

# Tracking Objects on a Deformable Surface using displacement and orientation data

Zachary DeStefano, Kyle Culter, Gopi Meenakshisundaram, Bruce Tromberg  
University of California, Irvine

August 19, 2014

## Abstract

Previous attempts at tracking objects as they move have used magnetic fields or a camera array. With deformable surfaces trying to track on them is especially difficult because the path is not directly following the original shape for the mesh. For this paper, we present an algorithm for tracking on a deformable surface as well as a way of tracking using only displacement and orientation sensors. We developed a probe that contained an optical mouse sensor for displacement on the surface and an accelerometer, gyroscope, and compass to find out the orientation of each displacement. We used this data to record paths and developed an algorithm for doing calibrations as well as following the surface.

## Introduction

In various medical applications, there is a need to track a probe as it moves across a patient's body and takes measurements. Currently they attach a grid to the patient and painstakingly take measurements at each point in the grid. This led us to explore ways of automating that process. Using a Kinect to try and track probe movement can be impractical because the probe needs to always be in view of the Kinect camera. Magnetic grids are expensive and cumbersome so that option would not have worked well. We then thought that using an ordinary optical mouse sensor on the surface itself could work well. Because the surface is not flat, we will need to know the orientation so we put a gyroscope, compass, and accelerometer into the probe. Using these simple tools as well as software that implemented the calibration algorithm shown in this paper, we were able to get a reasonable approximation of the probe's path on the surface itself.

Our process started with taking a 3D scan of the surface to get the mesh. We use a Kinect to get the data and the point cloud architecture **\*\*INSERT MORE DETAIL\*\*** to take the data and get a mesh and texture from it. We then take out some of the noise and use the QSlim **\*\*INSERT MORE DETAIL\*\*** algorithm to simplify the mesh. Once the mesh is simplified, we load it into our tracking environment. Our tracking environment is an application that takes in the mesh as well as probe data readings and gives live updates as to the probe's position and data. It operates similarly to a video game and was developed using JMonkeyEngine **\*\*INSERT DETAIL\*\***, a video game library written in Java that is similar to Ogre **\*\*INSERT REFERENCE\*\***.

The first step is to make sure the scale is calibrated. This is done in two steps. We move the probe a certain distance and see what the total displacement outputted by the mouse sensor was. We then measure the distance between known points on the surface in the virtual world and the real world and see what the ratio is. We then combine the two ratios so that we have a scale factor to go from displacement units read by the probe to displacement units in the virtual world. Once the scale is calibrated, we start recording paths on the mesh itself. We then have a problem. Because the rotation has not been calibrated, it is likely that the paths will have a shape similar to the part of the mesh they were on but they will not actually be on the mesh **\*\*INSERT PICTURE OF THIS\*\***. We thus need to figure out what rotation to apply to the probe reading's rotation in order to get the rotation in the virtual world. After that though, there is still a problem. Because

the surface is deformable, the path would still not perfectly follow the mesh. We thus need an algorithm that will approximate the probe's position on the mesh given the deformability.

This paper presents two algorithms. The first one takes a path that almost follows the mesh and projects it onto the mesh in such a way that preserves its orientation along the mesh as well as its arc length. The second one takes a recorded 3D path between two known points on the mesh and figures out how to rotate it and project it onto the mesh so that its start and end points match the ones on the mesh. The rotation found by the second algorithm gives us the rotation calibration we need. Once the rotation is calibration is found, we can then live track the probe and use the first algorithm to keep it along the mesh.

## Related Work

Commercial motion tracking for professional film makers use image-based tracking techniques [1]. This method has some serious disadvantages for our application. Being that is designed for use by artists, it does not have a guarantee of measurement accuracy, which is necessary for our case. Additionally, the fact that it is image based means the probe would always have to be in the view of the camera and in a clinical setting that can be hard to ensure. The post-processing of the images can be time consuming and in a clinical setting the results should come quickly. Finally, training the medical technicians taking the data to do all the steps that would be required is impractical.

Using the Kinect for tracking would have some advantages over image-based tracking. Because the Kinect gives us a depth map, we could more easily ensure measurement accuracy. Additionally, there would be much less processing required to get a final motion path. It still has the disadvantage that the probe would always have to be in view of the camera. It also has the disadvantage that the result is dependent on the shape of the probe. For this application, we do not know the shape of the final probe and it is impractical to try and accommodate all the different probe shapes.

Motion Capture is a popular technique for film making that tracks a character's motion in 3D space. Using a motion capture system we would get 3D coordinates for the probe's position as it moves across a patient. Current motion capture systems however are expensive and cumbersome to set up. Both of these make it impractical for the clinical setting. **\*\*INSERT REFERENCE\*\***

There is work being done on using magnets for motion tracking [4]. While we could get measurement accuracy with this method and the setup would not be too difficult, the system would be cumbersome for patients. It would also only be limited to tracking across an area as big as the magnetic coils. The method we are proposing can be used on any mesh that is scanned in 3D. Additionally, we want to easily be able to track across the entire mesh that was scanned.

Current work on deformable surface tracking is focused on tracking the deformations of the surface itself [3, 2]. This paper focuses on deformable surfaces that deform when pressure is applied but then retain

their shape afterwards. It is also focused on tracking objects on the surface rather than the surface itself.

Being that we are tracking on a 2D surface embedded into 3D space, we thought about trying to flatten the mesh first and then track on it. We found the following paper that described a mesh flattening algorithm. **\*\*INSERT CITATION\*\***. Even though the mesh flattening does not have any holes or gaps when using this algorithm, there is no way to assure that our path will not leave the boundary of the flattened mesh. Additionally, we need an algorithm to work on arbitrary meshes and the mesh flattening was only proven to work with a handful of meshes. Due to these difficulties, we decided not to flatten the entire mesh and only do local flattening when we want the path to follow the mesh, which means we care about the triangle where the path is currently as well as its neighboring triangles.

## Following a Deformable mesh

For the purposes of our application, we need to have paths on the mesh itself so that we know the location on the mesh of particular data points. Due to the deformability, the paths ends up being close to the mesh, but not on the mesh itself, no matter how accurate the calibration was. Additionally, in order to do the calibration, we need the path to follow the mesh itself so that the end points line up. We thus came up with an algorithm for projecting a path onto the triangles of the mesh.

A path is just a series of connected segments and the mesh is just a series of triangles. Thus the input of our algorithm is a segment with a start and end point as well as the triangle where it originates and the triangles neighbors. We need to make sure the projection preserves the length of the segment. We also want to preserve the orientation along the surface thus we are restricted to rotating the segment along the plane that it makes with the triangle's normal.

If we let  $v$  be the segment vector,  $N$  be the normal, and  $E$  be the resultant vector that is the projection of  $v$  onto the plane described by  $N$ , then the following equation will get us  $E$

$$E = v - proj_N(v)$$

This can be further simplified to say

$$E = v - N(v \cdot N)$$

We then find the intersection of the vector  $E$  with the current triangle. If the segment is entirely in the triangle, then we move onto the next segment starting from the endpoint of  $E$ . If the segment leaves the triangle, then we find which edge the segment intersects and repeat the projection procedure for the end part of the segment that is not in the triangle. This procedure of course relied on finding the intersection of the triangle with the segment which proved to be non-trivial.

When finding the triangle and segment intersection, we had a triangle in a 3D space as well as a segment that was supposed to be coplanar to the triangle but could be slightly off due to floating-point errors

with the coordinates. To find the intersection point, I converted the segment and triangle coordinates to the coordinate system where one of the vertices of the triangle is the origin and the basis vectors are the two vectors made by the segments coming off that vertex as well as the cross product of those vectors. In this coordinate system, the third coordinate should be zero. In practice, it was near zero due to floating point errors. Once in the new coordinate system, we just had to find the 2D intersection of the triangle and the segment.

The above procedure describes what we did with a single segment. With paths, we just iterated this procedure for each segment of the path. We assumed that the important aspect of each segment is its vector and not its origin point, since that is what we get from the probe. This meant that the end point of a projected segment was treated as the start point for the rest of the path when doing the loop to project the entire path onto the mesh.

## Calibration of Rotation and Scale

After taking a 3D scan and importing the mesh into our environment, we have our object in a virtual world with a virtual coordinate system. Our probe operates in the real world. We thus need to know how to translate between the two in order to get an accurate visualization. We first needed to translate displacement readings from the probe to displacement in the virtual world. Afterwards, we needed to get rotation calibrated so that the displacement vectors went in accurate directions in the virtual world.

The Scale Calibration was relatively straightforward. The probe gave numerical displacement readings that ranged from -127 to 127. We then made the probe travel 100 mm in every direction and summed up the displacement values. This gave us a probe unit to millimeters conversion factor. We then measured the distance between the calibration points in the virtual world and in the real world in millimeters and this gave us a virtual unit to millimeters conversion ratio. We then combined the two ratios and obtained a probe unit to virtual unit conversion ratio.

The Rotation Calibration was difficult. This involved recording a path that went between calibration points and finding the correct rotation to apply in the virtual world so that the start and end point of the path are the points on the mesh. Making the start point match is easy as you just translate the path. With making the end points match, the path has to follow the mesh in order to conclude that the end points truly match. We needed to find a rotation to apply to the path that would make end points match as close as possible while the path is projected onto the mesh. In order to do this, I first rotated the path onto the mesh meaning the first segment of the path was on the same plane as its triangle. I then rotated the path along the plane made by the first triangle until the end points matched up closely. I then took the rotated path and projected it onto the mesh. I then found the rotation required to match the projected path endpoint with the target endpoint. I used that angle and rotated the original rotated path along the surface that much. I then projected that path and repeated the process until I got convergence. The aggregate rotation that resulted was used as the rotation calibration and applied to the rotation found by the probe to get the rotation in the virtual world.

Here is the pseudo-code for the rotation calibration:

N: normal to the first triangle  
 P: original path  
 P': current path rotated  
 P'': the path P' projected onto the mesh

Rotations will be in axis-angle form as (T,E)  
 where T is the angle and E is the axis

1. Let p be the projection of the first segment of P onto the first triangle
2. Rotate P so that its first segment matches p and call the rotated path P'
3. Find the rotation (Theta,E) that will make the end point of P' match the target end point
4. Apply the rotation (Theta,N) to P' to rotate it along the same plane as the first triangle
5. Find the path P'' which is P' projected onto the mesh
6. Find the rotation (Theta,E) that will make the end point of P'' match the target end point
7. Apply the rotation (Theta,N) to P'
8. Repeat steps 5-7 until P' converges

Output the quaternion Q such that  $P' = Q * P$

## Experimental Results

After applying the calibration algorithms to obtain solid calibration numbers, we ended up being able to track somewhat well. There were two serious limitations to improving the tracking further. The end points were not rigid and thus the original probe path did not end up in the spot that we would have wanted. Additionally, it was difficult to tell exactly where the probe was on the model so when specifying calibration points, there were measurement errors.

## Conclusion and Future Work

This proved to be a good first step and proof of concept for tracking on a deformable surface using simple hardware. Once the process is refined, there are numerous applications to this hardware and software combo.

## References

- [1] Adobe Systems Incorporated. Tracking and stabilizing motion.
- [2] Salzmann M. Hartley R. Fua P. Convex optimization for deformable surface 3-d tracking.
- [3] Schulman J. Lee A. Ho J. Abbeel P. Tracking deformable objects with point clouds. *ICRA 2013*.
- [4] Huang J. Takashima K. Hashi S. Kitamura Y. Im3d: Magnetic motion tracking system for dextrous 3d interactions.

## Appendix