Spring, 2011 Haptics Class Project Paper presented at
the University of South Florida, May 2, 2011.

Development of Volumetric Models in a Tele-operated Environment with Force Smoothing Playback

Eric W. Petersen

Abstract—The purpose of this project was to develop a method of creating a virtual model from user exploration in a tele-operated environment. The virtual model is developed by a creating a finite and discrete voxel matrix. The model developed after sufficient exploration, would be replayed back to the user with a force shading technique. This voxel based rendering system can possible be used to fill in extraneous time delays in tele-operated systems.

I. Introduction

When developing a virtual model by measuring a surface rarely do you encounter a smooth and simple surface. Finite and point based models are typical for data retrieved from a real surface. This data is usually non-uniform and can present issues when rendering these surfaces haptically. The discrete points of the models can create force discontinuities and gaps. Many algorithms have been developed to render these forces by interpolating the values of the nearest points, such as the methods proposed by Kim and Sukhatme [1]. By comparing the force at the current position within the virtual model with the nearest points surrounding the current position, the rendered force vectors reduce the discontinuities produced when using the previous traditional single point rendering techniques.

By applying these finite element rendering techniques to a tele-operated environment, a sufficient model can then be developed representing the surface can be developed in real-time. The model development algorithm produced in this paper can ultimately used to produce a smooth and continuous surface within a static tele-operated environment. When using a tele-operated system over a variable time delay environment such as communication via the internet. This voxel model can be used to smooth the environment for the end user when the delay becomes too great for the tele-operation to operate smoothly. The model can be updated continuously when the data from the tele-operated environment is available.

The algorithm uses discrete points at regular intervals to store rendered force data pertaining to the forces generated when an object is contacted within the tele-operated environment. Using points at regular intervals provides unique advantages when developing the model, such as:

- Reducing computational time for finding the nearest point.
- Data can easily be called upon using three dimensional arrays defined by the physical position within the model.
- Models can be continuously and quickly updated and smoothed as they are received from the tele-operated environment.

II. Related Work

A. Background

When working within the defined workspace, the model of the surface is developed when the force generated reach a value over a pre-defined threshold. This is similar to the work completed by Mukherjee in which they developed algorithms for an artificially intelligent tele-operated robot object detection system for its defined workspace using a Volumetric modeling [2]. Voxels are volumetric pixels defined by specific positions within the work space. These Voxels can store object data such as the force components as well as normal vector components to the surface at that particular point. Mukherjee subsequently used these Voxels to store data about objects within the workspace so the tele-operated robotic arm can move within the workspace and avoid contacting these objects.

Some of the techniques considered before developing this algorithm were the finite polygons methods developed by Kim and Sukhatme [3]. These polygons were defined by a minimum of three points to create the surface normal. They subsequently applied their previous methods of force smoothing and shading techniques to these polygonal surfaces removing the force discontinuities present when only rendering the single normal to the polygon. This method was difficult to reproduce using the Voxel method since it would require finding the closest previously defined point within the Voxel matrix, finding the normal of the polygon at the current position, finding the adjacent polygons, calculating and then smoothing the target polygon's normal with the surrounding polygons and their respective normal vectors. This is an acceptable method for uses such as individual model development where the individual polygon positions are initially defined as well as which points they are connected to. The object will have a static number of polygons and only allows for updates of the position of the corresponding nodes. With the Voxel method, points are only defined when the force threshold is reached. Therefore, the number of polygons would be extremely variable within the Voxel matrix and would require continuous updates until every Voxel is defined. This would realistically never be achieved because the only Voxels should be generated at the surface of the object. This method would require additional computational time which

could be better allotted to other processes such as the force rendering and smoothing.

B. Computational Methods

Much of the computation when using the proposed method in this paper, places the computational load in looking up values and modifying them in place of calculating the exact solution for each subsequent iteration. Some of the computational techniques proposed by El-Far, and Georganas were considered for this program [4]. Their methods included solving systems of linear equations for a finite element deformation model. The computational load is highly dependent on the number of elements within the array. If a deformable surface was modeled, an elastic component would be required for each component within the Voxel array. A system of linear equations would need to be solved in order to define this component for each of the defined Voxels. This elastic model would most likely be the subject of future work. By allowing the force values to be searched for instead of computed for the Voxel matrix, computational power can be allotted for solving the system of linear equations only if needed.

III. Implementation of the Voxel Matrix with Tele-Operation Model Development

The tele-operation of the Phantom Omni's used in the development of this algorithm utilized the position exchange equation (eqn. 1) which provided force feedback for the master Omni. When operating in playback mode an additional component was added into the equation in order to provide the required force obtained from the Voxel matrix (eqn. 2).

$$F_m = -K_{pm}(X_m - X_s) - K_{dm}(\dot{X}_m - \dot{X}_s) \quad \text{(eqn. 1a)}$$

$$F_s = K_{ps}(X_m - X_s) + K_{ds}(\dot{X}_m - \dot{X}_s) \quad \text{(eqn. 1b)}$$

$$F_m = \overline{Fn}_{voxposition}(X_{proxy} - X_m) - K_{pm}(X_m - X_s) - K_{dm}(\dot{X}_m - \dot{X}_s) \quad \text{(eqn. 2a)}$$

$$F_s = K_{ps}(X_m - X_s) + K_{ds}(\dot{X}_m - \dot{X}_s) \quad \text{(eqn. 2b)}$$

The normalized force vector and the distance between the defined proxy position and the master position are used to replay the forces defined by the Voxel matrix. This allows for the incorporation of the forces experienced by the end user set by the current position within the Voxel matrix.

A. Development of the voxel matrix

Initially, the program starts by loading a blank voxel matrix. This is accomplished by using the struct() command and defining a three dimensional array of these structures. The three dimensional array is used to define the position within the array as well as the size and shape of the workspace. It was not necessary to define the actual position of each voxel within the struct() command since it was handled by the three dimensional array. Only seven values were stored within each voxel, the three components (denoted as fx, fy, fz) of the force vector, a single bit character which allows for quick access indicating whether or not that particular voxel has been set, and finally three points for the normalized unit vector components (denoted as n_fx, n_fy, n_fz) to the applied force. Since the position values received from the Omni are in units of millimeters, a voxel was placed at regular intervals of one millimeter. The matrix used in this program was set for a workspace of 10 x 10 x 10 centimeters. This resulted in one million data points with a total of six double variables and one bit variable each. The origin of the voxel matrix was defined through constant at the beginning of the program locating the array within the Omni's workspace. Also, a scaling of the voxel resolution is possible by multiplying the integer values from the three dimensional array by a scaling factor. This was not necessary for this program since the Omni's have three un-actuated degrees of freedom which hinder the ability to operate efficiency in tele-operation.

B. Recording forces and setting values to the voxel matrix

If the user is exploring the tele-operated environment via the master Omni and comes into contact with an object, the slave is prevented from reaching its desired position mimicking the master's position. This difference in position between the master and the slave Omni allows for a force to be generated proportional to the difference. The forces generated are measured and compared to a preset threshold value. If this generated force is greater than the threshold value, the position of the slave is measured and truncated, as shown in figure 1. The truncated position value provides an integer which can then be used to directly call the particular voxel in which the Omni is in contact with.

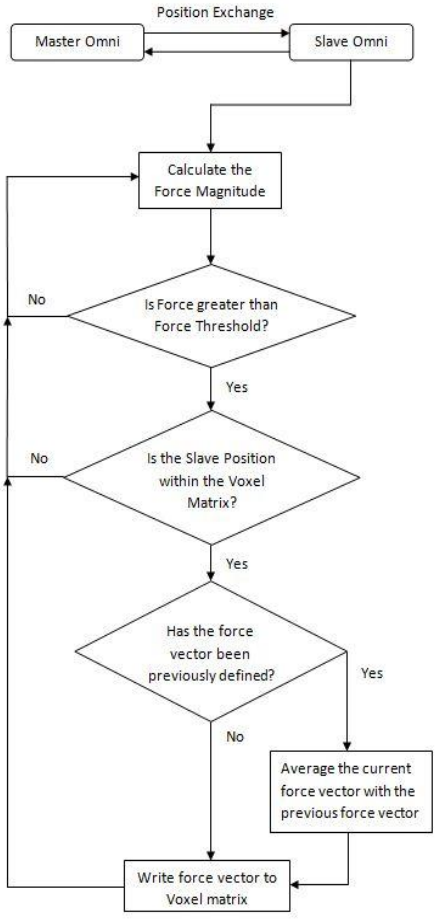

Figure 1: Flow chart for the development of the Voxel matrix.

If the position of the Omni is within the predefined voxel array, then the program checks to see if the value has been previously set. The value is either set for the first time or averaged with the previous value. The averaging of this force vector will help reduce the error between

the true normal to the surface and the measured value. These errors can arise from the dynamics of the system or the friction on the surface of the object. It is also worth mentioning that some of unintended triggered Voxels can be avoided by either placing a velocity or an acceleration threshold on top of the force threshold which would only allow for the Voxels being set only under specific conditions. A velocity threshold was employed when testing this algorithm but was ignored for two reasons; first the user rarely would exceed this threshold value while exploring the remote surface and second if the motion of the Omni was quickly reversed, the forced accelerating the Omni would be greater than the force threshold while the velocity would momentarily dip below the velocity threshold. A much better alternative would be to set an acceleration threshold, but since the Omni only reads position values the acceleration would require a second derivative of the position. This requires a minimum of three cycles in the callback loop and would create a large amount of error.

C. Playback of the rendered forces and applying force feedback to the user

Once a sufficient number of Voxels have been defined by the user exploring the surface, the resulting forces can be generated and displayed to the user. At the termination of the recording loop, the direction of each force vector is reversed and normalized. The file is then saved with all seven values of each voxel. This is the most computationally demanding part of the algorithm, therefore it is held back until the end of the recording loop. Once these values are defined the playback of these rendered normal forces can then be generated.

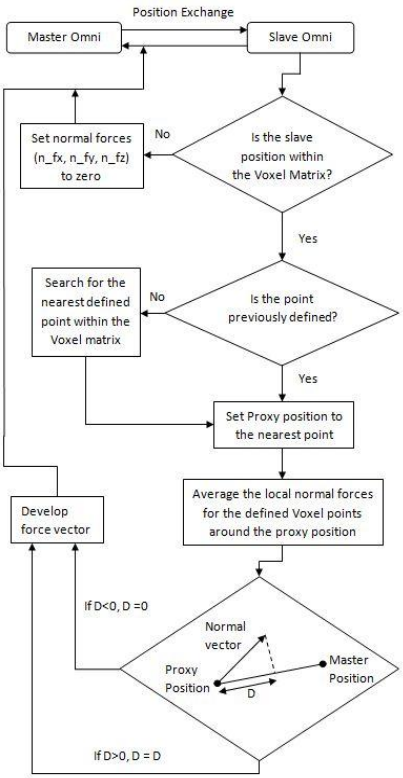

Figure 2: Flow chart for virtual surface playback from the Voxel matrix.

The file is first loaded into the array since reading straight from the output file would take too long. When the file completes its loading procedure, the Omni's continue operating in position exchange, except now the equation contains the value for the nearest defined point within the array. The algorithm first checks to see if the position is within the workspace. If the position is outside of the voxel matrix, force direction components are simply set to zero which allows the master and slave to continue operating in position exchange operation.

Once the position is within the voxel array, the position is checked against the array to see if has been previously defined. If the particular value has not been set, the algorithm is checked to see if another value of the array has been set within the pre-specified end effector size. Instead of calculating the distance to the nearest point for every point within the maximum size of the end effector, another three dimensional array was employed. Using MATLAB, a simple bubble sort method was used to sort every point within a 10 x 10 x 10 millimeter array from shortest to the furthest distance with respect to the center point of the array (example of this code can be found in the Appendix). This yielded a thousand x, y, z points which can quickly be checked against in order to find the closest point. This results in the main program searching in circular fashion around the end effector and stopping when a set value is found.

Once the closest point has been located, it is then set as the proxy position. This proxy position is used to generate a proportional response in the direction of the normal force vector at that proxy position relative to the position of the master Omni. Also, using the a proxy position the force direction is averaged with the surrounding defined force vectors in order to smooth the forces experienced by the end user.

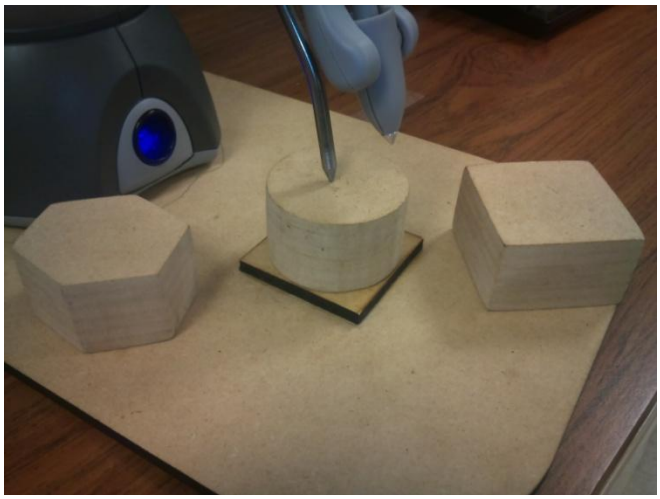

Figure 3: Testing Apparatus. Simple shapes and extended end effector.

Then a simple if then statement is employed to check the direction between the force vector and the vector developed by the difference between the endpoint and the proxy position. For example, if the differences between the position components are in the negative direction and the normal components are also in the negative direction the multiplied result is positive. Summarily, if the position component is positive and the force vector is positive the resultant is positive. Only when the position and force components are opposite will a negative result be produced. If the result is found negative, the force component is set to zero. This avoids producing a force value within the position

exchange equation the case where you are above the defined voxel array. If this were allowed, the user would feel a force pulling the user back to the defined proxy position.

IV. Results

The voxel array has proven to be successful in properly displaying the forces back to the user. Unfortunately, the accuracy was highly dependent on the number of Voxels generated when the user explored the object. If an area with only a few defined Voxels were found, the resulting forces were not very smooth. This effect could have been accounted for by simply letting the user explore the surface for a longer period of time. This would generate more points in which to reference, removing the discontinuities experienced by the end user.

The Omni's were not particularly suited for this task. Since only three degrees of freedom were available for control, the objects tested were very limited (Figure 3). A modified end effector was employed through an attachment to reduce the amount of error caused by the Omni's end effector (Figure 3). Also, the forces generated by the Omni's were very low which made the friction forces a determining factor in the development of the appropriate force vector.

The resulting forces and the defined Voxels were rendered graphically using MATLAB (Figure 5). Figure 5 also shows the skewing of the cylindrical object due to the extension of the Omni's working length.

V. Future Work

Some of the future work includes reducing the amount of error generated by the sparse voxel matrix. This can be reduced further and produce a more homogeneous response for the end user.

Instead of running this algorithm as two different programming modes, the voxel matrix can be employed in real-time in order to test if the voxel matrix would be an appropriate stand in if the delay between the master and slave Omni's were increased. This could prove to be a good compromise for the effects to extraneous delays in communication time. Ultimately this method could produce a smoother experience for the end user.

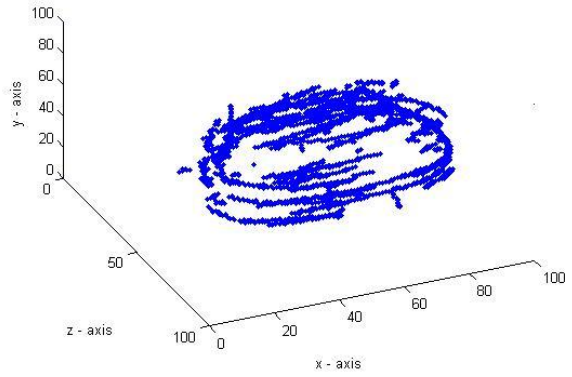

Figure 4: Cylindrical object rendered in MATLAB.

Figure 5: Slave Omni in operation.

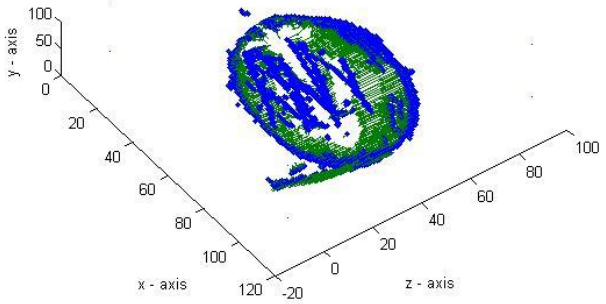

Figure 6: Measured forces represented by vectors in green.

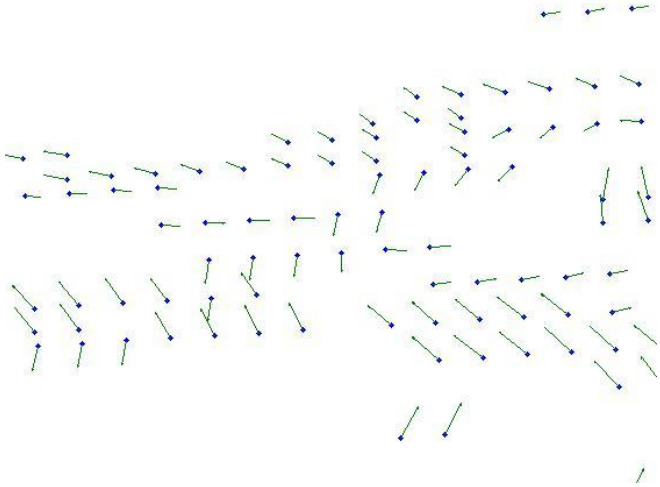

Figure 7: Close up rendering of the corresponding normal unit vectors for each voxel.

Acknowledgment

Special thanks to Dr. Reed for the advisement and assistance in generating the necessary code.

REFERENCES

[1] Laehyun Kim; Kyrikou, A.; Sukhatme, G.S.; Desbrun, M.; , "An implicit-based haptic rendering technique," *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on* , vol.3, no., pp. 2943- 2948 vol.3, 2002.

[2] Mukherjee, J.K.; , "AI based tele-operation support through voxel based workspace modeling and automated 3D robot path determination," *TENCON 2003. Conference on Convergent Technologies for Asia-Pacific Region* , vol.1, no., pp. 305- 309 Vol.1, 15-17 Oct. 2003.

[3] Laehyun Kim; Sukhatme, G.S.; Desbrun, M.; , "A haptic-rendering technique based on hybrid surface representation," *Computer Graphics and Applications, IEEE* , vol.24, no.2, pp. 66- 75, March-April 2004.

[4] El-Far, N.R.; Georganas, N.D.; El Saddik, A.; , "An algorithm for haptically rendering objects described by point clouds," *Electrical and Computer Engineering, 2008. CCECE 2008. Canadian Conference on* , vol., no., pp.001443-001448, 4-7 May 2008.