MoveIt! and consisted in moving the iCub's right hand towards its left, palms facing each other, until the tips of both thumbs touched, and then rotating the right hand inward, causing the fingers of the right hand to intersect and pass through the fingers of the left. This ensured plenty of collisions between the different links of the fingers. Figure 5.3 depicts part of this movement, up to the point the first finger links interpenetrate. The movements were manually performed in the RViz interface. Queries were timed only when the considered pair of surfaces were intersecting.

The hand representations proposed in section 5 were considered, with mesh, implicit and parametric versions tested separately using the following hand representations: the simplified meshes of the iCub hand from (VisLab, ISR Lisboa 2014) (i); superellipsoids (Figure 5.2 (b)) using the superellipsoid-specific implementation, which is mathematically simpler (ii); the same representation but with the general superovoid implementation (iii); and the mixed representation (Figure 5.2 (c)) using the superovoid implementation for the fingertips and the superellipsoid implementation for the other links (iv). As

mentioned in section 5, each finger link is represented with an average of 605 vertices, in the case of the mesh representation.

A total of 8 tests were performed in MoveIt! with ROS Indigo, both compiled from source, on a Lubuntu 14 x32 system with an Intel® Core™ i7-4700MQ @ 2.40 GHz and 4 GB of RAM. The time results of these tests are shown in Table 2 and Figure 6.3. An isolated test of the geometric accuracy of the algorithms was run, using 2 superovoids and comparing them to the 2 corresponding meshes generated from discretizations of varying resolution. Its results can be seen in Figure 6.4.

Table 2 – Minimum distance computation times (average and standard deviation) and number of Newton-Raphson iterations (average), taken from around 1600 queries, for FCL's mesh implementations and for the proposed algorithms.

Representation	**Average call time (μs)**	**σ of call time (μs)**	**Average iterations**
Meshes (BVH collision)	28.29	21.98	-
Meshes (minimum distance)	181.03	95.21	-
Superellipsoids (implicit)	127.71	299.89	3.7
Superovoids (implicit)	123.55	309.49	3.5
Mixed (implicit)	128.62	277.83	2.7
Superellipsoids (parametric)	39.01	96.27	2.3
Superovoids (parametric)	45.67	95.11	2.1
Mixed (parametric)	49.55	116.65	2.3

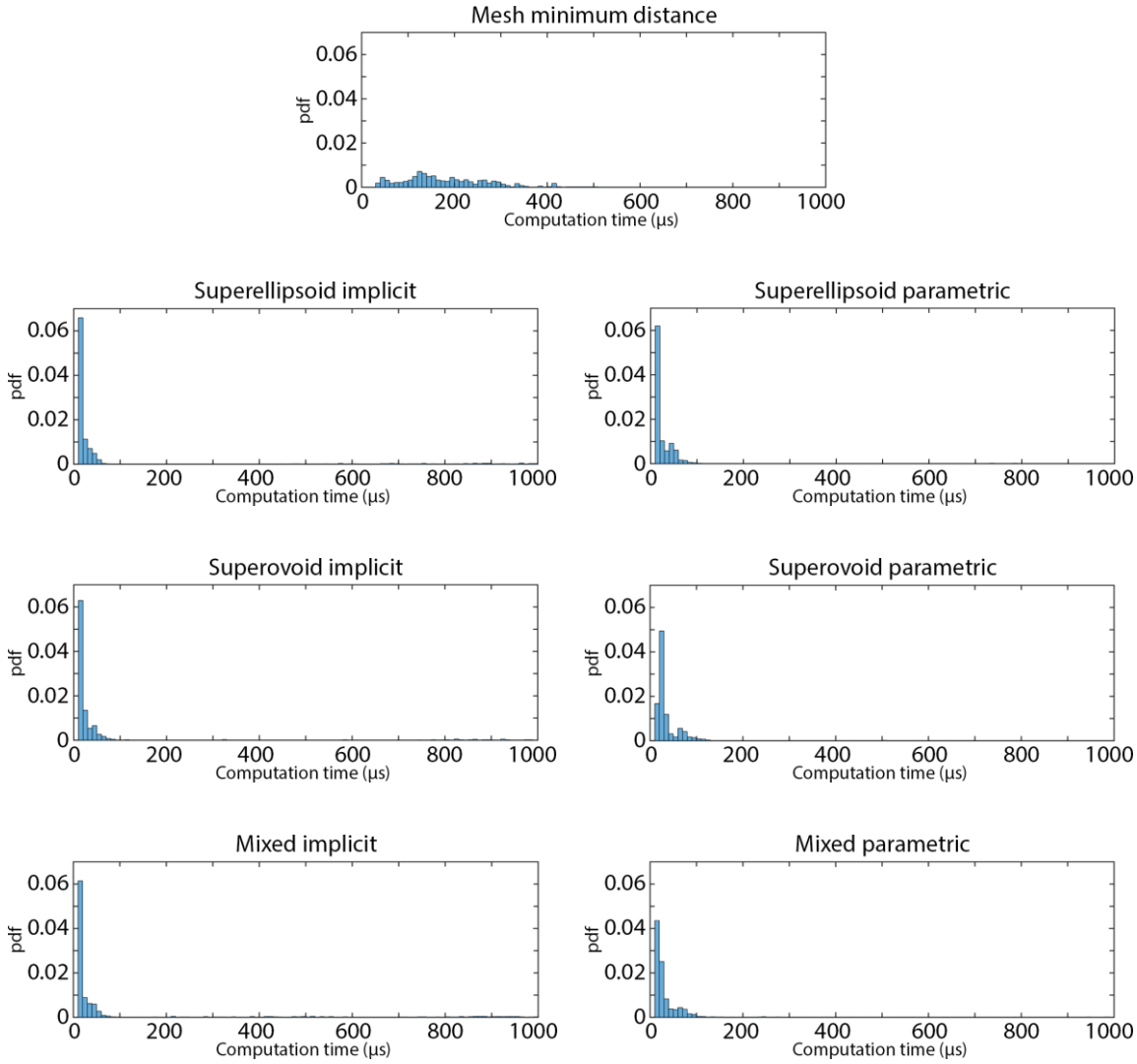


Figure 6.3 – Computation time distributions of the minimum distance queries, comparing FCL's mesh minimum distance implementation with the proposed methods. In all cases the average and median times are lower compared to the mesh algorithm. In some rare cases the proposed methods can have high computation times (≈1 ms) if the Newton-Raphson procedure diverges and has to start over from a new initial estimation (more noticeable in the implicit representations).

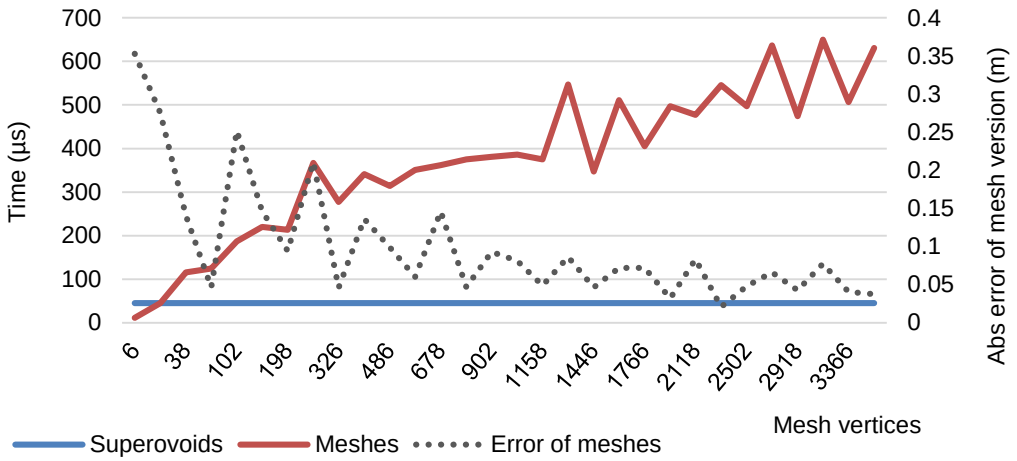


Figure 6.4 – Comparison of computation times and geometric accuracy between the proposed parametric superovoid algorithm and the FCL mesh minimum distance query. The meshes are in the shapes of superovoids, of length parameters $a_1, a_2, a_3 \in [0.96, 1.4]$ m, with different resolutions.

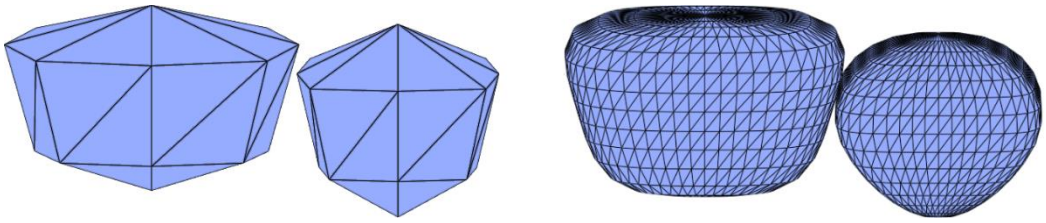


Figure 6.5 – Meshes of superovoids used for the accuracy benchmark of Figure 6.4, in their relative positions. On the left, meshes with 18 vertices, 32 triangles. On the right, meshes with 786 vertices, 1568 triangles.

The results in Table 2 indicate that the proposed algorithms run 1.5 to 4 times faster than their mesh equivalent. As for FCL's mesh collision detection query, despite being significantly faster than the minimum distance queries, it uses an OBB hierarchy to accelerate the computation at the expense of geometric accuracy, hence it cannot be considered a minimum distance technique but rather a proximity query method.

The algorithm for implicit surfaces is considerably slower than the parametric version, due to the octree-based initial estimation being quite expensive (500 µs per estimation, on average), and a new estimation must be calculated if the Newton-Raphson algorithm does not converge using the result from the previous time-step as the initial iteration. The parametric quadtree-based estimation method is simpler, as it is defined in a 2D domain, and the parametric geometric constraints, while requiring more non-linear function evaluations due to the co-sines and sines, guarantee that the potential points are on the surfaces and the numerical method diverges less often. Still, these spurious needs for new estimations increase the time standard deviation of the proposed algorithms, especially for the implicit versions.

The simpler superellipsoid versions were not significantly different than the general superovoid ones in terms of computation time, even though the mathematical expressions for computing normals and values of the implicit function were slightly faster when tested in isolation.

Regarding the geometric precision of the superovoid and superellipsoid queries, these can be easily adjusted, while for mesh-based algorithms, this would involve regenerating the mesh with a different number of vertices and triangles, which might not be feasible in real-time. The reported contact normal for mesh-based algorithms can also be noisy due to the faceted nature of polygonal meshes and strictly depends on its tessellation, which does not happen for the proposed algorithms. As shown in Figure 6.4, the mesh-based algorithm only achieves a precision of 10^{-1} m using more than 786 vertices (see Figure 6.5); the superovoid version's precision is 10^{-6} m and takes an order of magnitude less time to compute. Furthermore, the superovoid query is faster than the mesh one, except for meshes with 18 vertices or less, and these meshes lead to very low geometric precision, as Figure 6.5 visually suggests.

Finally, it is worth noting that, using either the implicit or parametric representations, it is possible to fully describe a superovoid storing only 7 (shape) + 6 (rigid body) parameters stored as floating point numbers, while polygonal meshes occupy considerably more memory to store vertices, triangles and eventual BVH geometries. Thus, the pros and cons of using implicit or parametric superellipsoids or superovoids for minimum distance computations, compared to meshes, are qualitatively summarized in Table 3.

Table 3 – Qualitative comparison of the proposed and studied geometries and respective algorithms, in terms of computation time (average and standard deviation), required memory for representation, and geometric accuracy.

Representation	**Time (avg)**	**Time (σ)**	**Memory**	**Accuracy**
Meshes (BVH collision detection)	fastest	low	high	low
Meshes (minimum distance)	slow	low	high	low
Superellipsoids/superovoids implicit	fast	high	low	high
Superellipsoids/superovoids parametric	faster	high	low	high

7. Conclusions and future work

7.1. Conclusions

Simulations of physical systems demand increasingly more geometric accuracy, combined with efficient usage of computational resources; in particular, processing time has strict constraints in real-time interactive applications, such as video games and robotic controllers, which were targets of study on this thesis. As far as collision detection is concerned, polygonal meshes and bounding volume hierarchies are considered gold standards both in literature and industry, but they both entail difficulties: meshes must be very detailed to achieve high geometric precision, and this slows down mesh-based algorithms, and BVH are, by design, implemented for performing very fast, low-accuracy proximity queries.

This thesis analyzed the usage of smooth convex surfaces and their implicit and parametric representations for efficient real-time collision detection. In particular, the superellipsoid and superovoid geometries were studied, and applied in various case-studies, demonstrating that SCS are adequate for both 3D modeling and efficient collision detection. The results are summarized in this section.

Two prototypes of interactive sketch-based applications were developed, with the aim of studying the usage of smooth convex surfaces for contact geometry modeling: Multibody Sketcher and Terrain Cloth. The first makes it easy and quick to create an articulated skeleton and attach SCS collision volumes to its bones. The second accelerates the process of producing terrains, by creating and laying out SCS under a simulated cloth, which collides and drapes on top of the SCS. Terrain Cloth was also used to benchmark the superellipsoid-cloth collision detection algorithm, showing it is adequate for real-time interactive applications.

A superovoid minimum distance algorithm was implemented in FCL, and compared to mesh-based alternatives in a robotics case-study involving the iCub hand. Both the implicit and parametric versions of the proposed algorithm performed faster than FCL's mesh minimum distance query. In addition, the SCS representation requires considerably less memory when compared to mesh geometries, and allows computation of arbitrarily accurate contact points and normal directions. It was noted that the parametric representation led to noticeably faster algorithm than the implicit one.

Collision detection is an important and useful tool in a multitude of fields involving computer science and simulation tools. Thus, the studied methodologies and algorithms can be employed in many different applications, heightening the importance of the contributions exposed in this thesis.

7.2. Future work

The explored algorithms and methods leave room for improvement and open up interesting investigation avenues, which are proposed in this section.

Extensive geometric benchmarks on superovoid algorithms – The presented tests compared the implicit and parametric versions of the superovoid/superellipsoid collision detection algorithm, when applied to a robotics case-study, and using one particular set of movements between the tested surfaces. A more robust set of benchmarks should be performed to further compare these algorithms, in particular when the same initial estimation of the solution is provided to the numerical Newton-Raphson procedure for each version of the algorithm. These benchmarks should include many different relative spatial configurations of the surfaces.

Reliability of superovoid algorithms – The quality of the initial estimation used to start off the Newton-Raphson procedure is critical for the convergence of the algorithms. This thesis presented two methods to find an acceptable initial estimation, based on the Euclidean distance between points on the two superovoid surfaces, using the implicit and parametric representations of the surfaces. Proposals of other methods, in particular methods taking into account the common normal condition, or faster and more accurate methods, should improve the robustness and overall quality of the algorithm. Furthermore, the analytical computation of the Jacobian matrix used in the numerical procedure should boost its speed, when compared to the finite-differences approximation employed in this thesis.

Multibody Sketcher application – The presented Multibody Sketcher prototype should be refined and tested with users, to appropriately assess its advantages for contact geometry modeling when compared to traditional software suites. It is also worth exploring the potential of Multibody Sketcher as an animation tool, where users can pose and animate the created models using techniques such as the line-of-action proposed in (Guay et al. 2013).

Terrain Cloth application – While a functional prototype of Terrain Cloth was presented, and this prototype suggests the cloth metaphor is useful to expeditiously create terrains, no user-tests were performed. To fully support this claim, the application should be polished and tested on users, for example, tasked with creating a certain terrain feature in Terrain Cloth, and in another existing terrain modeling software for comparison. The implementation of superellipsoid-edge and superellipsoid-triangle collision detection modules is also worth considering, to improve the visual accuracy of the simulated mesh-based cloth.

Robot modeling using SCS – Smooth convex surfaces were used to model the iCub's fingers for the case-study presented in this thesis, and their usage brought measurable benefits. Representing other robots and robotic parts may also be valuable – in particular, the iCub itself features smooth arms and torso parts, which may be accurately modelled with superovoids and superellipsoids.

Robot physical simulation using SCS – The presented robotics case-study was completed in the MoveIt! (Sucan & Chitta 2015) motion planning software, which only performs geometrical collision detection. The adjustable accuracy of smooth convex contact geometries will surely be of even greater value if applied to physics-based tools such as Gazebo (Koenig & Howard 2004), which simulate contact forces, torques and friction for robotic applications.

8. References

33rd Square (2015), *iCub photo*, viewed 14 October 2015, <http://www.33rdsquare.com/2015/02/icub-becomes-master-of-balancing.html>.

Anderson, E, Bai, Z, Bischof, C, Blackford, S, Demmel, J, Dongarra, J, Du Croz, J, Greenbaum, A, Hammarling, S, McKenney, A and Sorensen, D (1999), *LAPACK Users' Guide*, 3rd edn, Society for Industrial and Applied Mathematics, Philadelphia, PA.

Autodesk, Inc. (2015), *Autodesk Maya*, <http://www.autodesk.com/products/maya/overview>.

Barr, AH (1981), 'Superquadrics and Angle-Preserving Transformations', *Computer Graphics and Applications, IEEE*, vol 1, no. 1, pp. 11-23.

Bender, J, Koschier, D, Charrier, P and Weber, D (2014), 'Position-based simulation of continuous materials', *Computers & Graphics*, vol 44, pp. 1-10.

Blender Online Community (2015), *Blender 2.73 - a 3D modelling and rendering package*, <http://www.blender.org>.

Boon, CW, Houlsby, GT and Utili, S (2013), 'A new contact detection algorithm for three-dimensional non-spherical particles', *Powder Technology*, vol 248, pp. 94-102.

Campos, M & Passerin, J (2015), *mGear: Rigging Framework for Autodesk Maya*, <http://www.miquel-campos.com/post/109768264793/mgear-10-rigging-framework-for-autodesk-maya>.

Capcom Co., Ltd. (2008), *Devil May Cry 4*, Japan, <http://www.devilmaycry.com/>.

Capcom Co., Ltd. (2014), *Ultra Street Fighter IV*, <http://www.streetfighter.com/us/usfiv>.

Chen, S, Xin, S, He, Y and Wang, G (2012), 'The Closest and Farthest Points to an Affine Ellipse or Ellipsoid', *Tsinghua Science and Technology*, vol 17, no. 4, pp. 481-484.

Chittawadigi, RG & Saha, SK (2013), 'An analytical method to detect collision between cylinders using dual number algebra', *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, IEEE, Tokyo.

Choi, Y-K, Wang, W, Mourrain, B, Tu, C, Jia, X and Sun, F (2014), 'Continuous collision detection for composite quadric models', *Graphical Models*, vol 76, no. 5, pp. 566 - 579.

Counter-Strike Wiki (2014), *Hitbox*, viewed 1 January 2015, <http://counterstrike.wikia.com/wiki/Hitbox>.

CustomTerrain (2012), *Experimental Textured Blanket Gaming Surface, fallout terrain rpg*, viewed December 2014, <https://www.youtube.com/watch?v=qPL9jHUtsyE>.

Decapre (2014), *Ryu Hitboxes Normals - Ultra Street Fighter IV*, viewed December 2014, <http://decapre.com/watch?v=MW8WwWNO0nE>.

Duncan, M (2005), *Applied Geometry for Computer Graphics and CAD*, 2nd edn, Springer Undergraduate Mathematical Series, USA.

Epic Games, Inc (2014), *Static Mesh Collision Reference*, viewed December 2014, <https://docs.unrealengine.com/latest/INT/Engine/Physics/Collision/Reference/index.html>.

Epic Games, Inc (2014), *Unreal Engine 4*, <https://www.unrealengine.com/>.

Ericson, C (2004), *Real-time collision detection*, CRC Press, Boca Raton.

Farin, G (2002), *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*, 5th edn, Academic Press, San Diego, CA.

Fire and Fury Games (2014), *Building Attractive Wargame Terrain*, viewed 9 October 2015, <http://www.fireandfury.com/conventions/terain.php>.

Fonseca, MJ, Pimentel, C and Jorge, JA (2002), 'CALI: An online scribble recognizer for calligraphic interfaces', *AAAI spring symposium on sketch understanding*, pp. 51-58.

Gamebase USA & Gamebase Co., Ltd. (2012), *Gamebryo*.

Gilitschenski, I & Hanebeck, UD (2012), 'A Robust Computational Test for Overlap of Two Arbitrary-dimensional Ellipsoids in Fault-Detection of Kalman Filters', *Information Fusion (FUSION), 2012 15th International Conference on*, IEEE, Singapore.

Guay, M, Cani, M-P and Ronfard, R (2013), 'The Line of Action: an Intuitive Interface for Expressive Character Posing', *ACM Transactions on Graphics*, ACM, New York, NY, USA.

Handle project (2009), *Photo gallery*, viewed 01 January 2015, <http://www.handle-project.eu/index.php/publications/photogallery>.

Havok (2011), *Havok*, Dublin, Ireland, <http://www.havok.com/>.

Hornung, A, Wurm, KM, Bennewitz, M, Stachniss, C and Burgard, W (2013), 'OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees', *Autonomous Robots*, vol 34, no. 3, pp. 189-206.

Igarashi, T, Matsuoka, S and Tanaka, H (1999), 'Teddy: A Sketching Interface for 3D Freeform Design', *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press/Addison-Wesley Publishing Co., Los Angeles, California.

Jacobson, A, Panozzo, D, Glauser, O, Pradalier, C, Hilliges, O and Sorkine-Hornung, O (2014), 'Tangible and Modular Input Device for Character Articulation', *ACM Trans. Graph.*, ACM.

Jaklič, A, Leonardis, A and Solina, F (2000), 'Superquadrics and Their Geometric Properties', in *Segmentation and Recovery of Superquadrics*, Springer Netherlands.

Jia, X, Choi, Y-K, Mourrain, B and Wang, W (2011), 'An algebraic approach to continuous collision detection for ellipsoids', *Computer Aided Geometric Design*, vol 28, no. 3, pp. 164-176.

Johnson, KL (1985), *Contact Mechanics*, Cambridge University Press.

Kimoto, T & Yasuda, Y (1997), 'Shape description and representation by ellipsoids', *Signal Processing: Image Communication*, vol 9, no. 3, pp. 275-290.

Kodam, M, Curtis, J, Hancock, B and Wassgren, C (2012), 'Discrete element method modeling of bi-convex pharmaceutical tablets: Contact detection algorithms and validation', *Chemical Engineering Science*, vol 69, pp. 587-601.

Koenig, N & Howard, A (2004), 'Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator', *IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, Sendai, Japan.

Leithinger, D & Ishii, H (2010), 'Relief: A Scalable Actuated Shape Display', *Proceedings of the Fourth International Conference on Tangible, Embedded, and Embodied Interaction*, ACM, Cambridge, Massachusetts.

Lopes, DS (2013), 'Smooth convex surfaces for modeling and simulating multibody systems with compliant contact elements', PhD Thesis, Instituto Superior Técnico, Universidade de Lisboa, Portugal.

Lopes, DS, Neptune, RR, Ambrósio, JA and Silva, MT (2015a), 'A superellipsoid-plane model for simulating foot-ground contact during human gait', *Computer Methods in Biomechanics and Biomedical Engineering*, pp. 1-10.

Lopes, DS, Neptune, RR, Gonçalves, AA, Ambrósio, JA and Silva, MT (2015b), 'Shape Analysis of the Femoral Head: A Comparative Study Between Spherical, (Super)Ellipsoidal, and (Super)Ovoidal Shapes', *Journal of Biomechanical Engineering*, vol 137, no. 11, pp. 114504-114504-8.

Lopes, DS, Silva, MT and Ambrósio, JA (2013), 'Tangent vectors to a 3-D surface normal: A geometric tool to find orthogonal vectors based on the Householder transformation', *Computer-Aided Design*, vol 45, no. 3, pp. 683-694.

Lopes, DS, Silva, MT, Ambrósio, JA and Flores, P (2010), 'A mathematical framework for contact detection between quadric and superquadric surfaces', *Multibody System Dynamics*, vol 24, no. 3, pp. 255-280.

Lu, G, Third, JR and Müller, CR (2012), 'Critical assessment of two approaches for evaluating contacts between super-quadric shaped particles in DEM simulations', *Chemical Engineering Science*, vol 78, pp. 226–235.

Lu, G, Third, JR and Müller, CR (2015), 'Discrete element models for non-spherical particle systems: From theoretical developments to applications', *Chemical Engineering Science*, vol 127, pp. 425-465.

Machado, MMF (2012), 'A multibody approach to the contact dynamics: a knee joint application', PhD Thesis, Escola de Engenharia, Universidade do Minho.

Macklin, M, Müller, M, Chentanez, N and Kim, T-Y (2014), 'Unified particle physics for real-time applications', *ACM Trans. Graph.*, vol 33, no. 4, pp. 153:1-153:12.

Mao, C, Qin, SF and Wright, D (2009), 'A sketch-based approach to human body modelling', *Computers & Graphics*, vol 33, no. 4, pp. 521-541.

Metta, G, Natale, L, Nori, F, Sandini, G, Vernon, D, Fadiga, L, Hofsten, CV, Rosander, K, Lopes, M, Santos-Victor, J, Bernardino, A and Montesano, L (2010), 'The iCub humanoid robot: An open-systems platform for research in cognitive development', *Neural Networks*, vol 23, no. 8-9, pp. 1125 - 1134.

Moustakas, K, Nikolakis, G, Koutsonanos, D, Tzovaras, D and Strintzis, MG (2005), 'Haptic feedback using an efficient superquadric based collision detection method', *Eurohaptics Conference, 2005 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2005. World Haptics 2005. First Joint*, IEEE.

Moustakas, K, Tzovaras, D and Strintzis, MG (2007), 'SQ-Map: Efficient Layered Collision Detection and Haptic Rendering', *IEEE Transactions on Visualization and Computer Graphics*, vol 13, no. 1, pp. 80-93.

Murray, RM, Sastry, SS and Zexiang, L (1994), *A Mathematical Introduction to Robotic Manipulation*, 1st edn, CRC Press, Inc., Boca Raton, FL, USA.

Nikravesh, P (1988), *Computer-Aided Analysis of Mechanical Systems*, Prentice Hall, Englewood Cliffs.

Nintendo Co., Ltd. (2008), *Super Smash Bros. Brawl*, <http://www.smashbros.com/wii/en_us/>.

Nvidia Corporation (2014), *PhysX*, <http://www.geforce.com/hardware/technology/physx>.

Ogarko, V & Luding, S (2012), 'A fast multilevel algorithm for contact detection of arbitrarily polydisperse objects', *Computer Physics Communications*, vol 183, no. 4, pp. 931-936.

Olsen, L, Samavati, FF, Sousa, MC and Jorge, JA (2009), 'Sketch-based modeling: A survey', *Computers & Graphics*, vol 33, no. 1, pp. 85-103.

Pan, J, Chitta, S and Manocha, D (2012), 'FCL: A general purpose library for collision and proximity queries', *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, IEEE, Saint Paul, MN.

Pazouki, A, Mazhar, H and Negrut, D (2012), 'Parallel collision detection of ellipsoids with applications in large scale multibody dynamics', *Mathematics and Computers in Simulation*, vol 82, no. 5, pp. 879-894.

Pfeiffer, F & Glocker, C (1996), *Multibody dynamics with unilateral constraints*, New York, John Wiley and Sons.

Quigley, M, Conley, K, Gerkey, BP, Faust, J, Foote, T, Leibs, J, Wheeler, R and Ng, AY (2009), 'ROS: an open-source Robot Operating System', *ICRA Workshop on Open Source Software*, IEEE.

Razzaq, S, Shah, AA and Bin Mansoor, S (2011), 'Real time collision detection in buildings using polygon presence grid', *Multitopic Conference (INMIC), 2011 IEEE 14th International*, IEEE, Karachi.

RobotCub (2014), *iCub sourceforge repository*, viewed August 2015, <http://sourceforge.net/projects/robotcub>.

Selle, A, Su, J, Irving, G and Fedkiw, R (2008), 'Robust High-Resolution Cloth Using Parallelism, History-Based Collisions, and Accurate Friction', *Visualization and Computer Graphics, IEEE Transactions on*, vol Volume 15, no. 2, pp. 339-350.

Shadow Robot Company (2013), *Dexterous Hand*, <http://www.shadowrobot.com/products/dexterous-hand/>.

SharpDevelopment (2014), *uPhysicsPro*, SharpDevelopment, <https://www.assetstore.unity3d.com/en/#!/content/17419>.

Shi, J & Koonjul, GS (2014), 'Real-Time Grasp Planning with Environment Constraints', *IROS 2014 Workshop: Real-time Motion Generation & Control*, Chicago.

SmashWiki (2013), *Mario Hitboxes*, viewed December 2014, <http://www.ssbwiki.com/Mario_%28SSBB%29/Hitboxes>.

SmashWiki (2014), *Hitbox*, viewed December 2014, <http://www.ssbwiki.com/Hitbox>.

Srivatsan, RA & Bandyopadhyay, S (2013), 'An Analytical Formulation for Finding the Proximity of Two Arbitrary Cylinders in Space', *Proceedings of Conference on Advances In Robotics*, ACM, Pune, India.

Steinbach, K, Kuffner, J, Asfour, T and Dillmann, R (2006), 'Efficient Collision and Self-Collision Detection for Humanoids Based on Sphere Trees Hierarchies', *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, IEEE, Genova.

Stein, E, Borst, R and Hughes, TJR (2004), *Encyclopedia of computational mechanics*, John Wiley.

Sucan, IA & Chitta, S (2015), *MoveIt!*, viewed 30 August 2015, <http://moveit.ros.org>.

Taeubig, H & Frese, U (2012), 'A New Library for Real-time Continuous Collision Detection', *Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on*, VDE, Munich.

Tang, TD (2014), 'Algorithms for collision detection and avoidance for five-axis NC machining: A state of the art review', *Computer-Aided Design*, vol 51, pp. 1-17.

Técnico Lisboa (2015), *ISR - iCub hand*, viewed 14 October 2015, <https://www.flickr.com/photos/tecnicolisboa/17136400735/>.

Tsuji, T, Uto, S, Harada, K, Kurazume, R, Hasegawa, T and Morooka, K (2014), 'Grasp planning for constricted parts of objects approximated with quadric surfaces', *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, IEEE, Chicago, IL.

TVTropes (2013), *Hitbox Dissonance*, viewed 6 October 2015, <http://tvtropes.org/pmwiki/pmwiki.php/Main/HitboxDissonance>.

TVTropes (2015), *There Are No Bedsheets*, viewed 9 October 2015, <http://tvtropes.org/pmwiki/pmwiki.php/Main/ThereAreNoBedsheets>.

Unity Technologies (2014), *Unity*, San Francisco, <http://unity3d.com/>.

Unity Technologies (2015), *Cloth - Unity Documentation*, viewed 9 October 2015, <http://docs.unity3d.com/Manual/class-Cloth.html>.

Unity Technologies (2015), *Height Tools*, viewed 11 October 2015, <http://docs.unity3d.com/Manual/terrain-Height.html>.

Valve Corporation (2012), *Counter-Strike: Global Offensive*.

Valve Corporation (2015), *Reanimated*, viewed 26 September 2015, <http://blog.counter-strike.net/index.php/2015/09/12492/>.

Valve Developer Community (2012), *Hitbox*, viewed December 2014, <https://developer.valvesoftware.com/wiki/Hitbox>.

van den Bergen, G (1997), 'Efficient collision detection of complex deformable models using AABB trees', *Journal of Graphics Tools*, vol 2, no. 4, pp. 1-13.

VisLab, ISR Lisboa (2014), *iCub MoveIt! URDF model*, viewed 30 August 2015, <https://github.com/vislab-tecnico-lisboa/icub-moveit>.

Wang, H (2014), 'Defending Continuous Collision Detection Against Errors', *ACM Trans. Graph.*, vol 33, no. 4, pp. 122:1-122:10.

Weisstein, EW (2015), *"Point-Plane Distance." From MathWorld--A Wolfram Web Resource.*, viewed 10 October 2015, <http://mathworld.wolfram.com/Point-PlaneDistance.html>.

Wellmann, C, Lillie, C and Wriggers, P (2008), 'A contact detection algorithm for superellipsoids based on the common-normal concept', *Engineering Computations*, vol 25, no. 5, pp. 432-442.

Wilson, AD, Izadi, S, Hilliges, O, Garcia-Mendoza, A and Kirk, D (2008), 'Bringing Physics to the Surface', *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*, ACM, Monterey, CA.

Xue, Z, Marius Zoellner, J and Dillmann, R (2007), 'Grasp planning: Find the contact points', *Robotics and Biomimetics, 2007. ROBIO 2007. IEEE International Conference on*, IEEE.

Xue, Z, Woerner, P, Zoellner, JM and Dillmann, R (2009), 'Efficient grasp planning using continuous collision detection', *Mechatronics and Automation, 2009. ICMA 2009. International Conference on*, IEEE, Changchun.

Yang, R & Wunsche, BC (2009), 'Automatic joint and skeleton computation for the animation of sketch-based 3D objects', *Image and Vision Computing New Zealand, 2009. IVCNZ '09. 24th International Conference*, IEEE, Wellington.

Yumer, ME & Kara, LB (2012), 'Co-abstraction of Shape Collections', *ACM Trans. Graph.*, vol 31, no. 6, pp. 166:1--166:11.

Zhang, X & Kim, YJ (2012), 'k-IOS: Intersection of spheres for efficient proximity query', *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, IEEE, Saint Paul, MN.

Zheng, Q, Li, FWB and Lau, RWH (2010), 'Sketching-Based Skeleton Generation', *Ubi-media Computing (U-Media), 2010 3rd IEEE International Conference on*, IEEE, Jinhua.

Appendix A. Pseudo-code of presented collision detection algorithms

Table 4 – Pseudo-code of the superellipsoid-plane minimum distance computation algorithm.

SuperellipsoidPlane_minimum_distance **(** Superellipsoid $\mathbf{s_i}$, Plane $\mathbf{s_j}$, Transforms $\mathbf{t_i}$, $\mathbf{t_j}$ **)**	
Pseudo-code	**Comments**
n ← $\mathbf{s_i}$.Global_To_Local(tj.Local_To_Global($\mathbf{s_j}$.normal))	*get plane normal on superellipsoid's coordinate system*
$\boldsymbol{\varphi_1, \varphi_2}$ ← $\mathbf{s_i}$.Get_Point_with_Normal(**n**)	*see Equation* (9)
Q ← $\mathbf{t_i}$.Local_to_Global($\mathbf{s_i}$.Get_Point($\boldsymbol{\varphi_1, \varphi_2}$))	*see Equation* (7)
P ← $\mathbf{t_j}$.Local_to_Global($\mathbf{s_j}$.Project_to_Plane(**P**))	*see Equation* (17)
return P, **Q**	

Table 5 – Pseudo-code for the minimum distance computation between two superovoids, using the implicit representation. The Transform inputs store information about the position and orientation of the superovoids, and the *initial_guess* input is optional. This algorithm is valid for superellipsoids as well.

SuperovoidSuperovoid_minimum_distance **(** Superovoids $\mathbf{s_i}$, $\mathbf{s_j}$, Transforms $\mathbf{t_i}$, $\mathbf{t_j}$, [6x1 vector **initial_guess**] **)**	
Pseudo-code	**Comments**
if no **initial_guess** was given	
q ← Get_initial_guess($\mathbf{s_i}$, $\mathbf{s_j}$, $\mathbf{t_i}$, $\mathbf{t_j}$)	*see section 3.4.1*
else	
q ← **initial_guess**	
q_final ← Solve_Newton_Raphson($\mathbf{s_i}$, $\mathbf{s_j}$, $\mathbf{t_i}$, $\mathbf{t_j}$, **q**)	
if newton-raphson did not converge **& initial_guess** was given	
q ← Get_initial_guess($\mathbf{s_i}$, $\mathbf{s_j}$, $\mathbf{t_i}$, $\mathbf{t_j}$)	*ignore given guess, get new one*
q_final ← Solve_Newton_Raphson($\mathbf{s_i}$, $\mathbf{s_j}$, $\mathbf{t_i}$, $\mathbf{t_j}$, **q**)	*and attempt to solve again*
P ← $\mathbf{t_i}$.Local_to_Global(**q_final**[1..3])	*q_final has the local coordinates*
Q ← $\mathbf{t_j}$.Local_to_Global(**q_final**[4..6])	*of the minimum distance solution*
impl ← $\mathbf{s_i}$.Implicit_Function($\mathbf{t_i}$.Global_to_Local(**Q**))	*test if Q, belonging to superovoid s_j*
colliding ← **true** if **impl** <= 0	*is also inside superovoid s_i*
return P, **Q**, **colliding**	

Table 6 – Pseudo-code for the numerical Newton-Raphson procedure, referenced in the superovoid-superovoid minimum distance algorithm (Table 5).

Solve_Newton_Raphson **(** Superovoids $\mathbf{s}_i$, $\mathbf{s}_j$, Transforms $\mathbf{t}_i$, $\mathbf{t}_j$, 6x1 vector $\mathbf{q}_0$ **)**	
Pseudo-code	**Comments**
$\mathbf{q}_k \leftarrow \mathbf{q}_0$	
$\mathbf{q}_{best} \leftarrow \mathbf{q}_0$	*save the best candidate found*
while $\lvert\Delta\mathbf{q}_k\rvert$ < tolerance & **k** < max_iterations	
$\mathbf{\Phi} \leftarrow \Phi(\mathbf{s}_i, \mathbf{s}_j, \mathbf{t}_i, \mathbf{t}_j, \mathbf{q}_k)$	*6x1 vector of constraints, Equation* (21)
$\mathbf{\Phi}_\mathbf{q} \leftarrow$ Jacobian($\mathbf{\Phi}$, $\mathbf{s}_i$, $\mathbf{s}_j$, $\mathbf{t}_i$, $\mathbf{t}_j$, $\mathbf{q}_k$)	*6x6 Jacobian matrix of* $\Phi(\dots)$
$\Delta\mathbf{q}_k \leftarrow$ Solve_Linear_System($\mathbf{\Phi}_\mathbf{q}\,\mathbf{x} = \mathbf{\Phi}$, solve for **x**)	*6x1 vectors, see Equation* (24)
$\mathbf{q}_k \leftarrow \mathbf{q}_k - \Delta\mathbf{q}_k$	
if all computations converged	*may not converge due to numerical instability, or poor starting iteration*
$\mathbf{q}_{best} \leftarrow \mathbf{q}_k$	
else	
break	
return $\mathbf{q}_{best}$	

Table 7 – Pseudo-code of the quadtree-based method to obtain an initial guess for the parametric minimum distance algorithm, as described in section 3.4.1 and shown visually in Figure 3.6.

Get_initial_guess_parametric **(** Superovoids $\mathbf{s}_i$, $\mathbf{s}_j$, Transforms $\mathbf{t}_i$, $\mathbf{t}_j$ **)**	
Pseudo-code	**Comments**
for N iterations	*around 6 is adequate*
$\mathbf{centers}_i \leftarrow$ Get_Quadtree_Centers($\mathbf{selected}_i$)	*lists of four 2x1 vectors, containing parametric coordinates of points inside the selected quadtree cells*
$\mathbf{centers}_j \leftarrow$ Get_Quadtree_Centers($\mathbf{selected}_j$)	
for each pair ($\mathbf{c}_i$, $\mathbf{c}_j$) of lists ($\mathbf{centers}_i$, $\mathbf{centers}_j$)	*total of 16 pairs*
$\mathbf{P} \leftarrow \mathbf{t}_i$.Local_to_Global($\mathbf{s}_i$.Get_Point($\mathbf{c}_i$))	*c_i and c_j of the form* (φ_1, φ_2) *(Equation* (12))
$\mathbf{Q} \leftarrow \mathbf{t}_j$.Local_to_Global($\mathbf{s}_j$.Get_Point($\mathbf{c}_j$))	*P and Q of the form* (x, y, z)
distance $\leftarrow \lvert\mathbf{P} - \mathbf{Q}\rvert_2$	
if distance is the lowest found in this iteration	
$\mathbf{selected}_i \leftarrow \mathbf{c}_i$	
$\mathbf{selected}_j \leftarrow \mathbf{c}_j$	
return [$\mathbf{c}_i$, $\mathbf{c}_j$]	*4x1 vector, see Equation* (23)

Appendix B. Duper Bowl Pong

The superellipsoid-plane collision detection algorithm presented in 3.2 was adapted to a 2D case featuring a superellipse and a finite line segment. The first phase of the algorithm is analogous to the superellipse-infinite line method, where the infinite line overlaps the finite line segment. When the minimum distance point lying on the infinite line is also inside the considered finite line segment, then these computed points are reported. However, if the point projected onto the infinite plane lies outside the line segment, then the calculated minimum distance points are incorrect. In this case, the two ends of the line segment are considered, and the inside-outside function of the superellipse (25) is computed for these two points. A collision is reported if any of these yield a negative result.

$$F_{SE-2D}(x,y) = \left|\frac{x}{a_1}\right|^{\frac{2}{\epsilon_1}} + \left|\frac{y}{a_2}\right|^{\frac{2}{\epsilon_1}} - 1 \tag{25}$$

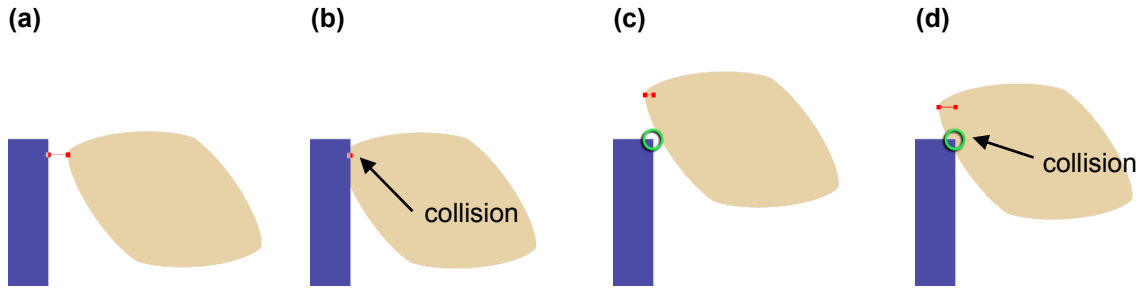


Figure 8.1 – Configurations for the superellipse-finite line segment collision detection algorithm. In (a, b), considering the blue finite line segment or an infinite line yields the same result for the minimum distance points shown in red. In (c, d), the minimum distance points do not lie on the line segment anymore, and the inside-outside function of the superellipse is computed on the corner point (highlighted in green).

This superellipse-line segment algorithm was applied in a simple video game named Duper Bowl Pong, produced in Unity3D, and based on the old classic arcade game Pong. In this game, two players engage in a tennis-like match, in which the objective is to bounce a ball into the other player's court, while not allowing the ball to pass through the backside line on the player's own court. Each player controls a rectangular paddle in a vertical fixed path. Unlike its inspirational predecessor, the developed game features a superelliptical ball, instead of a round/squared one, which changes shape and eccentricity as the game progresses. The game is designed to be played with a multi-touch-screen interface, and the player can move and/or rotate his or her paddle to hit the ball at an angle, using one or two fingers, respectively. The second player is controlled by the computer. Screenshots of the game are shown in Figure 8.2.

Duper Bowl Pong has been published on the Amazon (http://amzn.com/B00X1JUII6) and Aptoide (http://kyuu840.store.aptoide.com/app/market/com.newbark.sellypong/1/9170520/Duper+Bowl+Pong) Android app stores, from where it was downloaded 10 times. It was presented at the Microsoft Games Pizza Night event on May 8th 2015, and at the Montra de Jogos do IST on May 27th and 28th 2015, where

36 people, of which 4 teachers, played the game. Feedback was mostly positive: while some people reported using 2 fingers to control the paddle's rotation was awkward, the novelty of playing with a superelliptical ball was well received. The author plans to release updates to Duper Bowl Pong, improving the rotation controls and the physics of the collision reaction. A more active promotion of the game will also be considered.

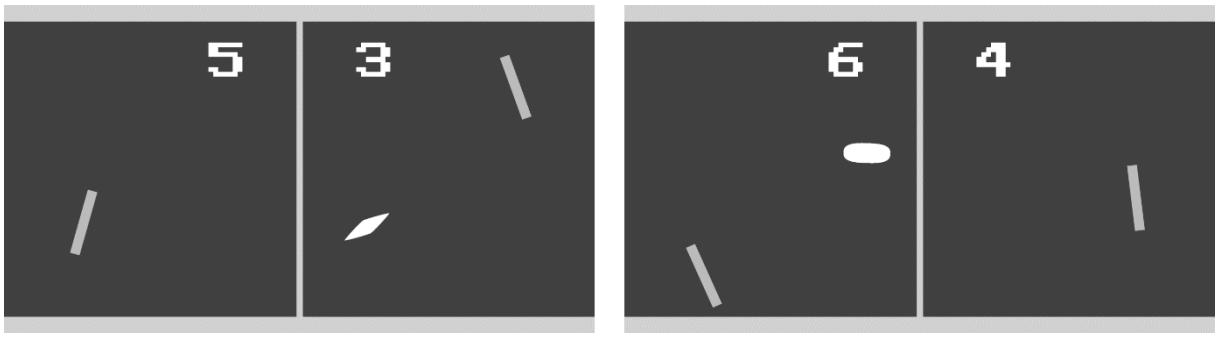


Figure 8.2 – Two screenshots of Duper Bowl Pong, running on an Android smartphone. The player controls the left-side paddle, while the computer guides the right-side one. Notice the superelliptical shape of the ball, which gradually changes as the game progresses, and the angle at which the players' paddles are placed to hit it.