7 REFERENCES

[1] O. A. J. v. d. Meijden and M. P. Schijven, "The value of haptic feedback in conventional and Robot-assisted minimal invasive surgery and virtual reality training: a current review," *Surg Endosc*, pp. 1180-1190, 2009.

[2] A. LIu, F. Tendick, K. Cleary, and C. Kaufmann, "A Survey of Surgical Simulation: Applications, Technology, and Education," vol. 12 No.6, pp. 599-614, 2003.

[3] E. P. Westebring, V. D. Putten, R. H. M. Goossens, J. J. Jakimowicz, and J. Dankelman, "Haptics in minimally invasive surgery - a review," *Minimally Invasive Therapy*, vol. 17, pp. 3-16, 2008.

[4] M. Guiatni, V. Riboulet, and A. Kheddar, "Design and Evaluation of a Haptic Interface for Interactive Simulation of Minimally- Invasive Surgeries," in *2009 IEEE/ASME International Conference on Advanced Intelligent Mechatronics* Suntec Convention and Exhibition Center, Singapore, 2009, pp. 1336-1341.

[5] F. Kockerling, "Robotic Vs Standard Laparoscopic Tehnique- What is better?," *Front Surg*, vol. i, 2014.

[6] C. Basdogan, M. Sedef, M. Harders, and S. Wesarg, "VR- Based Simulators for Training in Minimally Invasive Surgery," *IEEE Computer Graphics and Applications*, pp. 54-66, 2007.

[7] J. Dankelman, "Surgical Simulator Design and Development," *World Journal of Surgery*, vol. 32, pp. 149-155, 2008.

[8] H. K. Kim, D. W. Rattner, and M. A. Srinivasan, "Virtual-reality-based laparoscopic surgical training: The role of simulation fidelity in haptic feedback," *Computer Aided Surgery*, vol. 9, pp. 227-234, 2004.

[9] L. Zhang, C. Grosdemouge, V. S. Arikatla, W. Ahn, G. Sankaranarayanan, S. De, D. Jones, S. Schwaitzberg, and C. G. L. Cao, "The Added Value of Virtual Reality Technology and Force Feedback for Surgical Training Simulators," 2012.

[10] T. R. Coles, D. Meglan, and N. W. John, "The Role of haptics in Medical Training Simulators: A Survey of the State-of- the-art," *IEEE Transaction on Haptics,* vol. 4 N0.1, pp. 51-66, 2011.

[11] E. Hiemstra, E. M. Terveer, M. K. Chmarra, J. Dankelman, and F. W. Jansen, "Virtual reality in laparoscopic skills training: Is haptic feedback replaceable?," *Minimally Invasive Therapy,* vol. 20, pp. 179-184, 2011.

[12] C. E. Buckley, E. Nugent, D. Ryan, and P. C. Neary, "Virtual Reality- A New Era in Surgical Training," 2012.

[13] S. M. B. I. Botden, S. N. Buzink, M. P. Schijven, and J. J. Jakimowicz, "Augmented versus Virtual Reality Laparoscopic Simulation: What is the Difference? ," *World Journal of Surgery,* vol. 31, pp. 764-772, 2007.

[14] M. Downes, M. C. Cavusoglu, W. Gantert, L. W. Way, and F. Tendick, "Virtual Environments for Training Critical Skills in Laparoscopic Surgery " in *Medicine Meets Virtual Reality* San Diego, CA, 1998, pp. 316-322.

[15] W. McMahan, J. Gewirtz, D. Standish, P. Martin, J. A. Kunkel, M. Lilavois, A. Wedmid, D. I. Lee, and K. J. Kuchenbecker, "Tool Contact Acceleration Feedback for Telerobotic Surgery," *IEEE TRANASACTIONS ON HAPTICS,* vol. 4, pp. 210-220, 2011.

[16] N. Enayati, E. D. Momi, and G. Ferrigno, "Haptics in Robot - Assisted Surgery : Challenges and Benefits," *Accepted for publication in a future issue journal " IEEE reviews in Biomedical Engineering",* 2015.

[17] C. Brino, S. Dargar, G. Sankaranarayanan, K. Matthes, and S. De, "Haptic Interaction During Natural Orifice Translumenal Endoscopic Surgery," in *IEEE Haptic Symposium* Houston, Tx, USA, 2014, pp. 617-622.

[18] C. Vapenstad and S. N. Buzink, "Procedural virtual reality simulation in minimally invasive surgery," in *Surg Endosc*, 2012.

[19] R. M. Satava, "Historical Review of Surgical Simulators - A Personal Perspective," *World J Surg,* vol. 32, pp. 141-148, 2008.

[20] H. d. Visser, M. O. Watson, O. Salvado, and J. D. Passenger, "Progres in virtual reality simulators for surgical training and certification," *The Medical Journal of Australia,* vol. 194(4), 2011.

[21] F. Wang, E. Burdet, R. Vuillemin, and H. Bleuler, "Knot - Tying with Visual and Force Feedback for VR Laparoscopic Training," in *Engineering in Medicine and Biology 27th Annual Conference* Shanghai, China, 2005, pp. 5778-5781.

[22] S. M. B. I. Botden and J. J. Jakimowicz, "What is going on in augmented reality simulation in laparoscopic surgery?," *Surg Endosc,* vol. 23, pp. 1693-1700, 2009.

[23] K. Moustakas, P. Daras, D. Tzovaras, and M. G. Strintzis, "Deformable Objects, Interface Simlation of," in *Wiley Encyclopedia of Biomedical Engineering*, 2006.

[24] J. Vozenilek, J. S. Huff, M. Reznek, and J. A. Gordon, "See One, Do One, Teach One: Advanced Technomogy in Medical Education," *ACAD EMERG MED* vol. 11, pp. 1149-1154, November, 2004.

[25] P. Dubois, Q. Thommen, and A. C. Jambon, "In Vivo Measurement of Surgical Gestures " *IEEE Transaction on Biomedical Engineering,* vol. 49, pp. 49-54, 2002.

[26] S. Misra, K. T. Ramesh, and A. M. Okamura, "Modeling of Tool-Tissue Interaction for Computer - Based Surgical Simulation: A Literature Review," National Institutes of Health, 2008, pp. 1-41.

[27] D. Morris, C. Sewell, F. Barbagli, K. Salisbury, N. H. Blevins, and S. Girod, "Visuohaptic Simlation of Bone Surgery for Training and Evaluation," *IEEE Computer Graphics and Applications,* pp. 48-57, 2006.

[28] Z. Pezzementi, D. Ursu, S. Misra, and A. M. Okamura, "Modeling Realistic Tool-Tissue Interactions With Haptic Feedback: A Learning-based Method," in *Symposium on Haptic Interfaces for Virtual Environments and Teleoperator Systems* USA, 2008, pp. 209-215.

[29] M. Graafland, J. M. Schraagen, and M. P. Schijven, "Systematic review of serious games for medical education and surgical skills training," *British Journal of Surgery,* vol. 99, pp. 1322-1330, 2012.

[30] M. Scandola, L. Gasperotti, M. Vicentini, and P. Fiorini, "The Role of Visual-Haptic Discrepancy in Virtual Reality Environments," in *IEEE Haptics Symposium*, 2012, pp. 289-195.

[31] H. Song, K. Kim, and J. Lee, "DEvelopment of the dexterious manipulator and the force sensor for Minimally Invasive Surgery," in *4th International Conference on Autonomous Robotics and Agents*, Wellington, New Zealand, 2009, pp. 524-528.

[32] F. G. Hamza-Lup, A. Seitan, D. M. Popovici, and C. M. Bogdan, "Medical Simulation and Training: "Haptic" Liver," in *The 7th International Conference on Virtual Learning ICVL*, 2012, pp. 27-33.

[33] C. Basdogan, D. Suvranu, J. Kim, M. Muniyandi, H. Kim, and M. A. Srinivasan, "Haptics in Minimally Invasive Surgical Simulation and Training," *IEEE Computer Graphics and Applications,* pp. 56-64, 2004.

[34] G. R. D.-. La-Torre, "The Important of the Sense of Touch in Virtual and Real Environments," in *Haptic User Interface for Multimedia Systems*, 2006, pp. 24-30.

[35] M. A. Srinivasan, "What is Haptics?," Laboratory for Human and Machine Haptics: The Touch Lab, Massachusetts Institute of Technology, pp. 1-11.

[36] A. E. Saddik, "The Potential of Haptics Technologies," in *IEEE Instrumentation and Measurement Magaine*, 2007, pp. 10-17.

[37] P. Strom, L. Hedman, L. Sarna, A. Kjellin, T. Wredmark, and L. Fellander-Tsai, "Eaely exposure to haptic feedback enhances performance in surgical simulator training: a prospective randomized crossover study in surgical residents," *Surg Endosc,* vol. 20, pp. 1383-1388, 2006.

[38] H. K. Kim, D. W. Rattner, and M. A. Srinivasan, "Virtual - reality- based laparoscopic surgical training: The role of simulation fidelity in haptic feedback," *Computer Aided Surgery,* vol. 9, pp. 227-234, 2004.

[39] M. Yiasemidou, D. Glassman, P. Vasas, S. Badiani, and B. Patel, "Faster simulated laparoscopic cholecystectomy with haptic feedback technology," *Open Access Surgery,* vol. 4, pp. 39-44, 2011.

[40] P. Lamata, W. Ali, A. Cano, J. Cornella, J. Declerck, O. J. Elle, A. Freudenthal, H. Furtado, D. Kalkofen, E. Naerum, E. Samset, P. Sanchez-Gonzalez, F. M. S.-. Margallo, D. Schmalstieg, M. Sette, T. Studeli, J. V. Sloten, and E. J. Gomez, "Augmented Reality for Minimally Invasive Surgery: Overview and Some Recent Advances," in *Augmented Reality*. vol. chapter5, S. Maad, Ed., 2010, pp. 73-98.

[41] C. A. Linte, J. White, R. Eagleson, G. M. Guiraudon, and T. M. Peters, "Virtual and Augmented Medical Imaging Environments: Enabling Technology for Minimally Invasive Cardiac Interventional Guidance " *IEEE Reviews in Biomedical Engineering,* vol. 3, pp. 25-47, 2010.

[42] S. M. B. I. Botden, I. H. J. T. d. Hingh, and J. J. Jakimowicz, "Suturing Training in Augmented Reality: gaining proficiency in suturing skills faster," *Surg Endosc,* vol. 23, pp. 2131-2137, 2009.

[43] E. P. W. v. d. Putten, J. J. v. d. Dobbelsteen, R. H. M. Goossens, J. J. Jakimowicz, and J. Dankelman, "The Effect of Augmented Feedback on

Grasp Force in Laparoscopic Grasp Control," *IEEE Transactions on Haptics,* vol. 3, pp. 280-291, 2010.

[44] G. C. Burdea, "Haptic Feedback for Virtual Reality," in *Virtual Reality and Prototyping Workshop* France, !999.

[45] T. Horeman, S. P. Rodrigues, F. W. Jansen, J. Dankelman, and J. J. v. d. Dobbelsteen, "Force measurement platform for training and assessment of laparoscopic skills," *Surg Endosc,* vol. 24, pp. 3102-3108, 2010.

[46] C. Laugier, C. Mendoza, and K. Sundaraj, "Towards a Realistic Medical Simulator using Virtual Environment and Haptic Interaction," pp. 1-14.

[47] M. Mahvash and V. Hayward, "High-Fidelity Haptic Synthesis of Contact with Deformable Bodies," *IEEE Computer Graphics and Applications,* pp. 48-55, 2004.

[48] T. R. Coles, N. W. John, D. A. Gould, and D. G. Caldwell, "Integrating Haptics with Augmented Reality in a Femoral Palpation and Needle Insertion Training Simulation," *IEEE Transactions on Haptics,* vol. 4, pp. 199-209, 2011.

[49] R. J. Lapeer, P. D. Gasson, and V. Karri, "A Hyperelastic Fiinte-Element Model of Human Skin for Interactive Real-Time Surgical Simulatiion," *IEEE TRANASACTIONS ON BIOMEDICAL ENGINEERING,* vol. 58, pp. 1013-1022, 2011.

[50] T. Hu, J. P. Desai, A. E. Castellanos, and A. C. W. Lau, "Modeling In vivo Soft Tissue Probing ", 2006.

[51] P. Marshall, S. Payandeh, and J. Dill, "A Study on Haptic Rendering in a Simulated Surgical Training Environment," in *Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems,* 2006, pp. 241-247.

[52] S. M. Perera, K. K. D. A. R. Swaris, S. R. Wanniarachchi, and N. D. Weerasekara, "Hardware Inteface for Laparoscopic Surgery Simulator," 2011.

[53] S. Technologies, "www.3Dsystems.com," 2013.

[54] S. Technologies, "OpenHaptics Toolkit version 3.0, Programmer's Guide," 1999-2008.

[55] C. Todd, S. Mallya, S. Majeed, J. Rojas, and K. Naylor, "VirtuNav: A Virtual Reality Indoor Navigation Simulator with Haptic and Audio Feedback for the Visually Impaired," IEEE, 2014.

[56] V. Spitzer, M. J. Ackerman, A. L. Scherzinger, and D. Whitlock, "The Visible Human Male: A Technical Report," *Journal of the American Medical Information Association,* vol. 3, pp. 118-130, 1996.

[57] G. Riva, "Applications of Virtual Environments in Medicine," in *Methods MIM 0161*, 2003, pp. 524-534.

[58] M. J. Ackerman, "Accessing the Visible Human Project," National Library of Medicine, Bethesda, MD 1995.

[59] J. A. Aucar, N. R. Groch, S. A. Troxel, and S. W. Eubanks, "A Review of Surgical Simulation with Attention to Validation Methodology," *Surg laparosc Endosc Percutan Tech,* vol. 15, pp. 82-89, 2005.

[60] C. A. Diaz, D. Posada, and H. Trefftz, "Development of a Surgical Simulator to training Laparoscopic Procedures," *International Journal of Education and InformationTechnologies,* vol. 2, pp. 95-103, 2008.

[61] C. Diaz, H. Trefftz, J. Bernal, and S. Eliuk, "General algorithms for laparoscopic surgical simulators," *Revista Ingenieria Biomedica,* vol. 4, pp. 57-70, 2010.

[62] Z. Mayooran, L.Watterson, P.Withers, J. Line, W. Arnett, and R. Horley, "MediseusEpidural:Full-Procedure Training Simulator Epidural Analgesiain Labour," in *SimTecT Healthcare simulation Conference* 2006.

[63] Mentice Ab."ProcdicusMISTReference List, "http:www.mentice.com/arcieve/pdf_produts/MIS_reference_080904.pdf._," 2008.

[64] F. H. Halvorsen, O. J. Elle, and E. Fosse, "Simulators in surgery " *Minimaly Invasive Therapy and Allied Technologies,* vol. 14, pp. 214-223, 2005.

[65] S. S. Ab., " "Lapsim Validation Studies" http://www. surgical Sciencs.com/index.cfm/en/lapsim_a_validated_training_system/," 03/12/2008.

[66] E. Verdaasdonk, J. dankalmen, J. lange, and L. Stassen, "Transfer validity of laparoscopic Knot Tying Training on a VR simulator to a realistic Environment :P A Ranomised controlled Trial," *Surgical Endoscopy,* vol. 22, pp. 1636-1642, 2008.

[67] T. Pham, L. Roland, K. A. Benson, R. W. Webster, A. G. Gallagher, and R. S. Haluck, "Smart tutor: A Pilot Study of a Novel Adaptive simulation Environment," *Stod Health Technol Inform,* vol. 111, pp. 385-389, 2005.

[68] M. M. Hammoud, F. S. Nuthalapaty, A. R. Goepfert, P. M. Casey, S. Emmons, E. L. Espey, J. M. Kaczmarczyk, N. T. Katz, J. J. Neutens, and E. G. Peskin, "To the point:medical education review of the role of simulators in surgical training," *American Journal of Obstertrics & Gynecology,* pp. 1-6, 2008.

[69] C. R. Wagner, N. Stylopouslos, and R. D. Howe, "The Role of Force Feedback in Surgery: Analysis of Blunt Dissection," in *Haptic Interface for Virtual Einironment & Teleoperatory Systems,* 2002.

[70] G. Tholey and J. P. Desail, "Design and Development of a General Purpose 7 DOF Haptic Device," in *Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems* Virginia, USA, 2006, pp. 95-101.

[71] B. D. Varalakshmi, J. Thriveni, K. R. Venugopal, and L. M. Patnaik, "Haptics: State of the Art Survey," *IJCSI International Jouranal of Computer Science* vol. 9, pp. 234 - 244, 2012.

[72] E. S. Clapan and F. G. Hamza-Lup, "Simulation and Training with Haptic Feedback - A Review," in *International Conference on Virtual Learning*, 2008.

[73] F. Bello, T. R. Coles, D. A. Gould, and C. J. Hughes, "THe Need to Touch Medical Virtual Environment?," report.

[74] F. G. Hamza-Lup, C. M. Bogdan, D. M. Popoviei, and O. D. Costea, "A Survey of Visuo-Haptic Simulationin Surgical Training," in *The Third International Conference on Mobile, Hybrid and On-line Learning*, 2011, pp. 57-62.

[75] C. Basdogan, C. H. Ho, and M. A. Srinivasan, "Virtual Environments for Medicalo Training : Graphical and Haptic Simulation of Laparoscopic Common Bile Duct Exploration," *IEEE/ASME TRANSACTIONS ON MECHATRONICS,* vol. 6, pp. 269-285, 2001.

[76] A. Talvas, M. Marchal, and A. Lecuyer, "A Survey on Bimanual Haptic Interaction," *IEEE TRANASACTIONS ON HAPTICS,* vol. 7, pp. 285-300, 2014.

[77] H. P. Chen and D. C. Tseng, "Laparoscopic SurgicalSimulation on Gallbladder Removal," *Laparoscopic Surgical Simulation on..* vol. 6, pp. 38-54, 2000.

[78] R. Webster, R. S. Haluck, B. Mohler, J. Boyd, J. Reeser, A. Benson, and D. DeSanto, "A Haptic Surgical Simulator for Operative Setup and Exposure in Laparoscopic Cholecystectomy," in *Anual Medicine Meets Virtual Reality Conference (MMVR 2003)* California, 2003.

[79] J. Berkley, G. Turkiyyah, D. Berg, M. Ganter, and S. Weghorst, "Real-Time Finite Element Modeling for Surgery Simulation: An Appliocation to

Virtual Suturing," *IEEE TRANASACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS,* vol. 10, pp. 314-325, 2004.

[80] S. Cotin, H. Delingette, and N. Ayache, "Real - Time Elastic Deformations of Soft Tissues for Surgery Simulation," *IEEE TRANSACTION ON VISUALIZATION AND COMPUTER GRAPHICS,* vol. 5, pp. 62-73, 1999.

[81] T. Hu, J. P. Desai, A. E. Castellanos, and A. C. W. Lau, "Modeling In vivo Soft Tissue Probing," 2006.

[82] P. Lamata, E. J. Gomez, F. L. Hernandez, A. O. Pastor, F. M. Sanchez-Margallo, and F. D. P. Guerrero, "Understanding Perceptual Boundaries in Laparoscopic Surgery," *IEEE TRANASACTIONS ON BIOMEDICAL ENGINEERING,* vol. 55, pp. 866-873, 2008.

[83] P. Boonvisut, R. Jackson, and M. C. Cavusoglu, "Estimation of Soft Tissue Mechanical Parameters form Robotic Manipulation Data," in *IEEE International Conference on Robotics and Automation* USA, 2012, pp. 4667-4674.

[84] M. Carassiti, R. Quarta, A. Mattei, M. Tsei, P. Saccomandi, C. Massaroni, and R. Steola, "Ex vivo anmal-modeel assesment of a non-invasive system for loss of resistance detection during epidural blockade," 2017.

[85] M. Hollensteiner, D. Fuerst, and A. Schremptf, "Artificial muscles for a novel simulator in minimally invasive spine surgery," in *IEEE*, 2014, pp. 506-509.

[86] S. Misra, A. M. Okamura, and K. T. Ramesh, "Force Feedback in Noticeably Different for Linear versus Nonlinear Elastic Tissue Models ", 2007.

[87] M. Carbone, S. Condino, M. Ferrari, and F. Mosca, "Surgical Simulators Integrating Virtual and Physical Anatomies," *EICS4Med*, pp. 13-18, 2011.

[88] J. Zhang, W. Huang, J. Zhou, J. Qin, B. H. Lee, T. Yang, J. Liu, Y. Su, C. K. Chui, and S. Chang, "GPU- Friendly Gallbladder Modeling for Laparoscopic Cholecystectomy Simulation," in *3rd International Conference on Biomedical Engineering and Informatics*, 2010, pp. 1872-1876.

[89] J. Zhang, W. Huang, J. Zhou, T. Yang, J. Liu, Y. Su, C. K. Chui, and S. Chang, "Gallbladder Modeling and Simulation in Laparoscopic Cholecystectomy," in *6th IEEE Conference on Industrial Electronicsand Applications*, 2011, pp. 128-131.

[90] K. Kossi and M. Luostarinen, "VIRTUAL REALITY LAPAROSCOPIC SIMULATOR AS AN AID IN SURGICAL ERSIDENT EDUCATION: TWO YEARS' EXPERIENCE " *Scandinavian Journal of Surgery,* vol. 98, pp. 48-54, 2009.

[91] M. W. Salkini, C. R. Doarn, N. Kiehl, T. J. Broderick, J. F. Donovan, and K. Gaitonde, "The Role of Haptic Feedback in Laparoscopic Training Using the LapMentor II," *JOURNAL OF ENDOUROLOGY,* vol. 24, pp. 99-102, 2010.

[92] J. R. Thompson, A. C. Leonard, C. R. Doarn, M. J. Roesch, and T. J. Broderick, "Limited value of haptics in virtual laparoscopic cholecystectomy training," *Surg Endosc,* vol. 25, pp. 1107-1114, 2011.

[93] C. H. Park, K. L. Wilson, and A. M. Howard, "Examing the Learning Effects of a Low-Cost Haptic-Based Virtual Reality Simulator on Laparoscopic Cholecystectomy," 2013, pp. 233-238.

[94] C. H. Park, K. L. Wilson, and A. M. Howard, "Pilot Study: Supplementing Surgical Training for Medical Students Using a Low-Cost Virtual Reality Simulator," in *IEEE 27th International Symposium on Computer-Based Medical System*, 2014, pp. 125-127.

[95] Y. Youssef, G. Lee, C. Godinez, E. Sutton, R. V. Klein, I. M. George, F. J. Seagull, and A. Park, "Laparoscopic Cholecystectomy Poses Physical Injury

Risk to Surgeons: Analysis of hand Technique and Standing Position," *Surg Endosc*, vol. 25, pp. 2168-2174, 2011.

[96] J. Rosen, J. D. Brown, D. Smita, M. Sinanan, and B. Hannaford, "Biomechanical Properties of Abdomen Organs In Vivo and Postmortem Under Compression Loads," *Journal ocf Biomechanical Engineering,* vol. 130, pp. 021020-1 - 021020-17, 2008.

[97] J. Kim, C. Choi, S. De, and M. A. Srinivasan, "Virtual Surgery Simulationfor Medical Training Using Multi-ResolutionOgan Models," *THE INTERNATIONAL JOURNAL OF MEDICAL, ROBOTICS AND COMPUTER ASSISTED SURGERY,* vol. 3, pp. 149-158, 2007.

[98] S. M. Kannangara, S. C. Ranasinghe, S. K. Kumarage, and N. D. Nanayakkara, "Hardware Interface for Haptic Feedback in Laparoscopic Surgery Simulators," in *IEEE Region 10 Technical Symposium, Kuala Lumpur*, 2014, pp. 376-380.

[99] S. Technologies, "Specifications for the PHANTOM Omni Haptic Device," 2013.

[100] D. Hyuk Lee, U. Kim, T. Gulrez, W. J. Yoon, B. Hanaford, and H. R. Choi, "A Laparoscopic Grasping Tool With Force Sensing Capability," in *IEEE/ASME TRANSACTION ON MECHATRONICS,* vol. 21, pp. 131-140, 2016.

[101] A. Takacs, P. Galambos, I. J. Rudas, and T. Haidegger, "A Novel Methodology for Usability Assesment of Rheological Soft Tissue Models," in *IEEE 15th International Symposium on Applied Machine Inteligence and Informatics,* 2017, pp. 271-278.

]102] M. Li, H. Liu, J. Li, L. D. Seneviratne,and K. Althoefer, " Tissue Stiffnes Simulation and Abnormality Localization using Pseudo- haptic Feedback," in *IEEE International Conference on Robotics and Automation, USA,* 2012, pp. 5359-5364.

[103] S. Liao, Y. Chen, P. Sun, D. Liao, and X. Chen, " Development of a patient -Specific Surgical Simulator Based on Virtual Reality," in *10th IEEE International Congress on Image and Signal Processing , Biomeical Engineering and Informamatic,* 2017.

[104] G. I. Eugen, R. Ionut, and B. N. George, *" haptic Devices Synchronization into a Software Simulator,"* IEEE, pp. 440-445, 2017.

[105] Q. Q. Cheng, P. X. Liu, P. H. Lai, and N. Zou, " An Interactive Meshless Model Cutting Model for Nonlinear Viscoelastic Soft Tissue in Surgical Simulators," *IEEE* , pp. 16359-16371, 2017.

[106] S. Guo, X. Cai, B. Gao, and Y. Jiang, "An improved VR Training System for Vascular Interventional Surgery," in *IEEE International Conference on Robotics and Biomimetics,* pp. 1667-1672, 2016 .

[107] S. Dargar, A. C. Akyildiz, and Suvranu de, "In Situ Mechanical Characterization of Multilayer Soft Tissue Using Ultrasound Imaging," in *IEEE TRANSACTION ON BIOMEDICAL ENGINERING,* vol .64, pp. 2595-2606, 2017

8 APPENDIX A: FEATURES OF THE AUGMENTED REALITY SIMULATORS PROVIDED BY THEIR MANUFACTURES

Feature		ProMIS	Blue Dragon	CELTS	LTS3e
Modules and tasks of the simulator	Basic Skills:				
	Navigation/coordination	√	√	√	√
		√	√	√	√
	Touching	√	√	√	√
	Grasping	√	√	√	√
	Stretching/traction		√	√	√
	Translocation		All laparoscopic skills can be measured		
	Other				
	Advanced skills:				
	Clip Application	√	√	-*	-
	Transection/cutting	√	√	-*	√
	Dissection	√	√	-*	-
	Diathermia	√	√	-	-
	Suturing	√	√	√	√
	Knot Tying	√	√	√	√
	Other	Hand-assisted laparoscopic olectomy	All procedural component tasks		Canulation
Recorded parameters	Time	√	√	√	√
	Path length	√	√	√	-
	Smoothness	√	√	√	-
	Economy of movement	√	-	√	-
		√	-	-	√

	Errors Other	Hand dominance	Tool/tissue interaction. Opening/closin gof instruments	Instrument orientation, ambidexterity	-
Feedback	Progression curve of recorded parameters				
	Real playback of the task	√	_	√	√
	Virtual playback of the task	√	_	_	_
	Other	√	_	_	_
overview of measurements					
Need for observer	Is an 'expert' observer needed for evaluation of the performance of the tasks?	No	No	No	Yes
	An 'expert' observer is only needed for feedback /help with problem	Yes	Yes	Yes	Yes
	Trainees can train and evaluate modules without an "expert" observer	Yes	Yes	Yes	Yes
Instruction	Written instruction of the task on the screen	Yes Yes	No	No	Yes
	Demonstration video	Yes	Yes	No	Yes
	Spoken instruction during the task	Yes	No	No	No
	Guiding lines on the screen during the task	Animation to	No	No	No

	Othher	illustrate the task			
Validation	Is the simulator completely validated? If no, what part is ?	Yes	Yes	No	Under research

Source: S. M. B. I. Botden and J. J. Jakimowicz, "What is going on in augmented reality simulation in laparoscopic surgery?," *Surg Endosc*, vol. 23, pp. 1693-1700, 2009

APPENDIX B: C++ CODES FOR THE SIMULATOR WITH EMBEDDED HAPTIC MODELS

```
Copyright (c) 2004 SensAble Technologies, Inc. All rights reserved.

OpenHaptics(TM) toolkit. The material embodied in this software and use of
this software is subject to the terms and conditions of the clickthrough
Development License Agreement.

For questions, comments or bug reports, go to forums at:
    http://dsc.sensable.com

Module Name:

  HapticMaterial.cpp

Description:

  This example demonstrates setting haptic material properties on a group of
  objects and dragging objects with contraints.

  This example displays a group of objects.  When the user touches one and
  holds the stylus button down, the user can move it around the screen.  While
  the button is down, constraint axes are drawn allowing more precise
  control over dragging.  The user can change the haptic material
  properties of the object via the GLUI user interface.

  This example shows the integration with GLUI, a user interface
  toolkit, from University of North Carolina.

  This example shows the integration with haptic device as a mouse.

******************************************************************************
*/


#include <math.h>
#include <assert.h>
#include <iostream>
#include <fstream>
#include <ctime>
#include <string>


#ifdef WIN32
#include <windows.h>
#endif

#include <GL/gl.h>
#include <GL/glut.h>
#include <GL/glui.h>
```

```
#include <HL/hl.h>
#include <HDU/hduMath.h>
#include <HDU/hduMatrix.h>
#include <HDU/hduQuaternion.h>
#include <HDU/hduError.h>
#include <HLU/hlu.h>

#include <HapticMouse/HapticMouse.h>

#include <vector>
#include <fstream>
#include <iostream>

/*Assimp Includes*/
#include "assimp/Importer.hpp"
#include "assimp/postprocess.h"
#include "assimp/scene.h"

using namespace std;

/* Function prototypes. */
void glutDisplay();
void glutReshape(int width, int height);
void glutIdle();
void glutMouse(int button, int state, int x, int y);
void glutMotion(int x, int y);
void glutKeyboard(unsigned char key, int x, int y);
void glutSpecialKeys(int key, int x, int y);



void initHL();
void initGL();
void initScene();
void drawScene();
void drawString(const char* string);
void drawPrompts();
void updateCamera();
void updateHapticMapping();

void createDraggableObjects();
void drawDraggableObjects();
void drawCubes(double size, double R, double G, double B);
void _cdecl buttonCallback(int control);
void _cdecl control_cb(int control);
void _cdecl descriptor_cb(int control);

void redrawCursor();

void HLCALLBACK buttonDownCollisionThreadCallback(HLenum event, HLuint object,
                                                  HLenum thread, HLcache
*cache,
                                                  void *userdata);
void HLCALLBACK buttonDownClientThreadCallback(HLenum event, HLuint object,
                                               HLenum thread, HLcache *cache,
                                               void *userdata);
void HLCALLBACK buttonUpClientThreadCallback(HLenum event, HLuint object,
                                             HLenum thread, HLcache *cache,
                                             void *userdata);
void updateDragObjectTransform();


void __cdecl exitHandler();

void updateStiffness();
void initStiffness();
void loadIdentityToViewRotate();
void quaternion_to_rotationMat(float x, float y, float z, float angle);


/*Assimp Function Prototypes*/
void get_bounding_box (aiVector3D* min, aiVector3D* max);
void get_bounding_box_for_node (const aiNode* nd, aiVector3D* min, aiVector3D*
max);
void color4_to_float4(const aiColor4D *c, float f[4]);
void set_float4(float f[4], float a, float b, float c, float d);
bool Import3DFromFile( const std::string& pFile, const std::string&
pFile2,const std::string& pFile3, const std::string& pFile4,const std::string&
pFile5);
void recursive_render (const  aiScene *sc, const aiNode* nd);
void assimpDisplay(void);
void apply_material(const aiMaterial *mtl);
/*------------------------------------------*/




static hduVector3Dd gCameraPosWC;
static hduVector3Dd gCenterPosWc;
static int gWindowWidth, gWindowHeight;
static int gViewportWidth, gViewportHeight;

#define CURSOR_SIZE_PIXELS 20
static double gCursorScale;
static GLint gCursorDisplayList = 0;

/* Variables used by the trackball emulation. */
static hduMatrix gCameraRotation;
float view_rotate[16] = { 1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1 };
float obj_pos[] = { 0.0, 0.0, 0.0 };

static double gCameraScale = 1;
static double gCameraTranslationX = 0;
static double gCameraTranslationY = 0;
static bool gIsRotatingCamera = false;
static bool gIsScalingCamera = false;
static bool gIsTranslatingCamera = false;
static int gLastMouseX, gLastMouseY;

static bool gIsProxyManipPoint = false;

/* Haptic device and rendering context handles. */
static HHD ghHD = HD_INVALID_HANDLE;
static HHLRC ghHLRC = NULL;
```

```
HLdouble gInitWorkspace[6];

static const double kPI = 3.1415926535897932384626433832795;

static HLuint gAxisId;
static hduVector3Dd gAxisCenter(0,0,0);


/* Live variables passed into GLUI. */
int   obj_type = 0;
int   organval = 0;
int   selected_organ_num = 0;
int   profession = 0;
char  name[200] = {"Enter your name"};
char  experience[200] = {"Enter your experience in years"};
char  selected_organ[40] = {"Liver"};
char  prof_name[20] = {"Surgeon"};
int   rightCubeVal = 0;
int      stifnessChoice = 0;
int   light0_enabled = 1;
int   light1_enabled = 1;
float light0_intensity = 1.0;
float light1_intensity = 1.0;
int   main_window;
int   initial_run = 0;
/******Rotational Angles*********/
float phi = 0;
float theta = 0;
float psi = 0;
/* Haptic material properties. */
float hap_stiffness = 0.9;
float hap_damping = 0.0;
float hap_static_friction = 0.0;
float hap_dynamic_friction = 0.0;

/* GLUI: Pointers to the windows and some of the controls. */
GLUI            *glui;
GLUI_Panel      *obj_panel;
GLUI_Panel      *des_panel;
GLUI_RadioGroup *radio1;
GLUI_Spinner *spinnerStiffness;
GLUI_RadioGroup *radio3;
GLUI_Checkbox   *checkbox;
GLUI_EditText *name_text, *exp_text;
GLUI_Listbox *prof_listbox;
GLUI_Panel *start_panel, *next_panel, *finish_panel, *attempt_no;


/* haptic material properties. */
GLUI_Panel      *haptic_mat_panel;
GLUI_Spinner    *stiffness_spinner;
GLUI_Spinner    *damping_spinner;
GLUI_Spinner    *static_friction_spinner;
GLUI_Spinner    *dynamic_friction_spinner;
GLUI_Control *lastGluiCon = NULL;


/*File handling*/
ofstream results;
```

```
int test_no = 0;
int stiffness_array[3];
int sensed_stiffness[3];
//string organ_list[4] = {"Liver","Stomach","Goldbladder","Skin"};
//string profession_list[4] = {"Surgeon","Doctor","Medical Student","Other"};
//string stiffness_property[3] = {"soft","mild","hard"};
float stiffness_values[3] = {0.1,0.45,0.95};
bool started = false;
int test_counts = 0;

/* User IDs for callbacks */
#define OBJECT_TYPE_ID      199
#define ORGAN       100
#define RIGHT_OBJECT 110
#define MIDDLE_OBJECT       120
#define START_BUTTON 130
#define RECORD_BUTTON              131
#define FINISH_BUTTON       132
#define RESET_BUTTON    133
#define LIGHT0_ENABLED_ID    200
#define LIGHT1_ENABLED_ID    201
#define LIGHT0_INTENSITY_ID  250
#define LIGHT1_INTENSITY_ID  251
#define STIFFNESS_SPINNER        133
#define FRICTION_SPINNER         134
#define NAME_TEXT                135
#define EXP_TEXT                 136
#define PROFESSION_INT           137

#define VIEW_ROTATE_ID       252
#define VIEW_PAN_ID          253
#define VIEW_ZOOM_ID         254

#define LIVER               320
#define STOMACH         321
#define GALLBLADDER     322
#define BOWELS          323
#define SPINE           324


/* Haptic Stiffness */
#define SOFT         0.1
#define MILD         0.4
#define HARD         0.8

/* Lighting parameters */
GLfloat light0_ambient[] =  {0.1f, 0.1f, 0.3f, 1.0f};
GLfloat light0_diffuse[] =  {.6f, .6f, 1.0f, 1.0f};
GLfloat light0_position[] = {.5f, .5f, 1.0f, 0.0f};

GLfloat light1_ambient[] =  {0.1f, 0.1f, 0.3f, 1.0f};
GLfloat light1_diffuse[] =  {.9f, .6f, 0.0f, 1.0f};
GLfloat light1_position[] = {-1.0f, 0.9f, 1.0f, 0.0f};


/*assimp global variables*/
const  struct aiScene* scene = NULL;
const struct aiScene* scene2 = NULL;
const struct aiScene* scene3 = NULL;
const struct aiScene* scene4 = NULL;
const struct aiScene* scene5 = NULL;
aiVector3D scene_min, scene_max, scene_center;
int initialRun = 0;

#define aisgl_min(x,y) (x<y?x:y)
#define aisgl_max(x,y) (y>x?y:x)

// Create an instance of the Importer class
Assimp::Importer imp;
Assimp::Importer imp2;


// scale factor for the model to fit in the window
float scaleFactor;

/*----------------------------------*/

/* Struct representing a shape in the scene that can be felt,
   touched, transformed, and drawn. */
struct DraggableObject
{
    HLuint shapeId;
    GLuint displayList;
    hduMatrix transform;

       int show;
    float hap_stiffness;
    float hap_damping;
    float hap_static_friction;
    float hap_dynamic_friction;
};

/* List of all draggable objects in scene. */
std::vector<DraggableObject> draggableObjects;

/* Object currently being dragged (index into draggableObjects). */
int gCurrentDragObj = -1;

/* Position and orientation of proxy at start of drag. */
hduVector3Dd gStartDragProxyPos;
hduQuaternion gStartDragProxyRot;

/* Position and orientation of drag object at start of drag. */
hduMatrix gStartDragObjTransform;

/* Flag for enabling/disabling axis snap on drag. */
bool gAxisSnap = true;

/* Flag for enabling/disabling rotation. */
bool gRotate = true;

/*****************************************************************************
**
 Main function.
******************************************************************************
*/
int main(int argc, char *argv[])
{
    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);

    glutInitWindowSize(850, 500);

    main_window = glutCreateWindow("Haptic Material");

    // Set glut callback functions.
    glutDisplayFunc(glutDisplay);
    glutMotionFunc(glutMotion);

    GLUI_Master.set_glutReshapeFunc(glutReshape);
    GLUI_Master.set_glutKeyboardFunc(glutKeyboard);
    GLUI_Master.set_glutMouseFunc(glutMouse);
    GLUI_Master.set_glutSpecialFunc(glutSpecialKeys);


    // The GLUT main loop won't return control, so we need to perform cleanup
    // using an exit handler.
    atexit(exitHandler);

    initScene();

    // egister the idle callback with GLUI (not with GLUT).
    GLUI_Master.set_glutIdleFunc(glutIdle);

    glutMainLoop();

    return 0;
}


/*****************************************************************************
**
 GLUI control callback.
*****************************************************************************
*/
void _cdecl control_cb(int control)
{

       if( control == ORGAN )
    {
              printf("%d",organval);
              switch (organval){
              case 0:
                     hap_stiffness = 0.5;
                     updateStiffness();
                     strcpy(selected_organ,"Liver");
                     next_panel->enable();
                     break;
        case 1:
                     hap_stiffness = 0.5;
                     updateStiffness();
                     strcpy(selected_organ,"Stomach");
                     next_panel->enable();
```

```
                 break;
          case 2:
                 hap_stiffness = 0.5;
                 updateStiffness();
                 strcpy(selected_organ,"Goldbladder");
                 next_panel->enable();
                 break;
          case 3:
                 hap_stiffness = 0.5;
                 updateStiffness();
                 strcpy(selected_organ,"Bowels");
                 next_panel->enable();
                 break;
          case 4:
                 hap_stiffness = 0.5;
                 updateStiffness();
                 strcpy(selected_organ,"Muscle");
                 next_panel->enable();
                 break;

          }
   }

   else if( control == STIFFNESS_SPINNER )
{
    if (hap_stiffness==0) hap_stiffness = 0.1;
          printf("%f\n",hap_stiffness);
          updateStiffness();

}
   else if( control == FRICTION_SPINNER )
{
    printf("%f\n",hap_static_friction);
          updateStiffness();
}
   else if(control == 0)
   {
          if (results.is_open()){
                 results.close();
          }
          exit(EXIT_SUCCESS);
   }

   if (control == LIVER){
          DraggableObject& dro = draggableObjects[0];
          dro.show = -1*dro.show;
          printf("%d",dro.show);
   }
   if (control == SPINE){
          DraggableObject& dro = draggableObjects[1];
          dro.show = -1*dro.show;
   }
   if (control == GALLBLADDER){
          DraggableObject& dro = draggableObjects[2];
          dro.show = -1*dro.show;
   }
   if (control == BOWELS){
          DraggableObject& dro = draggableObjects[3];
          dro.show = -1*dro.show;
```

```
       }
       if (control == STOMACH){
              DraggableObject& dro = draggableObjects[4];
              dro.show = -1*dro.show;
       }
}

/*****************************************************************************
****
GLUI callback for details of the user
******************************************************************************
****/

void _cdecl descriptor_cb(int control)
{

       if(control = PROFESSION_INT){
              switch (profession){
              case 0:
                     strcpy(prof_name,"Surgeon");
                     break;
        case 1:
                     strcpy(prof_name,"Senior Registar");
                     break;
              case 2:
                     strcpy(prof_name,"Registar");
                     break;
              case 3:
                     strcpy(prof_name,"MO");
                     break;
              case 4:
                     strcpy(prof_name,"Medical Student");
                     break;
              case 5:
                     strcpy(prof_name,"Other");
                     break;
              }

       }


}
/*****************************************************************************
*
Callback function for the button press
*When a button is pressed this function is called
******************************************************************************
*/
void _cdecl buttonCallback(int control)
{

       if (control == START_BUTTON)
       {
              results.open("results.csv",ios::out | ios::app);
              time_t t = time(0);   // get time now
              struct tm * now = localtime( & t );
              results<< (now->tm_year + 1900) << '-'
         << (now->tm_mon + 1) << '-'
         <<  now->tm_mday << ' '
```

```
                 << now->tm_hour << ":" << now->tm_min
          << endl;

                hap_stiffness = 0.5;
                updateStiffness();
                start_panel->disable();
                next_panel->enable();
                finish_panel->enable();
                started = true;


        }
        else if (control == RECORD_BUTTON)
        {

                results<<name_text->get_text()<<","<<prof_name<<","
                       <<exp_text-
>get_text()<<","<<selected_organ<<","<<hap_stiffness<<","
                       <<hap_static_friction<<endl;

                updateStiffness();
                next_panel->disable();

        }
        else if (control == FINISH_BUTTON)
        {
                started = false;

                results.close();
                initStiffness();
                test_counts = 0;
        }


        if (control == RESET_BUTTON)
        {
                loadIdentityToViewRotate();
                obj_pos[0] = 0.0;
                obj_pos[1] = 0.0;
                obj_pos[2] = 0.0;
                phi = 0;
                theta = 0;
                psi = 0;
                updateCamera();
        }

        // Put an exit function here
}


/*****************************************************************************
**
 Initializes GLUI user interface.
 * Here buttons are created and callback functions are assigned
***************************************
****************************************/
void initGLUI()
{
	int viewport_w, viewport_h,x,y;
    printf("GLUI version: %3.2f\n", GLUI_Master.get_version());

    glui = GLUI_Master.create_glui_subwindow(main_window,
GLUI_SUBWINDOW_RIGHT);


    glui->add_statictext("TEST PROGRAM 01");

	//Description pannel to record the details of the person
	des_panel = glui->add_panel( "Enter your details bellow" );
	des_panel->set_alignment(GLUI_ALIGN_LEFT);
	name_text = glui-
>add_edittext_to_panel(des_panel,"Name       :",GLUI_EDITTEXT_TEXT,name);
	name_text->set_w(300);
	name_text->set_alignment(GLUI_ALIGN_LEFT);
	prof_listbox = glui-
>add_listbox_to_panel(des_panel,"Profession :",&profession,PROFESSION_INT,desc
riptor_cb);
	prof_listbox->add_item (0,"Surgeon");
	prof_listbox->add_item (1,"Senior Registar");
	prof_listbox->add_item (2,"Registar");
	prof_listbox->add_item (3,"Medical Officer");
	prof_listbox->add_item (4,"Medical Student");
	prof_listbox->add_item (4,"Other");
	exp_text = glui-
>add_edittext_to_panel(des_panel,"Experience:",GLUI_EDITTEXT_TEXT,experience);
	exp_text->set_w(300);


	obj_panel = glui->add_panel( "Adjust the stiffness and friction for the
selected organ");
	obj_panel->set_alignment(GLUI_ALIGN_LEFT);

    GLUI_Panel *organ = glui->add_panel_to_panel(obj_panel, "Organ");
	organ->set_alignment(GLUI_ALIGN_LEFT);

    radio1 = glui->add_radiogroup_to_panel(organ,&organval, ORGAN, control_cb);
    glui->add_radiobutton_to_group(radio1, "Liver");
    glui->add_radiobutton_to_group(radio1, "Stomach");
    glui->add_radiobutton_to_group(radio1, "Goldbladder");
	glui->add_radiobutton_to_group(radio1, "Bowels");
	glui->add_radiobutton_to_group(radio1, "Muscle");

	glui->add_column_to_panel(obj_panel,false);

	GLUI_Panel *stiffnessVal = glui->add_panel_to_panel(obj_panel,
"Stiffness Value",GLUI_PANEL_NONE);
	stiffnessVal->set_alignment(GLUI_ALIGN_RIGHT);

      stiffness_spinner  =  glui->add_spinner_to_panel(stiffnessVal,"Stiffness",
GLUI_SPINNER_FLOAT,&hap_stiffness,STIFFNESS_SPINNER, control_cb);
    stiffness_spinner->set_float_limits(0.0,1.0,GLUI_LIMIT_CLAMP);
	stiffness_spinner->set_speed(.2);
	stiffness_spinner->set_float_val(0.5);

	dynamic_friction_spinner = glui-
>add_spinner_to_panel(stiffnessVal,"Friction",
GLUI_SPINNER_FLOAT,&hap_static_friction,FRICTION_SPINNER, control_cb);
```

```
    dynamic_friction_spinner->set_float_limits(0.0,1.0,GLUI_LIMIT_CLAMP);
       dynamic_friction_spinner->set_speed(.2);

       stiffnessVal->disable();

       GLUI_Panel *organ_selection_panel = glui->add_panel("Select the organ
to be visualized");
       GLUI_Checkbox *checkbox = glui-
>add_checkbox_to_panel(organ_selection_panel, "Liver", NULL,LIVER,
control_cb );
       checkbox->set_int_val(1);
       checkbox = glui->add_checkbox_to_panel(organ_selection_panel,
"Stomach",NULL,STOMACH, control_cb );
       checkbox->set_int_val(1);
       checkbox = glui->add_checkbox_to_panel(organ_selection_panel,
"Gallbladder",NULL,GALLBLADDER, control_cb );
       checkbox->set_int_val(1);
       checkbox = glui->add_checkbox_to_panel(organ_selection_panel, "Bowels",
NULL,BOWELS, control_cb );
       checkbox->set_int_val(1);
       checkbox = glui->add_checkbox_to_panel(organ_selection_panel, "Spine",
NULL,SPINE, control_cb );
       checkbox->set_int_val(1);


    GLUI_Panel *reset_panel = glui->add_panel("Selection
Panel",GLUI_PANEL_NONE);
    glui->add_button_to_panel(reset_panel,"Reset View",
RESET_BUTTON,(GLUI_Update_CB) buttonCallback);

    GLUI_Panel *selection_panel = glui->add_panel("Selection
Panel",GLUI_PANEL_NONE);
    start_panel = glui->add_panel_to_panel(selection_panel,"Start
Panel",GLUI_PANEL_NONE);
       glui->add_button_to_panel(start_panel,"Start",
START_BUTTON,(GLUI_Update_CB) buttonCallback);
       glui->add_column_to_panel(selection_panel,false);
       next_panel = glui->add_panel_to_panel(selection_panel,"Next
Panel",GLUI_PANEL_NONE);
       glui-
>add_button_to_panel(next_panel,"Record",RECORD_BUTTON,(GLUI_Update_CB)
buttonCallback);
       glui->add_column_to_panel(selection_panel,false);
       finish_panel = glui->add_panel_to_panel(selection_panel,"Finish
Panel",GLUI_PANEL_NONE);
       glui-
>add_button_to_panel(finish_panel,"Finish",FINISH_BUTTON,(GLUI_Update_CB)
buttonCallback);



       // A 'quit' button.
       glui->add_button("Quit",0,(GLUI_Update_CB) exit);

       glui->add_separator();
       attempt_no = glui-
>add_panel_to_panel(obj_panel,"attempt",GLUI_PANEL_NONE);
```

```
    // Link windows to GLUI, and register idle callback.


    glui->set_main_gfx_window(main_window);
}

/*****************************************************************************
**
 GLUT callback for redrawing the view.  Use this to perform graphics rate
 processing.
*****************************************************************************
*/
void glutDisplay()
{
       drawScene();


       if (!started)
       {
              start_panel->enable();
              next_panel->disable();
              finish_panel->disable();

       }

}

/*****************************************************************************
**
 GLUT callback for reshaping the window.  This is the main place where the
 viewing and workspace transforms get initialized.
*****************************************************************************
*/
void glutReshape(int width, int height)
{
    static const double kFovY = 40;
    static const double kCanonicalSphereRadius = 2;

    gWindowWidth = width;
    gWindowHeight = height;

    int tx, ty;
    /****************************
       GLUI_Master.get_Viewport_area
       Determines the position and dimensions of the drawable area of the
current window.
       This function is needed when GLUI subwindows are used, since the
subwindows will
       occupy some of the area of a window, which the graphics app should not
overwrite.
       This function should be called within the GLUT reshape callback
function.
       ****************************/
       GLUI_Master.get_viewport_area(&tx, &ty, &gViewportWidth, &gViewportHeight);

    glViewport(tx, ty, gViewportWidth, gViewportHeight);

    // Compute the viewing parameters based on a fixed fov and viewing
    // sphere enclosing a canonical box centered at the origin.
    double nearDist = kCanonicalSphereRadius / tan((kFovY / 2.0) * kPI /
180.0);
    double farDist = nearDist + 2.0 * kCanonicalSphereRadius;
    double aspect = (double) gViewportWidth / gViewportHeight;


    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    gluPerspective(kFovY, aspect, nearDist, farDist);

    // Place the camera down the Z axis looking at center of the object
       gCameraScale = scaleFactor;
       gCameraPosWC[0] = 0;//scene_center.x*gCameraScale;
       gCameraPosWC[1] = 0;//scene_center.y*gCameraScale;
    gCameraPosWC[2] = 0+ nearDist +
kCanonicalSphereRadius;//scene_center.z*gCameraScale + nearDist +
kCanonicalSphereRadius;

    updateCamera();
}

/*******************************************************************************
**
 GLUT callback for idle state.  Use this to request a redraw.
********************************************************************************
*/
void glutIdle()
{
    // According to the GLUT specification, the current window is
    // undefined during an idle callback.  So we need to explicitly change
    // it if necessary.
    if (glutGetWindow() != main_window)
    {
        glutSetWindow(main_window);
    }

    glutPostRedisplay();

    glui->sync_live();
       updateStiffness();


}

/*******************************************************************************
**
 Initializes the scene.  Handles initializing both OpenGL and HDAPI.
********************************************************************************
*/
void initScene()
{
    initGL();
    initGLUI();
    initHL();

    createDraggableObjects();
}
```

```
/*****************************************************************************
**
 Cleans up.
******************************************************************************
*/
void exitHandler()
{
    // Shutdown the haptic mouse.
    hmShutdownMouse();

    // Free up the haptic rendering context.
    hlMakeCurrent(NULL);
    if (ghHLRC != NULL)
    {
        hlDeleteContext(ghHLRC);
    }

    // Free up the haptic device.
    if (ghHD != HD_INVALID_HANDLE)
    {
        hdDisableDevice(ghHD);
    }
}

/*****************************************************************************
**
 Sets up general OpenGL rendering properties: lights, depth buffering, etc.
******************************************************************************
*/
void initGL()
{
    glClearColor(0.5f, 0.6f, 0.0f, 1.0f);
    // Enable depth buffering for hidden surface removal.
    glDepthFunc(GL_LEQUAL);
    glEnable(GL_DEPTH_TEST);

    // Cull back faces.
    glCullFace(GL_BACK);
    glEnable(GL_CULL_FACE);

    // Set lighting parameters.
    glEnable(GL_LIGHTING);
    glEnable(GL_NORMALIZE);

    glEnable(GL_LIGHT0);
    glLightfv(GL_LIGHT0, GL_AMBIENT, light0_ambient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light0_diffuse);
    glLightfv(GL_LIGHT0, GL_POSITION, light0_position);

    glEnable(GL_LIGHT1);
    glLightfv(GL_LIGHT1, GL_AMBIENT, light1_ambient);
    glLightfv(GL_LIGHT1, GL_DIFFUSE, light1_diffuse);
    glLightfv(GL_LIGHT1, GL_POSITION, light1_position);


}
```

```
/*******************************************************************************
**
 Sets up/initializes haptic rendering library.
*******************************************************************************
*/
void initHL()
{
    HDErrorInfo error;
    ghHD = hdInitDevice(HD_DEFAULT_DEVICE);
    if (HD_DEVICE_ERROR(error = hdGetError()))
    {
        hduPrintError(stderr, &error, "Failed to initialize haptic device");
        fprintf(stderr, "Press any key to exit");
        getchar();
        exit(1);
    }

    // Create a haptic context for the device.  The haptic context maintains
    // the state that persists between frame intervals and is used for
    // haptic rendering.
    ghHLRC = hlCreateContext(ghHD);
    hlMakeCurrent(ghHLRC);

    // Generate a shape id to hold the axis snap constraint.
    gAxisId = hlGenShapes(1);

    // Add a callback to handle button down in the collision thread.
    hlAddEventCallback(HL_EVENT_1BUTTONDOWN, HL_OBJECT_ANY,
HL_COLLISION_THREAD,
                       buttonDownCollisionThreadCallback, NULL);

    // Initialize the haptic mouse.
    hmInitializeMouse(ghHLRC, "GLUT", "Haptic Material");

    // Save off the initial workspace, since we will be modifying it for the
    // haptic mouse to allow for motion outside of the viewport.
    hlGetDoublev(HL_WORKSPACE, gInitWorkspace);
}

/*******************************************************************************
**
 Creates the objects that can be seen, felt and dragged around.
 Here I have created three objects for our program.
*******************************************************************************
*/
void createDraggableObjects()
{
    // Create a bunch of shapes and add them to the draggable object vector.

       // Here we have imported the models

       Import3DFromFile("Liver.stl","bone.stl","Gallbladder.stl","bowels.stl",
"stomach.stl");



    // Add event callbacks for button down on each of the shapes.
    // Callbacks will set that shape to be the drag object.
    for (int i = 0; i < draggableObjects.size(); ++i)
    {
        // Pass the index of the object as userdata.
        hlAddEventCallback(HL_EVENT_1BUTTONDOWN, draggableObjects[i].shapeId,
HL_CLIENT_THREAD, buttonDownClientThreadCallback, reinterpret_cast<void *>(i));
    }

    // Add an event callback on button to clear the drag object
    // and end dragging.
    hlAddEventCallback(HL_EVENT_1BUTTONUP, HL_OBJECT_ANY, HL_CLIENT_THREAD,
buttonUpClientThreadCallback, NULL);

}

/*******************************************************************************
**
 Draws the objects that can be seen, felt and dragged around.
 Haptic object drawing. Draws at every frame
*******************************************************************************
*/
void drawDraggableObjects()
{
       //printf("Test1");
    int i = 0;
       hlTouchModel(HL_CONTACT);
    hlTouchableFace(HL_FRONT);


    for (int i = 0; i < draggableObjects.size(); ++i)
    {
        const DraggableObject& obj = draggableObjects[i];

        // Position and orient the object.
              if (obj.show>0){
        glPushMatrix(); // This one just save the current transformation for
this object only

                     glMultMatrixd(obj.transform);




                     // Draw the object graphically.
                     glCallList(obj.displayList);

                     // Draw the object haptically (but not if it is being
dragged).
                     if (i != gCurrentDragObj && !hmIsMouseActive())
                     {
                            hlBeginShape(HL_SHAPE_DEPTH_BUFFER, obj.shapeId);

                            hlMaterialf(HL_FRONT_AND_BACK, HL_STIFFNESS,
obj.hap_stiffness);
                            hlMaterialf(HL_FRONT, HL_DAMPING, obj.hap_damping);
                            hlMaterialf(HL_FRONT, HL_STATIC_FRICTION,
obj.hap_static_friction);
                            hlMaterialf(HL_FRONT, HL_DYNAMIC_FRICTION,
obj.hap_dynamic_friction);

                            glCallList(obj.displayList);
                            hlEndShape();
                     }

        glPopMatrix();

              }
              glutPostRedisplay();
       }
}

/******************************************************************************
**
 Sets the modelview transform from scratch.  Applies the current view
 orientation and scale.
*******************************************************************************
*/
void updateCamera()
{
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(gCameraPosWC[0], gCameraPosWC[1], gCameraPosWC[2],
              gCameraPosWC[0], gCameraPosWC[1], 0,

              0, 1, 0);

    glTranslatef(gCameraTranslationX, gCameraTranslationY, 0);
    glMultMatrixd(gCameraRotation);

    glTranslatef(obj_pos[0], obj_pos[1], obj_pos[2]);
    glMultMatrixf(view_rotate);

    glScaled(gCameraScale, gCameraScale, gCameraScale);

    updateHapticMapping();

    glutPostRedisplay();
}

/******************************************************************************
**
 Uses the current OpenGL viewing transforms to initialize a transform for the
 haptic device workspace so that it's properly mapped to world coordinates.
*******************************************************************************
*/
void updateHapticMapping(void)
{
    GLdouble modelview[16];
    GLdouble projection[16];
    GLint viewport[4];

    glGetDoublev(GL_MODELVIEW_MATRIX, modelview);
    glGetDoublev(GL_PROJECTION_MATRIX, projection);
    glGetIntegerv(GL_VIEWPORT, viewport);

    // Modify the workspace dimensions mapped to the view so that the user
    // can move outside of the viewport and access the GLUI interface.
    double t = (double) gViewportWidth / gWindowWidth;
    double xMaxProportion = hduLerp(gInitWorkspace[0], gInitWorkspace[3], t);
    hlWorkspace(gInitWorkspace[0], gInitWorkspace[1], gInitWorkspace[2],
                xMaxProportion, gInitWorkspace[4], gInitWorkspace[5]);

    // Fit haptic workspace to view volume.
    hlMatrixMode(HL_TOUCHWORKSPACE);
    hlLoadIdentity();
    hluFitWorkspace(projection);

    // Compute cursor scale.
    gCursorScale = hluScreenToModelScale(modelview, projection, viewport);
    gCursorScale *= CURSOR_SIZE_PIXELS;

    // Provide the current viewing transforms to HapticMouse so it can
    // map the device to the screen.
    hmSetMouseTransforms(modelview, projection, viewport);
}


/*****************************************************************************
*
 Displays a cursor using the current haptic device proxy transform and the
 mapping between the workspace and world coordinates
******************************************************************************
/
void redrawCursor()
{
    static const double kCursorRadius = 1;
       static const double kCursorHeight = 2.5;
    static const int kCursorTess = 15;

    HLdouble proxytransform[16];

    GLUquadricObj *qobj = 0;

    glPushAttrib(GL_CURRENT_BIT | GL_ENABLE_BIT | GL_LIGHTING_BIT);
    glPushMatrix();

    if (!gCursorDisplayList)
    {
        gCursorDisplayList = glGenLists(1);
        glNewList(gCursorDisplayList, GL_COMPILE);
        qobj = gluNewQuadric();

        gluCylinder(qobj, 0.0, kCursorRadius, kCursorHeight,
                    kCursorTess, kCursorTess);
        glTranslated(0.0, 0.0, kCursorHeight);
        gluCylinder(qobj, kCursorRadius, 0.0, kCursorHeight / 5.0,
                    kCursorTess, kCursorTess);

        gluDeleteQuadric(qobj);
        glEndList();
    }

    // Apply the local position/rotation transform of the haptic device proxy.
    hlGetDoublev(HL_PROXY_TRANSFORM, proxytransform);
    glMultMatrixd(proxytransform);

    // Apply the local cursor scale factor.
    glScaled(gCursorScale, gCursorScale, gCursorScale);

    glEnable(GL_NORMALIZE);
    glEnable(GL_COLOR_MATERIAL);
    glColor3f(0.0, 0.5, 1.0);

    glCallList(gCursorDisplayList);

    glPopMatrix();
    glPopAttrib();
}

/*****************************************************************************
*
 Draws coordinate axes using OpenGL.
******************************************************************************
/
void drawAxes(const hduVector3Dd& gAxisCenter)
{
    // Lines and points are best viewed without OpenGL lighting.
    glPushAttrib(GL_LIGHTING_BIT);
    glDisable(GL_LIGHTING);

    // Draw three lines - one along each coordinate axis from -1 to 1.
    glBegin(GL_LINES);
    {
        glVertex3f(gAxisCenter[0] - 1,gAxisCenter[1],gAxisCenter[2]);
        glVertex3f(gAxisCenter[0] + 1,gAxisCenter[1],gAxisCenter[2]);
        glVertex3f(gAxisCenter[0],gAxisCenter[1] - 1,gAxisCenter[2]);
        glVertex3f(gAxisCenter[0],gAxisCenter[1] + 1,gAxisCenter[2]);
        glVertex3f(gAxisCenter[0],gAxisCenter[1],gAxisCenter[2] - 1);
        glVertex3f(gAxisCenter[0],gAxisCenter[1],gAxisCenter[2] + 1);
    }
    glEnd();

    // Draw the origin.
    glBegin(GL_POINTS);
    {
        glVertex3f(gAxisCenter[0],gAxisCenter[1],gAxisCenter[2]);
    }
    glEnd();

    glPopAttrib();
}

/*****************************************************************************
*
 The main routine for displaying the scene.
******************************************************************************
/
void drawScene()
{
    hlBeginFrame();

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
       glMatrixMode(GL_MODELVIEW);
    // Any client thread button callbacks get triggered here.
    hlCheckEvents();

    // Draw any haptic mouse scene objects.
    hmRenderMouseScene();

    // Check if button on stylus is down - if so draw the coordinate axes and
    // move the drag object.
    HLboolean buttDown;
    hlGetBooleanv(HL_BUTTON1_STATE, &buttDown);

    if (buttDown && !hmIsMouseActive())
    {
        if (gAxisSnap)
        {
            // Graphically render the axes.
            drawAxes(gAxisCenter);

            // Make sure proxy resolution is on.  The event handler
            // turns it off but it must be on for shapes to be felt.
            hlEnable(HL_PROXY_RESOLUTION);

            // Haptically render the coordinate axes as a feedback buffer
            // shape.
            hlBeginShape(HL_SHAPE_DEPTH_BUFFER, gAxisId);

            // Make it a constraint to the cursor will stick to the axes.
            hlTouchModel(HL_CONSTRAINT);

            // Snap distance allows user to pull off of the constraint
            // if the user moves beyond that snap distance.
            hlTouchModelf(HL_SNAP_DISTANCE, 1.5);

            // Call the OpenGL commands to draw the axes, but this time
            // they will be used for haptics.
            drawAxes(gAxisCenter);

            hlEndShape();
        }


    }

    drawDraggableObjects();
       //updateCamera();

    if (!hmIsMouseActive())
    {
        redrawCursor();
    }

    //drawPrompts();

    glutSwapBuffers();

    hlEndFrame();

}
```

```
/*****************************************************************************
*
 GLUT callback for responding to mouse button presses. Detecst whether to
 initiate a point snapping, view rotation or view scale.
******************************************************************************
/
void glutMouse(int button, int state, int x, int y)
{
    if (state == GLUT_DOWN)
    {
        if (button == GLUT_LEFT_BUTTON)
        {
            gIsRotatingCamera = true;
        }
        else if (button == GLUT_RIGHT_BUTTON)
        {
            gIsScalingCamera = true;
        }
        else if (button == GLUT_MIDDLE_BUTTON)
        {
            gIsTranslatingCamera = true;
        }

        gLastMouseX = x;
        gLastMouseY = y;
    }
    else
    {
        gIsRotatingCamera = false;
        gIsScalingCamera = false;
        gIsTranslatingCamera = false;
    }
}

/*****************************************************************************
**
 This routine is used by the view rotation code for simulating a virtual
 trackball.  This math computes the z height for a 2D projection onto the
 surface of a 2.5D sphere.  When the input point is near the center of the
 sphere, this routine computes the actual sphere intersection in Z.  When
 the input point moves towards the outside of the sphere, this routine will
 solve for a hyperbolic projection, so that it still yields a meaningful
answer.
******************************************************************************
*/
double projectToTrackball(double radius, double x, double y)
{
    static const double kUnitSphereRadius2D = sqrt(2.0);
    double z;

    double dist = sqrt(x * x + y * y);
    if (dist < radius * kUnitSphereRadius2D / 2.0)
    {
        // Solve for sphere case.
        z = sqrt(radius * radius - dist * dist);
    }
    else
    {
        // Solve for hyperbolic sheet case.
        double t = radius / kUnitSphereRadius2D;
        z = t * t / dist;
    }

    return z;
}

/*****************************************************************************
**
 GLUT callback for mouse motion, which is used for controlling the view
 rotation and scaling.
******************************************************************************
*/
void glutMotion(int x, int y)
{
    if (0)//gIsRotatingCamera)
    {
        static const double kTrackBallRadius = 0.8;
              //printf("Position: %d\n",gLastMouseX);
        hduVector3Dd lastPos;
        lastPos[0] = gLastMouseX * 2.0 / gViewportWidth - 1.0;
        lastPos[1] = (gViewportHeight - gLastMouseY) * 2.0 / gViewportHeight -
1.0;
        lastPos[2] = projectToTrackball(kTrackBallRadius, lastPos[0],
lastPos[1]);

        hduVector3Dd currPos;
        currPos[0] = x * 2.0 / gViewportWidth - 1.0;
        currPos[1] = (gViewportHeight - y) * 2.0 / gViewportHeight - 1.0;
        currPos[2] = projectToTrackball(kTrackBallRadius, currPos[0],
currPos[1]);

        currPos.normalize();
        lastPos.normalize();

        hduVector3Dd rotateVec = lastPos.crossProduct(currPos);

        double rotateAngle = asin(rotateVec.magnitude());
        if (!hduIsEqual(rotateAngle, 0.0, DBL_EPSILON))
        {
            hduMatrix deltaRotation = hduMatrix::createRotation(rotateVec,
rotateAngle);
            gCameraRotation.multRight(deltaRotation);
            updateCamera();
        }
    }
    if (gIsTranslatingCamera)
    {
        gCameraTranslationX += 10 * double(x - gLastMouseX)/gViewportWidth;
        gCameraTranslationY -= 10 * double(y - gLastMouseY)/gViewportHeight;

        updateCamera();
    }
    else if (gIsScalingCamera)
    {
        float y1 = gViewportHeight - gLastMouseY;
        float y2 = gViewportHeight - y;

        gCameraScale *= 1 + (y1 - y2) / gViewportHeight;
        updateCamera();
    }


    gLastMouseX = x;
    gLastMouseY = y;
}

/******************************************************************************
**
 GLUT callback for key presses.
*******************************************************************************
*/
void glutKeyboard(unsigned char key, int x, int y)
{
    switch (key) {
    case 'a':
              obj_pos[0] = obj_pos[0] - 0.1;
              updateCamera();
              break;
       case 'w':
              obj_pos[1] = obj_pos[1] + 0.1;
              updateCamera();
              break;
       case 'd':
              obj_pos[0] = obj_pos[0] + 0.1;
              updateCamera();
              break;
       case 's':
              obj_pos[1] = obj_pos[1] - 0.1;
              updateCamera();
              break;
       case 'q':
              obj_pos[2] = obj_pos[2] - 0.1;
              updateCamera();
              break;
       case 'e':
              obj_pos[2] = obj_pos[2] + 0.1;
              updateCamera();
              break;
       default:
              printf("Nothing");

    }
}

/******************************************************************************
****
GLUT callback for special keys
*******************************************************************************
****/
void glutSpecialKeys(int key, int x, int y)
{
       switch (key) {
       case GLUT_KEY_LEFT:
              //loadIdentityToViewRotate();
              phi = phi - 0.1;
              view_rotate[0] = cos(phi);
              view_rotate[2] = -sin(phi);
              view_rotate[8] = sin(phi);
              view_rotate[10] = cos(phi);
              updateCamera();
              break;
       case GLUT_KEY_RIGHT:
              //loadIdentityToViewRotate();
              phi = phi + 0.1;
              view_rotate[0] = cos(phi);
              view_rotate[2] = -sin(phi);
              view_rotate[8] = sin(phi);
              view_rotate[10] = cos(phi);
              updateCamera();
              break;
       case GLUT_KEY_UP:
              //loadIdentityToViewRotate();
              theta = theta - 0.1;
              view_rotate[5] = cos(theta);
              view_rotate[6] = sin(theta);
              view_rotate[9] = -sin(theta);
              view_rotate[5] = cos(theta);
              updateCamera();
              break;
       case GLUT_KEY_DOWN:
              //loadIdentityToViewRotate();
              theta = theta + 0.1;
              view_rotate[5] = cos(theta);
              view_rotate[6] = sin(theta);
              view_rotate[9] = -sin(theta);
              view_rotate[5] = cos(theta);
              updateCamera();
              break;
       default:
              printf("Nothing");
       }
}


/*******************************************************************************
*
 Event callback triggered when styus button is depressed.  Called in collision
 thread to avoid kick.  (If the application instead waits to handle the button
 down in the client thread, the user can get a kick when the constraint is
 set if the user's hand moved from the position that the button down was
 originally recorded at.)
********************************************************************************
/
void HLCALLBACK buttonDownCollisionThreadCallback(HLenum event, HLuint object,
                                                   HLenum thread, HLcache
*cache,
                                                   void *userdata)
{
    // Don't proceed if the haptic mouse is active.
    if (hmIsMouseActive())
        return;

    if (gAxisSnap)
    {
        // Use the state cache to get the proxy position at the time
        // the event occured.
        hlCacheGetDoublev(cache, HL_PROXY_POSITION, gAxisCenter);

        // Temporarily turn off proxy resolution and set the
        // proxy position at the proxy position from the time
        // the event occured.  With proxy resolution disabled, the
        // proxy will not move unless told, so it will stick
        // to this spot until we place the constraint in the drawScene
        // routine in the client thread.  This basically allows us to hold
        // the proxy in place until the constraint can hold it there.
        hlDisable(HL_PROXY_RESOLUTION);
        hlProxydv(HL_PROXY_POSITION, gAxisCenter);
    }
}


/*****************************************************************************
*
 Event callback triggered when styus button is depressed and touching one of
 the draggable objects.  This callback is always called in client thread.
******************************************************************************
/
void HLCALLBACK buttonDownClientThreadCallback(HLenum event, HLuint object,
                                               HLenum thread, HLcache *cache,
                                               void *userdata)
{
    // Don't proceed if the haptic mouse is active.
    if (hmIsMouseActive())
        return;

    assert(gCurrentDragObj == -1);
    assert(object != HL_OBJECT_ANY);

    // Clicked while touching an object, set this object to be the current
    // object being dragged.  When event callback was registered, we set the
    // index of the drag object as the user data.
    gCurrentDragObj = reinterpret_cast<int>(userdata);

    // Store off proxy position so we can compute how much it moves each
    // frame (which is how much the drag object should also move).
    hlGetDoublev(HL_PROXY_POSITION, gStartDragProxyPos);
    hlGetDoublev(HL_PROXY_ROTATION, gStartDragProxyRot);

    // Store off initial position and orientation of drag object.
    gStartDragObjTransform = draggableObjects[gCurrentDragObj].transform;
}

/*****************************************************************************
*
 Event callback triggered when styus button is depressed and touching one of
 the draggable objects.
******************************************************************************
/
void HLCALLBACK buttonUpClientThreadCallback(HLenum event, HLuint object,
                                             HLenum thread, HLcache *cache,
                                             void *userdata)
{
    // Button up, done dragging, clear current drag object.
    gCurrentDragObj = -1;
```

```
}

/******************************************************************************
*
 Calculates updated object transform for drag object based on changes to
 proxy transform.
 Updated only if it is dragged
*******************************************************************************
/
void updateDragObjectTransform()
{
    // Exit if the dragged object is not one of the defined object

       assert(gCurrentDragObj >= 0 && gCurrentDragObj <
draggableObjects.size());

    // Calculated delta between current proxy pos and proxy pos at start
    // of drag.
    hduVector3Dd proxyPos;
    hlGetDoublev(HL_PROXY_POSITION, proxyPos);// Getting the proxy position to
local variable
    hduVector3Dd dragDeltaTransl = proxyPos - gStartDragProxyPos; // Position
and rotation is taken at the button down callback

    // Same for rotation.
    hduMatrix deltaRotMat;
    if (gRotate) // If rotation is enabled
    {
        hduQuaternion proxyRotq;
        hlGetDoublev(HL_PROXY_ROTATION, proxyRotq);
        hduQuaternion dragDeltaRot = gStartDragProxyRot.inverse() * proxyRotq;
        dragDeltaRot.normalize();
        dragDeltaRot.toRotationMatrix(deltaRotMat);

        // Want to rotate about the proxy position, not the origin,
        // so need to translate to/from proxy pos.
              // Not sure what is it.
        hduMatrix toProxy = hduMatrix::createTranslation(-gStartDragProxyPos);
        hduMatrix fromProxy  = hduMatrix::createTranslation(gStartDragProxyPos);
        deltaRotMat = toProxy * deltaRotMat * fromProxy;
    }

    // Compose rotation and translation deltas.
    hduMatrix deltaMat = deltaRotMat *
hduMatrix::createTranslation(dragDeltaTransl);

    // Apply these deltas to the drag object transform.
    draggableObjects[gCurrentDragObj].transform = gStartDragObjTransform *
deltaMat;
}

/******************************************************************************
*
 Draws a string using OpenGL.
 Used in the draw propmpts function
*******************************************************************************
/
void drawString(const char* string)
{
    for (;*string != '\0';++string)
    {
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, *string);
    }
}

/*****************************************************************************
*
 Draws string prompts at the bottom of the screen.
******************************************************************************
/
void drawPrompts()
{
    glPushAttrib(GL_ENABLE_BIT);
    glDisable(GL_LIGHTING);
    glDisable(GL_DEPTH_TEST);
    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
    glLoadIdentity();
    glOrtho(0, gViewportWidth, 0, gViewportHeight, 0, 1);
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glLoadIdentity();

       if (!started)
       {
    glRasterPos2f(4, 54);
    drawString("Touch the sphere and feel the stiffness and the friction of
the sphere.\n");
    glRasterPos2f(4, 36);
    drawString("Adjust the stiffness and the friction to match the actual
feeling.\n");
    glRasterPos2f(4, 18);
    drawString("Press 'Record' button to record the values.");
       }

       else if (started)
       {
              glRasterPos2f(4, 36);
              drawString("Select one organ from the set.\n");
              glRasterPos2f(4, 18);
              drawString("Adjust the stiffness to match the actual feeling of
the organ");
       }


    glMatrixMode(GL_PROJECTION);
    glPopMatrix();
    glMatrixMode(GL_MODELVIEW);
    glPopMatrix();
    glPopAttrib();
}

/*****************************************************************************
********
Update the stiffness of the object randomly to a previously assigned value
******************************************************************************
********/
void updateStiffness()
```

```
{

            DraggableObject& obj = draggableObjects[0];
            if (hap_stiffness == 0){
                hap_stiffness = .1;
            }
            obj.hap_stiffness = hap_stiffness;
            obj.hap_static_friction = hap_static_friction;

}
/*****************************************************************************
********
Initialize stiffness of the objects at the begining
******************************************************************************
********/
void initStiffness()
{

    DraggableObject& obj = draggableObjects[0];
    if (hap_stiffness == 0){
        hap_stiffness = .1;
    }
    obj.hap_stiffness = hap_stiffness;

    printf("Stiffness Initialized\n");


}


/************************************************************************
Assimp Functions
*************************************************************************/

void get_bounding_box_for_node (const aiNode* nd,
    aiVector3D* min,
    aiVector3D* max
){
    aiMatrix4x4 prev;
    unsigned int n = 0, t;

    for (; n < nd->mNumMeshes; ++n) {
        const aiMesh* mesh = scene->mMeshes[nd->mMeshes[n]];
        for (t = 0; t < mesh->mNumVertices; ++t) {

            aiVector3D tmp = mesh->mVertices[t];

            min->x = aisgl_min(min->x,tmp.x);
            min->y = aisgl_min(min->y,tmp.y);
            min->z = aisgl_min(min->z,tmp.z);

            max->x = aisgl_max(max->x,tmp.x);
            max->y = aisgl_max(max->y,tmp.y);
            max->z = aisgl_max(max->z,tmp.z);
        }
    }

    for (n = 0; n < nd->mNumChildren; ++n) {
```

```
              get_bounding_box_for_node(nd->mChildren[n],min,max);
       }

}

// ----------------------------------------------------------------------------
-
void get_bounding_box (aiVector3D* min, aiVector3D* max)
{
       aiMatrix4x4 trafo;


       min->x = min->y = min->z =  1e10f;
       max->x = max->y = max->z = -1e10f;
       get_bounding_box_for_node(scene->mRootNode,min,max);
}

// ----------------------------------------------------------------------------
-
void color4_to_float4(const aiColor4D *c, float f[4])
{
       f[0] = c->r;
       f[1] = c->g;
       f[2] = c->b;
       f[3] = c->a;
}

// ----------------------------------------------------------------------------
-

void set_float4(float f[4], float a, float b, float c, float d)
{
       f[0] = a;
       f[1] = b;
       f[2] = c;
       f[3] = d;
}

// ----------------------------------------------------------------------------
-

bool Import3DFromFile( const std::string& pFile, const std::string&
pFile2,const std::string& pFile3, const std::string& pFile4, const
std::string& pFile5  )
{

       //check if file exists
       std::ifstream fin(pFile.c_str());
       if(!fin.fail()) {
              fin.close();
       }
       else{
              printf("Couldn't open file: %s\n", pFile.c_str());
              printf("%s\n", imp.GetErrorString());
              return false;
       }

      scene = imp.ReadFile( pFile, aiProcessPreset_TargetRealtime_MaxQuality);
```

```
       // If the import failed, report it
       if( !scene)
       {
              printf("%s\n", imp.GetErrorString());
              return false;
       }

       // Now we can access the file's contents.
       printf("Import of scene %s succeeded.",pFile.c_str());

       DraggableObject dro;
       dro.shapeId = hlGenShapes(1);
    dro.transform = hduMatrix::createTranslation(0,0,0);
       dro.hap_stiffness = 0.49;
    dro.displayList = glGenLists(1);
       dro.show = 1;
    glNewList(dro.displayList, GL_COMPILE);
    recursive_render(scene, scene->mRootNode);
    glEndList();



    draggableObjects.push_back(dro);




       // This is to read the second model in to the scene\


       std::ifstream fin2(pFile2.c_str());
       if(!fin2.fail()) {
              fin2.close();
       }
       else{
              printf("Couldn't open file: %s\n", pFile2.c_str());
              printf("%s\n", imp.GetErrorString());
              return false;
       }

       scene2 = imp.ReadFile( pFile2,
aiProcessPreset_TargetRealtime_MaxQuality);

       // If the import failed, report it
       if( !scene2)
       {
              printf("%s\n", imp.GetErrorString());
              return false;
       }

       // Now we can access the file's contents.
       printf("Import of scene %s succeeded.\n",pFile2.c_str());

       // End of adding the second scene

       // Graphically drawing the organ

       dro.shapeId = hlGenShapes(1);
    dro.transform = hduMatrix::createTranslation(0,0,0);
    dro.displayList = glGenLists(1);
       dro.hap_stiffness = 0.83;
       dro.show = 1;
    glNewList(dro.displayList, GL_COMPILE);
       recursive_render(scene2, scene2->mRootNode);
    glEndList();

       draggableObjects.push_back(dro);

       // This is to read the third model in to the scene
       std::ifstream fin3(pFile3.c_str());
       if(!fin3.fail()) {
              fin3.close();
       }
       else{
              printf("Couldn't open file: %s\n", pFile3.c_str());
              printf("%s\n", imp.GetErrorString());
              return false;
       }

       scene3 = imp.ReadFile( pFile3,
aiProcessPreset_TargetRealtime_MaxQuality);

       // If the import failed, report it
       if( !scene3)
       {
              printf("%s\n", imp.GetErrorString());
              return false;
       }

       // Now we can access the file's contents.
       printf("Import of scene %s succeeded.\n",pFile3.c_str());


       // Graphically drawing the organ

       dro.shapeId = hlGenShapes(1);
    dro.transform = hduMatrix::createTranslation(0,0,0);
    dro.displayList = glGenLists(1);
       dro.hap_stiffness = 0.16;
       dro.show = 1;
    glNewList(dro.displayList, GL_COMPILE);
       recursive_render(scene3, scene3->mRootNode);
    glEndList();

       draggableObjects.push_back(dro);

       // Drawing the fourth organ
       /****************************************************/

       //check if file exists
       std::ifstream fin4(pFile4.c_str());
       if(!fin4.fail()) {
              fin4.close();
       }
       else{
              printf("Couldn't open file: %s\n", pFile4.c_str());
              printf("%s\n", imp.GetErrorString());
              return false;
       }

       scene4 = imp.ReadFile( pFile4,
aiProcessPreset_TargetRealtime_MaxQuality);

       // If the import failed, report it
       if( !scene4)
       {
              printf("%s\n", imp.GetErrorString());
              return false;
       }

       // Now we can access the file's contents.
       printf("Import of scene %s succeeded.\n",pFile4.c_str());


       dro.shapeId = hlGenShapes(1);
    dro.transform = hduMatrix::createTranslation(0,0,0);
       dro.hap_stiffness = 0.16;
    dro.displayList = glGenLists(1);
       dro.show = 1;
    glNewList(dro.displayList, GL_COMPILE);
    recursive_render(scene4, scene4->mRootNode);
    glEndList();



    draggableObjects.push_back(dro);


       //Drawing the fifth organ
       /*****************************************************/
       //check if file exists
       std::ifstream fin5(pFile5.c_str());
       if(!fin5.fail()) {
              fin5.close();
       }
       else{
              printf("Couldn't open file: %s\n", pFile5.c_str());
              printf("%s\n", imp.GetErrorString());
              return false;
       }

       scene5 = imp.ReadFile( pFile5,
aiProcessPreset_TargetRealtime_MaxQuality);

       // If the import failed, report it
       if( !scene5)
       {
              printf("%s\n", imp.GetErrorString());
              return false;
       }

       // Now we can access the file's contents.
       printf("Import of scene %s succeeded.\n",pFile5.c_str());


       dro.shapeId = hlGenShapes(1);
    dro.transform = hduMatrix::createTranslation(0,0,0);
```

```
       dro.hap_stiffness = 0.16;
    dro.displayList = glGenLists(1);
       dro.show = 1;
    glNewList(dro.displayList, GL_COMPILE);
    recursive_render(scene5, scene5->mRootNode);
    glEndList();



    draggableObjects.push_back(dro);

       /*****************************************************/
       //aiVector3D scene_min, scene_max, scene_center;
       get_bounding_box(&scene_min, &scene_max);
       scene_center.x = (scene_min.x + scene_max.x) / 2.0f;
       scene_center.y = (scene_min.y + scene_max.y) / 2.0f;
       scene_center.z = (scene_min.z + scene_max.z) / 2.0f;

       gCameraPosWC[0] = 0;//scene_center.x*gCameraScale;
       gCameraPosWC[1] = 0;//scene_center.y*gCameraScale;
    gCameraPosWC[2] = 5;//scene_max.z*gCameraScale + 5 ;


       printf("%f,%f,%f\n",scene_min.x,scene_min.y,scene_min.z);
       printf("%f,%f,%f\n",scene_max.x,scene_max.y,scene_max.z);
       printf("%f,%f,%f\n",scene_center.x,scene_center.y,scene_center.z);

       for (int i = 0; i < draggableObjects.size(); ++i)
    {
     DraggableObject& obj = draggableObjects[i];

              obj.transform = hduMatrix::createTranslation(-scene_center.x,-
scene_center.y,-scene_center.z);

    }

       assimpDisplay();
       // We're done. Everything will be cleaned up by the importer destructor
       return true;
       }

//------------------------------------------------------------------------

void assimpDisplay(void){
       float tmp;
       tmp = scene_max.x-scene_min.x;
       tmp = aisgl_max(scene_max.y - scene_min.y,tmp);
       tmp = aisgl_max(scene_max.z - scene_min.z,tmp);
       scaleFactor = 1.f / tmp;
       printf("scale is %f\n",scaleFactor);

}

//----------------------------------------------------------------------------
--
void recursive_render (const aiScene *sc, const aiNode* nd)
{
       unsigned int i;
       unsigned int n = 0, t;
       aiMatrix4x4 m = nd->mTransformation;
       aiColor4D color = (0.0,0.0,0.0,1.0);

       // update transform
       m = m.Transpose();
       glPushMatrix();
       glMultMatrixf((float*)&m);

       // draw all meshes assigned to this node
       for (; n < nd->mNumMeshes; ++n) {
              const aiMesh* mesh = scene->mMeshes[nd->mMeshes[n]];

              apply_material(sc->mMaterials[mesh->mMaterialIndex]);

              if(mesh->mNormals == NULL) {
                     glDisable(GL_LIGHTING);
              } else {
                     glEnable(GL_LIGHTING);
              }

              for (t = 0; t < mesh->mNumFaces; ++t) {
                     const aiFace* face = &mesh->mFaces[t];
                     GLenum face_mode;

                     switch(face->mNumIndices) {
                         case 1: face_mode = GL_POINTS; break;
                         case 2: face_mode = GL_LINES; break;
                         case 3: face_mode = GL_TRIANGLES; break;
                         default: face_mode = GL_POLYGON; break;
                     }

                     glBegin(face_mode);

                     for(i = 0; i < face->mNumIndices; i++) {
                         int index = face->mIndices[i];

                         if(mesh->mColors[0] != NULL){
                             glColor4fv((GLfloat*)&mesh-
>mColors[0][index]);

                         }
                         if(mesh->mNormals != NULL) {
                             glNormal3fv(&mesh->mNormals[index].x);
                         }


                         glVertex3fv(&mesh->mVertices[index].x);
                     }

                     glEnd();
              }

       }

       // draw all children
       for (n = 0; n < nd->mNumChildren; ++n) {
              recursive_render(sc, nd->mChildren[n]);
       }
```

```
        glPopMatrix();
        printf("Rendered");
}


// ---------------------------------------------------------------------------
-
void apply_material(const aiMaterial *mtl)
{
        float c[4];

        GLenum fill_mode;
        int ret1, ret2;
        aiColor4D diffuse;
        aiColor4D specular;
        aiColor4D ambient;
        aiColor4D emission;
        float shininess, strength;
        int two_sided;
        int wireframe;
        unsigned int max;

        set_float4(c, 0.8f, 0.8f, 0.8f, 1.0f);
        if(AI_SUCCESS == aiGetMaterialColor(mtl, AI_MATKEY_COLOR_DIFFUSE,
&diffuse))
                color4_to_float4(&diffuse, c);
        glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, c);

        set_float4(c, 0.0f, 0.0f, 0.0f, 1.0f);
        if(AI_SUCCESS == aiGetMaterialColor(mtl, AI_MATKEY_COLOR_SPECULAR,
&specular))
                color4_to_float4(&specular, c);
        glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, c);

        set_float4(c, 0.2f, 0.2f, 0.2f, 1.0f);
        if(AI_SUCCESS == aiGetMaterialColor(mtl, AI_MATKEY_COLOR_AMBIENT,
&ambient))
                color4_to_float4(&ambient, c);
        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, c);

        set_float4(c, 0.0f, 0.0f, 0.0f, 1.0f);
        if(AI_SUCCESS == aiGetMaterialColor(mtl, AI_MATKEY_COLOR_EMISSIVE,
&emission))
                color4_to_float4(&emission, c);
        glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION, c);

        max = 1;
        ret1 = aiGetMaterialFloatArray(mtl, AI_MATKEY_SHININESS, &shininess,
&max);
        if(ret1 == AI_SUCCESS) {
        max = 1;
        ret2 = aiGetMaterialFloatArray(mtl, AI_MATKEY_SHININESS_STRENGTH,
&strength, &max);
                if(ret2 == AI_SUCCESS)
                        glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, shininess *
strength);
         else
                glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, shininess);
```

```
    }
        else {
                glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 0.0f);
                set_float4(c, 0.0f, 0.0f, 0.0f, 0.0f);
                glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, c);
        }

        max = 1;
        if(AI_SUCCESS == aiGetMaterialIntegerArray(mtl,
AI_MATKEY_ENABLE_WIREFRAME, &wireframe, &max))
                fill_mode = wireframe ? GL_LINE : GL_FILL;
        else
                fill_mode = GL_FILL;
        glPolygonMode(GL_FRONT_AND_BACK, fill_mode);

        max = 1;
        if((AI_SUCCESS == aiGetMaterialIntegerArray(mtl, AI_MATKEY_TWOSIDED,
&two_sided, &max)) && two_sided)
                glDisable(GL_CULL_FACE);
        else
                glEnable(GL_CULL_FACE);
}

void loadIdentityToViewRotate(){
        view_rotate[0] = 1;
        view_rotate[1] = 0;
        view_rotate[2] = 0;
        view_rotate[3] = 0;
        view_rotate[4] = 0;
        view_rotate[5] = 1;
        view_rotate[6] = 0;
        view_rotate[7] = 0;
        view_rotate[8] = 0;
        view_rotate[9] = 0;
        view_rotate[10] = 1;
        view_rotate[11] = 0;
        view_rotate[12] = 0;
        view_rotate[13] = 0;
        view_rotate[14] = 0;
        view_rotate[15] = 1;
}


/*
This is to generate the rotation matrix from quaternion
*/

void quaternion_to_rotationMat(float x, float y, float z, float angle){
        float qw,qx,qy,qz;
        float temp1[16] = { 1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1 };
        float temp2[16] = { 1,0,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1 };


        qw = cos(angle/2);
        qx = x*sin(angle/2);
        qy = y*sin(angle/2);
        qz = z*sin(angle/2);

        temp1[0] = 1-2*qy*qy - 2*qz*qz;
```

```
       temp1[1] = 2*qx*qy + 2*qz*qw;
       temp1[2] = 2*qx*qz - 2*qy*qw;

       temp1[4] = 2*qx*qy - 2*qz*qw;
       temp1[5] = 1-2*qx*qx - 2*qz*qz;
       temp1[6] = 2*qy*qz - 2*qx*qw;

       temp1[8] = 2*qx*qz + 2*qy*qw;
       temp1[9] = 2*qy*qz - 2*qx*qw;
       temp1[10] = 1-2*qx*qx - 2*qy*qy;

       for(int j=0; j<16 ; j+=4){
              for (int i = j; i<j+4;i++){
                     temp2[i] = view_rotate[i]*temp1[j] + 
view_rotate[i+4]*temp1[j+1] + view_rotate[i+8]*temp1[j+2] + 
view_rotate[i+12]*temp1[j+3];
              }
       }

       for (int i = 0; i<16; i++){
              view_rotate[i] = temp2[i];
       }
}
```