Fast Contact Evolution for Piecewise Smooth Surfaces

by

Paul G. Kry

B.Math., University of Waterloo, 1997

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming
to the required standard

The University of British Columbia

August 2000

Abstract

Dynamics simulation of smooth bodies in contact is a critical problem in physically based animation and interactive virtual environments. We describe a technique which uses reduced coordinates to evolve a single continuous contact between Loop subdivision surfaces. The incorporation of both slip and no-slip friction into our algorithm is straightforward. The dynamics equations, though slightly more complex due to the reduced coordinate formulation, can be integrated easily using explicit integrators without the need for constraint stabilization. The use of reduced coordinates also confines integration errors to lie within the constraint manifold which is preferable for visualization.

Our algorithm is suitable for piecewise parametric or parameterizable surfaces with polygonal domain boundaries. Because a contact will not always remain in the same patch, we demonstrate how a contact can be evolved across patch boundaries. We also address the issue of non-regular parameterizations occurring in Loop subdivision surfaces through surface replacement with n sided S-patch surfaces.

Three simulations show our results. We partially verify our technique first with a frictionless system and then with a blob sliding and rolling inside a bowl. Our third simulation shows that our formulation correctly predicts the spin reversal of a rattleback top. We also present timings of the various components of the algorithm.

Contents

List of Figures

Acknowledgements

Many thanks go to my supervisor, Dinesh K. Pai, as this thesis would not have been possible without his guidance. I would also like to thank Uri Ascher for his invaluable comments and suggestions. Lastly, to all the others who are too numerous to name here, I am very grateful. This thesis was supported in part by scholarships from NSERC and BC-ASI.

Paul G. Kry

The University of British Columbia
August 2000

Chapter 1

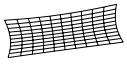
Introduction

Within the graphics community, interest in physical simulation of rigid bodies probably reached a peak sometime in the last ten years. Although some might consider it to be a solved problem, we feel that there are still important challenges to be met.

One of the difficult aspects of simulating rigid bodies is the case where there is contact. In this case the motion of the bodies must be constrained so that they do not interpenetrate. The combination of smooth surfaces, friction and multiple contacts can make this a very difficult problem.

Figure 1.1: Example of dynamic simulation using our algorithm. The objects are bounded by smooth Loop subdivision surfaces.

This thesis is about the dynamics of piecewise smooth surfaces in single continuous contact. We present a method of computing how a system of two rigid bodies evolves when their smooth boundaries are in contact at a single point. We call this problem the contact evolution problem. Figure 1 shows



an example of a smooth globe-like object rolling and sliding inside a bowl, simulated with our system.

We formulate the problem as an ordinary differential equation in the coordinates of the contact patches. By parameterizing the system in reduced coordinates, violation of the contact constraint is prevented as we integrate. Any errors that accumulate during integration instead lie entirely within the constraint manifold. The incorporation of dynamic Coulomb friction into our method is straightforward. Additionally, we can easily reformulate the equations to include no-slip and no-spin friction. By monitoring the constraint force we can accommodate changes in friction and object separation.

This is an attractive approach to the problem because it avoids the drawbacks of traditional methods which spend most of their time in collision detection and constraint computation. It is important to recognize that although collision detection is still necessary to identify new contacts, it can be performed at a much less frequent rate because it is not necessary for our single continuous contact evolution technique. Other techniques rely on collision detection or minimum distance computation at each time step to track the local contact points or for constraint stabilization.

Simulating realistic models constructed with subdivision surfaces or free-form parametric surfaces such as NURBS is important because of the popularity of these surface representations for modelling and design. These types of piecewise parametric and parameterizable surfaces are ideal for our single contact evolution technique. With respect to piecewise parametric surfaces, an important contribution of this thesis is a technique to evolve a contact across a patch boundary. The surface description of any object will almost always consist of a collection of several patches.

Part of the popularity of subdivision surfaces is attributable to the relative ease with which they can be edited and implemented. Unfortunately, this facility does not carry over to contact evolution. Almost every object defined by a subdi-

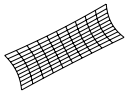
vision surface will have multiple patches that are non-regular (in the natural parameterization). In this thesis we focus on the use of Loop subdivision surfaces. For contact evolution to work we must avoid evolving the contact near non-regularities. We achieve this by replacing portions of the Loop subdivision surface with n sided S-patches which are more complicated but better behaved. The replacement surfaces we construct maintain tangent plane continuity at their boundary with the original subdivision surface. As a result their shape often matches quite closely that of the non-regular surface.

1.1 Chapter Summary

We provide an account of related work in Chapter 2. The related work generally falls into the three categories: dynamics simulation, contact kinematics, and collision detection.

Chapter 3 provides additional background and motivation. We start with a discussion on the representation of state. Sections 3.2 and 3.3 introduce the concept of a constraint and briefly discuss solving constrained systems. This leads into Section 3.4 where we motivate the use of reduced dimension parameterizations for Constrained Systems.

Chapter 4 consists of a detailed derivation of our algorithm. After some preliminaries, contact kinematics equations are derived in Section 4.2 and then incorporated as a constraint into the Newton-Euler equations of motion in Section 4.3. With the foundation of our technique explained in the first part of this chapter, we continue by demonstrating how to add Coulomb friction to the system in Section 4.4. We also present some ideas for implementing no-slip and no-spin friction in our framework as well as ideas on switching between different types of friction. Section 4.5 contains an explanation of how a contact traverses the boundary of a patch into an adjacent patch. Traversing boundaries is important as most models consist of a collection of parametric surfaces. Lastly, we conclude this chapter with



a summary of the algorithm.

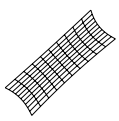
In Chapter 5 we examine the problems involved with using Loop subdivision surfaces with our algorithm. After a brief explanation of Loop subdivision surfaces, we discuss the standard method for evaluating Loop surface patches and the implications of a non-regular parameterization on our contact evolution algorithm. In Section 5.2 we introduce the idea of replacing parts of the surface with S-patches. This section provides a description of S-patches and explains how to set S-patch control points to maintain surface continuity via blossomed Loop patch evaluations. In Section 5.3 we provide additional detail on how we compute derivatives and blossoms. Since derivatives of S-patch functions are quite complicated we write this code with inspiration from automatic differentiation techniques. We wrap up this chapter with a brief overview of our Java implementation and its features.

Chapter 6 shows the results of our algorithm. As part of the evaluation of our algorithm, we measure how long different portions of the calculation take in Section 6.1. The timings show that a large proportion of the computation time is spent in surface evaluations and surface derivative evaluations which suggests that our algorithm is best suited for surfaces which are easy to evaluate. In Section 6.2 we partially validate our implementation with a Loop subdivision surface on a flat and frictionless Bézier patch. Section 6.3 demonstrates two Loop subdivision surfaces in contact with dynamic friction. Section 6.4 finishes this chapter by showing that our implementation correctly predicts the spin-reversing behaviour of a rattleback top.

Chapter 7 provides our conclusions and summarizes the impact of our algorithm. We also describe possible future work.

Appendix A describes an alternate and more useful derivation of the contact kinematics equations. This derivation starts by only considering half of the contact and takes the least complicated expressions for describing this half of the equations. The other half of the equations are very similar and only require transformation into the same coordinates as the first half.

In Appendix B we provide some sample files and a quick description of other data files needed for our implementation. A rattleback scene description file is provided in Section B.1 and a object file for a Loop subdivision surface is provided in Section B.2.



Chapter 2

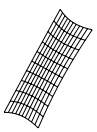
Related Work

In recent years there has been a good amount of work concerning simulation of contact. Being such a broad topic, the emphasis falls into several different areas. Papers coming from the robotics community tend to focus on the constraints and numerical methods while papers coming from the graphics community tend to focus on collision detection. We break the following short literature survey into three sections to help shed light on which areas our algorithm falls.

2.1 Dynamics Simulation

Multibody dynamics has been a bountiful area of research for many people over several decades. Numerous books have been written on this topic such as [15, 43, 38] to give an example of just a few. It is important for us to mention this work because our research has been largely influenced by these multibody algorithms. In particular, our algorithm is inspired in part by the analysis of rigid body methods by Ascher, Pai and Cloutier appearing in [4]. Note that reduced coordinate formulations such as the one we use in this thesis are well studied (for example, see [40] by Rabier and Rheinboldt). In the rest of this section we provide an account of recent algorithms and implementations relating to dynamics simulation.

Newton, [11], as described in Cremer's PhD thesis is a system architecture for



general purpose physical system simulation. His modular system incorporates collision detection with a variety of dynamics formulations consisting of open and closed loop kinematic chains with a variety of inter-object constraints. The simulation is based on events such as collisions and changes in contact and friction.

Lubich et al. present MEXX in [28]. MEXX is software for integration of constrained mechanical multi body systems. Its focus is on the numerical method. It is based on a method which is explicit in the differential equations and linearly implicit in the nonlinear constraints. It has the ability to detect events such as impacts, however, unlike Cremer's system it is the user's responsibility to reformulate their equations and constraints to deal with such events.

Sauer and Schömer in [46] describe GALILEO, a system for rigid body simulation with emphasis on the simulation of unilateral constraints for virtual reality applications. They formulate the problem of n polyhedral or curved objects (their examples are all portions of spheres or cylinders) with K contacts with friction in a single linear complementarity problem (they use a polyhedral approximation of a friction cone). They demonstrate their friction modeling though the simulation of a tippe-top.

Stewart and Trinkle in [53] present an interesting time-stepping method for rigid body simulation. Being based on impulse-momentum equations it does not need to explicitly resolve impulsive forces. It also does not require explicit collision detection and can handle simultaneous impacts. Since they use an impulse based method rather than a force based method they can set up their problem as a linear complementarity problem. Because of their formulation, it is not necessary to determine a time of impact (instead they maintain a set of constraints which are in danger of being violated). They use a polyhedral approximation of a friction cone.

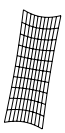
Hahn in [19] is an example of earlier work on rigid body simulation with polyhedral models. His work focuses on integrating physically based simulation with objects which are animated by hand. Although his system uses geometric

constraints as well as more general non-holonomic constraints, Hahn achieves non-interpenetration by modelling contact as a series of frequently occurring collisions.

Baraff's [5] is another example of some earlier work. The focus of this paper appears to be the formulation of resting contact constraints. He uses a linear complementarity problem to determine when contacts are breaking. This paper does not address friction and the the examples given are planar consisting of polygons.

Baraff, in [6], also looks at rigid body simulation with curved surfaces. Although the paper focuses on implicit surfaces a method of using parametric surfaces is described in one of the appendices. Central to the method is the tracking of all pairs of extremal points, that is the points where maximum penetration occurs between the surfaces. To solve multiple contacts they formulate the unilateral constraints as a linear complementarity problem (LCP). Here Lemke's algorithm for solving linear complementarity problems is preferred to using a heuristic solution to the positive semi definite (PSD) linear programming problem. Baraff points out, however, that the problem is non-PSD in the presence of friction and suggests a heuristic method would be useful in this case (which he presents in [8]). The paper also discusses a fast method for detecting collisions using cached "witness" separating planes. He also discuses a method for quickly and accurately determining the collision time as the algorithm requires this for backing up to the times at which collisions occur.

In [8], Baraff shows how to compute contact forces with friction for rigid bodies. His approach in this paper is to generalize the problem as little as possible (he does not formulate an LCP as before). This has the advantage of avoiding code for solving general optimization problems which can be overly complicated for the friction problem. The result is a much faster and simpler algorithm. Baraff admits that although they can not prove that the algorithm terminates they have not experienced this problem from any of their tests. They implemented an interactive planar algorithm, and an off line system for the three dimensional version of the



algorithm.

Anitescu, Cremer and Potra's [1] provides an example of more recent work in rigid body dynamics. This work focuses on the formulation of contact problems with friction as a linear complementarity problem. The friction cone is approximated by a polyhedron. They consider smooth shaped bodies and derive the contact kinematics equations for surface-surface contacts, curve-surface contacts, and curve-curve contacts. They also investigate conforming planar contacts with arbitrary boundaries. They give the example of a cylindrical object on a table with external forces to demonstrate the transition from the conforming planar contact to a curve-surface contact. Although the Newton-Euler formulation they use is in global coordinates, instead of using an optimization algorithm for determining the new contact points they use the contact kinematics equations. They do not address the problem of changing patches as the contact moves since they assume that all calculations are done within a single patch on each body.

Mirtich and Canny take an interesting approach to rigid body dynamics simulation in [29]. All interactions between the bodies, from colliding contact to all varieties of continuous contact are treated as collisions. Their method has the advantage of simplicity as a wide variety of systems can be simulated, including those that are difficult to simulate with constraint based methods. The disadvantage however is that it relies greatly on the collision detection algorithm to provide information about all collisions with accuracy. One of the many examples they use to test the algorithm is a rattleback top.

2.2 Contact Kinematics

Smooth surfaces in contact have been extensively studied in both graphics and robotics. We've already mentioned Anitescu's work with the equations of contact, but there are several others which should be mentioned. Many of the earlier papers derive general contact kinematic equations but then demonstrate them with simple

surfaces such as spheres.

Montana derived contact kinematics equations for orthogonal nets in [30] and showed how they could be used for fine grip adjustment and other robotics applications. The beauty of Montana's derivation is that if the contacting surfaces are described by orthogonal networks then the kinematics relationship can be described with matrices from differential geometry. See also [33, 10, 25, 22] for alternate derivations and applications.

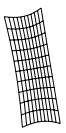
Goyal derives various constraints in [17] for the purpose of smooth surface simulation. The equations in this paper were derived to allow smooth surface interaction for Cremer's Newton dynamic simulator. The provided example interactions are limited to relatively simple objects such as a cylinder rolling on a block without slipping and a torus rolling on a plane.

An alternate version of the contact kinematics equations is derived by Nelson Johnson and Cohen in [35]. They use a separation distance as one of their contact coordinates allowing them to easily measure penetration distance during contact evolution. In this paper the penetration distance is used to compute a contact force in a haptic simulation of NURBS surfaces in contact.

2.3 Smooth Surface Contact, Collision and Distance

Hégron computes rolling motion of a convex object on a parametric surface using a prediction-correction schema in [20]. He demonstrates a sphere rolling without sliding on a collection of bi-cubic Bézier patches. The sphere is placed in contact with the surface by lowering it vertically using an iterative method. They assume that the parametric surface patch grid has C^1 continuity, and that the rolling rigid object is convex. A single point of contact is maintained during the animation.

Snyder examined the problem of placing smooth surfaces in contact in [49]. His algorithm allows the user to place smooth objects in stable non-interpenetrating configurations given a set of external forces. Shapes can be non-convex and place-



ment of complicated shapes can occur at interactive rates. The key is that the objects do not behave as physics would dictate. The example of placing a spoon in a bowl shows how this is useful for object placement. Trying to place a spoon in a bowl using a physically based animation algorithm may be difficult as the spoon may bounce out before the system comes to rest.

In an earlier paper, [50], Snyder and others at the California Institute of Technology investigated animating smooth surfaces. Their focus was on collision detection. Surfaces can be parametric or implicit and the shapes can vary over time. One of the key features of their solution is the generation of a set of uniformly distributed points on the interface of a conforming contact between smooth surfaces. Their method although slow is very powerful. The paper focuses solely on collision detection and does not address how to handle the collision response and contact evolution problems.

Johnson and Cohen, in [23], present a framework for minimum distance computations between parametric surfaces. Their approach is based on a lower-upper bound tree which allows them to quickly prune bounding volumes which are not involved in the minimum distance. They also show that their solution converges relatively quickly which is useful for time-critical applications.

In Nelson and Cohen's [34], constraints in Cartesian coordinates are used for interaction with smooth surfaced mechanical assemblies. They report that their approach has a compact implementation due to the re-use of constraints. They demonstrate their system for several small assemblies including a Stewart platform.

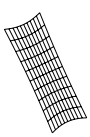
Chapter 3

Additional Background

This chapter provides additional background on rigid body dynamics and simulation issues in order to help motivate our work. We will show why we feel a solution to the single continuous contact problem is necessary. Additionally, the discussion should help give insight into the difficulty inherent in this simulation problem and related ones.

An important focus of our discussion will be on the choices for parameterizing the state of a system. This has a large impact on formulating dynamics equations. We will also discuss different types of constraints, and methods for solving the constrained equations of motion.

This extra motivation is helpful because our algorithm is not easily comparable to other existing techniques. This is because in focusing on single contact we avoid issues which are often emphasized in other algorithms. For example, friction is a well studied problem and is one that becomes very difficult when there are multiple contacts between two bodies (Baraff has shown that solving multiple contacts with friction is NP hard in [7]).



3.1 State Representation

Rigid body simulations require a method for parameterizing the possible positions and motions of bodies. An object free to move in space has six degrees of freedom. Three of these degrees of freedom are translational, while the other three are rotational.

Euler angles, for example, can be used to parameterize the rotation, although not without singularity problems. A higher dimensional parameterization is often used as it can be more convenient. The rotational component may be represented by a 3×3 rotation matrix giving us a parameter space of dimension 12. But only matrices which are orthogonal and have determinant equal to 1 are rotation matrices so we must be careful that our state vector preserves this property, despite any changes we make.

Alternatively unit length quaternions can be used to represent a rotation. This gives a total dimension of 7. Again, we must be careful to preserve the unit length of a quaternion, which happens to be a fair bit easier. If we make a change to the state which violates the condition by a small amount it is easy to project the state back onto the space of valid states as we can simply normalize the quaternion. See [55] for more on rigid body simulation and the issues of representing the state of the system.

3.2 Constraints

A constraint is an expression which describes a subset of states satisfying a certain property. It is often written in the form $f(x) = 0$ or $f(x) \geq 0$, where x is the state of the system. When the expression is an equality the constraint is bilateral, such as for a revolute joint. If the expression is an inequality then the constraint is unilateral, such as a non-interpenetration constraint.

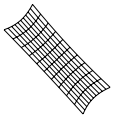
Holonomic constraints are essentially position constraints, or they are con-

straints that can be integrated to position constraints. Examples of this are revolute joints and some kinds of non-interpenetration constraints. A non-holonomic constraint, however, is one that is expressed as a velocity or acceleration constraint, and can not be integrated to get a position constraint. An example of this is a ball rolling without slipping on a table. The ball can achieve any position on the table with any rotation by some sequence of rolling and spinning. For more on constraints in rigid body dynamics see [24].

3.3 Solving Constrained Systems

Combining the Newton-Euler equations of motion with an algebraic constraint gives a differential algebraic equation (DAE). Such equations are not always easy to solve (see [2]). The most common approach is to differentiate the constraints so that they are stated in terms of accelerations. The differentiated constraints provide a system which can be solved as an ordinary differential equation. We must, however, take care to ensure the solution lies on the constraint manifold. This can be done with Baumgarte stabilization [9]. This stabilization technique acts like a spring damper system connecting our solution to the constraint manifold. As this type of stabilization has some undesirable properties (see [3]) there may be a desire to use more expensive and more mathematically sound techniques.

Sometimes it is easier to give a position constraint as a velocity constraint. This is the case for parametric surfaces in contact. The position constraint is essentially a distance function which lacks a simple closed form. The velocity constraint, on the other hand, is much easier to write using the normals of the contacting points (or the extremal points in the case of interpenetration).



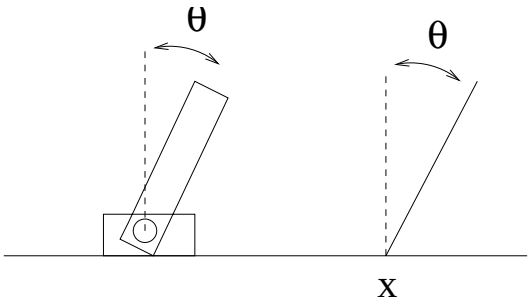


Figure 3.1: A robot arm with one degree of freedom is depicted on the left while a rod in contact with the plane having two degrees of freedom while in contact is depicted on the right. These are bilateral and unilateral constraints, respectively, for which we can express the configuration of the system more concisely with reduced coordinates.

3.4 Reduced Coordinates for Constrained Systems

It can be advantageous to choose a set of reduced dimension coordinates for representing the state of a constrained systems. A robot arm with one degree of freedom can be described by its joint angle rather than parameterizing its position as if it were a free body. Although the reduced coordinate dynamics equations are generally more expensive to evaluate, they do not require stabilization when they are integrated if the reduced coordinates exactly describe all possible configurations of the system. The robot arm with its bilateral constraint is an example of this situation. We might imagine a rod on a plane in a two dimensional environment as another example but with a unilateral constraint. Figure 3.1 shows a conceptual drawing of both these cases. For the robot arm case we only need to keep track of the angle. In the case of the rod, while it is in continuous contact with the surface we only need to keep track of the angle and the x position of the contact.

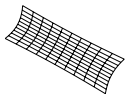
It is useful to consider the full implications of reduced coordinates for parametric surface non-interpenetration constraints.

A non-interpenetration constraint is unilateral. It is only active when there

is contact and it can be complicated as it depends on the contacting geometry. Multiple contacts mean multiple active constraints. If we use higher dimensional coordinates then although we do not need to change our state representation when the constraints become active, we do need to stabilize these constraints to ensure that our solution remains on the constraint manifold. The alternative of reduced coordinates unfortunately does not extend well to multiple contacts.

The constraint (or the constraint gradient in the case of some methods of solving this problem) may not be easy to compute for certain configurations of the given geometry. There will be a best way to handle each of these special contact configurations. For example, a screw in a threaded hole would best be treated as a system with one degree of freedom. A peg in a hole would be best treated as a two degree of freedom system. A cylinder on a table has three degrees of freedom until it is tipped up on edge in which case it has five. The problem of transitions between different formulations, however, is not trivial even for this relatively simple example (see [1]).

Trying to use reduced coordinates for systems with a reduced number of degrees of freedom is not always possible. The system may have multiple states for a given coordinate location, or singularities where there are reduced degrees of freedom. For example consider a five bar mechanism. This mechanism is a linkage of four bars (the fifth bar is actually the base which connects the two ends of the linkage). We might try to use the planar location of the end effector to describe the state of the system. An arbitrary position can be achieved with the left "elbow" bent in or out and the right elbow can be in or out. This can be seen in Figure 3.2. There are other positions which can not be achieved at all due to the limited length of the arm. Additionally, when one of the arms is straight there is one less degree of freedom at the end effector. We might consider using two angles to parameterize the system, but we do not really avoid any of our problems. There are still configurations which can not be reached and there are multiple possible



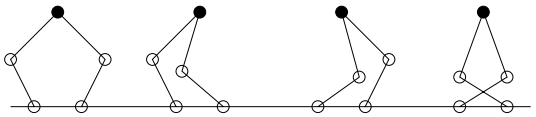


Figure 3.2: Four possible configurations of a five bar mechanism created by having the left and right "elbows" in or out. The planar position of the dark circle is the configuration space. It is inappropriate to use a two dimensional state representation for this system even although it really only has two degrees of freedom.

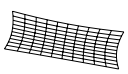
end effector positions for many settings of the two angles. It appears that a two dimensional state representation for the system is inappropriate.

This seems to be fairly strong motivation not to use reduced coordinates, at least for this scenario. Let us go back to the smooth surface contact constraint as this is our primary interest.

If there is only a single point of contact between two bodies, then the contact can be parameterized as a five degree of freedom joint (this is explained fully in Section 4.2.1). In this case the reduced coordinates uniquely describe a configuration. The additional advantage of these coordinates is that their dimension matches the number of degrees of freedom exactly, and the entire space parameterizes valid configurations. As a result, we can use these coordinates without having to worry about stabilization. Note that we will still accumulate errors in integrating the system but these errors will be inside the constraint manifold. These types of errors are acceptable because they are far less noticeable than interpenetration or separation errors.

Global changes such as new constraints becoming active can be detected at a slower rate than the rate at which we evolve contacts. This is possibly the most important feature of our technique as collision detection is often the most computationally expensive component of physically based simulations. Collision detection can be minimized or even omitted if we know that the object shapes do not allow multiple contacts.

Additionally, single contact can be useful in a variety of situations, such as haptics, [35], and certain manipulation examples, [22]. Fast contact evolution is interesting for real time interaction with smooth objects where contact evolution information is used for sound synthesis, [13, 14].



Chapter 4

Formulation

4.1 Preliminaries

In this thesis we use a notation for spatial dynamics which is similar to [15, 44, 21, 30, 32].

The motion of a rigid body $i = 1, ..., n$ is described using a reference frame labeled i attached to the body. Unless otherwise noted, we will let this body reference frame be located at the center of mass and with its axes aligned with the principle axes of rotation.

The homogeneous coordinates of frame i with respect to another frame j is given by the 4×4 matrix ${}^{j}_{i}\mathsf{E}$. We will always use leading subscripts and superscripts to indicate frames. The homogeneous coordinates of a three dimensional vector x in frame i are denoted ${}^{i}x$. The homogeneous coordinates of this vector in frame j are given by left multiplying by ${}^{j}_{i}\mathsf{E}$.

The spatial velocity $\phi(j, i)$ describes the relative motion of frame i with respect to frame j. In (non-homogeneous) coordinates of frame i it is given by the 6×1 matrix ${}^{i}\phi(j, i) = ({}^{i}\omega^T, {}^{i}v^T)^T$, where ω is the angular velocity and v is the linear velocity of the point at the origin of frame i. Spatial forces, called wrenches, are represented as $f = (f_r^T, f_t^T)^T$ where f_r is the (rotational) torque and f_t is the (translational) force.

It is useful to note that the spatial velocity vector parameterizes a twist that can be written in homogeneous coordinates as the 4×4 matrix,

$$\mathcal{O}\phi = \begin{pmatrix} [\omega] & v \\ 0 & 0 \end{pmatrix}. \tag{4.1}$$

Here, $[\omega]$ denotes the skew symmetric 3×3 matrix equivalent to the cross product $\omega\times$, i.e.,

$$[\omega] = \begin{pmatrix} 0 & -\omega^z & \omega^y \\ \omega^z & 0 & -\omega^x \\ -\omega^y & \omega^x & 0 \end{pmatrix}, \tag{4.2}$$

and $\mathcal{O}$ denotes the linear expansion operation which converts the 6×1 matrix into a 4×4 matrix representation. We will let $\mathcal{V}$ denote the linear operation which performs extraction of the 6 parameters, thus,

$$\phi = \begin{pmatrix} \omega \\ v \end{pmatrix} = \mathcal{V} \begin{pmatrix} [\omega] & v \\ 0 & 0 \end{pmatrix}. \tag{4.3}$$

If $\mathcal{V}$ is applied to a matrix whose upper 3×3 matrix is not skew symmetric then we can assume that $\mathcal{V}$ acts as a projection onto $so3$.

The time derivative of the change of coordinates from frame i to frame j is a twist when written in coordinates of frame i, i.e.,

$${}^i\phi(j,i) = \mathcal{V}({}^i_j\mathsf{E}\,{}^j_i\dot{\mathsf{E}}). \tag{4.4}$$

Spatial velocities and spatial wrenches transform according to the Adjoint transformation ${}^j_i\mathsf{Ad}$. Spatial velocities being contravariant quantities transform by left multiplying, ${}^j\phi = {}^j_i\mathsf{Ad}\ {}^i\phi$. Because we write spatial wrenches as column vectors, these covariant quantities are transformed by left multiplying with the inverse transpose, ${}^jf = {}^i_j\mathsf{Ad}^T\ {}^if$. The 6×6 Adjoint matrix is defined as,

$${}^j_i\mathsf{Ad} = \begin{pmatrix} \Theta & 0 \\ [p]\Theta & \Theta \end{pmatrix}, \quad \text{where} \quad {}^j_i\mathsf{E} = \begin{pmatrix} \Theta & p \\ 0 & 1 \end{pmatrix}.$$

Here Θ is a 3×3 rotation matrix and p is a 3×1 displacement.

Lastly, we define the spatial cross product of $\phi = \begin{pmatrix} \omega^T, & \mathsf{v}^T \end{pmatrix}^T$ as the linear operator with coordinate matrix

$$[\phi] \quad = \quad \begin{pmatrix} [\omega] & 0 \\ [v] & [\omega] \end{pmatrix}.$$

4.2 Contact Kinematics

We consider rigid bodies whose boundaries are defined by a collection of parametric surface patches. Although the surface patches will typically be polynomial or rational polynomial functions this does not necessarily need to be the case. We only require the ability to evaluate the function and its derivatives at a given point.

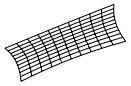
A single point of contact between two bodies will involve one patch from each body. We can ignore, for now, the fact that adjacent patches will overlap along their boundary curves as it is the location of the contact that we want to describe and if it lies on a boundary then we can choose either of the patches sharing the boundary to describe the contact location.

4.2.1 Contact Coordinates

Suppose body 1 and body 2 are in contact. Let the shape of the contacting patches be described by the functions ${}^{1}\mathsf{c} : (s,t) \to \mathbb{R}^3$ and ${}^{2}\mathsf{d} : (u,v) \to \mathbb{R}^3$. Note that the functions have a preceding superscript denoting that they define the surface shape in the coordinates of their respective body's reference frame. We drop this superscript when the coordinate frame is clear from context.

We use the notation $c_{,s} \stackrel{\text{def}}{=} \frac{\partial c}{\partial s}$. The orthonormal contact frame with coordinates (s,t) can be defined in frame *1*, the body frame, as the coordinate frame with origin at $c(s,t)$ and coordinate axes

$$\mathsf{x} = \frac{\mathsf{c}_{,s}}{\|\mathsf{c}_{,s}\|}, \quad \mathsf{y} = \frac{(\mathsf{c}_{,s} \cdot \mathsf{c}_{,s})\mathsf{c}_{,t} - (\mathsf{c}_{,s} \cdot \mathsf{c}_{,t})\mathsf{c}_{,s}}{\|(\mathsf{c}_{,s} \cdot \mathsf{c}_{,s})\mathsf{c}_{,t} - (\mathsf{c}_{,s} \cdot \mathsf{c}_{,t})\mathsf{c}_{,s}\|}, \quad \mathsf{z} = \frac{\mathsf{c}_{,s} \times \mathsf{c}_{,t}}{\|\mathsf{c}_{,s} \times \mathsf{c}_{,t}\|}. \tag{4.5}$$



Coordinates are in frame *1* (superscripts were omitted). The vectors x, y and z always form orthonormal axes, and will always exist if we require our parametric surface to be regular (that is $\mathsf{c}_{,s}(s,t) \neq 0$ and $\mathsf{c}_{,t}(s,t) \neq 0$ for all (s,t) in the domain). Also, y is orthogonal to z since it is a linear combination of $\mathsf{c}_{,s}$ and $\mathsf{c}_{,t}$. It is also orthogonal to x since its inner product with $\mathsf{c}_{,s}$ is equal to zero. Thus the homogeneous coordinates of this contact frame with respect to frame *1* are given by

$$\overset{1}{1c}\mathsf{E} = \begin{pmatrix} \overset{1}{}\mathsf{x} & \overset{1}{}\mathsf{y} & \overset{1}{}\mathsf{z} & \overset{1}{}\mathsf{c} \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4.6}$$

The matrix in equation 4.6 is a function of s and t. The contact frame on body 2 is defined in the same way yielding the matrix $\overset{2}{2c}\mathsf{E}$ which is a function of u and v in the domain of the patch $\mathsf{d}(u,v)$. Note that the c in the frame label *2c* stands for *contact* as opposed to representing patch c of body 1.

Because the contact frames are orthonormal and occur at the same location in space, they can be easily aligned by a rotation matrix (see Figure 4.1).

$$\overset{1c}{2c}\mathsf{E} = \begin{pmatrix} \cos\psi & -\sin\psi & 0 & 0 \\ -\sin\psi & -\cos\psi & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \overset{\text{def}}{=} \begin{pmatrix} R_\psi & 0 \\ 0 & 1 \end{pmatrix} \tag{4.7}$$

The matrix $\overset{1c}{2c}\mathsf{E}$ is idempotent and R_ψ is its upper 3×3 rotation matrix.

We can now describe any contact configuration between patches c and d of bodies 1 and 2 by the 2-dimensional location of the contact in the domain of each patch along with the angle of rotation ψ. Assembled in a column vector, we call these 5 independent variables the *contact coordinates* and we denote it q; i.e.,

$$q = \begin{pmatrix} s & t & u & v & \psi \end{pmatrix}^T. \tag{4.8}$$

4.2.2 Multiple Contacts

When there is more than one point of contact between two bodies the situation becomes more complex. When there are two contacts the number of degrees of

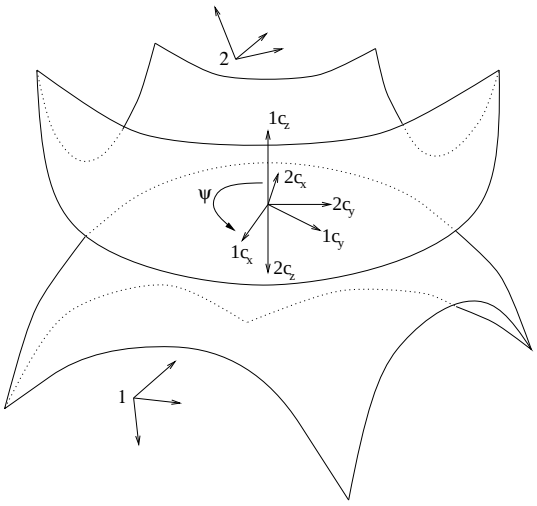
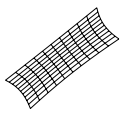


Figure 4.1: Contact Coordinates

freedom is reduced in general to four. We run into problems, however, if we try to represent our two contact configurations as a four dimensional generalized joint. Suppose we use the parametric location of one of the contacts on both surfaces as our contact coordinates. Valid unique configurations in the state space are those where there is only one rotation of the objects about each other at the specified point of contact for which the objects establish a second point of contact. If there are additional rotations which yield a second point of contact then we have the problem of not knowing which configuration we have from the contact coordinates alone. For some special cases of contact there can be in fact five degrees of freedom. An example of this is a sphere wedged between two flat parallel planes. In this case all rotations are possible giving us the extra degree of freedom. Although this case can be treated as a bilateral constraint (the sphere can not move in the direction normal to either contact), there are most certainly scenarios which can not be simplified so easily. Other attempts at a general parameterization of multi-contact configurations will have similar problems. Parameterizing multiple contact configurations is a difficult problem we do not propose to solve.



This all appears to be a good argument for avoiding a reduced coordinate formulation and sticking with simpler more reusable constraints that need stabilization. This avoids the problem of choosing and switching between different parameterizations. Nevertheless, we still recognize the single contact problem as interesting, important, and useful. If the objects spend most of the time in a given contact state then parameterizing the local contact manifold is useful.

4.2.3 Kinematics Equations

The contact kinematics equations relate the relative motion between two smooth contacting bodies to a change in the contact coordinates.

Because the contact coordinates are of reduced dimension this system of equations will be a linear transformation (a function of the contact coordinates) from the 5 dimensional space of contact coordinate velocities into a 5 dimensional subspace of the 6 dimensional space of spatial velocities.

Many different derivations of these equations with minor differences can be found in [30, 10, 1, 35].

For example, [30] showed that when the parametric surfaces are orthogonal networks these equations can be written using the fundamental forms of curvature of the two surfaces.

For another example, if the contact coordinates are extended to include a distance along the z direction between the coordinate frames then we get a bijective transformation relating all spatial velocities to the changes in these extended contact coordinates. A derivation of this can be found in [35] where it is used for haptics. Having a separation distance is useful in haptics since a negative distance measures penetration which can be used to compute a reaction force based on a penalty method.

In our simulation, however, because we use the contact kinematics equations as a constraint we do not include distance as one of the degrees of freedom. We will

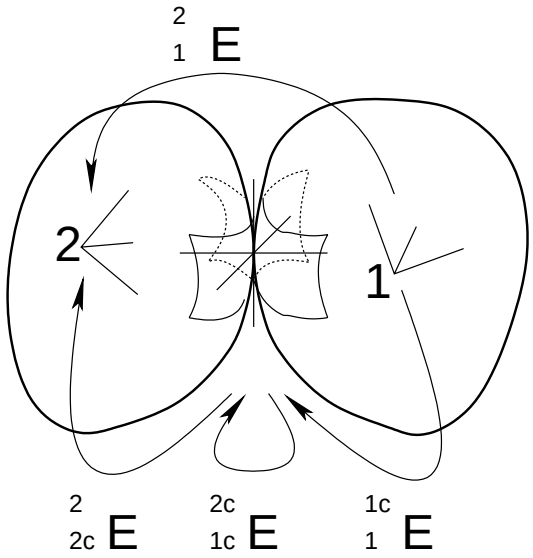


Figure 4.2: Change of reference frames

give our own derivation here, suitable for any type of parametric surface. A more detailed derivation can be found in Appendix A.

As illustrated in Figure 4.2 we compute a matrix representing a change in coordinates from frame *1* to frame *2*. This composition is written as

$$ {}^{2}_{1}\mathsf{E} = {}^{2}_{2c}\mathsf{E}\ {}^{2c}_{1c}\mathsf{E}\ {}^{1c}_{1}\mathsf{E}. \tag{4.9} $$

The three matrices on the right hand side are functions of the contact coordinates. They are defined by Equations 4.6 and 4.7.

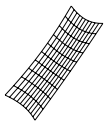
Recall that the relative spatial velocity can be defined by

$$ {}^{1}\phi(2,1) = \mathcal{V}\ {}^{1}_{2}\mathsf{E}\ {}^{2}_{1}\dot{\mathsf{E}}. \tag{4.10} $$

We take the time derivative by using the chain rule, thus

$$ {}^{1}\phi(2,1) = \mathcal{V}\sum_{j=1}^{5}\ {}^{1}_{2}\mathsf{E}\ \ {}^{2}_{1}\mathsf{E}_{,q_j}\ \dot{q_j}. \tag{4.11} $$

Since the contact coordinates can change independently of each other, each 4×4 matrix in the sum will be a twist and can have the extraction operator applied



directly, hence

$$ {}^{1}\phi(2,1) = \sum_{j=1}^{5} {}^{1}H_j \; \dot{q_j} \tag{4.12} $$

where,

$$ {}^{1}H_j = \mathcal{V} \; {}^{1}_{2}\mathsf{E} \;\; {}^{2}_{1}\mathsf{E}_{,q_j}. \tag{4.13} $$

The column vector ${}^{1}H_j$ tells us the contribution of a change in the j^{th} component of the contact coordinates to the relative spatial velocity of the two objects. Letting ${}^{1}H$ be a matrix whose columns are ${}^{1}H_j,\ j = 1 \ldots 5$, we can write Equation 4.12 in matrix notation as

$$ {}^{1}\phi(2,1) = {}^{1}H\dot{q}. \tag{4.14} $$

This 6×5 matrix H relates coordinate velocities to relative spatial velocities. It can be transformed to any other convenient frame by left multiplying with an appropriate adjoint.

Note that when H is written in one of the contact frames it takes on a simpler form. Its bottom row becomes zero because a velocity in the z direction is not achievable through a change in the contact coordinates (a non zero z velocity involves breaking contact).

The upper 5×5 sub-matrix of H is in general dense. In a contact frame, however, H will have several zeros if one of the surfaces is flat and without twists in its equiparameter lines.

If there is near conforming contact between the surfaces, H becomes poorly conditioned. When there is conforming contact the upper 5×5 sub-matrix becomes singular. Consider the example of a ball in a socket. If the surfaces have almost but not quite the same shape then very large parameter velocities are needed to account for small spatial velocities. If the surfaces conform exactly then we have a lower number of degrees of freedom and need a new parameterization for the contact (see [1] for an example of a cylinder on a flat surface going from conforming contact to point contact).

Although the derivation shown in Equations 4.9-4.14 is quick for demonstration, it lacks the convenience of the finer details necessary for an implementation. Appendix A contains a detailed derivation more suitable for implementation as it emphasizes ease of computation.

4.3 Constrained Dynamics

Let body *1* be free to move in contact with body *2* which is fixed. The Newton-Euler equations for a rigid body (see [36, 32]) can be written using spatial vectors as

$$f_{ext} + f_c = M\dot{\phi} - [\phi]^T M\phi \tag{4.15}$$

where f_{ext} is the external force, f_c is the constraint force, ϕ is the spatial velocity of the body relative to the world (inertial) frame, M is the spatial inertia, and all coordinates are with respect to the body fixed frame, frame *1*. Note that $\dot{\phi}$ is the time derivative of ϕ in the body frame.

We can combine this equation with the non-holonomic constraint

$$^1\phi = {}^1H\dot{q} \tag{4.16}$$

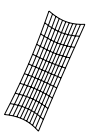
to get a differential algebraic equation. Differentiating the constraint once, in the body frame, we get

$$^1\dot{\phi} = {}^1\dot{H}\dot{q} + {}^1H\ddot{q}, \tag{4.17}$$

$$\text{where} \quad {}^1\dot{H} = {}^1_{1c}\dot{\mathsf{Ad}}\ {}^{1c}H + {}^1_{1c}\mathsf{Ad}\ {}^{1c}\dot{H}. \tag{4.18}$$

Combining Equation 4.15 with the constraint Equation 4.17 (instead of Equation 4.16) gives us an ordinary differential equation instead of a differential algebraic equation.

In the frictionless case the constraint force must not do any work on the system. The result of this is that f_c will be orthogonal to all of the degrees of freedom. This can be expressed as $H^T f_c = 0$. If interpreted in the contact frame,



f_c is easily seen to be perpendicular to the surface as the first five rows of H are linearly independent and the sixth row is zero.

We will write these equations in the following matrix form below where all quantities are expressed in frame *1* coordinates, so we drop the frame label for clarity.

$$\begin{pmatrix} I & M & 0 \\ 0 & I & H \\ H^T & 0 & 0 \end{pmatrix} \begin{pmatrix} f \\ -\dot{\phi} \\ \ddot{q} \end{pmatrix} = \begin{pmatrix} b \\ -a \\ 0 \end{pmatrix} \tag{4.19}$$

$$b = -[\phi]^T M\phi - f_{ext} \tag{4.20}$$

$$a = \dot{H}\dot{q} \tag{4.21}$$

Performing block row elimination on the last row produces

$$\begin{pmatrix} I & M & 0 \\ 0 & I & H \\ 0 & 0 & H^T MH \end{pmatrix} \begin{pmatrix} f_c \\ -\dot{\phi} \\ \ddot{q} \end{pmatrix} = \begin{pmatrix} b \\ -a \\ r \end{pmatrix}. \tag{4.22}$$

Here,

$$r = H^T(f_{ext} + [\phi]^T M\phi - M\dot{H}\dot{q}).$$

The last row yields an ordinary differential equation in the contact coordinates which can be integrated directly; it's equivalent to the ODE

$$\ddot{q} = (H^T MH)^{-1} H^T(f_{ext} + [\phi]^T M\phi - M\dot{H}\dot{q}). \tag{4.23}$$

Note that although $H^T MH$ is an invertible 5×5 matrix we use a matrix factorization instead of explicit inversion.

To summarize, we choose our constraint as the description of how two surfaces move when they are in contact. Because we constrain the motion of our objects to remain in contact, we will achieve this type of motion when we integrate while applying an external force. In this approach we avoid stabilization since we evolve the

system in a set of reduced coordinates which parameterize the constraint manifold exactly.

Recall that we only wanted a unilateral non-interpenetration constraint. Although the contact kinematics equations are much less complicated than the distance computation required for a position level constraint, our formulation effectively gives us a bi-lateral constraint. We resolve this problem by monitoring the constraint force to check for when the surfaces are breaking contact.

4.4 Friction

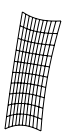
Friction is a necessary component for realism in any physically-based animation. We present a simple way of incorporating dry Coulomb friction into our algorithm. Sections 4.4.2 through 4.4.4 discuss ideas related to static friction which we have not implemented.

4.4.1 Dynamic Friction

With only a single point of contact between two contacting bodies and isotropic friction, we can make the assumption that the friction force opposes the relative velocity.

When the relative translational velocity is non zero, we are in sliding or dynamic friction mode. When the magnitude of the relative translational velocity falls below a small threshold, we might consider that the contact enters sticking or non-slip or static friction mode. In this mode the system has fewer degrees of freedom as the contact will move at the same speed across both surfaces. We discuss how no-slip friction could be added to our algorithm in Section 4.4.2. See also [17] for more on no-slip contact evolution.

We observed results which are adequate for many graphics applications by allowing the surfaces to be in frictionless contact when the relative translational velocity falls below a threshold. When the bodies are close to a static friction



configuration they will appear to be sticking but will in fact be slipping on one another and the contact will slowly drift from its current position.

For translational friction due to sliding we have the equation,

$$f_t = -\mu \hat{v} |f_n|, \quad v \neq 0, \tag{4.24}$$

where f_t is the tangent translational force due to friction, f_n is the normal force, and $\hat{v}$ is the normalized relative translational velocity. Summing tangent and normal components gives us the translational component of the constraint force.

This equation becomes very simple when written in coordinates of a contact frame since the constraint wrench in coordinates of a contact frame is already decomposed into a component normal to the constraint manifold and a tangential component due to friction. The normal component is simply the z translational component (the 6th entry of ${}^{1c}\mathsf{f}_c$). The remaining components are due to friction.

Let ${}^{1c}f_{c_i}$ be the i^{th} component of the constraint wrench written in frame *1c*. Equation 4.24 becomes

$${}^{1c}f_{c_4} = -\mu \ \hat{v}_x \ {}^{1c}f_{c_6}, \tag{4.25}$$

$${}^{1c}f_{c_5} = -\mu \ \hat{v}_y \ {}^{1c}f_{c_6}. \tag{4.26}$$

We may also like to have friction slow down an object which is rolling or spinning. This is most commonly achieved through viscous damping due to air resistance. It is not difficult, however, to have a rolling or spinning friction torque proportional to the normal force instead of the angular velocity (there is physical justification for this in some special cases, see [31]).

When written in the contact frame, the relative angular velocity, ω, decomposes into a rolling component, $\rho \stackrel{\text{def}}{=} (\omega_x, \omega_y)^T$, and a spinning component, ω_z. Letting μ_r and μ_s represent the coefficients of rolling and spinning friction we can

write,

$$ {}^{1c}f_{c_1} = -\mu_r \ \hat{\rho}_x \ {}^{1c}f_{c_6}, \tag{4.27} $$

$$ {}^{1c}f_{c_2} = -\mu_r \ \hat{\rho}_y \ {}^{1c}f_{c_6}, \tag{4.28} $$

$$ {}^{1c}f_{c_3} = -\mu_s \ \mathrm{sgn}(\omega_z) \ {}^{1c}f_{c_6}. \tag{4.29} $$

Recall that the constraint wrench in Equation 4.15 is in coordinates of frame *1*. Thus we will need to left multiply by the appropriate adjoint. We can use the adjoint and Equations 4.25-4.29 to write our new constraint equation in matrix form.

$$ \boldsymbol{\mu} \stackrel{\text{def}}{=} \begin{pmatrix} \mu_r \ \hat{\rho}_x \\ \mu_r \ \hat{\rho}_y \\ \mu_s \ \mathrm{sgn}(\omega_z) \\ \mu \ \hat{v}_x \\ \mu \ \hat{v}_y \end{pmatrix}, \tag{4.30} $$

$$ \begin{pmatrix} & \vdots & \\ I_{5\times5} & & 0 \\ & \vdots & \end{pmatrix} \ {}^{1}_{1c}\mathsf{Ad}^T \ {}^{1}\mathsf{f}_c = - \begin{pmatrix} & \vdots & \\ 0_{5\times5} & & \boldsymbol{\mu} \\ & \vdots & \end{pmatrix} \ {}^{1}_{1c}\mathsf{Ad}^T \ {}^{1}\mathsf{f}_c. \tag{4.31} $$

The left hand side extracts the tangential component of the constraint force, while the right hand side computes what the frictional component should be based on the normal restoring force. There are only five rows in the matrix because we do not place any restrictions on the normal force. The normal force is determined solely by the fact that the surfaces are not allowed to interpenetrate. The essence of this equation is to make the constraint wrench point in a slightly off normal direction such that it opposes the relative motion of the bodies. If the normal component needs to be larger to prevent the bodies from interpenetrating the tangential portion will also become larger due to the larger normal force.

Bringing everything to the left hand side in Equation 4.31 we get,

$$ F \ {}^{1}\mathsf{f}_c = 0, \tag{4.32} $$

$$\text{where } F \stackrel{\text{def}}{=} \begin{pmatrix} & \vdots & \\ I_{5\times 5} & \boldsymbol{\mu} & \\ & \vdots & \end{pmatrix} {}^{1}_{1c}\mathsf{Ad}^{T}. \tag{4.33}$$

The 5×6 matrix F thus replaces H^T in Equation 4.19. Repeating the block row simplification performed in Section 4.3 on the modified matrix results in the following equation which we use instead of Equation 4.23.

$$FMH\ddot{q} = F(f_{ext} + [\phi]^T M\phi - M\dot{H}\dot{q}) \tag{4.34}$$

Note that F is a function of ϕ. In Equation 4.34, a time explicit integration method would use the relative spatial velocity from the previous time step. This is what we have implemented.

4.4.2 No-Slip Friction

No-slip friction imposes a non-holonomic unilateral constraint on motion. Because it is non-holonomic, it only places a restriction on the contact coordinate velocities. We still need the five dimensional contact coordinates, however, to keep track of the configuration of the contacting objects. The contact coordinate velocities on the other hand are no longer independent since they only have three degrees of freedom.

In this section, unless otherwise specified, we let H be in a contact frame (say ${}^{1c}H$, in the contact frame *1c*). Let $H_{i..j,k..l}$ be the sub-matrix consisting of rows i through j and columns k through l. For example, $H_{4..5,1..2}$ is the portion of H which relates the contribution of the contact coordinate velocities $(\dot{s}, \dot{t})$ to the tangential translational velocity of the contact, (v_x, v_y).

When the two objects in contact are moving without slipping, the relative translational velocity of the bodies in the contact frame will be zero. We can use this fact to find a relationship between $(\dot{s}, \dot{t})$ and $(\dot{u}, \dot{v})$. Because $\dot{\psi}$ does not contribute

at all to this velocity we can write this relationship as

$$H_{4..5,1..2} \begin{pmatrix} \dot{s} \\ \dot{t} \end{pmatrix} = -H_{4..5,3..4} \begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix}. \tag{4.35}$$

$$\text{Hence,} \quad \begin{pmatrix} \dot{s} \\ \dot{t} \end{pmatrix} = D \begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix} \tag{4.36}$$

$$\text{where} \quad D \stackrel{\text{def}}{=} -H^{-1}_{4..5,1..2} H_{4..5,3..4}. \tag{4.37}$$

We can now rebuild a smaller version of H which relates the relative spatial velocity of the objects to a set of independent contact coordinate velocities. That is, we can write

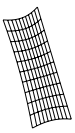
$$\phi = H_{1..6,1..2}\ D \begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix} + H_{1..6,3..4} \begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix} + H_{1..6,5}\ \dot{\psi}, \tag{4.38}$$

which has the matrix form

$$\phi = H' \begin{pmatrix} \dot{u} \\ \dot{v} \\ \dot{\psi} \end{pmatrix}, \tag{4.39}$$

$$\text{where} \quad H' \stackrel{\text{def}}{=} \begin{pmatrix} \vdots & \vdots \\ (H_{1..6,1..2}\ D + H_{1..6,3..4}) & H_{1..6,5} \\ \vdots & \vdots \end{pmatrix}. \tag{4.40}$$

The no-slip equations of motion are now simply the equations in Section 4.3 with all occurances of H and $\dot{q}$ replaced with this 6×3 matrix H' and the reduced set of contact coordinates velocities $(\dot{u}, \dot{v}, \dot{\psi})^T$ from Equation 4.39. We solve this smaller system to compute the reduced contact coordinate accelerations. We still need to integrate the original $\dot{q}$, however, to get the new contact configuration q. This is done at each derivative computation step by computing $\dot{q}$ using Equation 4.36. We use the reduced contact coordinate velocities resulting from the integration of the reduced contact coordinate accelerations.



4.4.3 No-Slip and No-Spin Friction

We can likewise additionally restrict the free object from spinning. This reduces the degrees of freedom to two. We get no-spin by requiring ω_z to be zero in the contact frame. We can write this requirement as

$$H_{3,1..2} \begin{pmatrix} \dot{s} \\ \dot{t} \end{pmatrix} + H_{3,3..4} \begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix} + H_{3,5}\ \dot{\psi} = 0. \tag{4.41}$$

Since ${}^{1c}H_{3,5} = 1$ (see Section 4.2 or Appendix A for more details), and using D from Equation 4.36 we can write

$$\dot{\psi} = -(H_{3,1..2}\ D + H_{3,3..4}) \begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix}. \tag{4.42}$$

Thus Equation 4.39 with the additional requirement of no spin becomes,

$$\phi = H'' \begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix} \tag{4.43}$$

$$\text{where} \quad H'' = \begin{pmatrix} \vdots \\ H_{1..6,1..2}\ D + H_{1..6,3..4} + H_{1..6,5}(-(H_{3,1..2}\ D + H_{3,3..4}) \\ \vdots \end{pmatrix}. \tag{4.44}$$

This 6×2 matrix along with $\dot{u}$ and $\dot{v}$ replace H and $\dot{q}$ in our dynamics equations of Section 4.3. After integrating the computed $\ddot{u}$ and $\ddot{v}$ to get $\dot{u}$ and $\dot{v}$, we use Equations 4.36 and 4.42 to compute $\dot{q}$.

4.4.4 Changes in Friction

We can simulate a better friction model by changing between our static and dynamic friction models at appropriate times. As mentioned before, although our sliding friction model can do a reasonable job imitating no-slip friction, objects which appear to be at rest will be slowly moving.

To switch between models we must monitor the constraint force and slip velocity. When in no-slip friction mode we check to see if

$$\frac{|f_t|}{|f_n|} \leq \mu_{\text{static}}. \tag{4.45}$$

Recall that the tangential and normal components are easily extracted from the constraint force when it is written in the contact frame. If the inequality is not satisfied then we switch to sliding friction. When in sliding friction mode, if the magnitude of the slip velocity falls below some threshold then we switch back to no-slip friction. Note that when switching to slipping friction, the objects should be given a chance to start sliding before switching back to no-slip friction.

The no-spin condition suggests that the contact area is large enough to transmit frictional torques. In this case we might also want to compare the torque about the z axis with the magnitude of the normal force. That is, if the inequality

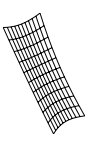
$$\frac{|{}^{1c}f_{c_3}|}{|f_n|} \leq \mu_{\text{spin static}} \tag{4.46}$$

is violated then we switch to sliding friction.

4.5 Traversing Patch Boundaries

Object models are commonly a collection of surface patches. In Figure 4.3, the bowl at the left consists of 56 triangular patches and the plate on the right consists of 192 triangular patches. A robust method of evolving a contact across patch boundaries is needed.

Although the surfaces depicted in Figure 4.3 are subdivision surfaces, recall that any surface that can be evaluated parametrically is suitable for our contact evolution technique. For our boundary traversal method, an object can consist of any mixture of patches, from simple Bézier or NURBS patches (see [39, 45]) to more complicated surfaces, as long as tangent plane continuity exists between all pairs of neighbouring patches.



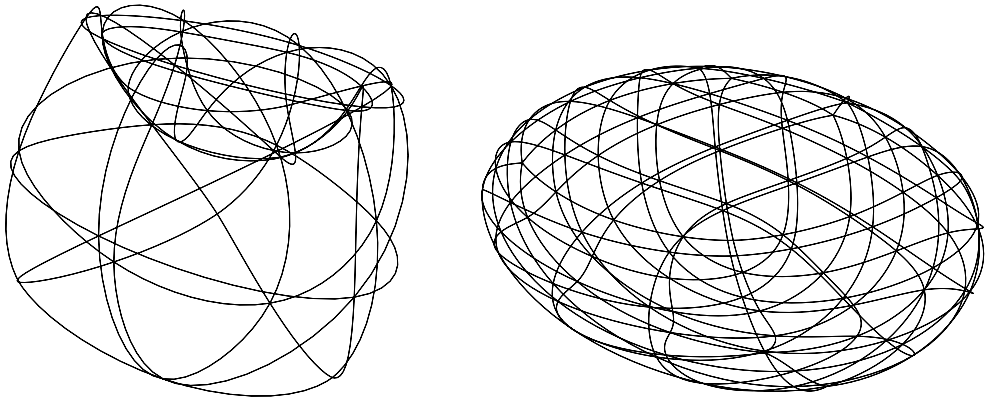


Figure 4.3: A bowl shape and a complex shallow bowl showing patch boundaries.

We assume that the domain of a patch is a polygon, which we describe by a counter clockwise list of points in the plane, $P_i \in \mathbb{R}^2, i = 1 \ldots n$. For every edge in the domain polygon we need to know both the adjacent patch and the corresponding domain edge. That is, for a given patch A and its domain edge $P_i^A P_{i+1}^A$ mapping to a curve in $\mathbb{R}^3$, we have a corresponding patch B with domain edge $P_j^B P_{j+1}^B$ which maps to the same curve in $\mathbb{R}^3$. We call the edge $P_i^A P_{i+1}^A$ edge i of patch A.

We also need to know how to find points on edge j of patch B which map to the same location in $\mathbb{R}^3$ as points on edge i of patch A. One way of describing the relationship between these two boundaries is to write the boundary curve as a single parameter function. Edge i of patch A whose shape is described by $\mathsf{c}^A(u, v)$ can have its boundary curve written as

$$\mathsf{c}_i^A(u) \stackrel{\text{def}}{=} \mathsf{c}^A(P_{i+1}^A u + P_i^A(1-u)). \tag{4.47}$$

Then we need to know the invertible reparameterization function

$$g : \mathbb{R} \to \mathbb{R} \tag{4.48}$$

$$\text{such that} \quad \mathsf{c}_j^B(u) = \mathsf{c}_i^A(g(u)). \tag{4.49}$$

In many cases this function will turn out to be quite simple. In our case, all boundary curves are of the same form and are equivalent to quartic Bézier curves.

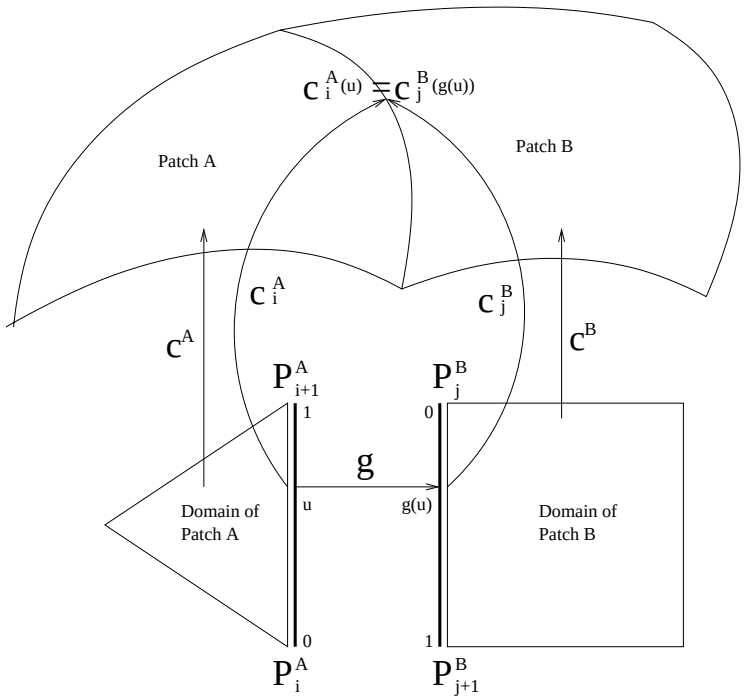
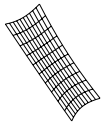


Figure 4.4: Domain boundaries and functions involved in a transition

The adjacent curves are identical (the control points in the Bézier basis will be the same) giving us a very simple function $g(u) = 1 - u$. It is not the identity because the domain polygons are described in counter clockwise order. Thus the curves, although identical, start at opposite ends.

Trimmed NURBS surfaces, as created by many commercial design packages, would cause a problem here. Trimmed NURBS are usually the result of a boolean constructive solid geometry operation and result in sharp edges. The sharp edged curves are often described in a piecewise linear fashion in the domain since surface surface intersections result in very high degree curves making it impractical to define the boundary by a curve in the domain (see [48]). That said, we do not address curve-curve or surface-curve contacts because they require a different contact coordinate parameterization (see [1]), but [35] addresses the issue of finding boundary points



in the domains of adjacent trimmed NURBS surfaces.

Here are the steps for doing a domain edge traversal:

- compute the state of the system at the time of crossing;
- compute the contact coordinates of the system using the adjacent patch;
- compute the new contact coordinate velocities.

We use the following notational convention. We assume, without loss of generality, that the contact moves from patch A to patch B on body 1 while the contact remains in patch C of body 2. When the state of the system is at a patch boundary we use a superscript to denote the patch whose coordinate system is being used. For example, q^A would denote the system configuration using patch A. Likewise H^A denotes the contact kinematics equations using patch A.

4.5.1 Time of Crossing

A boundary crossing is detected when the contact coordinates fall outside of the domain of the patch. This is easy to check when the domain polygon is convex in which case we check if the point falls to the left or right of the domain edges. If the point is on the right of any edge of the convex domain, then it is outside.

Once it is determined that the contact is leaving the domain of a patch we must back up the state of the system to the time of the traversal.

One approach is to revert to the previous system state and integrate with smaller time steps until the contact position in the domain of the patch being traversed is sufficiently close to the boundary. The point can then be projected onto the boundary.

An alternative is to assume that the evolution of the contact is linear on the scale of the step size of the integration technique. In this case we can compute a line segment intersection to compute the new state of the system. Suppose the line

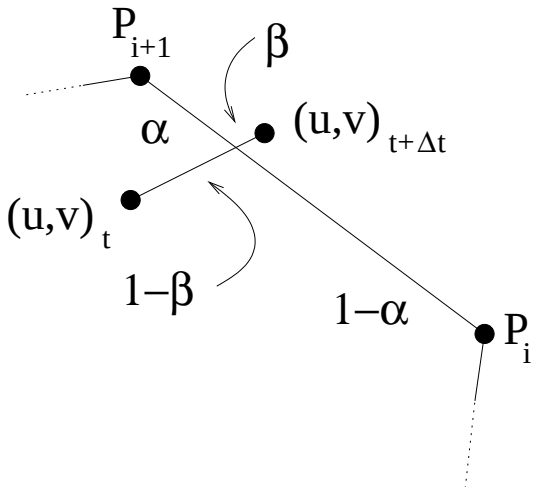


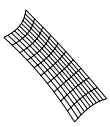
Figure 4.5: The boundary crossing as the location of intersection.

segment $P_i^A P_{i+1}^A$ is the domain boundary edge being crossed and $(u,v)_t(u,v)_{t+\Delta t}$ is the line segment described by the inside and outside contact locations. Let $\alpha, \beta \in [0,1]$ describe the location of the intersection on each segment as a proportion of the distance along each segment (see Figure 4.5). We can take the state of the system at the boundary to be

$$q^A \stackrel{\text{def}}{=} q^A_{t+\beta\Delta t} \approx \beta q^A_{t+\Delta t} + (1-\beta) q^A_t, \tag{4.50}$$

$$\dot{q}^A \stackrel{\text{def}}{=} \dot{q}^A_{t+\beta\Delta t} \approx \beta \dot{q}^A_{t+\Delta t} + (1-\beta) \dot{q}^A_t \tag{4.51}$$

and the location of the contact on the boundary will be $\mathsf{c}_i^A(\alpha)$. Note that we define q^A and $\dot{q}^A$ to be the state of the system at the boundary for the convenience of not specifying the time of crossing in a subscript. As mentioned before, the superscript denotes that the contact coordinate and contact coordinate velocity use patch A.



4.5.2 Contact Location on Adjacent Patch

We can write the state of the system at the boundary of patch B by using the function g defined in Equation 4.48.

$$q^B = \begin{pmatrix} P^B_{j+1} g(\alpha) + P^B_j (1 - g(\alpha)) \\ q^A_{3..4} \\ \psi^B \end{pmatrix} \tag{4.52}$$

Note that the third and fourth component of the contact coordinate remain unchanged. Note that if there is a simultaneous patch transition on the other object it will be handled after this transition is complete. Knowing the coordinates on the adjacent patch, we still need to compute ψ^B. This is done by adding to ψ^A a correction to reflect the angle between the x axes of the contact frames at the crossing position of the adjacent patches. Since $\cos^{-1}(\mathsf{x}^A \cdot \mathsf{x}^B)$ is computed as an angle in the range $[0, \pi]$[1], we check to see if $\mathsf{x}^A \times \mathsf{x}^B$ points in the opposite direction of the surface normal in which case we negate the correction angle. Thus we can write

$$\psi^B = \psi^A + \text{sgn}((\mathsf{x}^A \times \mathsf{x}^B) \cdot \mathsf{z}) \ \cos^{-1}(\mathsf{x}^A \cdot \mathsf{x}^B). \tag{4.53}$$

Note that if $\mathsf{x}^A \times \mathsf{x}^B = 0$ then $\cos^{-1}(\mathsf{x}^A \cdot \mathsf{x}^B)$ is either 0 or π making the sign correction irrelevant. Figure 4.6 shows the case where $\mathsf{x}^A \times \mathsf{x}^B$ points in the same direction as the normal.

4.5.3 Computing New Contact Coordinate Velocities

Now that the contact coordinates q are known on both sides of the boundary we can compute $\dot{q}^B$ from $\dot{q}^A$. We equate the relative spatial velocity on each side of the boundary using ${}^{2c}H^A$ and ${}^{2c}H^B$. The spatial velocity is computed in a contact frame allowing us to drop the z component of the velocity (which is zero) resulting in a system of equations which are not over-constrained. The contact frame of body

[1] e.g. by Java's Math.acos().

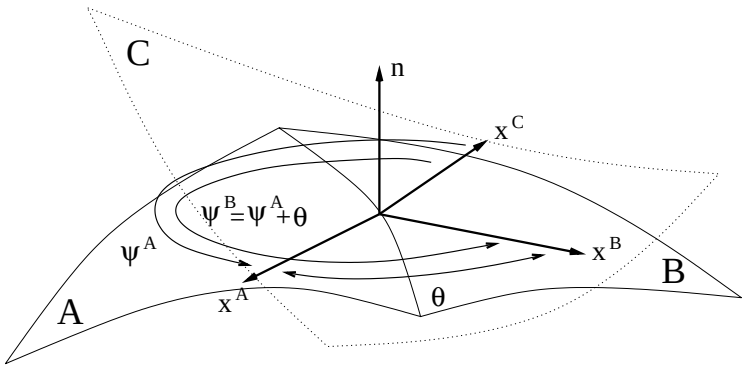


Figure 4.6: To compute contact coordinates on patch B, the angle ψ has θ added to it. We add positive θ because $\mathsf{x}^A \times \mathsf{x}^B$ points in the same direction as the normal.

2 is used as it is unaltered while the contact frame of body 1 changes from patch A to patch B. Hence, we solve for $\dot{q}^B$ in

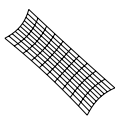
$$ {}^{2c}\hat{H}^A \; \dot{q}^A = {}^{2c}\hat{H}^B \; \dot{q}^B, \tag{4.54} $$

where ${}^{2c}\hat{H}$ is a 5×5 matrix equivalent to the matrix ${}^{2c}H$ with the bottom (zero) row removed.

Knowing the system state after the boundary traversal $(q^B, \dot{q}^B)$, we can run the integrator again with a step size of $(1 - \beta)\Delta t$ to bring us to the target time, $t + \Delta t$. If additional boundary crossings occur in this time interval, then they are treated in exactly the same manner.

4.5.4 Corner Transitions

We might ask what happens if the contact goes directly through a corner. In this case, although the contact may not end up in one of the adjacent patches (one sharing a boundary rather than just a vertex with our current patch), we still transfer the contact to an adjacent patch. Once in the adjacent patch the contact may evolve through the corner again. The process is repeated until we end up in a patch where the contact coordinate velocity evolves the contact into the interior of the domain.



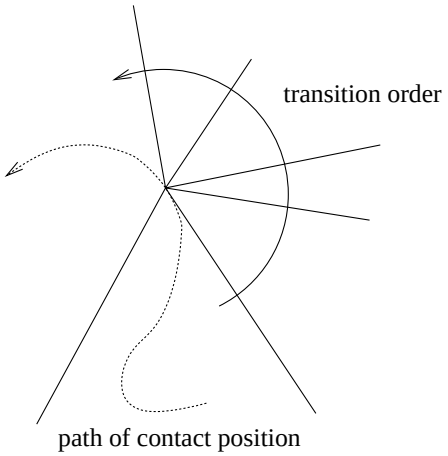


Figure 4.7: Transition through a corner

If we have n patches meeting at a point, we may need to perform $n-1$ traversals before we end up in the final destination patch (see Figure 4.7). This is because we take the first domain edge which intersects the state trajectory line segment. That is if the contact goes through P_{i+1} then both P_iP_{i+1} and $P_{i+1}P_{i+2}$ will intersect with the path. Consistently choosing the edge which ends rather than begins at the traversal point will prevent oscillating transitions between two adjacent patches.

4.5.5 Avoiding Traversal Problems

To increase robustness, we also allow the contact to remain on a patch if it is outside but within epsilon of the patch boundary. This is at a small expense of accuracy, however, as the patch shape outside the boundary does not describe the shape of the surface.

This epsilon extended boundary eliminates the problem of oscillating patch transitions which might occur when a contact exactly follows a patch boundary. To perform transitions when the patch is crossing the epsilon extended boundary we project the contact location back onto the boundary edge before making the

transition and then displace the contact location by an equivalent amount into the adjacent patch after the traversal. In doing this we must be careful not to compute a contact location in the new patch which is outside the epsilon extended boundary.

4.6 Algorithm Summary

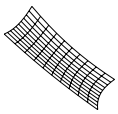
At this point it may be useful to review the entire algorithm. Everything is centered around the computation of $\ddot{q}$. For a single contact there is a constant amount of work to do this computation.

While the integrator has not achieved the target time for the next frame in the animation we try to step the integrator (note that this may result in crossing a patch boundary). The integrator gives us the current state and we must compute the derivatives of this state. That is we get $(q, \dot{q})$ and need to produce $(\dot{q}, \ddot{q})$. This only involves computing $\ddot{q}$ as we can use the $\dot{q}$ provided with our current state. The whole process can be summarized briefly by the steps that follow.

- First we evaluate the surfaces. That is, we must compute the position of the contact point on each surface and all the partial derivatives of order up to three. Because the basis functions of parametric surfaces are smooth and due to the symmetry of mixed partials ($\mathsf{c}_{,uv} = \mathsf{c}_{,vu}$), only ten sets of basis functions are needed to do this evaluation.

- Next we must compute the position and orientation of the bodies in world coordinates. This is necessary for computing any external forces such as gravity. We do this by computing ${}^{2}_{1}\mathsf{E}$ from Equation 4.9 using the surface evaluations we just computed. The absolute position and orientation of the objects are also used to render the animation.

- We then compute H with the method described in Appendix A. We transform H to the free body reference frame and compute ϕ using $\dot{q}$ from the current state.



- We now have everything we need to solve for $\ddot{q}$. We do this by solving Equation 4.23 for the frictionless case, or by solving Equation 4.34 for the friction case. This is accomplished with an LU decomposition of $H^T M H$ in the frictionless case, or FMH in the friction case.

  Recall that since contact coordinates are used the constraints are automatically satisfied and there is no need for constraint stabilization. Back substituting $\ddot{q}$ into the second row of Equation 4.22, we get the contact wrench f_c which can be monitored to detect contact breaking.

- If the integrator computes a new contact coordinate which is outside the current domain boundary then we backup to the time of the crossing, traverse the boundary as described in Section 4.5 and repeat the entire process from the first step until we have arrived at the target time.

Chapter 5

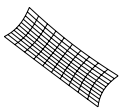
Surface Representation

We primarily use Loop subdivision surfaces to describe the boundaries of objects because of their popularity in the graphics community. Note that although subdivision surfaces are similar in many ways to other popular surface representations such as NURBS, they pose some special difficulties with respect to our contact evolution technique. In this chapter we describe the considerations that are needed to use Loop subdivision surfaces with our technique. We end the chapter with a general description of the features of our implementation.

5.1 Subdivision Surfaces

Subdivision surfaces have become popular in recent years for modelling objects [12]. This is because they provide an easy mechanism for converting meshes with arbitrary topology into smooth surfaces. The original mesh provides an easy way for the user to edit the shape of the smooth surface. Another advantage is that subdivision surfaces are not hard to implement. We will give a short introduction to subdivision surfaces in this section. The SIGGRAPH course notes [12] discuss the details in much greater depth (see also Schwitzer's PhD thesis, [47]).

A subdivision surface algorithm consists of rules for subdividing a polyhedral mesh where all faces have the same number of sides. The rules are applied recursively



to get finer and finer meshes. In the limit the rules define a smooth surface with nice continuity properties (they typically are C^1 or C^2 at patch boundaries, but C^∞ almost everywhere).

The mesh which defines the surface is usually but not necessarily a closed manifold. In the case of the Loop subdivision rules the faces are triangular, while for the Catmull Clark rules the faces are rectangular. These are the two most common types of subdivision. There are others with various interesting properties, such as the Butterfly scheme which interpolates mesh vertices.

The subdivision rules define a finer mesh by moving the existing vertices (or not in an interpolating scheme) and inserting new vertices to make new smaller faces (see figure 5.2). The rules state where a vertex moves, or where a new vertex should be created, based on the positions of vertices in a small neighbourhood. In the refined mesh, the new vertices are called *odd* vertices while the vertices which were in the original mesh are called *even* vertices. This naming convention comes from one dimensional subdivision curves because when the vertices are stored in a zero indexed array they occur at even and odd indices.

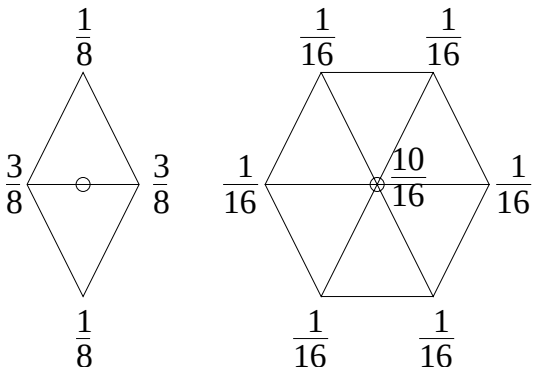


Figure 5.1: Loop subdivision rules

Figure 5.1 shows the affine combinations, also called vertex maps, for the Loop rules. The odd vertices have their position defined by an affine combination of their position and the positions of their neighbours. The even vertices are an affine combination of the surrounding vertices.

For the Loop subdivision rules, any vertex in the mesh which does not have a valence of six is referred to as an *extraordinary* vertex. Although there are slightly different subdivision rules at extraordinary vertices, the new position of the vertex

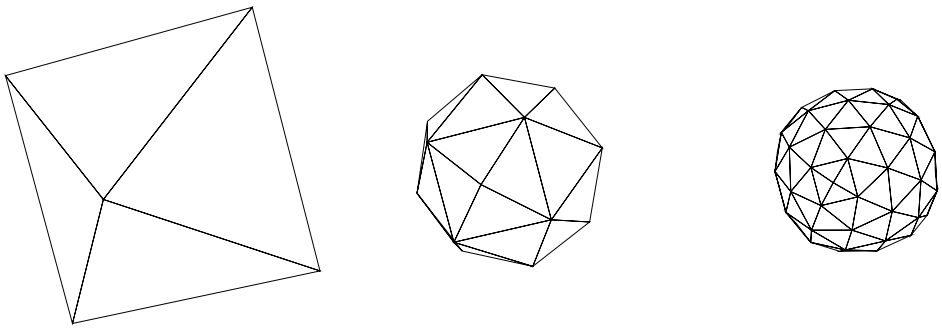


Figure 5.2: Three levels of subdivision of an octahedron. Note how the number of extraordinary vertices does not grow with subdivision.

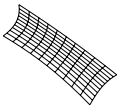
is still an affine combination of its position and the positions of its neighbours. Although there are multiple ways of choosing the weights (see [54]) the actual values are not important to us as we only require that the surface be smooth. Unfortunately, the resulting surfaces are typically less smooth at extraordinary vertices.

Most meshes will have extraordinary vertices since only shapes which are topologically a torus can have all mesh vertices with valence six. This can be seen by evaluating the Euler characteristic to compute the genus. This relationship is given in [18] by,

$$V - E + F = 2 - 2g \tag{5.1}$$

where V is the number of vertices, E is the number of edges, F is the number of faces and g is the genus. If a mesh consists of triangular faces and vertices of degree six, then we can count the edges and vertices from the number of faces. Each face has three vertices, each of which are shared among six faces ($V = \frac{3}{6}F$). Each face also has three edges which are shared with one other face ($E = \frac{3}{2}F$). Substituting these equalities into Equation 5.1 gives g equal to one, independent of the number of faces.

It is interesting to note that the number of extraordinary vertices does not change as the mesh is subdivided. This can be seen in the sample subdivision of an octahedron in Figure 5.2.



5.1.1 Parametric Evaluation of Subdivision Surfaces

Although subdivision surfaces were not designed with parametric evaluation in mind, it turns out that they can be evaluated parametrically. Loop and Catmull Clark surfaces happen to be quite simple away from extraordinary vertices. Away from extraordinary vertices, Loop surfaces can be described as quartic box splines, while Catmull Clark surfaces can be described by bi-cubic Bézier surfaces.

Because the subdivision rule for a vertex is a simple affine combination of its neighbouring vertices, a *subdivision matrix* can be used to describe how a neighbourhood of points moves at each subdivision step. In [52, 51], Stam shows that the eigenstructure of the subdivision matrix is useful for evaluating the surface next to an extraordinary vertex. The idea is to subdivide the surface until the point being evaluated is no longer in a patch with an extraordinary vertex. Subdivision to the desired depth is achieved by computing a power of the subdivision matrix. This is easily performed with the eigenvalue decomposition.

In Stam's formulation each face can only have one extraordinary vertex. This is easy to guarantee as we can perform one subdivision on our mesh to get a new mesh which has this property. In the new mesh the odd vertices will have degree six and no two even vertices will be adjacent.

For transitions between patches, we define the domain of a Loop patch to be the triangle with vertices $P_1 = (0, 0), P_2 = (1, 0)$ and $P_3 = (0, 1)$ (we treat extraordinary patches differently in Section 5.2). Although planar domain coordinates can be converted to barycentric coordinates for our quartic box spline representation, we do not need to do this in our implementation as we write the basis functions solely in terms of the two coordinates by simple substitution. That is, although the Loop basis functions are given by Stam (see [51]) in terms of barycentric coordinates u, v, and w, we replace occurances of w with $1 - u - v$.

5.1.2 Non-regular Parameterization

Because the contact coordinates use the parameterization of the surface, the algorithm is sensitive to poor parameterizations. Excessive twisting in a surface causes problems which are not always overcome with smaller step sizes. For an example where smaller step sizes work, consider a planar surface defined by a Bézier surface where the equiparameter lines twist left and right in the plane. For an object, such as a sphere, rolling in a straight line on this plane the contact coordinates will need to follow a non linear path in the domain of the twisty plane.

Loop surfaces near extraordinary vertices using the natural parameterization are unpleasant. When the valence of the extraordinary vertex is less than six the partial derivatives of the patch at the extraordinary vertex are zero. That is,

$$\lim_{(u,v)\to(0,0)} \mathsf{c}_{,s}^{\deg<6}(u,v) = 0, \quad \mathsf{c}_{,s}^{\deg<6}(0,0) = 0, \tag{5.2}$$

$$\lim_{(u,v)\to(0,0)} \mathsf{c}_{,t}^{\deg<6}(u,v) = 0, \quad \mathsf{c}_{,t}^{\deg<6}(0,0) = 0. \tag{5.3}$$

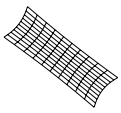
When the valence is greater than six the partial derivatives at the extraordinary vertex are undefined. In this case, the magnitude of the partial derivatives become large without bound as we approach the extraordinary vertex. That is,

$$\lim_{(u,v)\to(0,0)} |\mathsf{c}_{,s}^{\deg>6}(u,v)| = \infty, \quad \mathsf{c}_{,s}^{\deg>6}(0,0) = \text{undefined}, \tag{5.4}$$

$$\lim_{(u,v)\to(0,0)} |\mathsf{c}_{,t}^{\deg>6}(u,v)| = \infty, \quad \mathsf{c}_{,t}^{\deg>6}(0,0) = \text{undefined}. \tag{5.5}$$

An additional concern with both cases is that the isoparametric lines which come close to the extraordinary vertex make very sudden and sharp bends.

These effects are visible in Figures 5.3 and 5.4. The patches shown in these figures are flat with the extraordinary vertices in their lower right corners. The control points are placed such that n copies could be placed sharing and completely surrounding a degree n extraordinary vertex. The equiparameter lines become tightly bunched together near the extraordinary vertex in the degree three patch, while they become spaced further apart in the degree ten patch. The sharp bends within



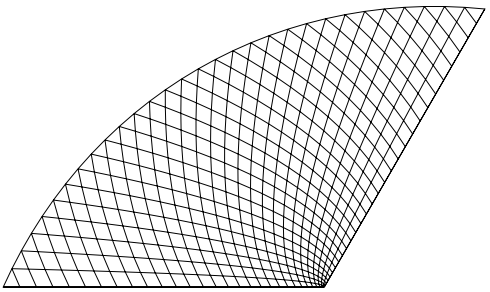


Figure 5.3: A planar valence 3 patch showing tight bunching of equiparameter lines near the extraordinary vertex.

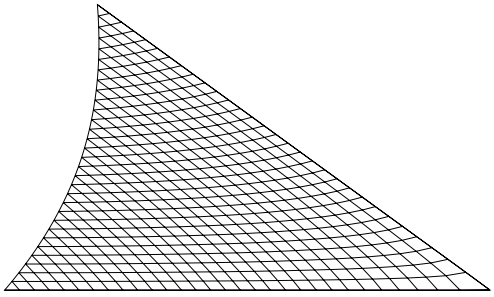


Figure 5.4: Valence 10 patch showing larger spacings between equiparameter lines near the extraordinary vertex.

the plane are also visible. In the valence three patch the isoparametric lines bend towards the extraordinary vertex, while they bend away from the extraordinary vertex in the degree ten case.

If a contact is moving at constant speed near an extraordinary vertex, this will result in either very large or very small contact coordinate velocities (depending on the valence). The more noticeable problem, however, is that $\dot{\psi}$ will also become very large to accommodate the twists of the isoparametric lines. When a contact is evolving near an extraordinary vertex, the contact coordinate ψ will have large accelerations. These accelerations may not integrate out leaving one body spinning like a top on the other. Another consequence is that H becomes poorly conditioned.

The interior of a regular Loop patch, being a polynomial surface, is C^{∞}. Note that there is normally a discontinuity in the third derivative at patch boundaries, except when the adjacent patches describe the same function. Because of the subdivisions necessary to get a regular patch near an extraordinary vertex, there exists a spider web of discontinuities in the third derivative.

One possible remedy is to reparameterize the extraordinary patch to make it regular. Stam suggests reparamaterizing based on the characteristic functions which correspond to the two partial derivatives (see the updated version of [51] which appears as a part of [12]). This unfortunately involves inverting these functions which

is impractical since they are quartic polynomials in two variables. Additionally, derivatives of the inverted characteristic functions are needed for computing the derivatives of our reparameterized surface.

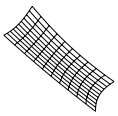
Another possible solution is to use a different subdivision rule at extraordinary vertices. Loop recently developed a method of using a vertex map with larger support which results in a surface with bounded curvature. This research can be found in the technical report [26]. To use Loop's result we would need the eigenstructure of the new subdivision matrices. Unfortunately, the rules are identical to the original rules when the valence is three. Thus the new scheme does not alleviate the non-regular parameterization problem for the valence three case. Additionly, we did not consider this as a possible solution to our problem since this report did not come to our attention until after we implemented the solution outlined in the following sections.

5.2 Surface Replacement

Given all the problems discussed in the previous section, we feel that the best option for avoiding these problems is to replace the portions of the subdivision surface near extraordinary vertices with surfaces which are better behaved.

Part of our justification for replacing the surface is that subdivision surfaces are not an exact model of anything. Their main purpose is to produce a smooth surface which approximates a desired shape, using a control mesh. If we are using a subdivision surface to model a real world object, it will be at best an approximation of this object. In this line of reasoning it is not unreasonable to change the shape of the surface slightly to create a surface with better properties.

This is a similar idea to what Peters does in [37]. He shows a method for building a minimal number of maximally sized NURBS surfaces which match a Catmull Clark subdivision surface exactly, except near extraordinary vertices where they are within a given small epsilon. This would be useful if we were using Catmull



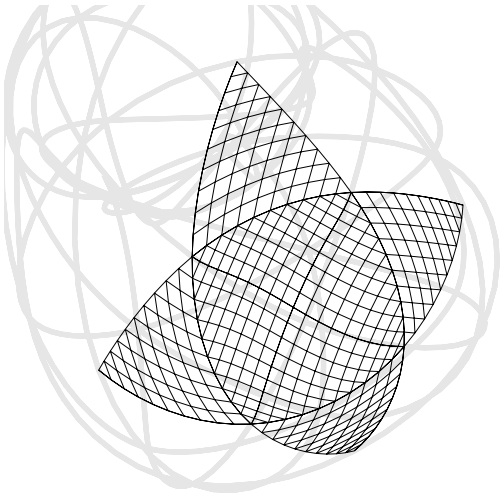


Figure 5.5: An four sided S-patch fit to an extra-ordinary vertex of a bowl. Note the gentle twisting in the equiparameter lines.

Clark subdivision surfaces.

Since we are using Loop subdivision surfaces, our solution involves cutting a hole in the surface around an extraordinary vertex and replacing it with an S-patch. S-patches, developed by Loop and DeRose in [27], are a multi sided generalization of a Bézier surface. We will give a description of S-patches in the next section.

Figure 5.5 shows the result of replacing the surface near one of the degree four extraordinary vertices in the bowl on the left of Figure 4.3. Although it is hard to tell from the figure, the four sided S-patch matches the original surface quite closely being only slightly flatter than the original surface.

5.2.1 S-patches

S-patches are defined by two maps. The first map is an embedding E from a polygon in the plane into an n-dimensional simplex. The second map, B, is the standard multidimensional Bézier map from the n-dimensional simplex into $\mathbb{R}^3$. When the

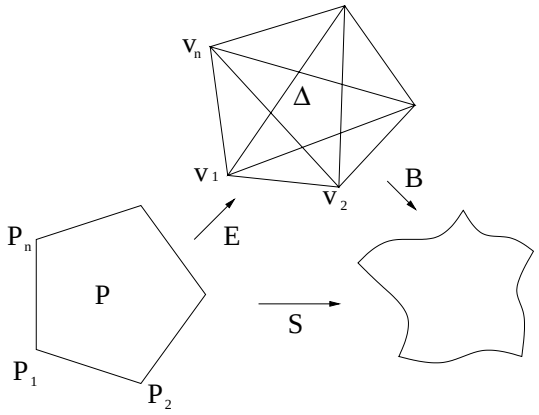


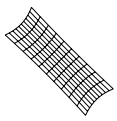
Figure 5.6: S-patch maps

maps are composed, the result is a surface given by the image of the two dimensional manifold within the simplex defined by the embedding. In other words, the S-patch function mapping the polygon P to $\mathbb{R}^3$ is given by the composition $S = B \circ E$ as shown in Figure 5.6.

The domain polygon is defined by the vertices of a regular n sided polygon centered at the origin and with the first point, P_1, lying on the positive u axis. We let the polygon be of unit area to help match the magnitude of derivatives of adjacent patches. This is desirable for contact evolution. We do not want our contact coordinate velocities to be dominated by any component as this will coincide with a less desirable condition number for H.

The simplex is defined by the vertices v_i. A convenient way to think about these n dimensional vertices is to let their coordinates all be zero except for a one in the i^{th} position.

The embedding is designed to map edges in the polygon to edges in the simplex (Loop and DeRose call this property *edge-preserving*). A walk around the polygon in the plane would map to a walk which goes in straight lines in each dimension and then returns to the starting point along the diagonal.



We will now give definitions of the maps. We will start with the multidimensional Bézier map. We use n dimensional barycentric coordinates,

$$(u_1, \dots u_n) \in \Big[0, 1\Big]^n \tag{5.6}$$

$$\text{where} \quad \sum_{j=1}^{n} u_j = 1, \tag{5.7}$$

to write the Bernstein polynomials of dimension n and degree d. We also need a n dimensional multi-index for the degree d polynomials. It is defined as

$$\vec{i} = (i_1, \dots i_n) \in \mathbb{N}^n \tag{5.8}$$

$$\text{with} \quad \sum_{j=1}^{n} i_j = d. \tag{5.9}$$

Note that $\mathbb{N}$ refers to non-negative integers including zero. We can now write the $\vec{i}^{\text{th}}$ degree d Bernstein basis function as,

$$Bern_{\vec{i}}^{(d)}(u_1, \dots u_n) = \binom{d}{\vec{i}} \prod_{j=1}^{n} u_j^{i_j}. \tag{5.10}$$

Here we use the multinomial coefficient instead of a binomial coefficient. It is defined as,

$$\binom{d}{\vec{i}} = \frac{d!}{i_1! i_2! \dots i_n!}. \tag{5.11}$$

Thus, given control points $\mathsf{V}_{\vec{i}}$ corresponding to the multi-indices, we can write the Bézier map as

$$B(u_1, \dots u_n) = \sum_{\vec{i}} \mathsf{V}_{\vec{i}}\, Bern_{\vec{i}}^{d}(u_1, \dots u_n). \tag{5.12}$$

The embedding giving us the n dimensional barycentric coordinates we need

is described by the following functions.

$$\alpha_i(p) = det \begin{pmatrix} p^u & P_i^u & P_{i+1}^u \\ p^v & P_i^v & P_{i+1}^v \\ 1 & 1 & 1 \end{pmatrix} \tag{5.13}$$

$$\pi_i(p) = \prod_{j \neq i-1,i} \alpha_j(p) \tag{5.14}$$

$$l_i(p) = \frac{\pi_i(p)}{\pi_1(p) + \cdots + \pi_n(p)} \tag{5.15}$$

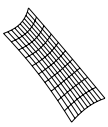
Here, p is a point in the polygon P. We use the superscripts u and v to denote the components of our two dimensional domain points. The function α_i defines the signed area of the triangle $P_iP_{i+1}p$ (all subscripts to the domain points are to be taken modulus n). Note that this area is zero when p falls on the edge P_iP_{i+1}. The function π_i being the product of all but two adjacent α_i functions will be zero around the entire boundary of the polygon except at the edges $P_{i-1}P_i$ and P_iP_{i+1}. Each π_i is normalized by dividing by the sum of all of the π_i functions to produce a function l_i. It is easily seen that the l_i functions partition unity as the sum of the numerators is equal to the common denominator. The functions l_i provide the barycentric coordinates we need for the multidimensional Bézier function. Loop and DeRose write the embedding function with the letter L when the domain polygon is a regular n-gon. This embedding, which we can write as

$$L(p) = \sum_{i=1}^{n} l_i(p) v_i, \tag{5.16}$$

has some useful properties. One such property is that

$$l_i(P_i(1-u) + P_{i+1}u) = u. \tag{5.17}$$

This makes it very easy to do transitions from the adjacent Loop patches into an S-patch. The fact that domain edges map to Bézier curves (this is true for any embedding E which maps edges to edges) combined with the property stated in Equation 5.17 implies that the function $g(u)$ introduced in Equation 4.48 is equal



to $1 - u$. The other important property is that we can easily maintain continuity between the S-patches and other adjacent polynomial patches (rational or non-rational).

5.2.2 Matching Continuity

The *blossomed* version of a polynomial function is the unique affine symmetric function which is equal to the original function when all the parameters are equal. For example the cubic function $f(u) = u^3 + u^2$ has the blossom

$$f^*(u_1, u_2, u_3) = u_1u_2u_3 + 1/3(u_1u_2 + u_2u_3 + u_1u_3). \tag{5.18}$$

The blossom is said to be affine because it is linear in each parameter when the others are held fixed. It is symmetric because the order of the parameters does not matter.

The power of the blossomed form is that the representation of f in the Bézier basis is easy to compute using the blossom. The weights for the Bernstein basis polynomials are computed by evaluating the blossom with various combinations of the domain end points. For our cubic curve above, defined on the interval $[0, 1]$, we have Bézier control points $f^*(0, 0, 0), f^*(1, 0, 0), f^*(1, 1, 0)$ and $f^*(1, 1, 1)$. A proof of this can be found in [41].

The result is similar for multi-variable functions. In this case they are multi-affine functions rather than affine functions since the parameters are no longer one dimensional. Ramshaw has written two tech reports, [41, 42] which discuss blossoms and their properties in much greater depth.

Using the blossomed form of an adjacent triangular Bézier patch, Loop and DeRose show that continuity can be maintained between an S-patch and an adjacent triangular patch. This technique (also demonstrated in [41] for maintaining continuity for curves and surfaces of the same type) involves placing the domains side by side and then evaluating the blossom with all combinations of the corners of

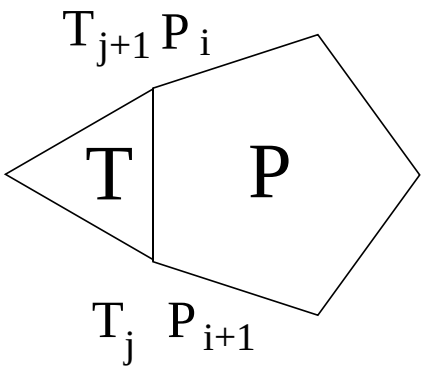


Figure 5.7: A triangular domain adjacent to a five sided domain

our domain. Note that these points will be on the boundary or outside the domain of the blossomed function.

To illustrate this further, suppose we have a triangular patch of degree d defined by the function f with blossom f^*. To find the control points for an adjacent n sided S-patch, we build an equilateral triangle domain polygon, T, for our triangular patch such that it shares an edge with the S-patch control polygon. Figure 5.7 shows this where we match up the desired edge j on the triangular patch with edge i on the n sided S-patch. When written in the barycentric coordinates of the triangle T, let the points P_i of the S-patch domain polygon be written as x_i. We can then write the control points of the S-patch which exactly match our function f as

$$\mathsf{V}_{\vec{i}} = f^*(\overbrace{\underbrace{x_1,\ldots,x_1}_{i_1},\underbrace{x_2,\ldots,x_2}_{i_2},\ldots\underbrace{x_n,\ldots,x_n}_{i_n}}^{d}). \tag{5.19}$$

The collection of parameters corresponding to a control point and used to evaluate a blossomed function is referred to as an *argument bag*. Recall the i_k for $k = 1 \ldots n$ are the components of the multi-index $\vec{i}$ and should not be confused with our arbitrary choice of edge i on the S-patch.

Setting all the control points for the S-patch in this way will give us an

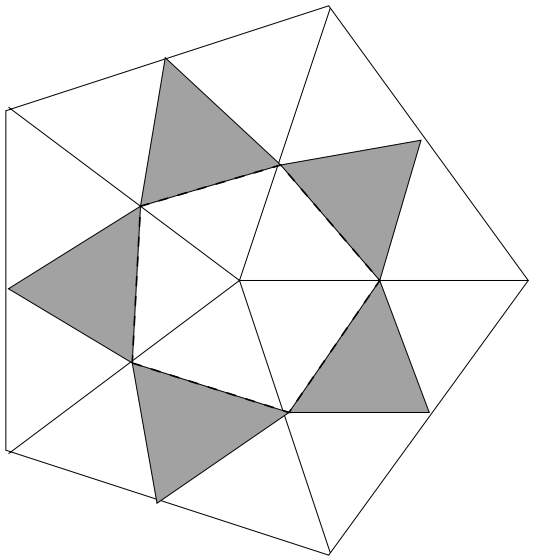


Figure 5.8: An example of 5 triangular patches meeting at an extraordinary vertex. The middle refinements are shown shaded and the boundary of the S-patch is shown with a dotted line.

adjacent patch which defines the exact same surface. If the S-patch is of equal or higher degree than the triangular patch then we will have C^∞ continuity at the join.

In our case, we want to match continuity with n patches for a degree n extraordinary vertex. The n patches we use are the center refinements of the extraordinary patches sharing the extraordinary vertex (see Figure 5.8). The center refinement not having any extraordinary vertices can be treated as a quartic triangular Bézier patch.

Through our choice of where to fit the replacement surface we define the four sided domain boundary of all extraordinary patches. The domain polygon has points $P_1 = (0,1), P_2 = (0,\frac{1}{2}), P_3 = (\frac{1}{2},0)$ and $P_4 = (1,0)$. Note that $(0,0)$ is always the domain point corresponding with the extraordinary vertex. Thus, each extraordinary patch at a given extraordinary vertex shares its P_2P_3 edge with an edge of the S-patch.

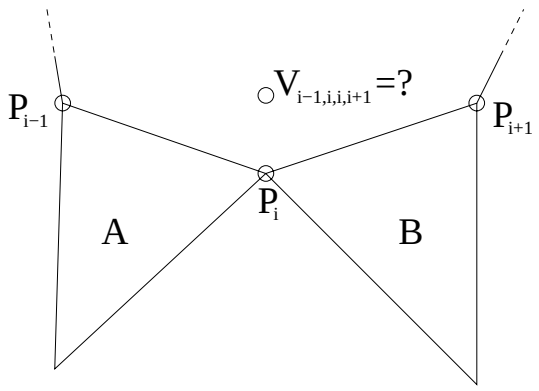


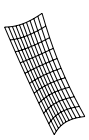
Figure 5.9: A non uniquely determined control point. This control point is at a distance of one from a boundary of both triangular patches A and B.

C^k continuity is maintained by adjusting the positions of control vertices which are within a distance of k of the boundary. That is these control points correspond to argument bags with a maximum of k points which are not endpoints of the shared edge.

By choosing to maintain C^2 continuity between the degree four S-patch and the surrounding Loop patches, we completely determine the shape of the S-patch. Although all control points are defined by a boundary, they are not defined uniquely. Consider two adjacent edges, $P_{i-1}P_i$ and P_iP_{i+1} as shown in Figure 5.9. The two edges border on different patches, but both patches will want to set the control point corresponding to the argument bag $(P_{i-1}, P_i, P_i, P_{i+1})$ because the control point corresponding to this bag is at a distance of one from both boundaries.

Our solution is to set these control points which have multiple possible settings on a first come first serve basis. The desired shape is preserved since the patches adjacent to our two consecutive edges are C^1, regardless which patch we use to set the control point.

We can demonstrate why this happens using a property of the Bézier curve control polygon. Consider a corner vertex where two triangular patches meet an



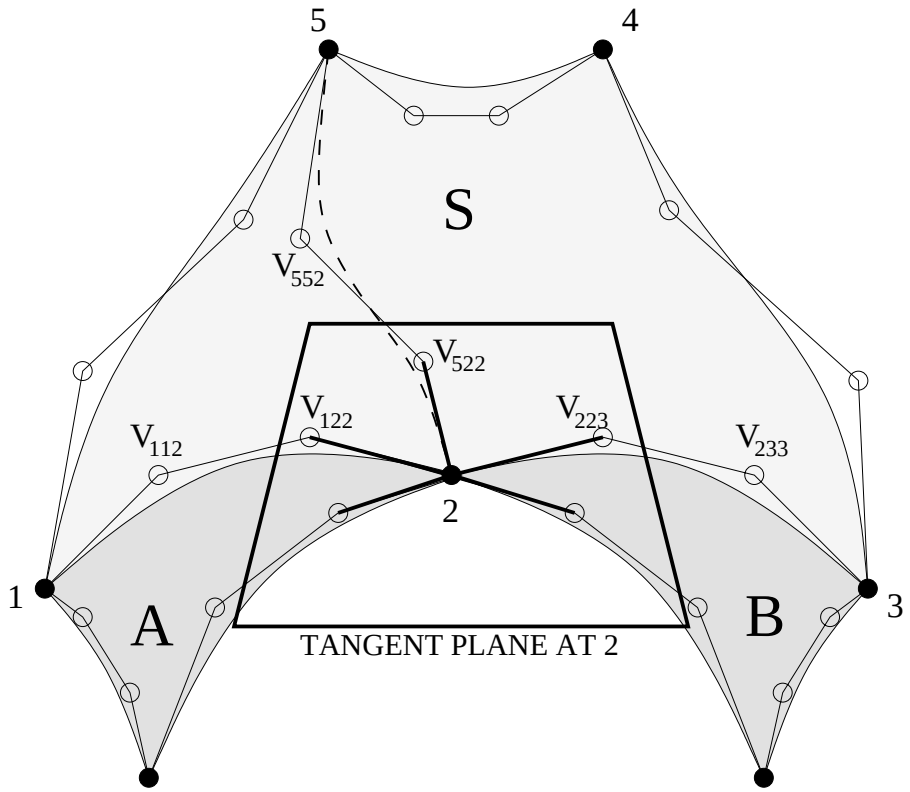


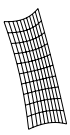
Figure 5.10: The dark rectangle shows the tangent plane at vertex 2. Corner control points are shown as black circles while boundary control points are shown as empty circles. For clarity, this diagram shows the case of cubic triangular patches. The interior control points of the triangular patches are omitted for clarity. The segment of all the boundary Bézier curves involving vertex 2 lie in the tangent plane. This is also true for the curve defined by $V_{555}, V_{552}, V_{522}, V_{222}$ even though it may not actually lie on the five sided S-patch.

S-patch as shown in Figure 5.10. The corner vertex and the control points at a distance of one on the triangular patches will all lie in the same plane if the surface is G^1 (there will be additional restrictions on their position if it is C^1). This is because the derivative of a Bézier curve at its endpoint is a multiple of the second last control point subtracted from the last control point. The multiple is equal to the order of the curve. All the control points of the S-patch at a distance of one from this corner vertex must also lie in the plane because of how we computed them. Consider the curve whose control polygon connects connecting the simplex vertex corresponding with the corner vertex to any other vertex in the simplex (vertex 2 to vertex 5 in the figure). This curve will be continuous at the join even though the curve as a whole may not lie in the S-patch. The control points will be the same as those we would compute if we were matching continuity for this curve alone. This is true for each curve corresponding to each direction v_i in the simplex. The result is that the embedding is irrelevant. A directional derivative taken at our join vertex (vertex 2 in the figure) with a direction s in the simplex will lie in the tangent plane. By definition,

$$\frac{\partial B}{\partial \mathsf{s}} \;=\; \hat{\mathsf{s}} \cdot \nabla B \;=\; \sum_{i=1}^{n} \frac{\partial B}{\partial u_i}\, \hat{\mathsf{s}}_i. \tag{5.20}$$

Evaluated at the corner vertex, each $\frac{\partial B}{\partial u_i}$ lies in the plane and thus their linear combination must lie in the plane too. Note that the function may not be regular if there is some combination of the partials equal to zero. This is only possible, however, with meshes having an irregular collection of obtuse angle triangles meeting at an extraordinary vertex. These meshes could be made suitable for surface replacement either through retriangulation or through moving control vertices.

The only side effect of our first come first serve rule is some minor twisting of equiparameter lines within the surface. This is not unexpected and can be seen in Figure 5.5. Although it may be possible to adjust the size and shape of the adjacent triangular domains T to achieve equal blossom evaluations for these shared control vertices, we have not explored this avenue. Averaging the control points may also



be a reasonable thing to do.

The only exception to the first come first serve rule is when a control point corresponds to an edge. In this case we let the adjacent patch define the position. If we did not do this then the control point corresponding to bag (P_i, P_i, P_i, P_{i+1}) on edge i, if set by the patch adjacent to edge $P_{i-1}P_i$ in order to maintain C^1 continuity, would break C^0 continuity along the edge P_iP_{i+1} leaving us a surface with a hole in it.

5.3 More on Computing Derivatives and Blossoms

Evaluating derivatives or blossoms of regular Loop patches is relatively easy. The terms of the basis polynomials have factors of the form u^i and v^j. For any given partial derivative we can simply compute the derivative or blossom of the terms and then evaluate the basis functions.

For example, any time u^4 appears as a factor within a term of a basis function we write `u4` instead. We do this for all powers of u and v. If we are evaluating a partial derivative with respect to u then we set `u4` to $4u^3$, `u3` to $3u^2$, and so on down to `u0` which we set to zero. When coding the basis polynomials it is important to include `u0` as a factor of terms which do not explicitly contain a power of u. For example, the term v^3 should be written as `v3*u0` if we are to have correct derivative computations.

This method also works for extraordinary patches since they use the same basis functions. We should mention, however, that we had a problem with derivative computations of extraordinary Loop patches due to a small error in [51]. Stam points out that the magnitude of derivatives must be corrected to take into account a remapping of the domain. This involves a sign change if the point evaluation is performed within a triangular patch which points in the opposite direction of the original patch. This sign change is not necessary for even derivatives, however, as applications of the chain rule will provide and even number of negations. The code

in the paper negates the result if the order of differentiation is odd but only if the number of subdivisions necessary to put the evaluation point in a regular patch is also odd.

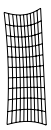
The evaluation of blossoms is achieved in a very similar way. Each factor is replaced with a symmetric affine polynomial whose diagonalization is equal to the factor. Blossoms of quartic Loop patches have four parameters. Instead of u we now have u_1, u_2, u_3, and u_4. For the factors corresponding to powers of u we set our variables as follows.

$$\begin{aligned}
\texttt{u0} &= 1 \\
\texttt{u1} &= \frac{1}{4}(u_1 + u_2 + u_3 + u_4) \\
\texttt{u2} &= \frac{1}{6}(u_1u_2 + u_2u_3 + u_3u_4 + u_1u_3 + u_2u_4 + u_1u_4) \\
\texttt{u3} &= \frac{1}{4}(u_1u_2u_3 + u_1u_2u_4 + u_1u_3u_4 + u_2u_3u_4) \\
\texttt{u4} &= u_1u_2u_3u_4
\end{aligned}$$

The factors corresponding to powers of v are set in the same manner.

Computing the blossom of an extraordinary patch, however, does not give expected results. Recall that we match continuity with the center refinement of an extraordinary patch. We tried to join an edge of an S-patch with the line segment $(0, \frac{1}{2})(\frac{1}{2}, 0)$ within an extraordinary patch. This line segment corresponds to a quartic Bézier curve. The degree would be much higher for an arbitrary line segment, but since it is an edge of the center refinement we are guaranteed that it is only quartic and thus it will be possible for us to construct an adjacent quartic S-patch that matches continuity.

We expected that we could match continuity with the center refinement in this manner without performing a mesh refinement to find the control points of the center refinement. This naive approach, however, does not yield the desired control points. We believe the control points are displaced due to the conversion of the extraordinary Loop patch's control points into the eigenbasis. Although the correct



control points can likely be computed by applying the inverse transformation, we chose to use the center refinement because it is easier to compute.

5.3.1 Derivatives of S-patches

Because S-patches are defined by a rather complex multivariate rational polynomial, special care is required to compute partial derivatives correctly. We used basic automatic differentiation techniques to write this code by hand. We tested the resulting code by comparing evaluations of all the partial derivatives at random points with evaluations of the derivatives computed symbolically with Maple.

For each assignment statement involved in a computation, an automatic differentiation programs creates a new variable to represent the derivative of the variable on the left hand side of the assignment and creates a new assignment statement that computes the derivative value for the new variable. For example, if we want a partial derivative with respect to u, the assignment `t=1` will cause the automatic differentiation program to create the new variable `tu` and the preceding statement `tu=0`. The new statement must precede the original because a variable might be used on both sides of an equation. Consider the statement `t=t*x[i]`. This statement results in the creation of the preceding statement `tu = tu * x[i] + t * xu[i]`. For our implementation we need both u and v partial derivatives in all combinations up to order three.

The computation of the $\pi_i(p)$ functions defined in Equation 5.14 is a representative example of applying the automatic differentiation technique. The block of code in Figure 5.11 shows how we compute the $\pi_i(p)$ functions.

5.4 Implementation Description

We have implemented this algorithm in Java using Java3D for some of the data structures and for displaying the animation.

We have a custom scene description file format for describing a system. A

```
for ( int i = 0; i < N; i++ ) {
    piuuu[i] = 0.0;
    piuuv[i] = 0.0;
    piuvv[i] = 0.0;
    pivvv[i] = 0.0;
    piuu[i] = 0.0;
    piuv[i] = 0.0;
    pivv[i] = 0.0;
    piu[i] = 0.0;
    piv[i] = 0.0;
    pi[i] = 1.0;
    for ( int j = 0; j < N; j++ ) {
        if ( j == i || j == (i+N-1)%N ) continue;
        // note that these are simplified due to the constant zero
        // second and third order partial derivatives of alpha
        piuuu[i] = piuuu[i] * alpha[j] + 3 * piuu[i] * alphau[j];
        piuuv[i] = piuuv[i] * alpha[j] + piuu[i] * alphav[j] +
                   2 * piuv[i] * alphau[j];
        piuvv[i] = piuvv[i] * alpha[j] + 2 * piuv[i] * alphav[j] +
                   pivv[i] * alphau[j];
        pivvv[i] = pivvv[i] * alpha[j] + 3 * pivv[i] * alphav[j];
        piuu[i] = piuu[i] * alpha[j] + 2 * piu[i] * alphau[j];
        piuv[i] = piuv[i] * alpha[j] + piu[i] * alphav[j] +
                  piv[i] * alphau[j];
        pivv[i] = pivv[i] * alpha[j] + 2 * piv[i] * alphav[j];
        piu[i] = piu[i] * alpha[j] + pi[i] * alphau[j];
        piv[i] = piv[i] * alpha[j] + pi[i] * alphav[j];
        pi[i] = pi[i] * alpha[j];
    }
}
```

Figure 5.11: Code produced with automatic differentiation techniques for computing the $\pi_i(p)$ functions

scene description file contains information about the objects in a system, how they should be rendered, and their initial conditions. An example file can be found in Appendix B. We used `javacc` to create a parser for this file format which makes it easily altered or extended.

Rigid bodies in this file can be defined by either Loop subdivision surfaces or they can have a limited portion of their boundary defined by a bi-cubic Bézier patch.

The Loop subdivision surfaces are defined by meshes stored in *obj* format. An example mesh file can also be found in Appendix B. These *obj* files can be written by hand for simple objects. For larger objects, however, we find it more convenient to use a modelling package such as 3D Studio Max.

Our interface is minimal as we only use it to test the underlying algorithm. Commands can be typed in a text box to change how objects are displayed, and to modify properties in the simulation such as friction.

Interaction with the simulation is possible by activating a spring force which connects the center of mass of the free object to a point which the user controls with the mouse. We find this minimal form of interaction useful for testing.

We also have an option for saving a selected time range of an animation as a Renderman *rib* file for rendering. The rendered images can be much more compelling because they show shadows that help convey where the contact is occurring. Unfortunately there is currently no support for shadows in Java3D. The ability to render images from the simulation to a file is also useful for creating animations. The ability to easily save Java3D images to a file has only recently become possible.

Chapter 6

Simulation Results and Evaluation

We present simulation results with a focus on how much time is spent in various parts of the algorithm. This is important as we wish to show that this algorithm can be used at interactive rates. We will also describe several simulation sequences which should help demonstrate the success of our algorithm in simulating single contact evolution.

6.1 Timings

Running the HotSpot Java virtual machine on a 350MHz Pentium II machine the entire computation of $\ddot{q}$ takes about 1.2 ms. With this computation time the simulation can run at 15 frames per second without using all available CPU cycles. The bulk of the time is spent in Java3d code to draw the system.

Table 6.1 shows the time necessary to evaluate various types of surface functions along with all the necessary partial derivatives for the algorithm. We performed all our tests on a 350 MHz Pentium II running Java 1.3. All times reported in this section are measured as an average of ten thousand computations to give the HotSpot virtual machine sufficient opportunity to optimize our code. Although

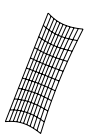


Table 6.1: Timings for calculating the ten partial derivatives necessary for the algorithm. Timings are with Hotspot optimizations, run on a 350MHz Pentium II. Times are an average of ten thousand successive computations and hence may appear smaller than they should due to caching effects.

surface type	evaluation time
Loop regular	0.2413 ms
Loop extraordinary (degree 3)	0.1683 ms
Loop extraordinary (degree 5)	0.1933 ms
Loop extraordinary (degree 7)	0.2223 ms
Loop extraordinary (degree 10)	0.2654 ms
S-patch (3 sides)	0.1632 ms
S-patch (4 sides)	0.4466 ms
S-patch (5 sides)	1.1006 ms
S-patch (10 sides)	20.0248 ms
Bi-cubic Bézier	0.0801 ms

timings in the table may be smaller than in practice due to caching effects, all other times reported in this section do not have this problem as they were measured during an actual simulation.

We found that evaluating the surface functions takes up a substantial portion of the time spent by the simulator. In our implementation these functions have quite a bit of room for optimization. Surface representations which are quick to evaluate are thus preferred for our algorithm. For two contacting cubic Bézier surfaces, the program computes H in 0.3485 ms on our test machine. Note that Table 6.1 reveals that half of this time is spent on surface evaluations of the two patches. The routine which computes $\ddot{q}$ takes about 0.235 ms (this includes the LU factorization and back substitution). The total computation time for the derivatives comes to about 0.9 ms in the Bézier on Bézier case. The time which is unaccounted for comes from checking boundaries, computing object positions, velocities, and external forces.

6.2 Bowl on Flat Frictionless Surface

The frictionless setting is useful for evaluating the correctness of a large part of our algorithm. Without friction, the center of mass of an object on a flat surface will not be able to have any horizontal translational motion. Observing purely vertical motion of the center of mass validates much of our implementation.

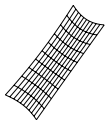
We simulated the system shown in Figure 6.1 consisting of an oddly shaped bowl sliding on a flat frictionless surface.

The bowl is a Loop subdivision surface while the flat surface is a Bézier patch. The bowl's center of mass is close to its base. We set gravity as the only external force in the simulation.

The upper strip shows a wire frame outline of each triangular patch so that the center of mass (the large sphere) and the contact point (the small sphere) can be seen. The triangular patch which is currently in contact is drawn with greater detail.

We observed that the center of mass has a purely vertical motion. This result is shown in Figure 6.2. Over time the horizontal position of the center of mass will change, however, as our numerical method has limited precision (this is not visible in the figure because of the error introduced by small cusps at S-patch boundaries). To prevent drift, we would either need to identify this special situation and formulate it as a three degree of freedom system, or we would need to use stabilization to keep the center of mass above the desired position in the plane.

The simulation will not handle the case of the bowl lying upside down on the table as the level lip of the bowl would establish contact with the flat plane all at once. In the simulation above, however, the system lacks the energy to reach this configuration because of the initial conditions.



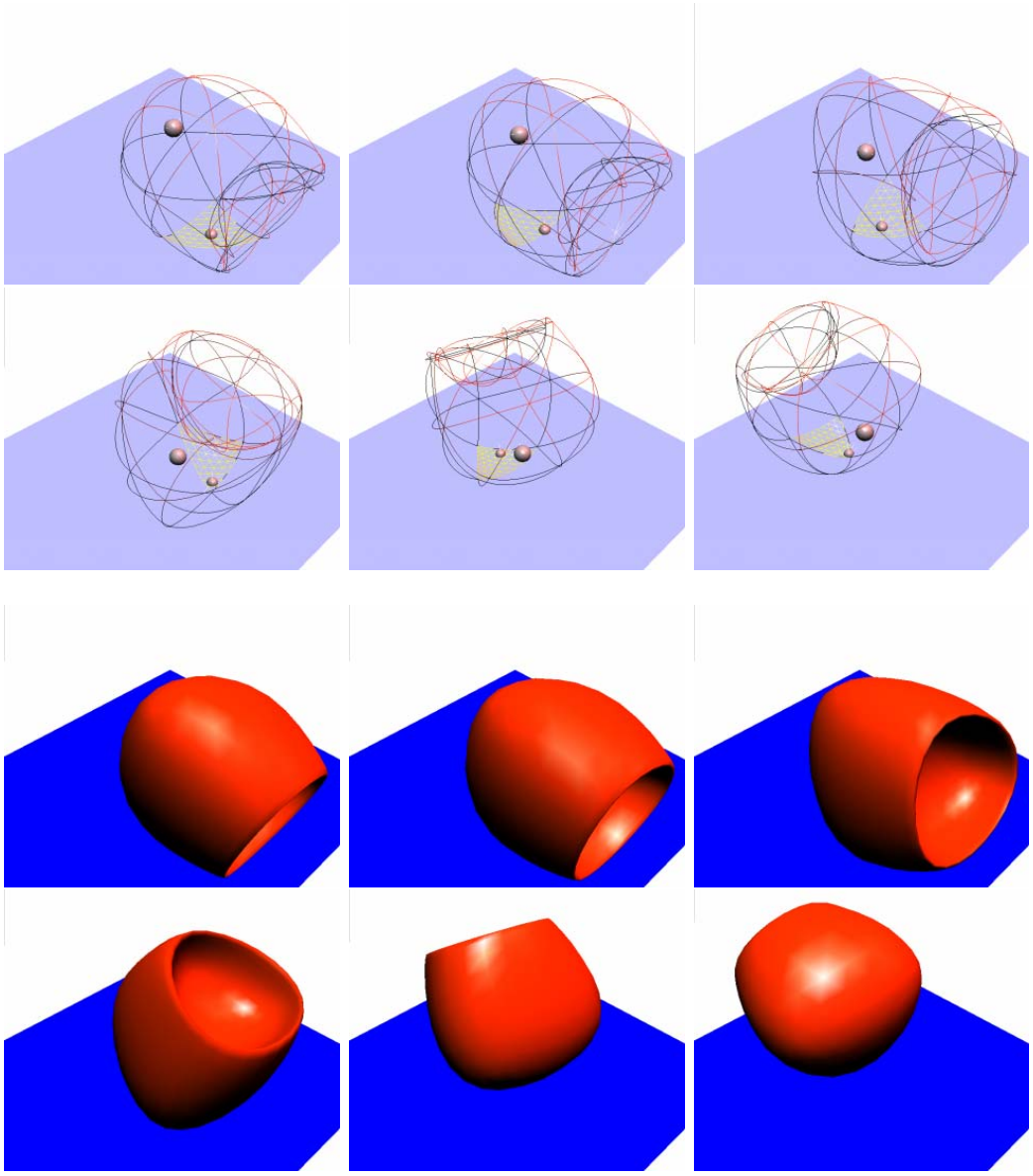


Figure 6.1: Simulation of a bowl sliding on a flat surface. Our system is validated by the vertical motion of the center of mass which can be seen as the larger sphere in the upper sequence. The smaller sphere shows the location of the contact point.

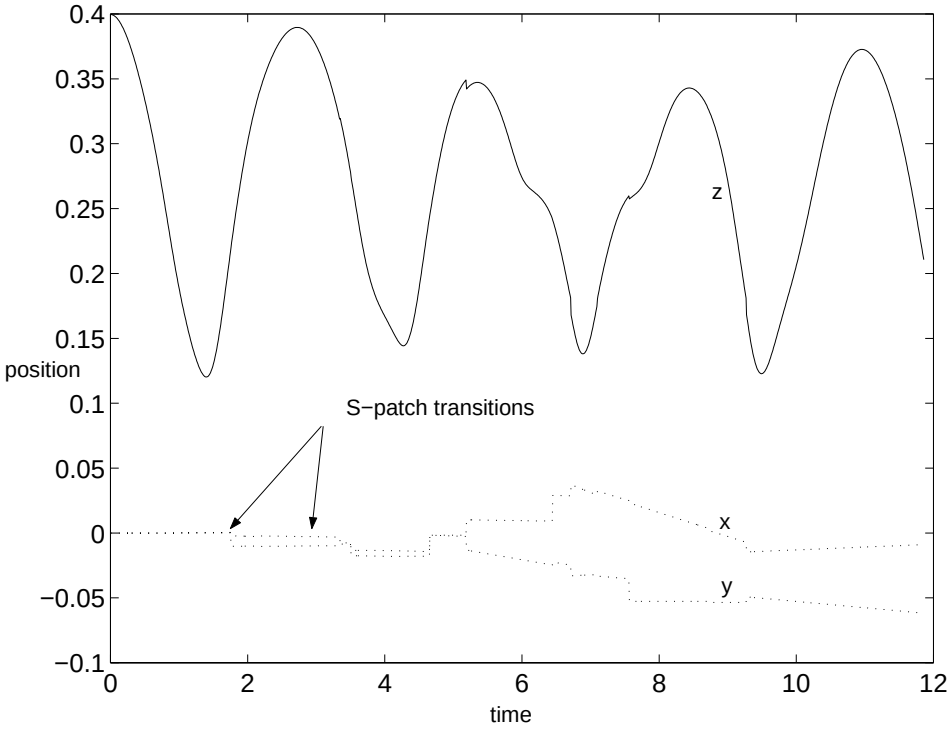
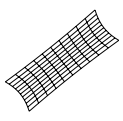


Figure 6.2: Graph showing the position of the center of mass in world coordinates over time. The solid line is the z position while the dotted lines are the x and y positions. The x and y positions do not move until a transition into an S-patch. This suggests a small problem with S-patch transitions. We attribute this problem to a cusp at the boundary due to an error in our implementation.



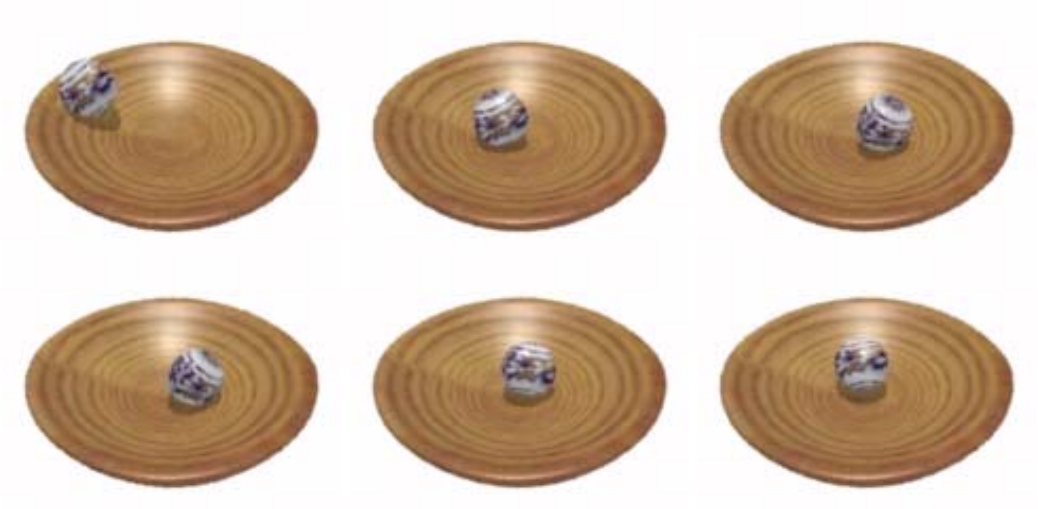


Figure 6.3: A lumpy marble rolling on a plate. Frames are one second apart. The free object can be seen to slide almost to the center of the plate before it starts to roll.

6.3 Blobs in Bowls

Because our simulator only handles single contact, we can only examine the small time segments involving a single continuous contact for interactions between arbitrary objects. An alternative is to restrict ourselves to objects which remain in contact and can only ever contact at a single point. This is an interesting test case because we can run our simulator for an arbitrary length of time.

Although these types of system may have only limited interest, they are more interesting than the curved-flat surface interaction in Section 6.2. They help validate our implementation as the matrix H is more complex in this case. The matrix in the flat surface case has more zero entries because the contribution of a flat surface's parameter velocities to ω is zero, except for ω_z which may be non-zero if the equiparameter lines of the flat surface curve within the plane.

Figure 6.3 shows an example of curved-curved contact between a lumpy marble and a shallow bowl. Both surfaces are Loop subdivision surfaces. Again gravity is the only external force, however due to friction the marble quickly starts to roll

Figure 6.4: Two different rattleback tops with asymmetric shape.

after a short period of sliding towards the center of the bowl.

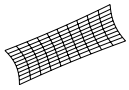
6.4 Rattleback Simulation

Spinning tops, in most cases, maintain a single continuous contact with the surface on which they are spinning. This makes tops an ideal object for us to simulate with our algorithm.

Rattleback tops, also known as celts or wobblestones, are interesting because they can reverse their spin. Some rattlebacks will reverse their spin in both directions while others have a spin bias and will only reverse their direction if spun in the direction opposite to their preferred direction.

As shown in Figure 6.4, rattlebacks have a long elliptical shape. Those in the figure have an asymmetric shape, but this is not necessary for spin reversal to occur.

When a rattleback is spun in the appropriate direction, its spin will slow and it will begin to wobble. The wobbling becomes larger until the top is no longer spinning and is instead rocking back and forth. The top then starts to spin in the



other direction and spins faster as the wobbles become smaller. It is due to the shape and/or inertial asymmetry of the top that the spinning energy is converted to a rocking motion and then back to a spinning motion.

Rattlebacks have been studied for a long time. The original paper which analyzed the motion of a rattleback appeared in 1896. More recently, Garcia and Hubbard in [16] explain how spin reversal can occur in one or both directions and discuss the limitations of the models in previous investigations. The limitations of previous work are mostly due to assumptions such as no-slip friction, assumed shape, and poor modelling of energy dissipation.

For our simulation we build a very simple rattleback model. Our model shown in Figure 6.4 consists of a single bi-cubic Bézier patch. It is longest in a direction 10 degrees off of the x axis and its width is one third of the length. Its curved shape comes from elevating the outer control points. We spin the model about its z axis. A scene description file including our rattleback model can be found in Appendix B.

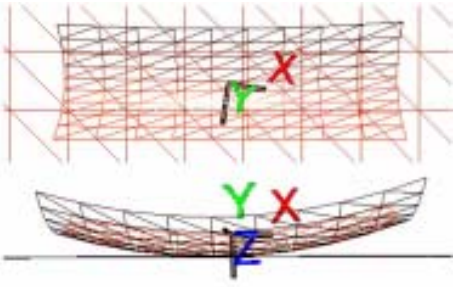


Figure 6.5: Our rattleback model, top and front views

Although the top is symmetric, we set the inertia tensor such that it is not in alignment with the planes of symmetry. In a physical model this can be achieved through a nonuniform mass density. The reversal effect is due to the misalignment of two of the principle axes of inertia with the principal axes of curvature at the point of contact.

With this simple model, using sliding friction and rotational dissipation we can simulate single or multiple spin reversals depending on the coefficients of rotational dissipation.

The preferred direction of spin can be found by giving it a tap on one end to

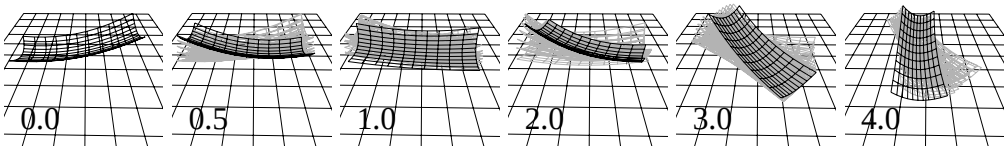


Figure 6.6: Tapping a rattleback to start it spinning. Previous positions are shown in grey. The top quickly starts spinning in a clockwise direction.

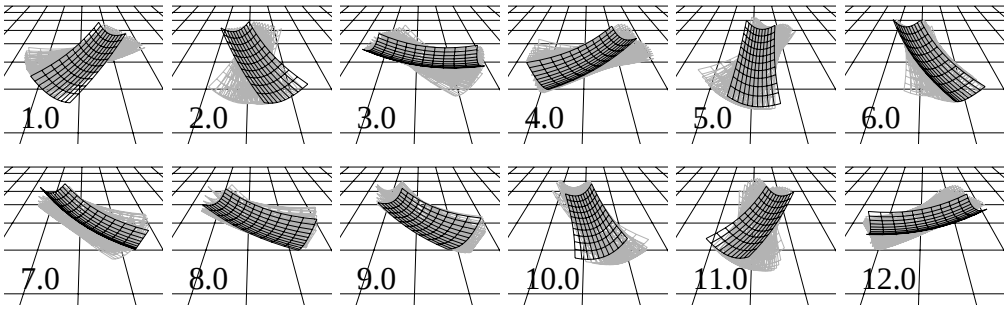


Figure 6.7: Spin start of a rattleback. Previous positions are shown in grey. The direction of the spin is initially counter clockwise and quickly changes to clockwise.

start it rocking. Even rattlebacks which reverse in both directions have a preferred spin direction. They take longer to reverse their spin when spinning in the preferred direction. Figure 6.6 shows a sequence of frames from a tap start, while Figure 6.7 shows a sequence of frames where it is spun to start. Several previous positions of the top are drawn in grey to give an indication of the direction of motion. In the latter simulation we start the spin slightly off the center of the top. This is necessary because a perfect spin about the z axis results in a stable rotation.

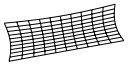
These figures have coefficients of friction set as follows.

$$\mu = 0.3$$

$$\mu_r = 0.0$$

$$\mu_s = 0.0$$

If we set $\mu_r = 0.003$ and $\mu_s = 0.0001$ we can prevent the second spin reversal. If all friction coefficients are set to zero we do not get any reversal at all.



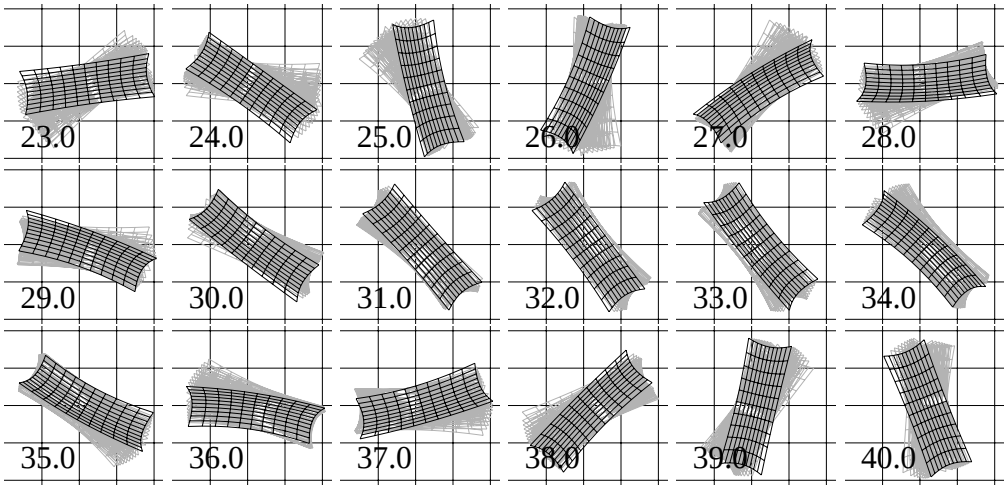


Figure 6.8: A sequence showing a second spin reversal. Previous positions are shown in grey. The top spins clockwise at 23 seconds and counter clockwise at 40 seconds.

Chapter 7

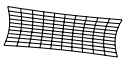
Conclusions

We have described a fast contact evolution algorithm for single contact between piecewise parametric surfaces. We first derived contact kinematics equations for arbitrary regular parametric surfaces. Then we showed that using these equations as acceleration constraints we can formulate the contact dynamics as an implicit ODE in the contact coordinates. The resulting ODE can be easily integrated using explicit integrators, without the need for constraint stabilization.

We can easily incorporate a Coulomb friction model into our formulation. Although our implementation of friction only provides sliding friction we find that we can still achieve a good approximation of no slip friction effects such as pure rolling. We also show how our formulation can be modified for no-slip and no-spin friction.

Because objects are made up of multiple patches, we derive a method for evolving a contact across patch boundaries. We also address special transitions such as going through a corner and propose extending patch boundaries by a small amount to increase stability through prevention of oscillating transitions.

We implement our algorithm using Loop subdivision surfaces. Unfortunately, the natural parameterization of Loop subdivision surfaces near extraordinary vertices is not regular. Our solution to this problem is to replace the portions of the



subdivision surface near extraordinary vertices with an n sided patch called an S-patch. This involves computing control points of the S-patch so that continuity is maintained at the join.

Several example simulations demonstrate the success of our formulation. We use a frictionless system to partially validate our implementation, and we use a simulation of a marble in a bowl to test our friction implementation. The results also show that our technique predicts the reversal behaviour of a rattleback top.

We also measured the time used performing various calculations for our simulation and discovered that a large proportion of the time, sometimes more than half, is spent computing partial derivatives of the parametric surfaces. This suggests that surfaces which are less expensive to evaluate are more desirable with our technique.

The main limitation of our algorithm is that we only treat single contact. Although only a small piece of the puzzle, we feel this method of dealing with smooth surface contact evolution is both an interesting and useful contribution to the rigid body dynamics community.

7.1 Future Work

Extending our implementation to handle other types of contact would make it much more useful. Flat conforming contacts as well as curve-surface and curve-curve contacts occur frequently in reality. Likewise, multiple contacts between two rigid bodies is a difficult problem which also occurs frequently and should be investigated. Lastly, an implementation capable of simulating many different kinds of contact would greatly benefit from a general framework for changing contact types.

Bibliography

[1] M. Anitescu, J. Cremer, and F. Potra. Formulating 3d contact dynamics problems. *Mechanics of Structures and Machines*, 24(4):405–437, 1996.

[2] U. Ascher. Stabilization of invariants of discretized differential systems. *Numerical Algorithms*, 14:1–23, 1997.

[3] U. Ascher, H. Chin, L. Petzold, and S. Reich. Stabilization of constrained mechanical systems with daes and invariant manifolds. *the Journal of Mechanics of Structures and Machines*, 23(2):135–157, 1995.

[4] U. Ascher, D. K. Pai, and B. Cloutier. Forward dynamics, elimination methods, and formulation stiffness in robot simulation. *International Journal of Robotics Research*, 16(6):749–758, December 1997.

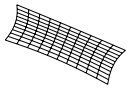
[5] D. Baraff. Analytical methods for dynamic simulation of non-penetrating rigid bodies. In *Proceedings of SIGGRAPH '89 (Boston)*, volume 23 of *Computer Graphics Proceedings, Annual Conference Series*, pages 223–232. ACM SIGGRAPH, July 1989.

[6] D. Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. *Computer Graphics*, 24(4):19–28, August 1990.

[7] D. Baraff. Issues in computing contact forces for non-penetrating rigid bodies. *Algorithmica*, 10:292–352, 1993.

[8] D. Baraff. Fast contact force computation for nonpenetrating rigid bodies. In A. Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 23–34. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[9] J. Baumgarte. Stabilization of constraints and integrals of motion in dynamical systems. *Computer Methods in Applied Mechanics and Engineering*, 1:1–16, 1972.



[10] C. Cai and B. Roth. On the spatial motion of rigid bodies with point contact. In *IEEE Conference on Robotics and Automation*, pages 686–695, 1987.

[11] J. F. Cremer. *An Architecture for General Purpose Physical System Simulation – Integrating Geometry, Dynamics, and Control*. PhD thesis, Cornell Univesity, 1989.

[12] T. DeRose. Subdivision surface course notes. SIGGRAPH Course Notes, 1998.

[13] D. DiFilippo. The AHI, an integrated audio haptics interface. Master's thesis, University of British Columbia, 2000.

[14] D. DiFilippo and D. K. Pai. The AHI: An audio and haptic interface for contact interactions. In *ACM Symposium on User Interface Software and Technology*, 2000.

[15] R. Featherstone. *Robot dynamics algorithms*. Kluwer, 1987.

[16] A. Garcia and M. Hubbard. Spin reversal of the rattleback: theory and experiment. *Proceedings of the Royal Society. London. Series A. Mathematical, Physical and Engineering Sciences*, 418(no. 1854):165–197, 1988.

[17] S. Goyal. Second order kinematic constraint between two bodies rolling, twisting and slipping against each other while maintaining point contact. Technical Report TR89-1043, Cornell University, Computer Science Department, October 1989.

[18] M. J. Greenberg and J. R. Harper. *Algebraic Topology, A First Course*, chapter Betti Numbers and Euler Characteristic. Number 58 in Mathematics lecture note series. Addison-Wesley, 1981.

[19] J. K. Hahn. Realistic animation of rigid bodies. *ACM Computer Graphics*, 22(4):299–308, 1988.

[20] G. Hegron. Rolling on a smooth bi-parametric surface. In *Eurographics Workshop on Animation and Simulation*, pages 205–213, 1991.

[21] A. Jain. Unified formulation of dynamics for serial rigid multibody systems. *Journal of Guidance, Control, and Dynamics*, 14(3):531–542, 1991.

[22] Y. Jia and M. A. Erdmann. Observing pose and motion through contact. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1998.

[23] D. E. Johnson and E. Cohen. A framework for efficient minimum distance computations. In *Proc. International Conference on Robotics and Automation*, pages 3678–3684. IEEE, May 1998.

[24] C. W. Kilmister and J. R. Reeve. *Rational Mechanics*. American Elsevier Publishing Company, Inc., 1966. Library of congress catalog card number 66-11823.

[25] Z. Li and J. F. Canny. Motion of two rigid bodies with rolling constraint. *IEEE Transactions on Robotics and Automation*, 6(1):62–72, 1990.

[26] C. T. Loop. Convex triangular subdivision surfaces with bounded curvature. Technical report, Microsoft Research, 2000.

[27] C. T. Loop and T. D. DeRose. A multisided generalization of bézier surfaces. *ACM Transactions on Graphics*, 8(3):204–234, July 1989.

[28] C. Lubich, U. Nowak, U. Pohle, and Ch. Engstler. Mexx – numerical software for the integration of constrained mechanical multibody systems. Tech. report SC92-12, Konrad-Zuse-Zentrum für Informationstechnik, Berlin., 1992.

[29] B. V. Mirtich and J. F. Canny. Impulse-based dynamic simulation of rigid bodies. In *Symposium on Interactive 3D Graphics*, 1995.

[30] D. J. Montana. The kinematics of contact and grasp. *International Journal of Robotics Research*, 7(3):17–32, 1988.

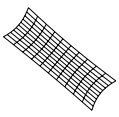
[31] D. F. Moore. *The Friction and Lubrication of Elastomers*, chapter Chapter 4, Rolling and Sliding. Pergamon Press Ltd., 1972.

[32] R. Murray, Z. Li, and S. S. Sastry. *A mathematical introduction to robotic manipulation*. CRC Press, 1994.

[33] Ju. I. Neimark and N. A. Fufaev. *Dynamics of Nonholonomic Systems*. American Mathematical Society, 1972.

[34] D. D. Nelson and E. Cohen. User interaction with cad models with nonholonomic parametric surface constraints. International Mechanical Engineering Congress and Exposition, November 1998. Haptic Symposium.

[35] D. D. Nelson, D. E. Johnson, and E. Cohen. Haptic rendering of surface-to-surface sculpted model interaction. In *Proceedings of the ASME Dynamic Systems and Control Division*, volume DSC-Vol. 67, pages 101–108, 1999.



[36] D. K. Pai, U. M. Ascher, and P. G. Kry. Forward dynamics algorithms for multibody chains and contact. In *IEEE International Conference on Robotics and Automation*, pages 857–863, 2000.

[37] J. Peters. Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. In *SIGGRAPH*, 1998.

[38] F. Pfeiffer and C. Glocker. *Multibody Dynamics with Unilateral Contacts*. Wiley series in nonlinear science. John Wiley and Sons. Inc., 1996.

[39] L. Piegl and W. Tiller. *The NURBS Book*. Monographs in Visual Communication. Springer, second edition, 1997.

[40] P. J. Rabier and W. C. Rheinboldt. On the numerical solution of the euler-lagrange equations. Technical report, University of Pittsburgh, February 1993.

[41] L. Ramshaw. Blossoming: A connect-the-dots approach to splines. Technical Report 19, Digital Systems Research Center, 130 Lytton Avenue, Palo Alto, California 94301, June 1987.

[42] L. Ramshaw. Blossoms are polar forms. Technical Report 34, Digital Systems Research Center, 130 Lytton Avenue, Palo Alto, California 94301, January 1989.

[43] R. E. Roberson and R. Schwertassek. *Dynamics of Multibody Systems*. Springer-Verlag, 1988.

[44] G. Rodriguez, A. Jain, and K. Kreutz-Delgado. A spatial operator algebra for manipulator modeling and control. *The International Journal of Robotics Research*, 10(4):371–381, 1991.

[45] D. F. Rogers. *An introduction to NURBS: with historical perspective*. Morgan Kaufmann, 2001.

[46] J. Sauer and E. Schömer. A constraint-based approiach to rigid body dynamics for virtual reality applications. In *ACM Symposium on Virtual Reality Software and Technology*, pages 153–161, 1998.

[47] J. E. Schweitzer. *Analysis and Application of Subdivision Surfaces*. PhD thesis, University of Washington, Seattle, Washington 98195, August 1996. Technical Report UW-CSE-96-08-02.

[48] T. W. Sederberg, D. C. Anderson, and R. N. Goldman. Implicit representation of parametric curves and surfaces. *Computer Vision, Graphics, and Image Processing*, 28:72–84, 1984.

[49] J. M. Snyder. An interactive tool for placing curved surfaces without interpenetration. In R. Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 209–218. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[50] J. M. Snyder, A. R. Woodbury, K. Fleischer, B. Currin, and A. H. Barr. Interval method for multi-point collisions between time-dependent curved surfaces. In *SIGGRAPH 93 Conference Proceedings*, pages 321–334. ACM SIGGRAPH, 1993.

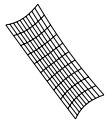
[51] J. Stam. Evaluation of loop subdivision surfaces. In *SIGGRAPH*, 1998. Included on course notes CD-ROM.

[52] J. Stam. Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. In *SIGGRAPH*, 1998.

[53] D. Stewart and J. C. Trinkle. An implicit time-stepping scheme for rigid body dynamics with coulomb friction. In *IEEE International Conference on Robotics and Automation*, pages 162–169, 2000.

[54] J. Warren. Subdivision methods for geometric design. Rice University, November 1995.

[55] A. Witkin and D. Baraff. Physically based modeling: Principles and practice. SIGGRAPH Course Notes, 1997.



Appendix A

Contact Kinematics Derivation

We use the same setup as described by Equations 4.5-4.8 in Section 4.2.

As the contact point moves, so do the contact frames. The spatial velocity of the contact frame *1c* relative to the body frame *1* in terms of contact parameter velocities, has a particularly simple form in the contact frame. If ${}^{1c}\phi(1,1c) = (\omega^T, v^T)^T$, then the spatial velocity of the contact frame *1c* relative to the body frame *1* in coordinates of *1c* is

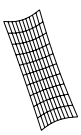
$$\begin{pmatrix} [\omega] & v \\ 0 & 0 \end{pmatrix} = {}^{1c}_{1}\mathsf{E}\ {}^{1}_{1c}\mathsf{E}_{,s}\ \dot{s} + {}^{1c}_{1}\mathsf{E}\ {}^{1}_{1c}\mathsf{E}_{,t}\ \dot{t} \tag{A.1}$$

Let Θ be the leading 3×3 sub-matrix of ${}^{1}_{1c}\mathsf{E}$, and we'll do the matrix multiplications in Equation A.1 to get a closer look at the contributions of $\dot{s}$ and $\dot{t}$ to ω and v. Recall that the rightmost column of ${}^{1}_{1c}\mathsf{E}$ is the location

$$[\omega] = \Theta^T\Theta_{,s}\ \dot{s} + \Theta^T\Theta_{,t}\ \dot{t}, \quad v = \Theta^T\mathsf{c}_{,s}\ \dot{s} + \Theta^T\mathsf{c}_{,t}\ \dot{t} \tag{A.2}$$

Lets first look at ω by examining the skew symmetric matrix $\Theta^T\Theta_{,s}$ ($\Theta^T\Theta_{,t}$ will be very similar).

$$\Theta^T\Theta_{,s} = \begin{pmatrix} \mathsf{x}\cdot\mathsf{x}_{,s} & \mathsf{x}\cdot\mathsf{y}_{,s} & \mathsf{x}\cdot\mathsf{z}_{,s} \\ \boxed{\mathsf{y}\cdot\mathsf{x}_{,s}} & \mathsf{y}\cdot\mathsf{y}_{,s} & \boxed{\mathsf{y}\cdot\mathsf{z}_{,s}} \\ \boxed{\mathsf{z}\cdot\mathsf{x}_{,s}} & \mathsf{z}\cdot\mathsf{y}_{,s} & \mathsf{z}\cdot\mathsf{z}_{,s} \end{pmatrix} \tag{A.3}$$



The least complicated expressions relating ω to $\dot{s}$ are those which are boxed in Equation A.3 ($\mathsf{x}_{,s}$ is less complicated than $\mathsf{z}_{,s}$ which is less complicated than $\mathsf{y}_{,s}$). Thus we can write ω as,

$$\omega = \begin{pmatrix} -\mathsf{y} \cdot \mathsf{z}_{,s} \\ -\mathsf{z} \cdot \mathsf{x}_{,s} \\ \mathsf{y} \cdot \mathsf{x}_{,s} \end{pmatrix} \dot{s} + \begin{pmatrix} -\mathsf{y} \cdot \mathsf{z}_{,t} \\ -\mathsf{z} \cdot \mathsf{x}_{,t} \\ \mathsf{y} \cdot \mathsf{x}_{,t} \end{pmatrix} \dot{t} \tag{A.4}$$

Now looking at the $\Theta^T c_{,s}$ and $\Theta^T c_{,t}$ components we can write,

$$\Theta^T \mathsf{c}_{,s} = \begin{pmatrix} \mathsf{x} \cdot \mathsf{c}_{,s} \\ \mathsf{y} \cdot \mathsf{c}_{,s} \\ \mathsf{z} \cdot \mathsf{c}_{,s} \end{pmatrix} = \begin{pmatrix} \mathsf{x} \cdot \mathsf{c}_{,s} \\ 0 \\ 0 \end{pmatrix} \quad \text{and} \quad \Theta^T \mathsf{c}_{,t} = \begin{pmatrix} \mathsf{x} \cdot \mathsf{c}_{,t} \\ \mathsf{y} \cdot \mathsf{c}_{,t} \\ \mathsf{z} \cdot \mathsf{c}_{,t} \end{pmatrix} = \begin{pmatrix} \mathsf{x} \cdot \mathsf{c}_{,t} \\ \mathsf{y} \cdot \mathsf{c}_{,t} \\ 0 \end{pmatrix}. \tag{A.5}$$

Equations A.4 and A.5 combine to give ${}^{1c}H_1$.

$${}^{1c}H_1 \stackrel{\text{def}}{=} \begin{pmatrix} -\mathsf{y} \cdot \mathsf{z}_{,s} & -\mathsf{y} \cdot \mathsf{z}_{,t} \\ -\mathsf{z} \cdot \mathsf{x}_{,s} & -\mathsf{z} \cdot \mathsf{x}_{,t} \\ \mathsf{y} \cdot \mathsf{x}_{,s} & \mathsf{y} \cdot \mathsf{x}_{,t} \\ \mathsf{x} \cdot \mathsf{c}_{,s} & \mathsf{x} \cdot \mathsf{c}_{,t} \\ 0 & \mathsf{y} \cdot \mathsf{c}_{,t} \\ 0 & 0 \end{pmatrix} \tag{A.6}$$

$${}^{1c}\phi(1, 1c) = {}^{1c}H_1 \begin{pmatrix} \dot{s} \\ \dot{t} \end{pmatrix} \tag{A.7}$$

We can analogously define ${}^{2c}H_2$. This transforms to frame *1c* as

$${}^{1c}H_2 = {}^{1c}_{2c}\mathsf{Ad}\ {}^{2c}H_2 = \begin{pmatrix} R_\psi & 0 \\ 0 & R_\psi \end{pmatrix} {}^{2c}H_2. \tag{A.8}$$

Finally, relative spatial velocity of the two contact frames is a pure rotation

about the surface normal, i.e.,

$$ {}^{1c}\phi(1c,2c) = \begin{pmatrix} 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \dot{\psi} \stackrel{\text{def}}{=} {}^{1c}H_{\psi}\dot{\psi} \tag{A.9} $$

We can now compute the contact matrices H_k as follows. The relative spatial velocity of the two bodies is given by

$$ \begin{aligned} \phi(1,2) &= \phi(1,1c) + \phi(1c,2c) + \phi(2c,2) \\ &= \phi(1,1c) + \phi(1c,2c) - \phi(2,2c). \end{aligned} $$

Substituting Equations A.7, A.8 and A.9 we have

$$ {}^{1c}\phi(1,2) = \begin{pmatrix} \vdots & \vdots & \vdots \\ {}^{1c}H_1 & -{}^{1c}H_2 & {}^{1c}H_{\psi} \\ \vdots & \vdots & \vdots \end{pmatrix} \dot{q} \stackrel{\text{def}}{=} {}^{1c}H\dot{q} \tag{A.10} $$

The contact matrix H can now be transformed to any convenient frame, for instance, frame *1*: ${}^{1}H = {}^{1}_{1c}\mathsf{Ad}\ {}^{1c}H$.

In Section 4.3 we need ${}^{1c}\dot{H}$ as we will take the time derivative of Equation A.10. Once ${}^{1c}\dot{H}_1$ and ${}^{2c}\dot{H}_2$ are computed with the current state (q and $\dot{q}$) using the chain and product rules, we can write

$$ {}^{1c}\dot{H} = \begin{pmatrix} \vdots & \vdots & \vdots \\ {}^{1c}\dot{H}_1 & -({}^{1c}_{2c}\dot{\mathsf{Ad}}{}^{2c}H_2 + {}^{1}_{1c}\mathsf{Ad}{}^{2c}\dot{H}_2) & 0 \\ \vdots & \vdots & \vdots \end{pmatrix}. \tag{A.11} $$

Appendix B

Example Files

This appendix shows example files used with our implementation. The scene description file format is the only new format that we use. Note that the subdivision surfaces use the standard *obj* format while stripified versions of these objects are stored in *objf* format as created by the program *stripe*. Texture files for Java3d and Renderman are in standard *jpeg* and *tiff*. Note that Renderman compliant rendering programs will generate the *txt* format texture files they need.

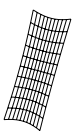
B.1 Scene Description File

Here is an example scene description file. Note there is a description of a Loop subdivision surface commented out.

```
// This file simulates a rattleback

view V1 {
    Translate [0 -0.355 1.2];
    Rotate -130 [1 0 0];
    ribFileName = "rb.rib";
    frameBaseName = "z:/pgkry/frames/rb";
    Format = "300 225 -1";  // size of the image... what is -1?
    PixelSamples = "2 2";
};

World RBSim {
    LightSource L1 ambientlight {
```

```
        intensity = 0.3;
        colour = [ 0.1 0.1 0.1 ];
        index = 0;
    };
    LightSource L2 arealight {
        point = [ 0 40 40 ];
        pointintensity = 1.0;
        colour = [ 1 1 1 ];
        intensity = 3585;
        polygonp = "[-10.0 10.0 40  10.0 10.0 40
                     10.0 -10.0 40  -10.0 -10.0 40]";
        index = 1;
    };

    /*
        Note that obj files have a .obj ending
        and stripfiles have a .objf ending
        also, if strip files of the form <stripfileroot><level>.objf
        don't exist then a <objfileroot><level>.obj will be created so
        that 'stripe' can be run.
     */
/*
    Object bowl Loop {
        scale = [ 0.05 0.05 0.05 ];
        offset = [ 0 0 0 ];
        objfileroot = "data/flatbowl";
        stripfileroot = "data/flatbowl";
        levels = 4;
        fixed = true;
        texture = "data/woodbowl.jpg";
        textureTiffFile = "data/woodbowl.tif";
        txtFileName = "data/woodbowl.txt";
    };
*/

    Object rattleback Bezier {
        scale = [ 0.6 0.2 0.2 ];
        rotate = -0.174 [ 0 0 1 ];
        offset = [0 0.0 0.2];
        mass = 5;
        inertia = [ 1 0 0 ] [ 0 10 0 ] [ 0 0 10 ];
        fixed = false;
        controlp =
    [ -1.5 -1.5 -2.0] [ -0.5 -1.5 -1.0] [ 0.5 -1.5 -1.0] [ 1.5 -1.5 -2.0]
    [ -1.5 -0.5 -1.0] [ -0.5 -0.5  0.0] [ 0.5 -0.5  0.0] [ 1.5 -0.5 -1.0]
    [ -1.5  0.5 -1.0] [ -0.5  0.5  0.0] [ 0.5  0.5  0.0] [ 1.5  0.5 -1.0]
    [ -1.5  1.5 -2.0] [ -0.5  1.5 -1.0] [ 0.5  1.5 -1.0] [ 1.5  1.5 -2.0];
    };
```

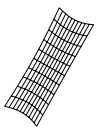
```
    Object floor Bezier {
        scale = [ 1.0 1.0 1.0 ];
        offset = [0 0 0];
        fixed = true;
        controlp =
    [ -1.5 -1.5 0.0] [ -0.5 -1.5 0.0] [ 0.5 -1.5 0.0] [ 1.5 -1.5 0.0]
    [ -1.5 -0.5 0.0] [ -0.5 -0.5 0.0] [ 0.5 -0.5 0.0] [ 1.5 -0.5 0.0]
    [ -1.5  0.5 0.0] [ -0.5  0.5 0.0] [ 0.5  0.5 0.0] [ 1.5  0.5 0.0]
    [ -1.5  1.5 0.0] [ -0.5  1.5 0.0] [ 0.5  1.5 0.0] [ 1.5  1.5 0.0];
    };

    Contact C2 {
        objectA = floor;
        objectB = rattleback;
        codeA = "oponoponoponoponoponoponopono";
        codeB = "onnononno";
        // tap start
//      coords = 0.15 0.5 0.5 0.5 0;
//      dcoords = 0 0 0 0 0;
        // spin start
        coords = 0.45 0.45 0.5 0.5 0;
        dcoords = 0 0 0 0 -1;
        // this works... but be sure to use
        // wmuxy = 0 and
        // wmuz = small (0 or -0.0001 is goo)
    };
};
```

B.2 Subdivision Surface File

Subdivision surface files use the *obj* file format. Here is the definition of the bowl seen on the left in Figure 4.3.

```
v -1.0 0.0 1.0
v 0.5 -0.866 1.0
v 0.5 0.866 1.0
v -1.0 0.0 0.0
v 0.5 -0.866 0.0
v 0.5 0.866 0.0
v -2.0 0.0 -1.0
v 1.0 -1.732 -1.0
v 1.0 1.732 -1.0
```

```
f 9 8 7
f 4 5 6
f 4 6 3
f 3 1 4
f 6 5 2
f 2 3 6
f 5 4 1
f 1 2 5
f 2 1 7
f 7 8 2
f 1 3 9
f 9 7 1
f 3 2 8
f 8 9 3
```