NOTES PROJECT MAIN

1. Discrete Hausdorff distance between triangular meshes

Let us briefly recall the mathematical definition of the Hausdorff distance between two closed bounded subsets A and B of $\mathbb{R}^3$. We recall that the distance between a point $x \in \mathbb{R}^3$ and a closed bounded subset A of $\mathbb{R}^3$ is the quantity:

$$d(x, A) = \inf_{y \in B} d(x, y)$$

where $d(,.,)$ is the usual Euclidean distance.
Denote $\rho(A, B)$ the quantity:

$$\rho(A, B) = \sup_{x \in A} d(x, B),$$

then, the Hausdorff distance between A and B is defined as:

$$d_H(A, B) = \sup(\rho(A, B), \rho(B, A)).$$

Let us now consider two discrete surfaces S, $S\prime$ described by two triangulations $\mathcal{T} = \cup_i K_i$, $\mathcal{T}\prime = \cup_k K_k\prime$. The discrete Hausdorff distance $d_H(S, S\prime)$ relies on the previous definition. Let us now describe a method to compute this quantity. Given a mesh triangle K_i of the triangulation $\mathcal{T}$, for each point $p \in K_i$, we are able to evaluate analytically the discrete distance:

$$d(p, \mathcal{T}\prime) = \inf_{K_k\prime \in \mathcal{T}\prime} d(p, K_k\prime).$$

Then the distance between K_i and $K_k\prime$ can be computed as:

$$d(K_i, K_k\prime) = \sup_{p \in K_i} d(p, K_k\prime)$$

where the point p is meant to belong to an opportune sampling of the triangle K_i. Finally the distance between the reference triangle K_i and the triangulation $\mathcal{T}\prime$ is given by:

$$d(K_i, \mathcal{T}\prime) = \inf_{K_k\prime \in \mathcal{T}\prime} d(K_i, K_k\prime).$$

The problem of computing the Hausdorff distance is then reduced to the problem to compute the distance between a sampled point (note that this point is not necessarily a vertex of the triangulation) and a reference triangulation.

2. Computation of the analytical distance between a point and a triangulation

Given a point $p \in \mathbb{R}^3$ known by its cartesian coordinates and a reference triangulation $\mathcal{T}\prime = \cup_k K_k\prime$, the goal is the numerical computation of the quantity :

$$d(p, \mathcal{T}\prime) = \inf_{K_k\prime \in \mathcal{T}\prime} d(p, K_k\prime)$$

The computation of the distance between a point and a given triangle is straightforward. We now want a strategy to compute the minimal distance between a point and a (big) set of triangles. The brute-force strategy to achieve this computation naively is described in algorithm 1.

One cannot think to apply this strategy on massive models (triangulations composed of thousands of triangles). In order to reduce the complexity of the algorithm, one could look for a way to reduce the number of point-triangle distance evaluations. Remember that here we don't need to know analytically all the distances $d_k = d(p, K_k)$, we are only interested in getting the minimum over k! So if compute an upper bound for this minimum, and we know that a triangle is further than this value from the point, then we can avoid the computation of the distance between the point and this triangle. An efficient way to drastically reduce the number of distance point-triangle evaluation is to take avantage of a bucket sort. The main idea of a bucket sort algorithm is to partition the spatial domain (say the bounding box of the triangulation in our

Data: point p, surface mesh $\mathcal{T}\prime$
Result: distance $d(p, \mathcal{T}\prime)$
set d = big value ;
for *each triangle* $K_k\prime of \mathcal{T}\prime$ **do**
  compute the distance $d_k = d(p, K_k)$;
  update the distance $d = \min(d, d_k)$;
end
return (d);

Algorithm 1: Brute force distance point-triangulation

case) with a uniform grid. For the sake of simplicity let $B = [0, 1]^3$ be the bounding box of a triangulation $\mathcal{T}\prime$. A uniform grid (of size N) of B is a partition of B into N^3 non-overlapping cubic cells (N subdivisions along each direction). Each cell of the grid (bucket) is labeled with a key, depending only on the cartesian coordinates of its vertices. Given a vertex of the triangulation it easy to compute the cell grid to which the point belongs. Then the vertices of the triangulation (or eventually the triangles) are sorted according to their position in the grid. The key idea is the following: the points are distributed into a number of buckets (cells of the grid in this case) depending on their spatial proximity. In algorithm 2 we describe a method for computing the distance between a point $p \in \mathbb{R}^3$ and a surface triangulation $\mathcal{T}\prime$ of bounding box B taking advantage of a bucket sort. The method is explained in [1]. We suppose that p is close enough from the triangulation, that is, p lies inside B. If p does not lie inside the box B one need to take a bigger box as a support for the bucket structure. We recall that the ball of a vertex p_0 is the list of all triangles sharing the vertex p_0.

Eventually instead of storing the vertices of the triangulation, one could store in the bucket structure the list of triangles intersecting each cell of the grid, as suggested in [2].

References

[1] H. Borouchaki and P. Frey, *Simplification of surface mesh using Hausdorff envelope*, Comput. Methods Appl. Mech. Engrg. 194 (2005) 4864?4884.

[2] P. Cignoni, C. Rocchhini, and R. Scopigno, *Metro: measuring error on simplified surfaces*, Computer Graphics Forum, Blackwell Publishers, Vol. 17, No. 2, pages 167-174, June 1998.

Data: point p, surface mesh $\mathcal{T}\prime$
Result: distance $d(p, \mathcal{T}\prime)$
create bucket structure and store in the bucket the list of the vertices of the triangulation $\mathcal{T}\prime$;
hash the triangles of $\mathcal{T}\prime$ and store adjacencies relations;
set $d =$ big value, $d_{app} =$ big value, $kel = 0$, $iel = 0$; ;
find the cell grid C to which p belongs;
starting from C explore the bucket and find the vertex triangulation p_0 closer to p and retain the distance $d_0 = d(p, p_0)$;
while p_0 **do**
  get the list of all triangles in the ball $B(p_0)$ of p_0;
  for *each triangle* $K_k\prime$ *in the ball* $B(p_0)$ **do**
    compute the distance $d_k = d(p, K_k\prime)$;
    if $d_k < d$ **then**
      update the distance $d = d_k$;
      store the index of current triangle $kel = k$;
    end
  end
  for *each vertex* p_k *(*$k \neq 0$*) belonging to the triangle kel* **do**
    get the list of all triangles in the ball $B(p_k)$ of p_k;
    for *each triangle* $K_{p_k}\prime$ *in the ball* $B(p_k)$ **do**
      compute the distance $d_{p_k} = d(p, K_{p_k}\prime)$;
      if $d_{p_k} < d_{app}$**then**
        update the distance $d_{app} = d_{p_k}$;
        store the index of current triangle $iel = k$;
      end
    end
    if $d_{app} > d$ **then**
      $p_0 = p_k$;
    else
      $p_0 = 0$;
    end
  end
end
return (d);

Algorithm 2: Bucket sort algorithm for the computation of the distance point-triangulation