A Fast and Robust Six-DoF God Object Heuristic for Haptic Rendering of Complex Models with Friction

Mikel Sagardia* and Thomas Hulin
German Aerospace Center (DLR), Institute of Robotics and Mechatronics
Wessling, Germany

Abstract

Collision detection and force computation between complex geometries are essential technologies for virtual reality and robotic applications. Penalty-based haptic rendering algorithms provide a fast collision computation solution, but they cannot avoid the undesired interpenetration between virtual objects, and have difficulties with thin non-watertight geometries. *God object* methods or constraint-based haptic rendering approaches have shown to solve this problem, but are typically complex to implement and computationally expensive. This paper presents an easy-to-implement *god object* approach applied to six-DoF penalty-based haptic rendering algorithms. Contact regions are synthesized to penalty force and torque values and these are used to compute the position of the *god object* on the surface. Then, the pose of this surface proxy is used to render stiff and stable six-DoF contacts with friction. Independently of the complexity of the used geometries, our implementation runs in only around 5 μs and the results show a maximal penetration error of the resolution used in the penalty-based haptic rendering algorithm.

Keywords: haptic rendering, virtual assembly, haptic devices, interaction techniques

Concepts: •**Software and its engineering** → **Virtual worlds training simulations;** •**Human-centered computing** → *Haptic devices;* •**Computing methodologies** → *Virtual reality;*

1 Introduction

Haptic rendering algorithms bring via haptic interfaces the sense of touch to the user who interacts in virtual manipulations. This helps comprehend more intuitively the assembled geometries, leading to increased accuracies [Sagardia et al. 2012]. One of the major challenges of haptic rendering is the fast computational speed required for stability: every 1 ms contacts must be processed and displayed to the user [Basdogan and Srinivasan 2002], which often leads to approximations and trade-off situations.

The majority of developed algorithms can be classified into three main paradigms depending on their force computation principle: impulse-based, penalty-based, and constraint-based approaches. Impulse-based approaches explicitly modify the velocity of objects when contacts are detected by applying small impulses, as done by [Mirtich 1996]. Penalty-based methods, on the other hand, detect the overlapping error (usually penetration or volume) to either compute a force or simulate a plausible motion upon contact using the Newton-Euler equations. A well known example is the Voxelmap-Pointshell (VPS) Algorithm presented by McNeely et al. [McNeely et al. 1999]. Finally, constraint-based approaches such as one the presented by Zilles and Salisbury [Zilles and Salisbury 1995], prevent overlap between objects. Basically, the user controls the *device pose* or the *haptic tool*, but a *proxy* or so-called *god object* is visualized. Even though the device object would go through the other geometry, that *god object* would always remain on the surface boundary. In this sense, the focus lies on determining the constrained movement of the object. Forces are rendered out of the difference between the *device* and the *proxy* pose.

Our work presents a constraint-based *god object* haptic rendering algorithm that can be applied to any penalty-based method. As a result, usually fast and easy to implement penalty-based approaches can benefit from more stable and harder contacts common for constraint-based algorithms. In addition, the tunneling effect or pop-through issues that typically arise when interacting with thin surfaces using penalty-based algorithms are overcome.

1.1 Related Work

To the best of our knowledge, Zilles and Salisbury coined the term *god object* for their constraint-based three-DoF haptic rendering algorithm [Zilles and Salisbury 1995] in 1995. This method gives rise to a series of works based on optimization approaches. In case of contact, in order to obtain the pose of the proxy on the surface, the authors minimized the energy of a spring between the penetrating point linked to the haptic device (known) and a parametrized proxy point, constrained to the collision plane.The problem is easily solvable by using the Lagrange multipliers. Similarly, Ruspini et al. [Ruspini et al. 1997] minimized the distance between the penetrating point and the proxy, but constrained the region outside of several contact half-planes. In both cases, the idea is related to the Gauss' principle of least constraints [Gauss 1829], which states that the motion of a mechanical system satisfies the minimum of the differences' norm between the constrained (proxy) and unconstrained (haptic device) accelerations. Redon et al. [Redon et al. 2002b] analyzed the advantages of this principle for rigid body simulations and Ortega et al. [Ortega et al. 2006] applied it for six-DoF haptic rendering. The method proposed by the last authors computes the force rendering in a separate asynchronous thread in order to achieve the 1 kHz update rate necessary for haptic interaction [Basdogan and Srinivasan 2002], since the used continuous collision detection [Redon et al. 2002a], in combination with the god object pose simulation, exceeds that performance threshold when contact regions increase. This decoupling opens up the possibility to testing other collision detection methods and experimenting with simplifications in the proxy pose computation.

In recent years, several optimization-based approaches have also been presented. Chan et al. [Chan et al. 2011] applied the Gauss' least constraints principle for six-DoF haptic rendering using volumetric medical imaging and point clouds formed by unordered ob-

*e-mail: mikel.sagardia@dlr.de

VRST '16,, November 02 - 04, 2016, Garching bei München, Germany
ISBN: ACM 978-1-4503-4491-3/16/11
DOI: http://dx.doi.org/10.1145/2993369.2993374

ject vertices. Rydén and Chizeck [Rydén and Chizeck 2013] applied the same basic principle of rigid body mechanics to streamed point clouds and voxelmaps to obtain six-DoF haptic rendering. Wang et al. [Wang et al. 2013] adapted the quadratic programming approach from [Ortega et al. 2006] to the six-dimensional configuration space. These authors use a sphere-based collision detection first and then minimize the distance between the proxy and the device.

It is also possible to constrain the god object to the surface without explicitly formulating the task as an optimization problem. In this sense, Salisbury and Tarr [Salisbury and Tarr 1997] presented a very interesting three-DoF haptic rendering algorithm for implicit surfaces. The method works as follows: first, when the device point penetrates the surface, its closest surface point is detected with a deepest descend algorithm, and a support tangent plane is computed on it; then, in each cycle, the projection of the device point on the tangent plane is computed, which leads to the closest surface point using the same deepest descend algorithm. The tangent plane is updated every cycle. Displayed forces are related to the distance between the device and the surface point. This algorithm has been exploited in recent years for three-DoF haptic interaction with streamed point clouds by Leeper et al. [Leeper et al. 2012], and with deformable volumetric medical image data by Chan et al. [Chan et al. 2013]. Our heuristic approach works on a similar idea to the one presented by Salisbury and Tarr to six-DoF haptic rendering.

Besides collision forces, friction is an important contact phenomenon which contributes to manipulation realism, particularly in virtual interactions with haptic feedback. We refer the reader to [Andersson et al. 2007] for a brief but through glimpse on basic concepts related to this topic. That work describes and presents simulation results of several friction models applied to a one-DoF system. Additionally, most important phenomena, model properties, advantages and disadvantages are discussed.

Unfortunately, many collision, movement and friction simulation methods from computer graphics such as the work in [Kaufman et al. 2005] usually require longer computation times than the 1 kHz necessary in haptics. Therefore, simplifications or heuristics are required. Hayward et al. [Hayward and Armstrong 2000] presented a very complete three-DoF friction model suited for haptic rendering. They improved the Dahl friction model cancelling the drifting effect and provided a set of useful approximations. Their model yields the four friction regimes observed in physical reality: sticking, creeping, oscillating, and sliding. Harwin and Melder [Harwin and Melder 2002] presented a three-DoF friction computation method similar to the one introduced in [Salisbury and Tarr 1997]. The method is easy to implement and applied upon the *god object* algorithm presented in [Zilles and Salisbury 1995]: a cone is placed on the penetrating *device* point and the *proxy* is allowed to move until the boundary of the intersection between the cone and the surface. This approach allows for static (dry) and kinetic (sliding) friction. Kawasaki et al. [Kawasaki et al. 2011] extended the previous approach to six-DoF. Their method is able to compute friction moment based on the torsion angle between the *god* and *device* reference frames. Additionally, they implemented the model in a hand-finger force feedback device to provide finger torque friction and conducted a user study on torque friction perception.

1.2 Overview and Key Contributions

Our heuristic operates in the configuration space (six-dimensional pose) of rigid bodies with arbitrary geometry and provides with six-DoF constraint (frictional) forces and correct *proxy* pose simulation with 1 kHz. Upon contact, the *proxy* is constrained to the surface using the forces and torques and the penetration depth computed by a penalty-based algorithm. These penalty forces inherently model contact geometry and, thus, restrict the motion of the object. Our contributions (and the overview of the paper) are summarized as follows:

- A six-DoF *god object* simulation and constraint force computation heuristic which is fast, robust and easy to implement on any penalty-based haptic rendering algorithm that provides with signed distances (Section 2).
- A six-DoF friction model applied to our *god object* heuristic that comprises static, kinetic viscous friction regimes (Section 2.7).
- Experimental results that show the behavior of our method in several usual and worst-case scenarios (Section 3).

As we conclude in Section 4, following the implementation steps we provide, it is possible to easily convert virtually any penalty-based haptic rendering algorithm to be a constraint based approach which benefits from the advantages of both paradigms: ease, speed, stability, and stiffness.

2 God Object Heuristic

This section presents step by step our *god object* simulation and force rendering method giving implementation details. Nevertheless, we omit initializations and division-by-zero, saturation, and similar checks for the sake of clarity. Furthermore, we consider the two-object scenario: the first object is moved by the user via the haptic device with respect to the second one; if we observe the relative movements, we can assume the second object stands still without loss of generality – although it may actually be moving.

As convention, bold capital symbols ($\mathbf{H}$) denote homogeneous transformation matrices in $\mathbb{R}^{4\times4}$, bold small symbols ($\mathbf{x}$, $\mathbf{h}$) vectors in $\mathbb{R}^3$ or $\mathbb{R}^6$, and small italic symbols (p) scalars in $\mathbb{R}$. Points, lines and surfaces in $\mathbb{R}^3$ are denoted with capital italic symbols (P, L) and a vector between two given points P and Q is denoted $\overrightarrow{PQ}$. In the case of poses (translation and rotation), we use the matrix representation for homogeneous coordinate transformations and the vector representation for all other transformations, as we believe this notation helps understand our method more intuitively. Values are transformed from one representation to another with functions like `setMatrix()`, `getRotation()` or `getTranslation()`. Additionally, a transformation from the coordinates W to D is denoted ${}^{\mathrm{W}}\mathbf{H}_{\mathrm{D}}$. All values correspond to the current cycle (k) except when properly indicated (e.g., $\mathbf{H} = \mathbf{H}(k)$ vs. $\mathbf{H}(k-1)$).

As shown in Figure 1, there are three main frames:

- W The *world frame*, which we will consider fixed in the center of mass of the still object.
- D The *haptic device frame*, which corresponds to the end-effector of the device moved by the user.
- S The *proxy, god* or *surface frame*, which corresponds to the object that remains on the surface. This frame is corrected to solve its penetration p, which leads to S'.

We also use the *body frame* B when deducing mass distribution properties of the object; the original geometry is supposed to be defined in this coordinates, located in its center of mass G. In the same line, the *eigen frame* E results from performing a principal component analysis of the body.

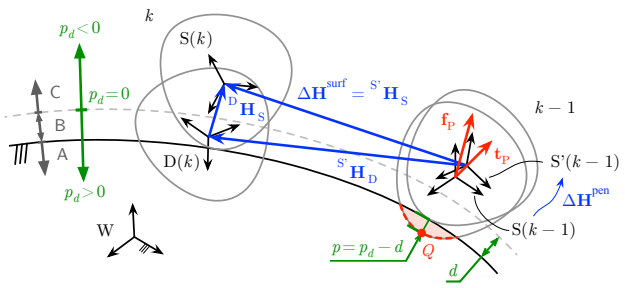

Figure 1: *Overview of the* god object *simulation in two consecutive time steps $k-1$ (previous) and k (current). Main frames corresponding to the* world (W)*, the* device (D) *and the* proxy, god object or surface (S) *are displayed. In our method, first the previous* surface *frame is corrected to* $\mathrm{S}'(k-1)$*. Then, the motion from this* $\mathrm{S}'(k-1)$ *to the current* device *pose* $\mathrm{D}(k)$ *is constrained with the force and torque values* $\mathbf{f}_{\mathrm{P}}, \mathbf{t}_{\mathrm{P}}$ *computed by the penalty-based collision detection algorithm. This yields the new current* god object *pose* $\mathrm{S}(k)$.

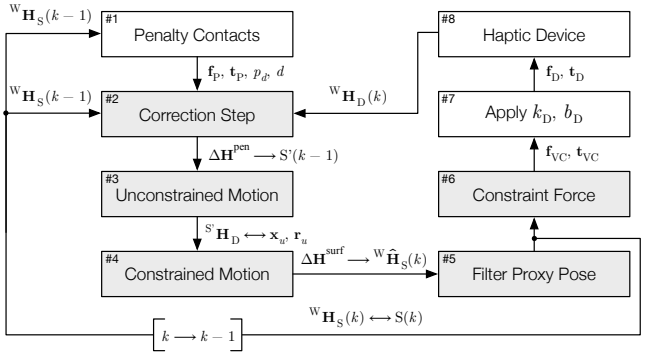

Figure 2: *Workflow of our* god object *simulation and force rendering method. Shadowed boxes #2 – #6 are core steps that define the our approach; the other steps could be changed without altering considerably the result, particularly our approach is suited for other penalty-based contact rendering algorithm (step #1). The whole procedure is repeated every 1 ms, being the contact computation (step #1) the one which lasts longer. Note that the procedure is fed with the* god object *pose of the previous cycle* $^{\mathrm{W}}\mathbf{H}_{\mathrm{S}}(k-1)$*, as well as the current device pose* $^{\mathrm{W}}\mathbf{H}_{\mathrm{D}}(k)$.

The goal of the method is to compute a virtual-constrained proxy pose $^{\mathrm{W}}\mathbf{H}_{\mathrm{S}}$ of the moved object with respect to the other (still) object given the coordinates of the haptic device's end-effector $^{\mathrm{W}}\mathbf{H}_{\mathrm{D}}$. $^{\mathrm{W}}\mathbf{H}_{\mathrm{D}}$ is the pose of the moved object commanded by the device, which can penetrate the other object. On the other hand, $^{\mathrm{W}}\mathbf{H}_{\mathrm{S}}$ is the pose of the *proxy or god-object* which tries to remain on the surface in case of collision. The computation of the *proxy* pose constrained to the surface results from restricting the transformation from the previous *proxy* pose to the current device pose with the penalty contact forces related to the *proxy*.

Figure 2 gives an overview of the whole procedure that is repeated every haptic cycle (1 ms). All eight steps depicted in it are described in detail in their respective subsections. We give a brief summary here to convey a global idea of the procedure:

#1 Penalty Contacts (Section 2.1): Penalty-based collision detection is performed using the previous *god object* pose. An important requirement is that we slightly dilate one object with a *safety margin* d to avoid real penetration between the objects approaching to each other and to obtain less noisy, more robust collision forces. The value of d could be optimized online, but we fix it to $d = 3$ mm in our simulations after empirical trials, since it already produces a stable a behavior. This step yields the signed distance or penetration value p_d and the penalty forces $\mathbf{f}_{\mathrm{P}}$ and torques $\mathbf{t}_{\mathrm{P}}$.

#2 Correction Step (Section 2.2): If the *god object* is penetrating ($p_d \geq d$) in the previous cycle, its corrected non-penetrating pose is computed, minimizing the error introduced in previous cycle. This step is the one that required the longest section in this work, but, still, it is the one with the fastest computation times, since analytical correction formulae for any geometry and contact configuration are derived.

#3 Unconstrained Motion (Section 2.3): The movement of the *god object* is computed as if no collision constraints were present.

#4 Constrained Motion (Section 2.4): The unconstrained motion is corrected with the contacts computed in step #1. The friction is also computed in this step #4 in the object configuration space (Section 2.7). The constrained motion leads to the current (unfiltered) *god object or proxy* pose.

#5 Filter Proxy Pose (Section 2.5): The current *god object* pose is smoothened with a low pass filter.

#6 Constraint Force (Section 2.6): Constraint forces $\mathbf{f}_{\mathrm{VC}}, \mathbf{t}_{\mathrm{VC}}$ are proportional to the difference between the current *device* and the *god object* pose, or, in other words, linear to the non allowed movement performed by the *device*. Friction forces are intrinsically considered (Section 2.7).

#7 Apply $k_{\mathrm{D}}, b_{\mathrm{D}}$ (Section 2.6): Stiffness and damping factors are multiplied to the constraint forces in order to achieve desired hard contact on device being still stable.

#8 Haptic Device (Section 2.6): Device forces $\mathbf{f}_{\mathrm{D}}, \mathbf{t}_{\mathrm{D}}$ are commanded to the haptic device and the pose of the end-effector is read every 1 ms.

The coordinates of the deepest colliding point Q are unknown but its penetration p_d is provided by the penalty-based collision computation. Note in Figure 1 that three regions are distinguished on its value:

A, $p_d \geq d$: There is overlapping between objects. If the *unconstrained* motion of the *device* frame moves in opposite direction of the penalty forces $\mathbf{f}_P$ and torques $\mathbf{t}_P$, it must be *constrained* to the surface.

B, $0 < p_d \leq d$: There is no overlapping between objects but the deepest colliding point Q is inside the *safety layer*, which has a width d over the surface. The approach is similar to the previous case, except for slight modifications in several steps, properly indicated.

C, $p_d < 0$: There is no overlapping between objects. In this case, the *god object* pose is the *device* pose, $\mathrm{S}(k) = \mathrm{D}(k)$, and therefore, there is no constraint coupling force to display, $\mathbf{f}_{\mathrm{VC}}, \mathbf{t}_{\mathrm{VC}} = \mathbf{0}$.

We will consider the cases in which Q lies on either A or B, since the last case of the region C has the mentioned trivial solution.

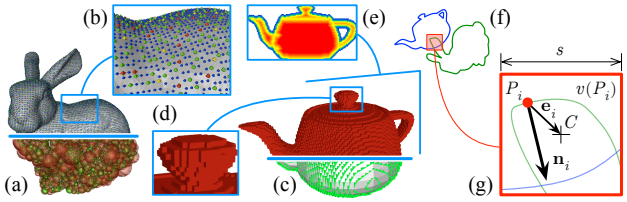

Figure 3: *Data structures of the penalty-based collision computation algorithm used. (a) Two sphere hierarchy levels of a Stanford Bunny; (b) Several point hierarchy levels; (c) Two voxel layers of a Utah Teapot; (d) Section of the signed distance field; (f) – (g) A colliding point P_i with normal $\mathbf{n}_i$ in a voxel with voxel layer value $v(P_i)$, center C and voxel edge size s.*

2.1 Penalty-Based Contact Computation (#1)

The first step is accomplished performing a penalty-based collision computation based on the Voxelmap-Pointshell (VPS) algorithm [McNeely et al. 1999], which was improved by several authors [Barbič and James 2008], [Sagardia et al. 2014].

Figure 3 shows the data structures we generate for any complex geometry (non-convex, hollow, millions of triangles) and the basic principle the algorithm works with achieving computation frequencies higher than 1 kHz.

In a colliding object pair, one of them is a *voxelmap* or a signed distance field embedded in a voxel grid, where each voxel contains a voxel layer value $v \in \mathbb{N}$; this value encodes the distance to the surface, being $v = 0$ for voxels on the surface, $v < 0$ outside of the object, and $v > 0$ inside. The other object is a *pointshell* or point-sphere tree with points and normals (or 6D points) that represent the surface of the object organized in a hierarchy. Each hierarchy level samples the whole object with a different resolution. Minimally bounding spheres contain these surface points and are used to speed up the detection of colliding areas. Once collision areas are detected, the signed distance of the points $V(P)$ that belong to them is evaluated in a level-of-detail manner (see Figure 3 (g)):

$$V(P) = v(P)s + \mathbf{n}^{\mathsf{T}}\mathbf{e} + d, \tag{1}$$

where s is the voxel edge size (constant in uniform grids) and $\mathbf{e}$ the vector from the point to the center C of the voxel where it is located. The *safety distance* d dilates the voxelmap artificially, as previously introduced.

All points with $V(P_i) > 0$ are colliding with the dilated *voxelmap*. The deepest colliding point Q has a penetration of $p_d = \max_i\{V(P_i)\}$. Additionally, single penalty forces and torques (expressed in the center of gravity G) associated to each points with $V(P_i) > 0$ are defined as

$$\mathbf{f}_i = V(P_i)\mathbf{n}_i, \qquad \mathbf{t}_i = \overrightarrow{GP_i} \times \mathbf{f}_i. \tag{2}$$

The total penalty force $\{\mathbf{f}_\mathrm{P}, \mathbf{t}_\mathrm{P}\}$ is the sum of all single forces $\{\mathbf{f}_i, \mathbf{t}_i\}$.

These values are already penalty forces that can be displayed to the user. However, penalty-based haptic rendering has the disadvantages mentioned in Section 1. Since we control the collision detection algorithm, a constraint-based approach could be developed using the whole contact manifold computed in this step. Nevertheless, we want to make our *god object* simulation method available for any penalty-based collision computation algorithm other than ours; therefore, we will use only the most common four values: $\mathbf{f}_\mathrm{P}, \mathbf{t}_\mathrm{P}, p_d, d$. Any algorithm able to provide them can be used instead of our reimplementation of the VPS.

2.2 Correction of the Previous Proxy Frame (#2)

In this section we compute the step $\Delta\mathbf{H}^{\text{pen}}$ necessary to obtain the corrected *surface* frame $\mathrm{S}'(k-1)$ out of the previous *god object* pose ${}^{\mathrm{W}}\mathbf{H}_\mathrm{S}(k-1)$. If the method were perfect, no correction would be necessary. However, since contacts are linearized, the predicted *proxy* might minimally penetrate the surface and it is necessary to resolve this overlap for minimizing the introduced error.

First, we define several parameters used throughout all sections. Then the generalized mass matrix in the center of mass G is composed of the real mass ($\mathbf{M}_\mathrm{B}$) and the inertia tensors ($\mathbf{J}_\mathrm{B}$) of the *body* with mass m, computed out of the data structures defined in Section 2.1:

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}_\mathrm{B} & \mathbf{0} \\ \mathbf{0} & \mathbf{J}_\mathrm{B} \end{bmatrix} = \begin{bmatrix} m\mathbf{I} & \mathbf{0} \\ \mathbf{0} & m\sigma\mathbf{J} \end{bmatrix} \in \mathbb{R}^{6\times 6}, \tag{3}$$

with $\mathbf{I}$ the identity matrix and

$$\sigma = \sqrt[3]{\det\left(\frac{1}{m}\mathbf{J}_\mathrm{B}\right)}. \tag{4}$$

We rather use the normalized version of the inertia tensor $\mathbf{J}$, of which all elements are close to 1. While $\mathbf{J}$ changes the direction of vectors when premultiplied, it is the inertia coefficient σ the factor that mainly changes their length. As the reader will see in the following sections, the mass m drops from the equations. Hence, it has no effect on the *god object* simulation, but only σ and $\mathbf{J}$ do.

In the same line, we define following normalized directions out of the penalty forces and torques (in object $\mathrm{S}(k-1)$ coordinates):

$$\mathbf{u}_\mathrm{x} = \mathbf{u}_\mathrm{f} = \frac{\mathbf{f}_\mathrm{P}}{\|\mathbf{f}_\mathrm{P}\|}, \quad \mathbf{u}_\mathrm{t} = \frac{\mathbf{t}_\mathrm{P}}{\|\mathbf{t}_\mathrm{P}\|}, \quad \mathbf{u}_\mathrm{r} = \frac{\mathbf{J}^{-1}\mathbf{u}_\mathrm{t}}{\|\mathbf{J}^{-1}\mathbf{u}_\mathrm{t}\|}. \tag{5}$$

Their associated magnitudes are the real effective penetration p and the force-torque lever distance δ:

$$p = p_d - d, \quad \delta = \frac{\|\mathbf{t}_\mathrm{P}\|}{\|\mathbf{f}_\mathrm{P}\|}. \tag{6}$$

At this point, we define the translation ($\Delta\mathbf{x}_p$) and rotation ($\Delta\mathbf{r}_p$) vectors necessary to solve the penetration p of the *god object* in the previous time stamp:

$$\begin{aligned} \Delta\mathbf{x}_p &= \lambda p \mathbf{u}_\mathrm{x}, \\ \Delta\mathbf{r}_p &= \theta(\lambda, p)\,\mathbf{u}_\mathrm{r}. \end{aligned} \tag{7}$$

In this last equation (7), we have two important (still) unknown parameters associated to the current penalty values: θ is the correction rotation step, whereas $\lambda \in [0, 1]$ is the translation-rotation distribution factor. If $\lambda = 1$, then $\theta = 0$, hence, the frame $\mathrm{S}(k-1)$ is only translated a distance p along the $\mathbf{u}_\mathrm{x}$ direction to obtain the corrected $\mathrm{S}'(k-1)$. On the other hand, if $\lambda = 0$, then $\theta = \theta_{\max}$, hence, the frame $\mathrm{S}(k-1)$ is only rotated $\theta_{\max}$ units around $\mathbf{u}_\mathrm{r}$ to obtain the corrected $\mathrm{S}'(k-1)$. Usually, the real values lie somewhere in between. We provide in the next two sections analytical closed form formulae for $\theta(\lambda, p)$ and λ.

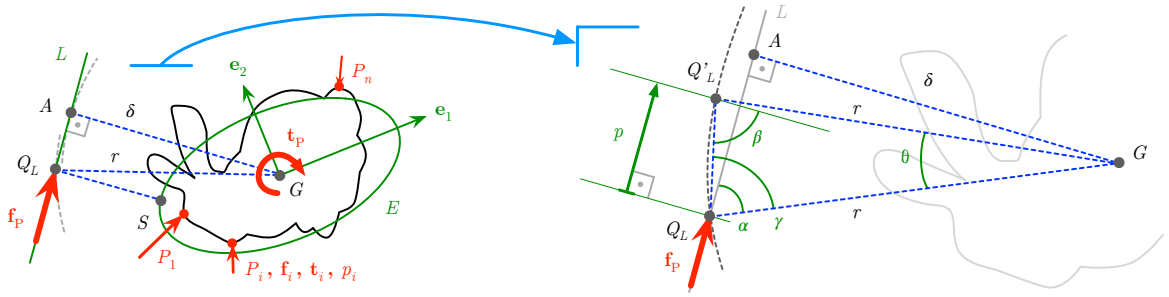

Figure 4: *Computation of the correction rotation θ. On the left, the equivalent system of a 2D Stanford Bunny is built, consisting of the eigen-ellipsoid $E(\mathbf{J}_\mathrm{B})$ and the force application line $L(\mathbf{f}_\mathrm{P}, \mathbf{t}_\mathrm{P})$, on which the equivalent deepest colliding point Q_L with penetration p lies. The distance from Q_L to the center of mass G is the rotation radius r. On the right, a region is zoomed where the point Q_L is rotated around G with its radius r (with exaggerated dimensions for the sake of clarity). The rotation θ brings Q to Q'_L, which is distance p away from Q_L along the force application line L. The rotation $\theta(p, \delta, r)$ should fully solve the penetration p; its value is deduced in Section 2.2.1.*

2.2.1 Computation of the Correction Rotation (θ)

In this section, we will consider the case $\theta = \theta_{\max}$, i. e., the penetration p is fully transformed into a rotation. We assume that the equivalent system presented here can fully solve the penetration by rotating. The correct optimum expression for $\theta(\lambda, p) \in [0, \theta_{\max}]$ that automatically regulates that assumption is provided at the end of the subsection.

A first approximation of the rotation required for fully solving the penetration could be

$$\theta_{\max} \simeq \frac{p}{\delta} \quad \Rightarrow \quad \theta \simeq \frac{p}{\delta}(1-\lambda). \tag{8}$$

Unfortunately, (8) overestimates the necessary rotation in some cases. Therefore, we improve that approximation working on the equivalent system shown in Figure 4. This system is built essentially using the eigen ellipsoid of the object and the penalty contact forces and torques from step #1 (Section 2.1). This eigen ellipsoid results from the principal axis analysis of the inertia tensor.

For that purpose, and before starting the simulation, the eigen values s_1, s_2, s_3 and eigen vectors $\mathbf{e}_1, \mathbf{e}_3, \mathbf{e}_3$ are computed using the inertia tensor $\mathbf{J}_\mathrm{B}$ of the *body* on G. The rotation which brings from the *body* coordinates B to the *eigen* coordinates E is

$$^\mathrm{B}\mathbf{R}_\mathrm{E} = [\mathbf{e}_1 \;\; \mathbf{e}_3 \;\; \mathbf{e}_3] \in \mathbb{R}^{3\times 3}. \tag{9}$$

On the other hand, the *eigen ellipsoid* E centered in G and expressed in the eigen coordinates $\{x_1, x_2, x_3\}$ is

$$E(\mathbf{J}_\mathrm{B}) \equiv \sum_{i=1}^{3} \frac{x_i^2}{a_i^2} = \frac{x_1^2}{a_1^2} + \frac{x_2^2}{a_2^2} + \frac{x_3^2}{a_3^2} = 1, \tag{10}$$

with the axis lengths a_i computed out of the eigen values s_j

$$a_i^2 = \frac{5}{2}\sum_{j=1}^{3}(-1)^\alpha s_j, \quad \alpha = 0 \text{ iff } j \neq i, \;\; \alpha = 1 \text{ otherwise.} \tag{11}$$

Additionally, the force application line L expressed parametrically in the *eigen* coordinates E is

$$\begin{aligned} L(\mathbf{f}_\mathrm{P}, \mathbf{t}_\mathrm{P}) \equiv \;\; & \delta\, ^\mathrm{E}\mathbf{u}_A + \mu\, ^\mathrm{E}\mathbf{u}_\mathrm{f} = \\ & \underbrace{\delta\, ^\mathrm{B}\mathbf{R}_\mathrm{E}\,(\mathbf{u}_\mathrm{t} \times \mathbf{u}_\mathrm{f})}_{A=\overrightarrow{GA}} + \mu\, ^\mathrm{B}\mathbf{R}_\mathrm{E}\mathbf{u}_\mathrm{f} = \\ & (\delta_1, \delta_2, \delta_3) + \mu\,(u_1, u_2, u_3). \end{aligned} \tag{12}$$

The unitary vector $^\mathrm{E}\mathbf{u}_A$ from (12) points from the center of mass G to A, whereas $^\mathrm{E}\mathbf{u}_\mathrm{f}$ is the unitary force direction. The point A is the closest single force application point for the wrench $\{\mathbf{f}_\mathrm{P}, \mathbf{t}_\mathrm{P}\}$. It is not a *material* point on the *body*, but we consider it to be a material point of our equivalent system, composed by $E(\mathbf{J}_\mathrm{B})$ in (10) and $L(\mathbf{f}_\mathrm{P}, \mathbf{t}_\mathrm{P})$ in (12). Note that in the *eigen* coordinates E, $G = \mathbf{0} = (0, 0, 0)$, however, we still name it for correctness and general validity.

Since E equivalently displays the mass distribution of the body, we define Q_L to be the closest point on the line L to the ellipsoid E. The distance from Q_L to the center of mass G is the rotation radius r, a key value to define the rotation θ:

$$r = \|\overrightarrow{GQ_L}\|, \quad \text{such that} \quad \min \|L_{|Q_L} - E\|^2. \tag{13}$$

As it can be seen in Figure 4, $r \geq \delta$. To find Q_L, we first substitute L in E, what leads to

$$\sum_{i=1}^{3} \frac{\delta_i^2 + 2\delta_i u_i \mu + u_1^2 \mu^2}{a_i^2} = 1. \tag{14}$$

This expression in (14) is a second order equation in μ, with all values δ_i and u_i known. If it has two real roots μ_1, μ_2, the line L intersects with the ellipsoid E, what leads to two possible force application points $Q_{L,1}$ and $Q_{L,2}$ substituting μ_1 and μ_2 in (12), respectively. In case both points are different, we select to be Q_L the one which satisfies the condition that the force is in opposite direction to the ellipsoid's surface normal:

$$Q_L = Q_{L,i} \;\text{ s. t. }\; \boldsymbol{\nabla} E_{|Q_{L,i}}\, ^\mathrm{E}\mathbf{u}_\mathrm{f} \leq 0, \;\; i = 1, 2, \tag{15}$$

being $\boldsymbol{\nabla} E = (2x_1/a_1^2, 2x_2/a_2^2, 2x_3/a_3^2)$ the gradient of the eigen ellipsoid, thus, its parametrized surface normal.

However, if no real roots exist for (14), the line L and the ellipsoid E are disjoint, hence, we are dealing with the case shown in Fig-

ure 4 (left). Instead of treating the problem as a constrained optimization, it is possible to solve Q_L using projective geometry. Few operations using homogeneous coordinates lead to the point S on E which is closest to L:

$$S = \overrightarrow{GS} = \frac{(a_1^2\delta_1, a_2^2\delta_2, a_3^2\delta_3)}{\sqrt{(a_1^2\delta_1^2 + a_2^2\delta_2^2 + a_3^2\delta_3^2)}}. \tag{16}$$

With S known, Q_L can easily be determined:

$$Q_L = \overrightarrow{GQ_L} = \overrightarrow{GA} + ((\overrightarrow{AS})^{\mathsf{T}\,\mathrm{E}}\mathbf{u}_{\mathrm{f}})^{\mathrm{E}}\mathbf{u}_{\mathrm{f}}. \tag{17}$$

The values of the rotation radius r and the force lever δ are enough to estimate the maximum rotation θ the body requires to fully solve a penetration p. We refer the reader to the right part of Figure 4, where Q_L is rotated an angle θ around G with a radius r, as if it was a material point. The resulting rotated position Q'_L is a distance p away from Q_L along the direction of the forces ${}^{\mathrm{E}}\mathbf{u}_{\mathrm{f}}$. From the figure, we can deduce the following relationships between angles and distances

$$\gamma = \frac{\pi - \theta}{2}, \quad \beta = \frac{\theta}{2} + \alpha, \quad \sin\alpha = \frac{\delta}{r}. \tag{18}$$

Since p is expected to be small with respect to the size of the object, so will be θ; therefore, we can consider

$$\alpha \gg \theta \quad \Rightarrow \quad \beta \simeq \alpha \quad \Rightarrow \quad \sin\beta \simeq \sin\alpha. \tag{19}$$

Additionally, due to the small value of θ, the arc and the segment joining Q_L and Q'_L will be very similar:

$$\overline{Q_L Q'_L} \simeq \widehat{Q_L Q'_L} \quad \Rightarrow \quad \frac{p}{\sin\beta} \simeq r\theta. \tag{20}$$

Therefore, using (18) and (19) in (20) yields

$$\theta \simeq \frac{p}{r}(1-\lambda)\sin\alpha = \frac{\delta}{r^2}p(1-\lambda). \tag{21}$$

Note that if $r = \delta$, we obtain the approximation we have done in (8), in other words, using the first approximation in (8) instead of (21) increases the possible correction rotation in a factor of r/δ.

2.2.2 Computation of the Correction Translation-Rotation Distribution (λ)

Since in the correction step the object is *moved* from $\mathrm{S}(k-1)$ to $\mathrm{S}'(k-1)$, we choose to distribute (λ) the translation and rotation of that movement by minimizing the required kinetic energy e, computed as

$$e = \frac{1}{2}m\|\dot{\mathbf{x}}_G\|^2 + \frac{1}{2}\boldsymbol{\omega}^{\mathsf{T}}\left(\underbrace{(m\sigma\mathbf{J})}_{\mathbf{J}_{\mathrm{B}}}\boldsymbol{\omega}\right). \tag{22}$$

The linear ($\dot{\mathbf{x}}_G$) and angular ($\boldsymbol{\omega}$) velocities required for the correction during time step Δt are

$$\begin{aligned} \dot{\mathbf{x}}_G &= \frac{\Delta\mathbf{x}_p}{\Delta t} = \frac{\lambda p}{\Delta t}\mathbf{u}_{\mathrm{x}}, \\ \boldsymbol{\omega} &= \frac{\Delta\mathbf{r}_p}{\Delta t} = \frac{\theta(\lambda, p)}{\Delta t}\mathbf{u}_{\mathrm{r}}. \end{aligned} \tag{23}$$

Introducing $\dot{\mathbf{x}}_G$ and $\boldsymbol{\omega}$ from (23) into (22), we obtain

$$e = \frac{m}{2\Delta t^2}\left(\lambda^2 p^2 + \sigma\tau\theta^2\right), \tag{24}$$

with

$$\tau = \frac{\mathbf{u}_{\mathrm{r}}\mathbf{u}_{\mathrm{t}}}{\|\mathbf{J}^{-1}\mathbf{u}_{\mathrm{t}}\|} = \frac{\left(\mathbf{J}^{-1}\mathbf{u}_{\mathrm{t}}\right)^{\mathsf{T}}\mathbf{u}_t}{\|\mathbf{J}^{-1}\mathbf{u}_{\mathrm{t}}\|^2}. \tag{25}$$

For a minimum value of kinetic energy e on λ, the equation (24) must satisfy

$$\frac{\partial e}{\partial\lambda} = 0 \quad \Rightarrow \quad \lambda p^2 + \sigma\tau\theta\frac{\partial\theta}{\partial\lambda} = 0. \tag{26}$$

If we introduce the definition of θ from (21) into (26) and solve for λ, we obtain the analytical value of the translation-rotation distribution:

$$\lambda = \frac{1}{1 + \dfrac{r^4}{\sigma\tau\delta^2}} \in [0, 1]. \tag{27}$$

2.2.3 Assembly of the Final Correction Step

At this point, the correction step transformation matrix $\Delta\mathbf{H}^{\text{pen}}$ which transforms from $\mathrm{S}(k-1)$ to $\mathrm{S}'(k-1)$ can be assembled using the translation and rotation step vectors from (7):

$$\Delta\mathbf{H}^{\text{pen}} \leftarrow \begin{cases} \texttt{setMatrix}(\eta\Delta\mathbf{x}_p,\ \eta\Delta\mathbf{r}_p) & \text{if } Q \in \mathsf{A} \\ \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \in \mathbb{R}^{4\times4} & \text{otherwise.} \end{cases} \tag{28}$$

The gain $\eta = 0.2$ helps regulate the speed with which the object is moved to the surface – instead of doing it suddenly, it is performed exponentially along several haptic cycles. Note that if the deepest point $Q \notin \mathsf{A}$ (i.e., $p_d \leq d \Leftrightarrow p \leq 0$), the corrected previous *god frame* must be the same as the uncorrected one: $\mathrm{S}'(k-1) = \mathrm{S}(k-1)$. However, the correction step vectors $\Delta\mathbf{x}_p$ and $\Delta\mathbf{r}_p$ from (7) still have to be computed, since they are used when computing the unconstrained motion in step #3 (next Section 2.3). In that case ($Q \notin \mathsf{A}$), since $p < 0$, their meaning does not refer to the movement required to solve penetration, but to the *movement allowed before collision occurs*.

With $\Delta\mathbf{H}^{\text{pen}}$ from (28) (in object coordinates $\mathrm{S}(k-1)$), the corrected pose of the *god object* in the previous iteration $(k-1)$ (in world coordinates W) is

$${}^{\mathrm{W}}\mathbf{H}_{\mathrm{S}'}(k-1) = {}^{\mathrm{W}}\mathbf{H}_{\mathrm{S}}(k-1)\Delta\mathbf{H}^{\text{pen}}, \tag{29}$$

being ${}^{\mathrm{W}}\mathbf{H}_{\mathrm{S}}(k-1)$ the *god object* pose delivered in the previous iteration $(k-1)$.

From here on, the corrected frame $\mathrm{S}'(k-1)$ is used instead of $\mathrm{S}(k-1)$, and all movement constraint direction vectors are transformed to it:

$$\begin{aligned} \mathbf{u}_{\mathrm{x}} &\leftarrow {}^{\mathrm{S}'}\mathbf{u}_{\mathrm{x}} = \Delta\mathbf{R}^{\text{pen}}\mathbf{u}_{\mathrm{x}} \\ \mathbf{u}_{\mathrm{r}} &\leftarrow {}^{\mathrm{S}'}\mathbf{u}_{\mathrm{r}} = \Delta\mathbf{R}^{\text{pen}}\mathbf{u}_{\mathrm{r}}, \end{aligned} \tag{30}$$

being

$$\Delta\mathbf{R}^{\text{pen}} \leftarrow \texttt{getRotationMatrix}(\Delta\mathbf{H}^{\text{pen}}). \tag{31}$$

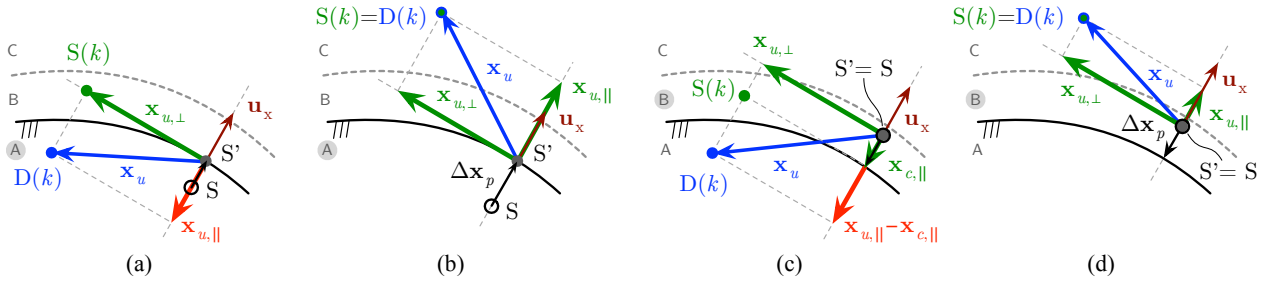

Figure 5: *Computation of the* constrained *movement* $\mathbf{x}_c$ *out of the* unconstrained $\mathbf{x}_u$ *using the movement direction constraint* $\mathbf{u}_x$. *For simplicity, a 2D version using only translations is shown – the computation of* constrained *rotation is analogous, but using* $\mathbf{r}_u$ *and* $\mathbf{u}_r$ *instead. Big dots represent the deepest colliding point* Q *(with penetration* p*) of the object in different frames. Blue vectors display the intended* unconstrained *movement from* $\mathrm{S}'(k-1)$ *to* $\mathrm{D}(k)$. *Green vectors are the parallel (*$\|$*) and orthogonal (*$\perp$*) components of* $\mathbf{x}_u$ *with respect to* $\mathbf{u}_x$ *which are allowed. Red vectors are the parallel components of* $\mathbf{x}_u$ *which are not allowed. The constrained motion* $\mathbf{x}_c$ *is the sum of all allowed components. Subfigures (a) and (b) correspond to the case* $Q \in \mathsf{A}$*, whereas (c) and (d) to the case* $Q \in \mathsf{B}$*, being* d *the safety distance.*

Although the deduction we provide in this subsection might appear relatively long, it is computed in few microseconds, because we have derived analytical formulae that provide a correction for any object in any configuration.

2.3 Computation of the Unconstrained Motion (#3)

The *unconstrained* motion of the *god object* is the transformation from the previous corrected *god frame* $\mathrm{S}'(k-1)$ to the current *device frame* $\mathrm{D}(k)$, as shown in Figure 1:

$$ {}^{\mathrm{S}'}\mathbf{H}_{\mathrm{D}} = {}^{W}\mathbf{H}_{\mathrm{S}'}^{-1\,W}\mathbf{H}_{\mathrm{D}}. \tag{32} $$

We break down this *unconstrained* motion into its translation ($\mathbf{x}_u$) and rotation ($\mathbf{r}_u$) parts

$$ \begin{aligned} \mathbf{x}_u &\leftarrow \texttt{getTranslation}({}^{\mathrm{S}'}\mathbf{H}_{\mathrm{D}}) \\ \mathbf{r}_u &\leftarrow \texttt{getRotation}({}^{\mathrm{S}'}\mathbf{H}_{\mathrm{D}}) \end{aligned} \tag{33} $$

and decompose each of them in parallel ($\|$) and orthogonal ($\perp$) components with respect to the movement constraint directions ($\mathbf{u}_x$ and $\mathbf{u}_r$, respectively):

$$ \mathbf{x}_u = \mathbf{x}_{u,\|} + \mathbf{x}_{u,\perp}; \qquad \mathbf{r}_u = \mathbf{r}_{u,\|} + \mathbf{r}_{u,\perp}, \tag{34} $$

where

$$ \begin{aligned} \mathbf{x}_{u,\|} &= (\mathbf{x}_u^{\mathsf{T}}\mathbf{u}_x)\mathbf{u}_x; & \mathbf{x}_{u,\perp} &= \mathbf{x}_u - \mathbf{x}_{u,\|}; \\ \mathbf{r}_{u,\|} &= (\mathbf{r}_u^{\mathsf{T}}\mathbf{u}_r)\mathbf{u}_r; & \mathbf{r}_{u,\perp} &= \mathbf{r}_u - \mathbf{r}_{u,\|}. \end{aligned} \tag{35} $$

2.4 Computation of the Constrained Motion (#4)

Figure 5 summarizes the computation of the *constrained* motion vectors $\mathbf{x}_c$ and $\mathbf{r}_c$. Essentially, the parallel components of the *unconstrained* vectors are cancelled or shortened in order to obtain the *constrained* vectors. For the sake of brevity, we explain the procedure with translation vectors ($\mathbf{x}$, $\mathbf{u}_x$) only; The computations with rotation vectors are completely analogous, but using rotation constraint direction vector $\mathbf{u}_r$.

As for the *unconstrained* motion, the *constrained* movement vector is the summation of its parallel and orthogonal components with respect to the motion constraints ($\mathbf{u}_x$):

$$ \mathbf{x}_c = \mathbf{x}_{c,\|} + \mathbf{x}_{c,\perp}, \tag{36} $$

where *always*

$$ \mathbf{x}_{c,\perp} = \mathbf{x}_{u,\perp}. \tag{37} $$

On the other hand, the parallel component of the *unconstrained* motion are allowed iff it does not increase penetration in a linearized contact model based on $\mathbf{u}_x$. If $Q \in \mathsf{A} \Leftrightarrow p \geq 0$ (see Figure 5 (a) and (b)), the contact constraint model leads to

$$ \mathbf{x}_{c,\|} = \begin{cases} \mathbf{x}_{u,\|} & \text{if } (\mathbf{x}_{u,\|})^{\mathsf{T}}\mathbf{u}_x \geq 0, \\ \mathbf{0} & \text{otherwise.} \end{cases} \tag{38} $$

Otherwise, if $Q \in \mathsf{B} \Leftrightarrow -d < p \leq 0$ (Section 5 (c) and (d)):

$$ \mathbf{x}_{c,\|} = \begin{cases} \mathbf{x}_{u,\|} & \text{if } (\mathbf{x}_{u,\|})^{\mathsf{T}}\mathbf{u}_x \geq 0, \\ -\min\{\|\mathbf{x}_{u,\|}\|, \|\Delta\mathbf{x}_p\|\}\mathbf{u}_x & \text{else.} \end{cases} \tag{39} $$

Recall from Section 2.2.3 that the correction vectors $\Delta\mathbf{x}_p, \Delta\mathbf{r}_p$ denote the minimum motion to solve penetration for the case $Q \in \mathsf{A}$ and the minimum motion to reach contact for the case $Q \in \mathsf{B}$.

We call *restricted* or *friction* parallel movement to the difference between the *unconstrained* and *constrained* motion vectors:

$$ \mathbf{x}_{r,\|} = \mathbf{x}_{u,\|} - \mathbf{x}_{c,\|}. \tag{40} $$

This vector, always displayed in red in Figure 5, is the parallel component which is not allowed and a key value for computing friction in Section 2.7.

At this point, we can assemble the transformation $\Delta\mathbf{H}^{\text{surf}}$ in Figure 1 which transforms from $\mathrm{S}'(k-1)$ towards $\mathrm{S}(k)$:

$$ \Delta\mathbf{H}^{\text{surf}} = {}^{\mathrm{S}'(k-1)}\mathbf{H}_{\mathrm{S}(k)} \leftarrow \texttt{setMatrix}(\mathbf{x}_c, \mathbf{r}_c). \tag{41} $$

With this step transformation along the surface, the current but still unfiltered *god object or proxy* is computed:

$$ {}^{\mathrm{W}}\hat{\mathbf{H}}_{\mathrm{S}}(k) = \Delta\mathbf{H}^{\text{surf}}\,{}^{\mathrm{W}}\mathbf{H}_{\mathrm{S}'}(k-1). \tag{42} $$

We could use this transformation to define the final *god object* frame, but we observed better results applying some filtering to it, as explained in next Section 2.5.

2.5 Filtering of the Proxy Pose (#5)

We apply a simple discrete exponential six-DoF low pass filter to ${}^{\mathrm{W}}\hat{\mathbf{H}}_{\mathrm{S}}(k)$ based on the previous *proxy* pose ${}^{\mathrm{W}}\mathbf{H}_{\mathrm{S}}(k-1)$ in order smoothen the movement of it on the surface. The filter was tested with many complex geometries and several parameter values; with the presented configuration no overlaps or artifacts have been observed. For an optimum behavior, the cut-off frequency f_{c} is lineally evaluated from the difference between ${}^{\mathrm{W}}\hat{\mathbf{H}}_{\mathrm{S}}(k)$ and ${}^{\mathrm{W}}\mathbf{H}_{\mathrm{D}}(k)$:

$$ {}^{\mathrm{W}}\mathbf{H}_{\mathrm{S}}(k) \leftarrow \texttt{LowPass}({}^{\mathrm{W}}\mathbf{H}_{\mathrm{S}}(k-1), \underbrace{{}^{\mathrm{W}}\hat{\mathbf{H}}_{\mathrm{S}}(k), {}^{\mathrm{W}}\mathbf{H}_{\mathrm{D}}(k)}_{\Delta \rightarrow f_{\mathrm{c}} \in [1,60]\,\mathrm{Hz}}). \quad (43) $$

The value taken by f_{c} depends on the stiffness of the haptic device and the contact configuration, but we scale it to the range $[1, 60]$ Hz. A typical contact situation with a distance from $\mathrm{S}(k)$ to $\mathrm{D}(k)$ of about 7 mm can reach a value of $f_{\mathrm{c}} \simeq 20$ Hz.

This filtered *god object* pose ${}^{\mathrm{W}}\mathbf{H}_{\mathrm{S}}(k)$ describes the *proxy frame* $\mathrm{S}(k)$. It is used

- (i) to visualize the non-penetrating object,
- (ii) to compute the coupling constraint force as explained in next Section 2.6,
- (iii) and to obtain the penalty contact manifold in the next cycle $(k+1)$, as explained in Section 2.1.

2.6 Coupling Forces Applied to the Haptic Device (#6, #7, #8)

Finally, the constraint coupling forces are computed in the step # 6 (see Figure 2). For that, the difference between the haptic *device* pose and the virtual *god-object* pose is computed first:

$$ {}^{\mathrm{D}}\mathbf{H}_{\mathrm{S}}(k) = {}^{\mathrm{W}}\mathbf{H}_{\mathrm{D}}^{-1}(k).{}^{\mathrm{W}}\mathbf{H}_{\mathrm{S}}(k). \quad (44) $$

Then, after obtaining the vector representation of (44),

$$ \begin{aligned} \mathbf{x}_{\mathrm{VC}} &\leftarrow \texttt{getTranslation}({}^{\mathrm{D}}\mathbf{H}_{\mathrm{S}}(k)), \\ \mathbf{r}_{\mathrm{VC}} &\leftarrow \texttt{getRotation}({}^{\mathrm{D}}\mathbf{H}_{\mathrm{S}}(k)), \end{aligned} \quad (45) $$

virtual coupling stiffness constants are applied to it:

$$ \mathbf{f}_{\mathrm{VC}} = k_{\mathrm{VC,x}}\,\mathbf{x}_{\mathrm{VC}}; \quad \mathbf{t}_{\mathrm{VC}} = k_{\mathrm{VC,r}}\,\mathbf{r}_{\mathrm{VC}}. \quad (46) $$

We have experimentally set the stiffness constants to be $k_{\mathrm{VC,x}} = 1$ and $k_{\mathrm{VC,r}} = 0.025$ (unit-less factors).

As a last step, the maximum virtual stiffness (k_{D}) and the corresponding damping (b_{D}) of the haptic device are applied to $\mathbf{f}_{\mathrm{VC}}, \mathbf{t}_{\mathrm{VC}}$ before displaying them to the user as $\mathbf{f}_{\mathrm{D}}, \mathbf{t}_{\mathrm{D}}$. In our experiments (see Section 3), we used a DLR/KUKA Light Weight Robot transformed as a haptic device [Hulin et al. 2008]. The usual constant values for the haptic device are $k_{\mathrm{D}} = 4000$ N/m and $b_{\mathrm{D}} = 20$ Ns/m. However, in the simulations shown in the attached video, moderate ($k_{\mathrm{D}} = 2000$ N/m) and low ($k_{\mathrm{D}} = 200$ N/m) stiffness values are used. Lower constants allow deeper penetrations of the device, which is more challenging, and it also becomes visually more noticeable how the *god object* remains on the surface.

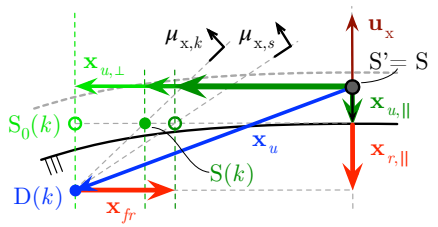

Figure 6: *Friction model which operates in the pose configuration space of the object (case $Q \in \mathsf{A}$ displayed, (c) from Figure 5). The apex of the friction cone is placed in $\mathrm{D}(k)$ and its axis is the opposite of the restricted parallel movement defined in (40): $-\mathbf{x}_{r,\parallel}$. The angle of the cone is defined with the static ($\mu_{\mathrm{x},s}$) and/or kinetic ($\mu_{\mathrm{x},k}$) friction coefficients. The current* proxy frame $\mathrm{S}(k)$ *is moved to the boundary of the cone using $\mathbf{x}_{fr}$. Static, kinetic and viscous friction are possible for both translations and rotations. In this figure, the final pose of $\mathrm{S}(k)$ due to kinetic friction is displayed with a filled green dot.*

2.7 Six-DoF Friction (#4)

In order to compute friction forces, we operate in the object configuration space and restrict the *constrained* movement of the *proxy* $\mathbf{x}_c$ with a *friction restriction* movement $\mathbf{x}_{fr}$:

$$ \mathbf{x}_c = \mathbf{x}_{c,\parallel} + \mathbf{x}_{c,\perp} + \mathbf{x}_{fr}. \quad (47) $$

This *friction restriction* movement is computed using the allowed perpendicular motion $\mathbf{x}_{u,\perp}$ and the not allowed parallel motion $\mathbf{x}_{r,\parallel}$ introduced in (40):

$$ \mathbf{x}_{fr} \leftarrow \texttt{computeFriction}(\mathbf{x}_{r,\parallel}, \mathbf{x}_{u,\perp}). \quad (48) $$

Algorithm 1 summarizes the computation of $\mathbf{x}_{fr}$ and Figure 6 illustrates it for translations. Our method is similar to approaches presented by [Salisbury and Tarr 1997], [Harwin and Melder 2002], [Kawasaki et al. 2011], all introduced in Section 1.1. The basic idea consists in, first, computing friction cones with apex in $\mathrm{D}(k)$ and axis parallel to $\mathbf{u}_{\mathrm{x}}$, and then, sliding $\mathrm{S}(k)$ to the cone boundaries, achieved by adding $\mathbf{x}_{fr}$ to the *constrained* movement $\mathbf{x}_c$. As shown in Algorithm 1, there can be friction only if there is not allowed parallel motion ($\|\mathbf{x}_{r,\parallel}\| > 0$), i.e., the user is applying a force/movement in opposite direction to the surface. In that case, the friction vector $\mathbf{x}_{fr}$ will always point in the opposite direction to $\mathbf{x}_{u,\perp}$ and its length will depend on the type of movement and friction associated to it:

- (i) Static friction – If the perpendicular movement is fully contained in the static friction cone, the object will not move perpendicularly, i.e. $\mathbf{x}_{fr} = -\mathbf{x}_{u,\perp}$. The aperture of the static cone is defined with the static friction coefficient $\mu_{\mathrm{x},s}$.
- (ii) Kinetic friction – If, on the contrary, the perpendicular movement is outside of the friction cone, a new (smaller) kinetic cone is computed and the length of $\mathbf{x}_{fr}$ is set to reach the boundary of the cone. Hence, the object will move, but less than in the frictionless case and tangential forces proportional to the length of $\mathbf{x}_{fr}$ are going to be present. The aperture of the kinetic cone is defined with the kinetic friction coefficient $\mu_{\mathrm{x},k}$.
- (iii) Viscous friction – In the case kinetic friction occurs, the length of $\mathbf{x}_{fr}$ is additionally increased proportionally to the perpen-

dicular velocity ($\mathbf{x}_{u,\perp}$) using the factor $\mu_{\mathrm{x},v}$.

Note that our friction model works also with rotations with the same algorithm but using the rotation vectors ($\mathbf{r}_c$, $\mathbf{r}_{r,\parallel}$, $\mathbf{r}_{u,\perp}$, $\mathbf{x}_{fr}$) and friction coefficients ($\mu_{\mathrm{r},s}$, $\mu_{\mathrm{r},k}$, $\mu_{\mathrm{r},v}$) instead. In the case of rotations, the computations are more difficult to illustrate, but, when applying friction, essentially, the amount of rotation from $\mathrm{S}'(k-1)$ to $\mathrm{S}(k)$ is decreased without altering the axis of rotation.

Friction coefficients can be measured or looked up in tables. In our experiments we have achieved most stable behaviors by choosing first the static friction coefficients $\mu_{\mathrm{x},s}$ and $\mu_{\mathrm{r},s}$ (≈ 0.2) and then fixing $\mu_k \approx 0.9\mu_s$ and $\mu_v \approx 0.01\mu_s$.

Algorithm 1: $\mathbf{x}_{fr}$ = `computeFriction`($\mathbf{x}_{r,\parallel}$, $\mathbf{x}_{u,\perp}$)

Data: Allowed perpendicular motion $\mathbf{x}_{u,\perp}$
and not allowed parallel motion, $\mathbf{x}_{r,\parallel}$.
Result: Motion restriction due to friction, $\mathbf{x}_{fr}$.

```
// Initialize default friction restriction
x_fr = 0
if ||x_{r,||}|| > 0 then
    if ||x_{u,⊥}|| < μ_{x,s} ||x_{r,||}|| then
        // Static friction
        x_fr = −x_{u,⊥}
    else
        // Kinetic friction
        x_fr = −μ_{x,k} ||x_{r,||}|| x_{u,⊥} / ||x_{u,⊥}||
        // Viscous friction
        x_fr ← x_fr − μ_{x,v} x_{u,⊥}
return x_fr
```

3 Experiments and Results

We have performed several benchmarking experiments, three of them shown in Figure 7 and in the additional video submitted with this paper. The first one consists in interacting with a Stanford Bunny and a Utah Teapot. Second, the same bunny hits a thin surface to check that it does not pop through. And finally, a peg object is introduced into a hole, in such a ways that roughly 90% of all its points collide. Figure 8 shows the results of the first bunny-teapot scenario. Contact and computation time values are plotted for hitting, sliding and ear-insertion tasks.

In all tested scenarios stable, stiff and realistic forces were always generated, and the *god object* remained visually on the surface. The penetration (p_d, Section 2.2) moved below the used voxel size, which defines our maximum resolution. Additionally, the total computation time (steps #1 – #7) stayed always easily below the 1 ms convention even when the several thousands of colliding points were reached. In particular, the bunny-teapot benchmark results in Figure 8 show that our *god object* method requires only around 5 μs for collision cases.

4 Conclusions

We presented an easy-to-implement *constraint-based* force computation method that can be applied to *penalty-based* haptic rendering algorithms. As shown in the results, our method computes six-DoF collision forces with the advantages of both approaches even with complex objects and collision situations: computational speed (only around 5 μs) and accuracy (error bounded by the resolution of data structures). Additionally, our algorithm computes six-DoF static and dynamic frictional forces. Our future work will deal with the comparison of our method with other available approaches and with its evaluation in user studies. Extending our algorithm to multi-body scenarios would be also a very interesting direction.

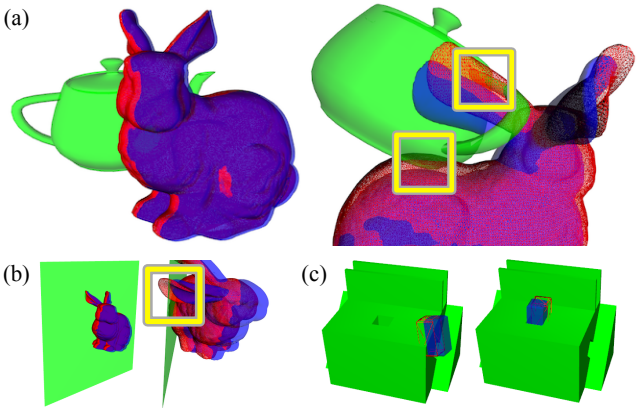

Figure 7: *Three benchmarking scenarios: (a) A Stanford Bunny collides against a Utah Teapot and the bunny's ear is introduced into the handle of the teapot, (b) The Stanford Bunny collides against a thin plane, (c) A classical peg-in-hole benchmark. The green object is always voxelized. The red mesh is the representation of the (unconstrained) object moved by the user, whereas the blue one is the god object constrained to the surface of the green object. Yellow boxes highlight challenging areas where the computation of the constrained object is successfully achieved.*

References

ANDERSSON, S., SÖDERBERG, A., AND BJÖRKLUND, S. 2007. Friction models for sliding dry, boundary and mixed lubricated contacts. *Tribology international 40*, 4, 580–587.

BARBIČ, J., AND JAMES, D. L. 2008. Six-dof haptic rendering of contact between geometrically complex reduced deformable models. *IEEE Trans. on Haptics 1*, 1, 39 –52.

BASDOGAN, C., AND SRINIVASAN, A. A. 2002. Haptic rendering in virtual environments. *Stanney, K. (Ed.), Handbook of Virtual Environments*, 117–134.

CHAN, S., CONTI, F., BLEVINS, N. H., AND SALISBURY, K. 2011. Constraint-based six degree-of-freedom haptic rendering of volume-embedded isosurfaces. In *Proc. IEEE World Haptics Conference*, IEEE, 89–94.

CHAN, S.-C., BLEVINS, N. H., AND SALISBURY, K. 2013. Deformable haptic rendering for volumetric medical image data. In *Proc. IEEE World Haptics Conference*, IEEE, 73–78.

GAUSS, C. F. 1829. Über ein neues allgemeines grundgesetz der mechanik. *Reine angewandte Mathematik 4*, 232–235.

HARWIN, W. S., AND MELDER, N. 2002. Improved haptic rendering for multi-finger manipulation using friction cone based god-objects. In *Proc. Eurohaptics Conf.*, 82–85.

HAYWARD, V., AND ARMSTRONG, B. 2000. A new computational model of friction applied to haptic rendering. In *Experimental Robotics VI*. Springer, 403–412.

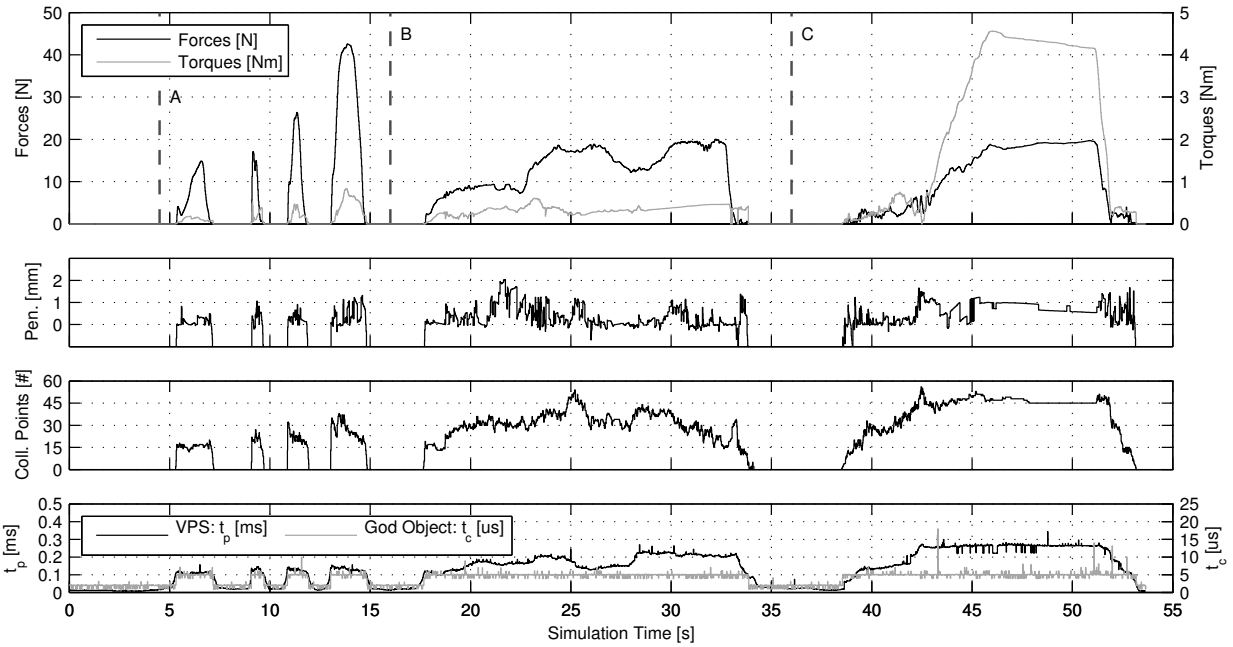

Figure 8: *Experimental results of our method for the scenario displayed in Figure 7(a), where a Stanford Bunny (point-cloud, 5587 points in 7 levels) collides against a Utah Teapot (voxelized distance field,* $360 \times 299 \times 315$ *voxels with 2 mm voxel edge length). Computed force, torque, penetration, colliding points and computation time are shown. The computation time is divided in the one related to the penalty-based method (step #1) and the one of the god object method (steps #2 – #7). From start time to marker A objects are separated; Between A – B the bunny knocks the teapot with increasing forces; Between B – C the bunny slides around the teapot; From C until the end of the simulation one ear of the bunny is introduced into the handle of the teapot. The used computer was an Intel(R) Core(TM) 2 Quad with CPUs at 2.66GHz and running Suse SLED 11 (not realtime).*

HULIN, T., SAGARDIA, M., ARTIGAS, J., SCHAETZLE, S., KREMER, P., AND PREUSCHE, C. 2008. Human-scale bimanual haptic interface. In *Proc. of the Int. Conf. on Enactive Interface*.

KAUFMAN, D. M., EDMUNDS, T., AND PAI, D. K. 2005. Fast frictional dynamics for rigid bodies. *ACM Trans. on Graphics 24*, 3, 946–956.

KAWASAKI, H., OHTUKA, Y., KOIDE, S., AND MOURI, T. 2011. Perception and haptic rendering of friction moments. *IEEE Trans. on Haptics 4*, 1, 28–38.

LEEPER, A., CHAN, S., AND SALISBURY, K. 2012. Point clouds can be represented as implicit surfaces for constraint-based haptic rendering. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, IEEE, 5000–5005.

MCNEELY, W. A., PUTERBAUGH, K. D., AND TROY, J. J. 1999. Six degree-of-freedom haptic rendering using voxel sampling. In *Proc. ACM SIGGRAPH*, ACM, 401–408.

MIRTICH, B. V. 1996. *Impulse-based Dynamic Simulation of Rigid Body Systems*. PhD thesis, University of Californya at Berkeley.

ORTEGA, M., REDON, S., AND COQUILLART, S. 2006. A six degree-of-freedom god-object method for haptic display of rigid bodies. In *Proc. IEEE Virtual Reality (VR)*, IEEE, 191–198.

REDON, S., KHEDDAR, A., AND COQUILLART, S. 2002. Fast continuous collision detection between rigid bodies. *Computer Graphics Forum 21*, 3, 279–287.

REDON, S., KHEDDAR, A., AND COQUILLART, S. 2002. Gauss' least constraints principle and rigid body simulations. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, 517–522.

RUSPINI, D. C., KOLAROV, K., AND KHATIB, O. 1997. The haptic display of complex graphical environments. In *Proc. ACM SIGGRAPH*, ACM, 345–352.

RYDÉN, F., AND CHIZECK, H. J. 2013. A method for constraint-based six degree-of-freedom haptic interaction with streaming point clouds. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, IEEE, 2353–2359.

SAGARDIA, M., WEBER, B., HULIN, T., PREUSCHE, C., AND HIRZINGER, G. 2012. Evaluation of visual and force feedback in virtual assembly verifications. In *Proc. IEEE Virtual Reality (VR)*, IEEE, 23–26.

SAGARDIA, M., STOURAITIS, T., AND E SILVA, J. L. 2014. A New Fast and Robust Collision Detection and Force Computation Algorithm Applied to the Physics Engine Bullet: Method, Integration, and Evaluation. In *EuroVR: Conf. and Exhibition of the European Association of Virtual and Augmented Reality*, Eurographics Association, 65–76.

SALISBURY, K., AND TARR, C. 1997. Haptic rendering of surfaces defined by implicit functions. In *Proc. Annual ASME Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, vol. 61, 61–67.

WANG, D., ZHANG, X., ZHANG, Y., AND XIAO, J. 2013. Configuration-based optimization for six degree-of-freedom haptic rendering for fine manipulation. *IEEE Trans. on Haptics 6*, 2, 167–180.

ZILLES, C. B., AND SALISBURY, J. K. 1995. A constraint-based god-object method for haptic display. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 146–151.