SoftSim Geometric Query Specifications

Alejandro M. Castro

August 24, 2020

1 Introduction

The purpose of this document is to describe a practical geometry query to be used in the simulation of soft bodies with contact. There is a large variety of mathematical formulations with strong foundations in the FEM community. While the Mortar method seems to be a standard in the FEM community [1, 2, 3, 4] here we'll follow an approach based on the work by Duriez [5] given it's simplicity and the accuracy requirements for the short-term applications. Further research on Mortar methods and its variants must be conducted but we believe this to be a good starting point, especially regarding the practical definition of a contact query.

1.1 Geometry Outline

A brief description of the query requirements are given in [5, §3.2].

In a nutshell, given two surface meshes A and B, the geometric query identifies a set of n_c point contact pairs. For each pair, the query reports points Ca and Cb on surfaces A and B respectively, the contact normals $\widehat{n}_A$ and $\widehat{n}_B$ at these points on each surface and the barycentric coordinates of Ca and Cb within the triangle the belong to. The physics solver is agnostic to how these pairs are obtained and, hopefully, robust to geometry induced perturbations such as changes in the normal directions especially at sharp corners.

Figure 1 (Fig. 2 in [5]) shows an example of two points P and Q identified by the query. While point Q in the figure corresponds to a surface vertex, point P is described by the barycentric coordinates in the triangle it belongs to.

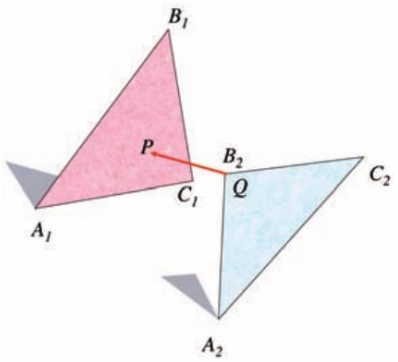

Figure 1: An example of two points P and Q identified by the query.

In [5] other contact modes are identified, summarized in Fig. 2 (Fig. 3 in [5]). The two top figures correspond to the most probable cases given that for the cases outlined in the bottom row triangles must be in perfect alignment. Even if those cases are considered, the information consumed by the physics engine is the same.

1.2 Geometry Specification by the Physics Engine

The FEM model for soft bodies is discretized using meshes of tetrahedra. The geometry used for contact queres does not necessarily need to be represented by the same mesh used for FEM, and

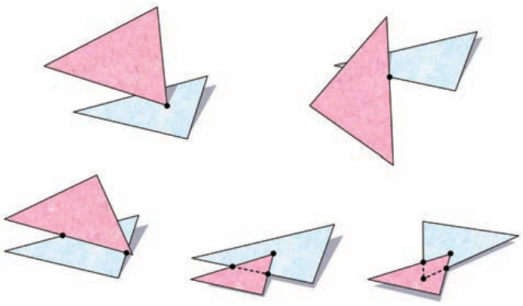

Figure 2: Different contact scenarios for triangle-triangle pairs.

actually in [6] usually a coarser mesh is used for contact queries. Whether this is true for us or not, the geometry engine does not need to know this details and the FEM solver could opt for using different representations with the additional bookkeeping.

It is very important that there is a precise correspondence between the mesh used by the physics engine and the mesh used by the geometry engine. That is, indexes reported back by the geometry engine must make sense to the physics engine. This leads to the conclusion:

- The physics engine must be able to specify the mesh in its entirety, including vertex positions and connectivity.

This is different from the typical approach for rigid contact, in which the physics engine does not know about the actual geometric representation.

Given the FEM solver knows and understands the structure of the geometry, it can already provide to the geometry engine:

1. Vertex positions in a *reference configuration* (typically the zero stresses state) and connectivity.

2. Surface vertex normals. Within each triangular face normals are linearly interpolated given their barycentric location when needed.

1.3 Geometric Queries

In this section we provide a more detailed description of the query. It is a "constructive" description in that we provide a methodology to compute it. The reason we need a constructive description is because the queries are usually performed at *unphysical states* at which interpenetration already ocurred. In the real world, contact leads to surfaces that deform conforming to each other, even if only in a microscopic region. Therefore, given two objects, the real physics leads to surfaces that conform and are *parallel* to each other. In that state, two closest neighbor points on each body have coincident locations and the normals on each body are exactly opposite to each other. This is generally not true in the discrete setting, however just an approximation to the real physics. Therefore in what follow we'll be specific to this constructive approach, with remarks to the physical approximation when it corresponds.

Consider two bodies A and B, each with their surface discretized by a triangle mesh. We will first find the set of surface vertices C_a of A that lie inside of B. We'll denote that set with ${}^A\mathcal{C}$. For each surface vertex C_a of A inside B the contact query must find a point C_b on the surface of B that is *close* to C_a. Even if C_b is the closest point to C_a on B, usually the surface normals evaluated at C_a and C_b will not be the opposite of each other. This is a consequence of the queries performed at *unphysical states* as outlined in the previous paragraph. However, these states are usually a good approximation in the limit in which the surfaces conform to each other. Now, this fact leads to a choice; "in what direction do we search C_b on B from C_a?" we could search for the *closest point*, i.e. the direction is aligned with the normal on B, or we could search in the direction to the normal to A. Other options are possible, but it is important to keep in mind that the normals on A and B do align in the limit in which the discrete solution converges as the spatial resolution increases. From a practical standpoint however, we do know the normal vector $\widehat{n}_A$ at C_a on A's surface. Therefore it makes sense to use $\widehat{n}_A$ as the search direction from C_a onto B's surface to find C_b. This approach has the nice effect of uniquely defining the result of the query.

Similarly, we define the set ${}^{B}\mathcal{C}$ by finding for each surface vertex C_b of B, a corresponding point C_a on surface A by essentially ray casting from C_b, in the direction of $\widehat{n}_B$ at C_b (since it's known), onto the surface of A. The set of all collision pairs will be denoted with $\mathcal{C} = {}^{A}\mathcal{C} \bigcup {}^{B}\mathcal{C}$.

For each of these pairs C_a/C_B the query must provide:

- Contact pair C_a, C_b.
- Indices of the surface triangles to which points C_a and C_b belong, in their respective surfaces.
- Barycentric coordinates of C_a and C_b in their triangles.
- Surface normals $\widehat{n}_A$ and $\widehat{n}_B$. This can be evalated by interpolation given we know in advance the surface vertex normals and the barycentric coordinates are a result of this query.

This information can be reported as point pairs by embeding this information in a data structure outlined in Listing 1. Notice that in this format, the physics engine is agnostic to the specifics of the query.

Listing 1: Contact feature data structure

```
template <typename T>
struct MeshMeshContactFeature {
    // Feature specs on mesh A.
    GeometryId id_A;              // Geometry identifier for mesh A.
    Vector3<T> p_WCa;             // Position of point Ca on A's surface.
    SurfaceFaceIndex Ca_face;     // A face to which Ca belongs.
    SurfaceMesh<T>::Barycentric
        Ca_barycentric;           //  Barycentric coordinates of
                                  //  point Ca in triangle Ca_face.
    Vector3<T> normal_Ca_W;       //  Outward normal at Ca, expressed in W.

    // Feature specs on mesh B.
    GeometryId id_B;              // Geometry identifier for mesh B.
    Vector3<T> p_WCb;             // Position of point Cb on B's surface.
    SurfaceFaceIndex Cb_face;     // A face to which Cb belongs.
    SurfaceMesh<T>::Barycentric
        Cb_barycentric;           //  Barycentric coordinates of
                                  //  point Cb in triangle Cb_face.
    Vector3<T> normal_Cb_W;       //  Outward normal at Cb, expressed in W.
};
```

1.3.1 Edge to Edge contact. Andy maybe other special cases?

The formulation to be presented in what follows can deal with edge-edge collisions within the same framework if we just only augment $\mathcal{C}$ to also include edge-edge point pairs, providing exactly the same information described above. This corresponds to the top-right corner in Fig. 2.

That is, if there is an edge-edge intersection, the collision engine should be able to provide point C_b on a given edge of B and point C_a on a given edge of A that are the closest within those edges. Once again, if we encapsulate this information as points that belong to given triangles, these triangles' indexes and the barycentric coordinates contain all the information we need.

References

[1] B. Yang, T. A. Laursen, and X. Meng, "Two dimensional mortar contact methods for large deformation frictional sliding," *International journal for numerical methods in engineering*, vol. 62, no. 9, pp. 1183–1225, 2005.

[2] L. De Lorenzis, P. Wriggers, and T. J. Hughes, "Isogeometric contact: a review," *GAMM-Mitteilungen*, vol. 37, no. 1, pp. 85–123, 2014.

[3] K. A. Fischer and P. Wriggers, "Mortar based frictional contact formulation for higher order interpolations using the moving friction cone," *Computer methods in applied mechanics and engineering*, vol. 195, no. 37-40, pp. 5020–5036, 2006.

[4] A. Popp, M. Gitterle, M. W. Gee, and W. A. Wall, "A dual mortar approach for 3d finite deformation contact with consistent linearization," *International Journal for Numerical Methods in Engineering*, vol. 83, no. 11, pp. 1428–1465, 2010.

[5] C. Duriez, F. Dubois, A. Kheddar, and C. Andriot, "Realistic haptic rendering of interacting deformable objects in virtual environments," *IEEE transactions on visualization and computer graphics*, vol. 12, no. 1, pp. 36–47, 2006.

[6] C. Duriez, *Real-time haptic simulation of medical procedures involving deformations and device-tissue interactions*. PhD thesis, 2013.

Todo list