Learning Soft-Tissue Simulation from Models and Observation

Jie Ying Wu, Adnan Munawar, Mathias Unberath, Peter Kazanzides

***Abstract*— Accurate soft-tissue simulation using biomechanical models is computationally expensive. This is unfortunate because accurate biomechanical models could model tool-tissue interaction during surgical procedures, thereby providing intraoperative guidance to surgeons. In this work, we present steps toward interactive soft-tissue simulation for specific models using a learning-based framework that learns from finite element method (FEM) simulations. We train a graph neural network that takes the position and velocity of a tracked tool as input and estimates the deformations of a base mesh at each time step. By using data augmentation, the network learns to self-correct for errors in estimation to maintain the stability of the simulation over time. This approach estimates soft tissue deformation with less than 1 mm mean error with respect to FEM simulation over an interaction sequence of 80 s. This error magnitude is within the accuracy of FEM in comparing to the in situ camera observations of the interaction. While the FEM took 15 h to simulate 80 s of interaction, the network-based simulator took 47 s. Despite several open challenges that will be the subject of our future work, this learning-based framework constitutes a step towards real-time biomechanics-based simulation for intraoperative surgical guidance.**

I. Introduction

Computer-assisted interventions are increasingly adopted for different procedures, yet clinical plans are developed on pre-operative images that are often hard to register with the surgical scene once tissue starts deforming. One approach is to use an intraoperative imaging modality, such as ultrasound, that can track the deformation in real-time and then be used to "warp" the preoperative image to match the deformed anatomy. Mohareri et al. [1] demonstrated this approach clinically using transrectal ultrasound to track prostate deformation during robot-assisted surgery, thereby enabling overlay of a preoperative magnetic resonance image (MRI). Alternatively, navigation systems can make use of partial observations [2] or track deformations through biomechanical models [3] and then register to the surgical scene to provide intraoperative guidance. The current gold standard for accurate soft-tissue modeling is the finite element method (FEM). Unfortunately, FEMs, especially volumetric models that can track topological changes such as cutting, are generally too slow to be used intraoperatively. While there are approaches that speed up FEM using pre-processing [4], multi-threading [5], and GPU [6], these have so far only supported a limited set of geometric shapes and interactions. Alternative methods for soft-tissue simulations that can run in real-time, such as ChainMail [7], have not been widely adopted, potentially due to their lower accuracy if the mesh is not optimized. Brunet et al. identified a target of reducing errors to within 5 mm for providing useful clinical guidance with augmented reality and soft-tissue simulation [8].

Another series of work explores obtaining FEM-like accuracy faster with deep neural networks (NN). These approaches often use NNs to estimate one deformation that reflects change over a large number of time steps [9], [10]. This work presents an alternative approach where the network simulates at small, successive time steps as a step towards providing interactive soft-tissue simulations. We outline how this step-wise framework could integrate with works to incorporate real-time observations [11]. As a NN can simulate new, complex tool-tissue interactions in seconds, it can run at video-frame rates. We use a dataset captured on a da Vinci Research Kit [12] (dVRK), which provides kinematic data of the instrument interacting with a gel phantom, augmented with a depth camera to observe the actual deformation.

A. Background

While FEM is widely adopted, mesh-less algorithms can be used for difficult-to-mesh cases [13]. Another alternative to FEM is to use neighborhood-aware cell neural networks [14]. Since we focus on modeling one base phantom well, we assume that a mesh is available. This is generally the case in clinical scenarios, as we can build patient-specific meshes from preoperative images. On the other hand, mass-spring models [15] and ChainMail algorithms [7] can achieve real-time soft-tissue simulation; however, to obtain high enough accuracy for clinical use, they require optimizing the geometry of the mesh which limits their generalizability. We limit our review here to FEM and deep learning methods, and refer readers to [16] for a more general review of soft-tissue simulation methods.

B. Related Work

Several works have looked at speeding up FEMs. Haferssas et al. uses domain decomposition [17] to speed up computation. Other works have used deep learning to predict deformation over larger time steps than for which the FEM is generally stable. Meister et al. uses learning to predict the solutions to Total Lagrangian Explicit Dynamics to speed up FEM simulations [10], while Mendizabal et al. predicts the deformations directly [9]. Both are trained from simulated examples where a virtual force is applied to a simulated mesh and predicts a single step. They do not explore whether the predictions can be made successively to perform step-wise simulations. This work aims to complement these existing works and estimate deformations in sequential updates.

Dept. of Computer Science, Johns Hopkins University, Baltimore, MD 21218, USA (email: {jieying, amunawar, unberath, pkaz}@jhu.edu

While these works have generally used voxel-based convolutions, graph-based neural networks [18] could be more topography-aware than voxel-based ones. Since meshes are modeled as tetra- and hexahedrals, they naturally have a graph structure. Graph-based neural networks have been used to learn propagation in particle-based simulators [19], [20]. Sanchez-Gonzalez et al. encode particles as graphs and use message passing to learn the acceleration of each simulation step [20]. While these show the potential of graph networks to simulate deformations over many time steps, they consider plastic deformations only. Elastic materials such as those considered in this work present different challenges. On the one hand, the forces acting on a node not only depend on external factors like gravity and on the node's collision with its neighbors, but also on each node's initial position. On the other hand, the edges are predefined and do not generally change. This avoids the need for dynamic graph building as described by Li et al. [19].

II. METHOD

The goal of our method is to predict successive update steps in the manner of a traditional simulator, such as an FEM. We start with the 3D mesh of the object. We construct a network that takes the mesh and the robot's kinematics as input and estimates an update step as output. The update step is added to the mesh and the updated mesh is passed with the next set of robot kinematics to the network for it to predict the next update step, as shown in Fig. 1. The sum should match the next step of the FEM. For stability, the update in each step is limited to be at most one voxel length, based on the mesh element's size, in each direction.

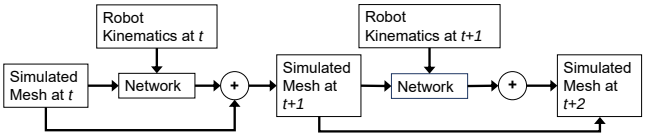

Fig. 1. Overview of the proposed system. The network takes the current mesh and robot kinematics and predicts an update step. The update step is added to the current mesh and the sum is passed to the network again, with the next robot kinematics, to do step-wise simulation.

A. Network Input

The input to the network is a graph where each node of the graph corresponds to a vertex on the mesh. The features of the node consist of mesh and robot kinematics information.

a) Mesh: The graph representation of the mesh consists of the relative node positions and edges. The edges are defined by the mesh. The relative node positions are represented by the differences between each node's current position and its position in the base mesh. Intuitively, this encourages the network to learn the physical property of elasticity without needing to implicitly learn where each node's resting position should be. Without an external force, the mesh should return to its original shape. Using the offset rather than absolute position of mesh nodes may also help with learning a generalizable convolutional kernel.

b) Robot kinematics: For each node in the network, we extend its feature representation with robot kinematics features. These kinematics features are initialized as zero for all points of the mesh. Then, if the robot is within a threshold of the mesh, we set the kinematics features of the closest node to the position and velocity of the end-effector. For our setup, we empirically determined the threshold size to be 5.5 mm based on the instrument tip. The velocity of the end-effector is computed separately from the position based on the method proposed in [21]. The more the robot moves, the more the mesh should deform. Since we limit our consideration to soft-tissue deformations, the deformation is largest where the robot touches the mesh and radiates outwards from there.

c) Data Augmentation: We use a data augmentation strategy that aims to capture the network error. Once the errors of the network converge in predicting the next FEM update, we introduce the following augmentation to the input. For 50% of the input, rather than loading the mesh at time t, the network instead loads a simulated mesh. The simulated mesh is constructed by applying the kinematics and FEM mesh from $t-1$ as inputs to the network from the last epoch. This is then used as the input to the current network to predict the next update step. Fig. 2 shows how the simulated mesh is generated and used as input for training the network.

The network predicts the next simulation update and its output is added to the simulated mesh. The sum is compared to the FEM output at the $t+1$ step. This step is important to prevent simulation errors from accumulating over time steps. Since the FEM should simulate step-wise deformation of the tissue, its inputs at test time would diverge from the training samples if its training data is drawn only from the FEM. By incorporating the network simulation in data augmentation, the network learns to correct for errors it introduces.

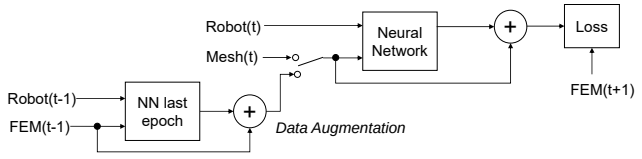

Fig. 2. Data augmentation strategy; once network is sufficiently trained, 50% of input uses simulated mesh predicted from previous FEM mesh.

B. Network Architecture

We use a U-Net architecture as it has been shown in the literature [2], [9] to work well in estimating mesh deformations. Unlike previous works, we use graph-based convolutions as we observed that the voxel-based network accumulated errors at the edges of the mesh due to padding. The network architecture, shown in Fig. 3, is based on the method described in [18]. We use ReLU for internal activation functions and sigmoid for last layer activation. The output should have the same number of nodes as the input and have feature size three representing the vertex position update step. The output of the sigmoid is shifted to be zero-centered and scaled based on the voxel size.

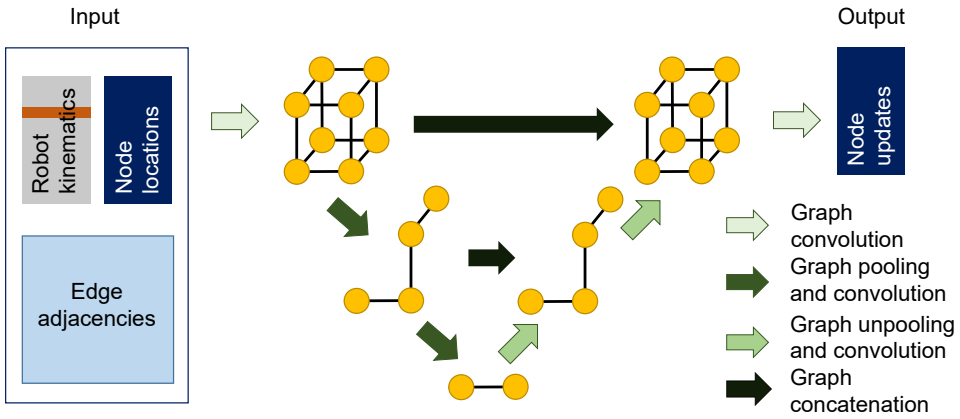

Fig. 3. Network architecture. The input consists of three parts. The node locations and edge adjacency are given by the mesh structure. While the node location is updated at each step, the edge adjacencies stay constant. The robot kinematics features are zero everywhere except where the robot end-effector is closest to the mesh, where it is the end-effector's position and velocity (indicated by the orange stripe). The network involves convolutions over the graph structure and pooling and unpooling between layers. Each of the network's hidden layers has 256 features. The output consists of three features for each mesh node to update its position for the simulation step.

C. Visualization

We implemented a visualization client in Blender [22] to show the similarities and differences between the simulations. Blender is a free, open-source and versatile tool that can be extended via its plugin-based API for different applications. We used this API to develop a visualizer (*blender visualizer*) for our *simulation frame* files. The graphical interface of the *blender visualizer* is shown in Fig. 4.

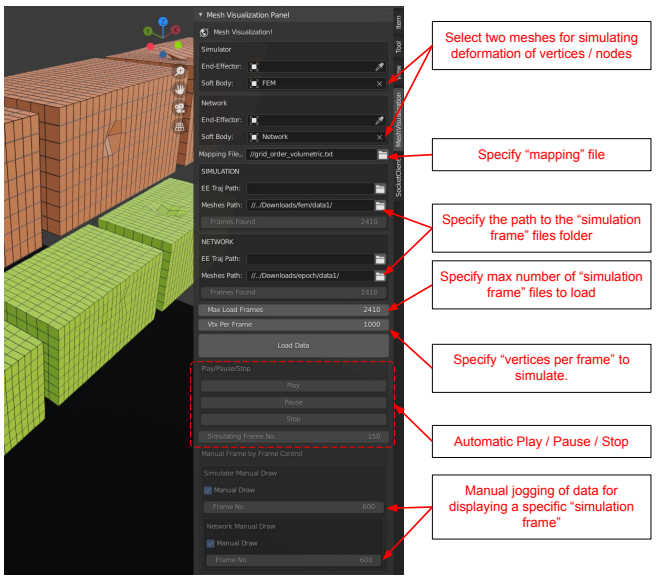

Fig. 4. The *blender visualizer* plugin. Users can pick two meshes to simulate at once. The plugin supports both automatic playing of data as well as manual jogging by providing the specific *frame number* to simulate.

Our custom visualizer was motivated by the fact that using existing simulators resulted in conflicts between the network updates and the simulator clock. We realized the importance of a portable package that could be used across different research projects. For this purpose, we first defined a simple data format for outputting the soft-tissue deformation data. This data format requires a predefined volumetric mesh, which could be created from pre-operative scans. Then for each time step, the data format contains a sequential array of vertex / node positions at each simulation time-frame. Thus, for example, a simulation episode consisting of 30 seconds with 10 frames per second (FPS) would result in 300 text files, where each file is called a *simulation frame* file.

To map the indices of the vertices / nodes in the *simulation frame* files and the mesh represented in Blender, we created another text file, called the *mapping* file. The format in this text file is a two column array, where the first column is the index of the vertex / node in the *simulation frame* files and the second column is the corresponding vertex index of the Blender mesh. Fig. 5 shows an example of the how the mapping file is used to match the vertices from the network output to the Blender scene.

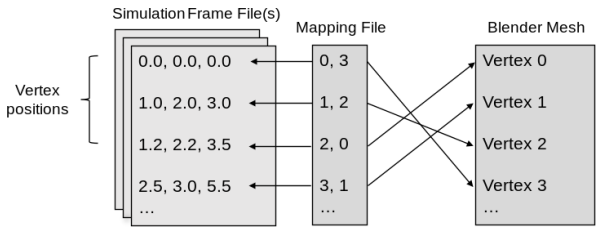

Fig. 5. Example of using the mapping file to match vertices from network to Blender visualization.

To generate these mappings, we created another Blender plugin (*blender client*) that provides an interface for querying vertex positions and indices of the mesh. We complemented this with a Python package (*user server*) that abstracts the socket interface and provides the user with an API for querying or commanding vertices of the Blender mesh. The *user server* can process multiple vertices / nodes at the same time. Fig. 4 and Fig. 6 show the graphical interface of the *blender visualizer* and *blender client*, respectively. Fig. 7 shows the API exposed by the *user server*.

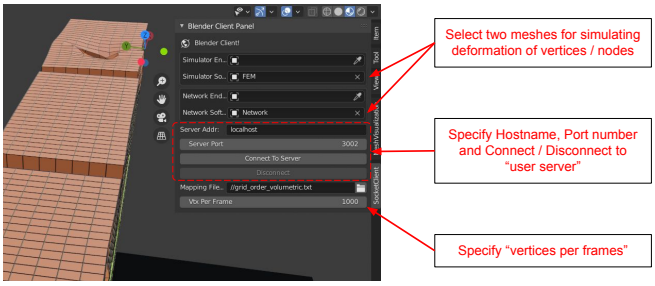

Fig. 6. The *blender client* plugin.

The *blender visualizer*, *user server* and *blender client* together allow for the visualization of any soft-body deformation data once it is ported to the *simulation frame* format. Blender's plugin API executes the callbacks in sequence with the graphics loop and as a result, any form of heavy processing within the callback method slows down Blender. A few examples are querying and updating multiple vertices at once and processing the socket interface. To support real-time visualizations, we implemented a combination of maintaining a shared data queue and populating it using separate threads. The plugin's callback method then reads a maximum number of vertex commands, specified by a user defined *vertex per frames* parameter, from the shared data queue. This allows for the real-time visualization of soft-tissue deformation. The source code of all these packages is provided [1]. This communication protocol allows us to use the neural network output to replace the physics backend and visualize estimated deformations in real time.

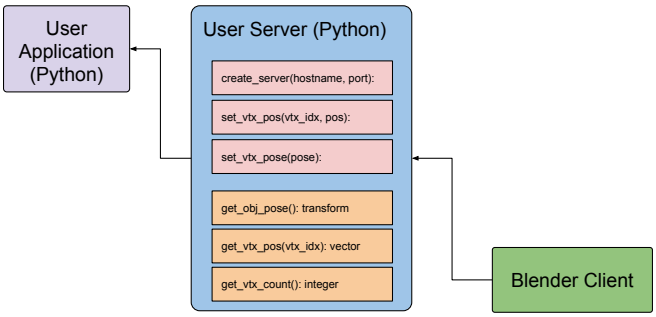

Fig. 7. The API of the *user server*.

III. Experiments

A. Data collection

We used the dataset described in [11] and we point the reader there for a more detailed description of the experiment setup and data processing. Briefly, an Intel Realsense SR300 RGBD camera (Intel, Santa Clara, CA) was mounted above the workspace of a da Vinci patient-side manipulator (PSM). The phantom was made from a homogeneous plastic material and its dimensions are $68.7 \times 35.8 \times 39.3$ mm. The setup is shown in Fig. 8. The depth resolution of the camera is 2 mm. Each frame of the depth camera is read out as a point cloud. We use the Point Cloud Library (PCL) [23] to remove points from the instrument and the table, as well as outliers, so that the only points that remain are the ones from the phantom. The scene is primarily composed of three objects: the table, the phantom, and the robot instrument. We cluster the points by a manually identified distance threshold. This is sufficient to separate the points into three groups, with each group belonging to one of the objects in the scene. As the phantom is the second largest object in the scene, our segmentation algorithm selects the second largest group of points as belonging to the phantom and removes all other points. Then we subsample the points to about 45k points, or about 16.5 points per mm^2.

The depth camera point cloud was recorded at 30 Hz, while the robot kinematics was recorded at 1 kHz as the PSM was manually moved to interact with a gel phantom. Each sequence was between one to two minutes long and contained several distinct interactions with the phantom. The tool was lifted to not touch the phantom between interactions. Each sequence is a set of palpation actions performed at different speeds. The robot kinematics were subsampled to match observations of the point cloud. We use ten sequences from the dataset that were captured at the same frequency. Eight sequences were used for training, and one each for validation and test. This results in around 12 minutes, or 21k frames, of video. Of those interactions, one sequence of 3794 frames is used as the validation set. One sequence of 2410 frames is used as the test set.

Fig. 8. Image of the setup. An Intel Realsense SR300 RGBD camera is mounted above the phantom and the robot. Reprinted with permission from Springer Nature from [11].

Following our previous approach [11], we created FEM simulations using the SOFA Framework [24]. We used the optimal parameters found in the previous study and set the Young's Modulus to 5×10^3 and the Poisson's ratio to 0.44. While a coarser mesh of $13 \times 5 \times 5$ was determined to be sufficient in our previous work, we increased the size of the mesh to $25 \times 9 \times 9$ here as it resulted in the network learning more stable simulations. We empirically set the FEM simulation time step to 0.1 ms as a balance between computation time and accuracy of the FEM. The robot and camera frames are calibrated to match the simulation frame. We use iterative closest point (ICP) [25] with manual initialization to register

[1] https://github.com/adnanmunawar/blender_socket_comm

between the point cloud and the simulation scene. We move the end-effector to each of the four corners of the block and calculate the rigid transformation between the robot and the simulation scene.

B. Implementation

This work was implemented on a workstation with a Xeon Processor E5-1630 v4 CPU (Intel, Santa Clara CA, USA) and a Titan V GPU (Nvidia, Santa Clara CA, USA). We implemented the network in PyTorch [26]. Since neural networks have been shown by previous work [10] to be stable over larger simulation steps compared with FEMs, we sampled the output of the FEM to match the camera's frequency of 30 Hz to create the training data for the simulator network. We used L2 loss between the network output and the FEM ground truth. We chose stochastic gradient descent as the optimizer, setting momentum to 0.9 and learning rate to 10^{-4}, with decay on plateau scheduler. We initially trained with each update step set to be at most half a voxel-step in each direction to speed up convergence. Once that has converged, we increased the step size to one full voxel.

IV. Results

We test the simulation network on the held out sequence, which is 80 s long and consists of several interactions with the phantom both on top and on the side. Fig. 9 shows simulation results from both the FEM and the network for samples from the test sequence, using the visualization software described in Section II.C.

Qualitatively, the network errors come from underestimating the deformations in some cases. Interactions to the side of the mesh may also introduce errors. We hypothesize that this is because interactions to different sides of the mesh have different effects and are thus harder to learn.

Although the entire sequence is performed as one simulation, we split the frames into 10 s segments for error calculations. This captures the range of errors resulting from different amounts of interaction with the phantom. In particular, the beginning and end sequences may have fewer interactions. This results in 8 sequences. As Table I shows, the error between the FEM and the network prediction for each sequence, as measured by Euclidean distance, is small. It is much smaller than the distance from point cloud to either the FEM or network simulation, which we measured as the average distance from each point in the point cloud to its closest neighbor on the mesh. This suggests that the error between the network prediction and the FEM is within the range of error from modeling using FEM. The advantage of the network simulation is that the total runtime to generate sequences with the network is 47 s, where each second consisted of 50 simulation steps. This is sufficient for real time performance as the video sequence covers a 80 s time span. Network predictions can be made faster than the video frame rate. In comparison, the FEM took just under 15 h to run. The runtime is large for the mesh size as we use collision models between the mesh and the instrument, rather than directly applying forces on the mesh. The runtime could be reduced by applying the forces directly but direct force sensing is not currently available in most clinical surgical systems. Using a method for force estimation at the instrument tip, such as proposed in [27] [28], and applying the force directly to the mesh could reduce the simulation time. A larger step size could also be used to reduce run time, but at the cost of larger error when compared to the real observations.

TABLE I

Average (standard deviation) error in mm for each sequence between the FEM, the network simulation meshes, and the observed point cloud (PC). Error between FEM and network prediction meshes is measured as the Euclidean distance between each vertex. Point cloud to mesh error is measured as average distance of every point on the point cloud to the closest point on the mesh.

Seq.	**FEM** to **Net**	**PC to FEM**	**PC to Net**
1	0.51 (0.38)	4.56 (2.61)	4.48 (2.55)
2	0.60 (0.58)	4.15 (1.52)	4.00 (1.42)
3	0.54 (0.28)	5.85 (0.69)	5.63 (0.67)
4	0.65 (0.37)	5.31 (1.47)	5.05 (1.53)
5	1.04 (1.24)	3.48 (1.40)	3.19 (1.43)
6	0.81 (0.83)	3.97 (1.61)	3.88 (1.67)
7	0.86 (0.83)	4.28 (2.12)	4.03 (2.24)
8	0.59 (0.34)	5.92 (1.78)	5.70 (1.74)

Additionally, we evaluate how incorporating real time observations to correct the meshes would affect each case. We implement the 2D U-Net version of the correction network described in our prior work [11] and compare the error between the point cloud and both the corrected FEM and the corrected network outputs in Table II. Since we use a finer mesh, here, we skip the mesh refinement step in training. Due to limited data size, we use the same training and validation sequences for both the simulator and correction networks, which may reduce the performance of the correction network for the network predictions.

TABLE II

Average (standard deviation) error in mm for each sequence between the observed point cloud (PC) and corrected meshes. Point cloud to mesh error is measured as average distance of every point on the point cloud to the closest point on the mesh.

Seq.	**Corrected FEM**	**Corrected Network**
1	4.07 (2.52)	4.13 (2.26)
2	3.41 (1.40)	3.39 (1.27)
3	4.89 (0.56)	4.85 (0.54)
4	4.51 (1.41)	4.44 (1.41)
5	2.94 (1.34)	2.67 (1.33)
6	3.40 (1.53)	3.22 (1.59)
7	3.75 (2.02)	3.64 (2.13)
8	5.23 (1.69)	5.46 (1.60)

We observe that both sets of meshes also show similar performance after correction. This paves the path for a network-based soft-tissue simulator that can run at video-frame rates and can incorporate real-time observations.

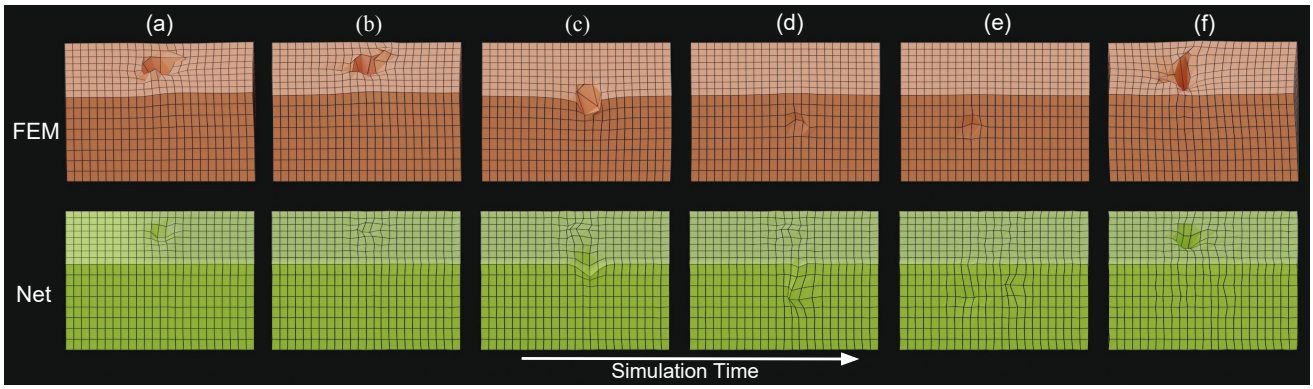

Fig. 9. Sample frames of the simulation sequence over 80 s and multiple interactions with the phantom. The network generally matches the FEM predictions. Comparing frames (a) and (b), we see that the network initially detected the interaction but returned to the undeformed mesh too quickly. In frame (d), we see that there is an error in the direction of the deformation and that propagated through the remaining frames as the network is slow to recover from unexpected errors.

V. Discussion and Conclusion

The proposed method is able to accurately simulate soft-tissue deformation by learning from FEM simulations. Its ability to run at video-frame rates at test time makes this approach also suitable for providing real-time feedback to users, such as for intraoperative guidance or a surgical simulator. After correction, its errors generally fall below the 5 mm identified as clinically useful [8]. As this is similar to the FEM error, improving the FEM by using a finer mesh may also improve the network predictions. Although the overall error is lower, we observe that the network predictions are smoother than that of the FEM. One potential solution for the over-smooth problem could be to train a GAN to sharpen the mesh, such as that used in [29]. Currently, we assume that the robot tracks the instrument end-effector position accurately and four points are sufficient to match the robot to the simulation frame. Incorporating tracking kinematics errors [30] and more extensive calibration techniques such as [31] may reduce the error for both the FEM and the network.

In this work, we used fairly coarse meshes as we focused on having enough data with which to train a network, similar to the cantilever used in [9]. An interesting extension is to include elements from networks used in plastic deformation such as in [20], which builds a graph in each step. This way, one could adopt the adaptive meshing used in FEMs [32] to better capture details in more deformed areas. The drawback is that the graph discovery step remains an open question for elastic materials. In plastic materials, the interactions only depend on each node's neighbors at each time step whereas in elastic materials, it also depends on a node's starting position.

Currently, all our training data sequences were collected at the same frequency. An extension to this work could incorporate time as a parameter of the network to account for simulations at different time steps. A next step can also look at generalizing to different geometries and material parameters. More work is needed to determine whether a network trained on one phantom could be fine-tuned to phantoms of different geometry and material.

Using real-time volumetric observations may improve data availability. Future work could use 3D ultrasound or other imaging sources to train directly from observations. This would skip the dependence on FEMs and entirely avoid the need to estimate material parameters and tissue boundaries. As collecting 3D observations of deformation is time-consuming, e.g. by using 3D ultrasound, or may require high radiation exposure, e.g. by using CT scans, we are also considering options to augment limited observations. One potential augmentation is methods that estimate the entire mesh from partial observation, such as [2], thereby replacing training from FEM with observations. Combining this with methods that construct point clouds from endoscope videos, such as [33], could allow data from any robotic procedure on a phantom with known geometry to be a potential source of training data. Future work may also consider pre-training on a phantom organ and using real observations to update the network for patient-specific characteristics. This may reduce the amount of patient training data required. For example, pre-training followed by training for patient-specific corrections has shown potential for force sensing [28].

This work has shown a first step of a deep-learning based soft-tissue simulator that can provide step-wise estimations of deformations. It has been integrated with the dVRK environment and can be incorporated in the open-source libraries as an interactive, accurate soft-tissue simulator. Additionally, an open source Blender client is provided to facilitate visualization of deforming meshes. These contributions can be used to provide real-time simulations and would enable future research that requires interacting with phantoms in simulated settings, such as training tasks and surgical challenges.

Acknowledgment

This research was supported in part by NSF OISE 1927354 and by a collaborative research agreement with the Multi-Scale Medical Robotics Center in Hong Kong. The Titan V used for this research was donated by Nvidia Corporation.

REFERENCES

[1] O. Mohareri, J. Ischia, P. C. Black, C. Schneider, J. Lobo, L. Goldenberg, and S. E. Salcudean, "Intraoperative registered transrectal ultrasound guidance for robot-assisted laparoscopic radical prostatectomy," *The Journal of Urology*, vol. 193, no. 1, pp. 302–312, 2015.

[2] M. Pfeiffer, C. Riediger, J. Weitz, and S. Speidel, "Learning soft tissue behavior of organs for surgical navigation with convolutional neural networks," *International Journal of Computer Assisted Radiology and Surgery*, vol. 14, no. 7, pp. 1147–1155, 2019.

[3] S. Suwelack, S. Röhl, S. Bodenstedt, D. Reichard, R. Dillmann, T. dos Santos, L. Maier-Hein, M. Wagner, J. Wünscher, H. Kenngott *et al.*, "Physics-based shape matching for intraoperative image guidance," *Medical Physics*, vol. 41, no. 11, p. 111901, 2014.

[4] S. Cotin, H. Delingette, and N. Ayache, "Real-time elastic deformations of soft tissues for surgery simulation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 1, pp. 62–73, 1999.

[5] S. Fialko and V. Karpilowskyi, "Multithreaded parallelization of the finite element method algorithms for solving physically nonlinear problems," in *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*, 2018, pp. 311–318.

[6] H. Courtecuisse, J. Allard, P. Kerfriden, S. P. Bordas, S. Cotin, and C. Duriez, "Real-time simulation of contact and cutting of heterogeneous soft-tissues," *Medical Image Analysis*, vol. 18, no. 2, pp. 394–410, 2014.

[7] J. Zhang, Y. Zhong, J. Smith, and C. Gu, "A new ChainMail approach for real-time soft tissue simulation," *Bioengineered*, vol. 7, no. 4, pp. 246–252, 2016.

[8] J.-N. Brunet, A. Mendizabal, A. Petit, N. Golse, E. Vibert, and S. Cotin, "Physics-based deep neural network for augmented reality during liver surgery," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2019, pp. 137–145.

[9] A. Mendizabal, P. Márquez-Neila, and S. Cotin, "Simulation of hyperelastic materials in real-time using deep learning," *Medical Image Analysis*, vol. 59, p. 101569, 2020.

[10] F. Meister, T. Passerini, V. Mihalef, A. Tuysuzoglu, A. Maier, and T. Mansi, "Deep learning acceleration of Total Lagrangian Explicit Dynamics for soft tissue mechanics," *Computer Methods in Applied Mechanics and Engineering*, vol. 358, p. 112628, 2020.

[11] J. Y. Wu, P. Kazanzides, and M. Unberath, "Leveraging vision and kinematics data to improve realism of biomechanic soft tissue simulation for robotic surgery," *International Journal of Computer Assisted Radiology and Surgery*, vol. 15, no. 5, pp. 811–818, 2020.

[12] P. Kazanzides, Z. Chen, A. Deguet, G. S. Fischer, R. H. Taylor, and S. P. DiMaio, "An open-source research kit for the da Vinci® surgical system," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 6434–6439.

[13] G. Joldes, G. Bourantas, B. Zwick, H. Chowdhury, A. Wittek, S. Agrawal, K. Mountris, D. Hyde, S. K. Warfield, and K. Miller, "Suite of meshless algorithms for accurate computation of soft tissue deformation for surgical simulation," *Medical Image Analysis*, vol. 56, pp. 152–171, 2019.

[14] J. Zhang, Y. Zhong, and C. Gu, "Neural network modelling of soft tissue deformation for surgical simulation," *Artificial Intelligence in Medicine*, vol. 97, pp. 61–70, 2019.

[15] G. San-Vicente, I. Aguinaga, and J. T. Celigueta, "Cubical mass-spring model design based on a tensile deformation test and nonlinear material model," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 2, pp. 228–241, 2011.

[16] J. Zhang, Y. Zhong, and C. Gu, "Deformable models for surgical simulation: a survey," *IEEE Reviews in Biomedical Engineering*, vol. 11, pp. 143–164, 2017.

[17] R. Haferssas, P.-H. Tournier, F. Nataf, and S. Cotin, "Simulation of soft tissue deformation in real-time using domain decomposition," Dec. 2019, working paper or preprint. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02400982

[18] H. Gao and S. Ji, "Graph U-Nets," in *International Conference on Machine Learning*. PMLR, 2019, pp. 2083–2092.

[19] Y. Li, J. Wu, R. Tedrake, J. B. Tenenbaum, and A. Torralba, "Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids," in *Intl. Conf. on Learning Representations*, 2019.

[20] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia, "Learning to simulate complex physics with graph networks," in *International Conference on Machine Learning*. PMLR, 2020, pp. 8459–8468.

[21] J. Y. Wu, Z. Chen, A. Deguet, and P. Kazanzides, "FPGA-based velocity estimation for control of robots with low-resolution encoders," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 6384–6389.

[22] B. O. Community, *Blender - a 3D modelling and rendering package*, Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. [Online]. Available: http://www.blender.org

[23] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE Intl. Conf. on Robotics and Automation*, 2011, pp. 1–4.

[24] J. Allard, S. Cotin, F. Faure, P.-J. Bensoussan, F. Poyer, C. Duriez, H. Delingette, and L. Grisoni, "Sofa-an open source framework for medical simulation," in *MMVR 15-Medicine Meets Virtual Reality*, vol. 125. IOP Press, 2007, pp. 13–18.

[25] P. J. Besl and N. D. McKay, "Method for registration of 3-D shapes," in *Sensor fusion IV: control paradigms and data structures*, vol. 1611. International Society for Optics and Photonics, 1992, pp. 586–606.

[26] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

[27] N. Yilmaz, J. Y. Wu, P. Kazanzides, and U. Tumerdem, "Neural network based inverse dynamics identification and external force estimation on the da Vinci Research Kit," in *Intl. Conf. on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 1387–1393.

[28] J. Y. Wu, N. Yilmaz, U. Tumerdem, and P. Kazanzides, "Robot force estimation with learned intraoperative correction," in *International Symposium on Medical Robotics (ISMR)*. IEEE, 2021.

[29] L. Fink, S. C. Lee, J. Y. Wu, X. Liu, T. Song, Y. Velikova, M. Stamminger, N. Navab, and M. Unberath, "Lumipath–towards real-time physically-based rendering on embedded devices," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2019, pp. 673–681.

[30] M. Hwang, B. Thananjeyan, S. Paradis, D. Seita, J. Ichnowski, D. Fer, T. Low, and K. Goldberg, "Efficiently calibrating cable-driven surgical robots with RGBD fiducial sensing and recurrent neural networks," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5937–5944, 2020.

[31] A. Roberti, N. Piccinelli, D. Meli, R. Muradore, and P. Fiorini, "Improving rigid 3-d calibration for robotic surgery," *IEEE Transactions on Medical Robotics and Bionics*, vol. 2, no. 4, pp. 569–573, 2020.

[32] S. Lo, "Finite element mesh generation and adaptive meshing," *Progress in Structural Engineering and Materials*, vol. 4, no. 4, pp. 381–399, 2002.

[33] X. Liu, A. Sinha, M. Ishii, G. D. Hager, A. Reiter, R. H. Taylor, and M. Unberath, "Dense depth estimation in monocular endoscopy with self-supervised learning methods," *IEEE Transactions on Medical Imaging*, 2019.