

Tablet-Based Training for Open Reduction Internal Fixation:
A Model for Complex Pelvic Trauma Surgery

by
Vondel Shadrach Edwin Mahon

A thesis submitted to Johns Hopkins University
in conformity with the requirements for
the degree of Master of Arts

Baltimore, Maryland
March, 2019

© 2019 Vondel Shadrach Edwin Mahon
All Rights Reserved

ABSTRACT

Open Reduction Internal Fixation (ORIF) surgery of the pelvis is a highly complex procedure that should only be performed by the most highly skilled and experienced orthopaedic surgeons. To develop the appropriate skills, surgeons must complete at least five years of orthopaedic residency training including mentorship and exposure to a wide array of surgical cases. The expectation of residency training is that new surgeons will develop the skills and competencies, and attain the experience that will enable them to be proficient in their future role. However, the current training system is outdated and often does not provide sufficient opportunities to master skills, particularly those that require surgeons to convert 2D information into 3D mental models. As a result, some orthopaedic surgeons may be inadequately prepared to perform ORIF surgeries.

This thesis explores the design of a tablet-based 3D interactive surgical training tool that teaches ORIF procedures. It will allow orthopaedic residents to improve geometric and spatial understanding of fractures, skeletal anatomy, and surgical hardware, in the context of ORIF surgery; enable investigation of optimal positioning of plates and screws, and allows users to confirm their placement by means of x-ray views; promote higher levels of confidence in reduction and fixation of unstable pelvic fractures. The overarching goal of this interactive surgical training tool is to enable surgical residents to practice ORIF surgery on virtual patients created from Digital Imaging and Communications in Medicine (DICOM) derived anatomical models and virtual surgical hardware. This approach allows DICOM data to be developed into virtual cases and used as training material, thus allowing residents to increase their exposure to relevant surgeries and improve their skills in ORIF procedures.

The research project is a work in progress. It is part of a larger study at Johns Hopkins University School of Medicine. The interactive surgical training tool has the potential to truly impact the quality of patient care by helping orthopaedic residents gain the skills and experience necessary while using simulated cases. Once fully tested and developed, this interactive surgical training tool has the potential to be used by orthopaedic residency programs around the globe.

CHAIRPERSONS OF THE SUPERVISORY COMMITTEE

Greg M. Osgood, MD, *Preceptor*

Chief, Orthopaedic Trauma, Department of Orthopaedic Surgery, Johns Hopkins Hospital

Associate Professor of Orthopaedic Surgery, Johns Hopkins University School of Medicine

David A. Rini MFA, CMI, FAMI, *Faculty Advisor*

Professor, Graduate Program Director, Department of Art as Applied to Medicine, Johns Hopkins School

of Medicine

Mathias Unberath, PhD, *Technical Advisor*

Executive Director, The Johns Hopkins Center for Bioengineering, Innovation, and Design

Assistant Professor of Biomedical Engineering, The Johns Hopkins University

ACKNOWLEDGEMENTS

The success of this research project would not have been possible without my faith in my Lord and Savior, Jesus Christ; and the support, guidance, and encouragement that I received from a team of exceptional individuals. These individuals have dedicated their time, resources, and expertise during the research process. I would like to express my sincere gratitude to them for helping me to bring this project to fruition.

David A. Rini, Department Advisor and Professor, Graduate Program Director, Department of Art as Applied to Medicine, Johns Hopkins School of Medicine. David was a tremendous inspiration to me. His steadfastness, comprehensive guidance, and wholesome coaching was invaluable.

Greg M. Osgood, Chief, Orthopaedic Trauma, Department of Orthopaedic Surgery, Johns Hopkins Hospital and Associate Professor of Orthopaedic Surgery, Johns Hopkins University School of Medicine. Dr. Osgood played an essential role as my preceptor. His meaningful assistance imparting knowledge and providing expert advice deepened my understanding of the subject matter throughout the process.

Mathias Unberath, Assistant Research Professor, Department of Computer Science, Laboratory for Computational Sensing and Robotics, Johns Hopkins University played a tremendous role in this project. As my primary Technical Advisor, his encouragement and leadership has helped me to develop higher levels of creativity and greater appreciation for research.

Kristen Browne from the National Institute of Health, U.S. National Library of Medicine, Lister Hill National Center for Biomedical Communications. Her generosity in providing the much needed digital anatomical models allowed me to develop a deeper understanding of creative processes.

Elizabeth Weissbrod, **Eric Acosta**, and **Jamie Cope** from the Uniformed Services University of Health Sciences, Val G. Hemming Simulation Center. Their guidance and expert knowledge significantly impacted the project.

Patrick McCarthy from STAR Academy. Without his assistance with programming and helping me understand the scripts used in the simulation, this project would not have been as successful as it was.

The faculty and staff of the Department of Art as Applied to Medicine. Their patience, kindness, and instruction equipped me with all the tools necessary to complete this project.

My classmates, Insil Choi, Cecilia Johnson, Alisa Brandt, Brittany Bennett, and Lohitha Kethu. Their collegiality, creativity, and encouragement has been a tremendous blessing to me.

Dan Hermansen, Julia Lerner, Hillary Wilson, Amanda Slade. Their guidance and support provided during this process was tremendous.

The Vesalius Trust. Their financial support of this project was extremely generous.

My family and friends. Their support and encouragement has been bountiful throughout my studies. Without them none of this would have been possible.

TABLE OF CONTENTS

Abstract	ii
Chairpersons of the Supervisory Committee	iii
Acknowledgements	iv
Table of Contents	vi
Index of Figures	x
Introduction	1
Background.....	1
Statement of Problem.....	1
Solution.....	1
Objectives and Scope of this Thesis Project.....	2
Review of Literature	4
The Apprenticeship Model of Surgical Training.....	4
Functional and Clinical Significance of the Pelvis	5
Unstable Pelvic Fracture Epidemiology and Residency Training	5
2D Training Material in Orthopaedic Training	6
Potential of Radiological Imaging	6
Effectiveness and Benefit of Virtual Training.....	7
3D Visualization for 3D Implementation.....	7
Materials and Methods	9
Early Stage Medical Software Accelerator Program	10
<i>Business Solution</i>	10
<i>Brief Summary of the Business Model</i>	10
1. <i>Identifying the target audience</i>	10
2. <i>Identifying market opportunity and target customers</i>	10

3. <i>Determining partnerships</i>	10
4. <i>Identifying key competitors</i>	11
5. <i>Conducting user interviews</i>	11
6. <i>Identifying the program’s technical specifications</i>	12
7. <i>Identifying the Minimum Viable Product (MVP)</i>	12
Creative Software and Units of Measurement.....	13
Creating the User Interface (UI) Elements... ..	13
Creating the Virtual Bone Models	13
<i>Data Acquisition</i>	13
<i>Fracture Creation</i>	14
<i>Approximation and Recreation of the Fracture Fragments</i>	15
<i>Creating the Low Res Skeletal Model from High Res Model</i>	18
Creating the Virtual Instrument Models.....	20
Creating Plates..... ..	20
Creating Screws	22
Creating the IOS application.....	22
<i>Import into Unity</i>	22
<i>Unity Project Set-up and Unity Remote</i>	23
<i>Importing 3D Models</i>	23
<i>Importing UI Elements</i>	23
<i>Scene Development</i>	23
<i>Menu Scene Development</i>	24
<i>Simulation Scene Development</i>	24
<i>Script Functions and Usage</i>	26
<i>Touch Gestures</i>	27
<i>3D Orbit Fracture Selection</i>	27

<i>Pan Orbit Script</i>	27
<i>Pan Rotate</i>	28
<i>Window Manager Script</i>	28
<i>Snap in Place and Lerp Script</i>	29
<i>Spawn Object Script</i>	29
<i>Destroy All Tagged Script</i>	30
<i>OR Timer Script</i>	30
<i>Enable Disable Object Script</i>	31
<i>Change Material Script</i>	31
<i>Load Scene Script</i>	31
Results and Discussion	32
Case Selection Menu System.....	32
Simulation Scene	34
Conclusion	37
Future Research on the Interactive Surgical Training Tool	38
Asset Referral Info.....	39
Appendices	40
Appendix A: 3D Orbit Fracture Selection	40
Appendix B: Pan Orbit Script.....	42
Appendix C: Pan Rotate.....	50
Appendix D: Window Manager Script	56
Appendix E: Snap in Place	57
Appendix F: Lerp Script	58
Appendix G: Spawn Object Script.....	59
Appendix H: Destroy All Tagged Script.....	60
Appendix I: OR Timer Script.....	60

Appendix J: Enable Disable Object Script.....	61
Appendix K: Change Material Script	61
Appendix L: Load Scene Script.....	62
References	63
Vita	68

INDEX OF FIGURES

Figure 1. Orthopaedic templating materials.....	
Figure 2. Competitive Landscape	
Figure 3. models received from CAMP and NIH-NLM-LHNCBC.....	
Figure 4. Superimposed models.....	
Figure 5. Sculpted cortical and cancellous bone areas.....	
Figure 6. Cube3D transferred to high-resolution model	
Figure 7. Created fracture fragments	
Figure 8. Seamless effect of the systematic approach.....	
Figure 9. Painted high-resolution model.....	
Figure 10. Low resolution state of newly created model.....	
Figure 11. Pivot/axis points centered	
Figure 12. Rigged plate.....	
Figure 13. Collider modifications	
Figure 14. RigidBody parameter settings	
Figure 15. Empty fields of the skeleton gameobject.....	
Figure 16. Main menu window	
Figure 17. Skeletal Regions window	
Figure 18. Pelvis window.....	
Figure 19. Case selection interface	
Figure 20. Case 0001	
Figure 21. Simulation instructions	
Figure 22. Instrument selection window.....	
Figure 23. X-ray vision of screw fixation	

INTRODUCTION

Background

Patients with unstable pelvic fractures typically undergo Open Reduction Internal Fixation (ORIF) surgery. This is an uncommon and complex procedure wherein an orthopaedic surgeon specializing in trauma surgery first cleans the fracture, reduces the components (puts them back in place), then fixes (stabilizes) them with surgical hardware. This procedure is performed to restore the normal anatomy and volume of the pelvis.

To perform ORIF surgeries the surgeon must be highly skilled and experienced and possess the ability to mentally reconfigure 2D images such as radiographs into 3D mental models in order to visualize the 3D anatomy and volume of the pelvis and its associated structures. In addition, the surgeon must also be able to visualize the 3D spatial orientation of surgical hardware when planning and performing fixations.

Statement of Problem

The ability to mentally reconfigure 2D images of the pelvic anatomy into 3D mental models is not appropriately cultivated during residency training. This disadvantage of residency training is due to (1) the limitations of the apprenticeship model of physician training, (2) reliance on chance and on availability of relevant and appropriate training cases, (3) the complexity of unstable pelvic surgery, and (4) the relatively low number of unstable pelvic fracture cases, which is further limited by specialty case restrictions during residency. In addition, a resident's understanding of the spatial dynamics of the pelvis and surgical hardware in the context of ORIF may be undermined by ineffective and outdated 2D training materials such as textbooks, 2D imagery, and orthopaedic templating kits (Figure 1). These 2D representations of 3D objects impair the trainee's perception of depth and spatial volume. Together these factors severely limit opportunities to develop critical ORIF skills in the context of complex pelvic trauma surgery.

Solution

In the past, various groups have attempted to address this problem but have fallen short of providing a reliable and effective solution that takes into consideration all aspects of the problem. The Computer Aided Medical Procedures laboratory (CAMP) began investigating this problem in June of 2018, in a

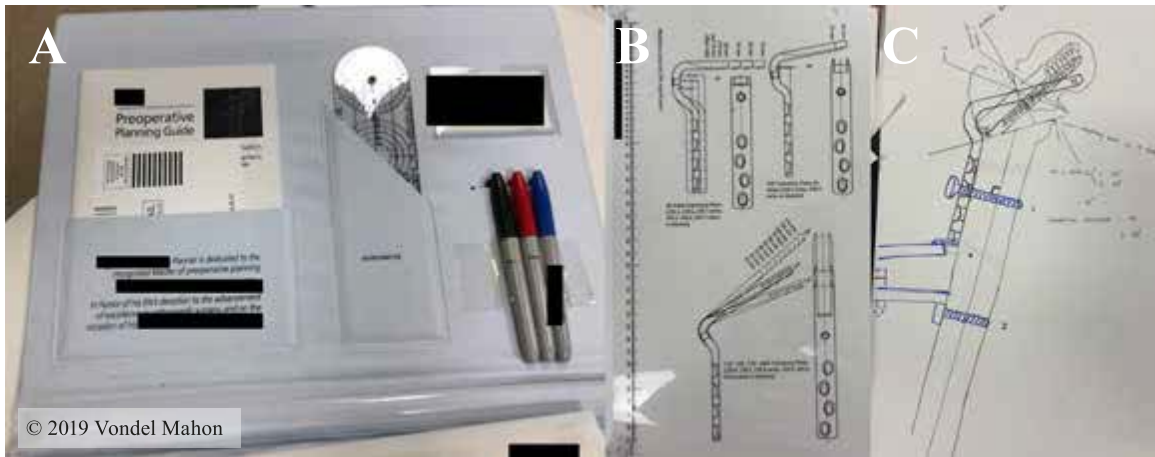


Figure 1. Sample of an orthopaedic templating kit. Courtesy of an orthopaedic surgeon at Johns Hopkins University. (A) Templating kit, (B) 2D acetate template, (C) drawings made from using templates. Text not intended to be read.

collaborative effort by Mathias Unberath, PhD, Nassir Navab, PhD, and Greg M. Osgood, MD. The team began by first addressing the cognitive and spatial needs via an Augmented Reality (AR) ORIF simulation module; however, because the availability of AR hardware is still limited even in healthcare settings, other digital delivery platforms were considered. This thesis project was proposed to develop a working training module for tablet delivery and to provide a direct comparison with the AR delivery system to assist CAMP in determining the most viable solution to the ORIF surgery training dilemma.

This thesis explores the design of a sustainable 3D interactive surgical simulation tool which teaches ORIF procedures. The tool will allow surgical residents to practice ORIF on virtual Digital Imaging and Communications in Medicine (DICOM)-derived anatomical models using virtual surgical hardware.

Objectives and Scope of this thesis Project

Due to the magnitude of the problem, market potential, and technical complexity, the thesis project was conceived in the form of a startup business, as an appendage of The Computer Aided Medical Procedures laboratory (CAMP). Due to accessibility considerations, it was decided that mobile-based platform should be explored. Upon this consideration, the thesis objectives were formulated. These were to design a tablet-based interactive surgical training tool that will allow the user to:

- Improve geometric and spatial understanding of the fracture, anatomy, and hardware crucial for correct execution of ORIF.

- Enable interactive investigation of optimal positioning of plates, screw paths, and x-ray views.
- Gain experience and attain higher levels of confidence in reduction and fixation of unstable pelvic fractures.
- Improve surgical considerations when approaching ORIF procedures.
- Engage in discovery through a quasi-open world experience.
- Access multiple fracture cases requiring ORIF.
- Transfer knowledge into the clinical setting.

Review of the Literature

THE APPRENTICESHIP MODEL OF SURGICAL TRAINING

Surgeons have trained under the apprenticeship model since the 16th century. This system of learning operates on the concept of “see one, do one, teach one” (LeCompte, Stewart, Harris, Rivers, Guth, Ehrenfeld, Sexton, & Terhune, 2019). In this system, the trainee learns a procedure through direct observation, then performs the procedure in the fashion observed. Eventually the learner would be called upon to then teach the procedure to another individual to complete the learning process. Although this system was later modified by Sir William Halsted in 1887, the core principles are still the building blocks of residencies to this day (Polavarapu, Kulaylat, Sun & Hamed, 2013). In the apprenticeship model, real patients serve as the primary teaching model; thus, trainees practice on the very individuals that they are training to care for.

Although using patients as training material is the most hands-on and authentic experience a trainee can receive, the apprenticeship model is inadequate. The model fails to provide continuity in the learning process (Rassie, 2017). Appropriate procedures for effective learning are not always available in a timely way because the apprenticeship model by design relies on actual patients. Therefore, residents must depend on chance to observe and practice procedure-specific skills, a problem especially acute for relatively rare procedures such as Open Reduction Internal Fixation (ORIF).

There are alternative methods of training currently implemented in other non-medical industries. In aviation, for instance the practitioner (pilot) undergoes hundreds of hours of simulation training before ever applying their skills in a situation where the lives and safety of people are at stake (de Montbrun & Macrae, 2012). Indeed, the use of virtual simulation has been used in aviation since the 1950s (de Montbrun & Macrae, 2012). Unlike the apprenticeship model which relies on contingent training material for skills training, aviation and other high-stakes industries use virtual simulation, which is available at the convenience of the trainee while achieving the same learning goals, all without placing lives at risk. (Weinstock, 2017; Issenberg, McGaghie, Hart, Mayer, Felner, Petrusa, Waugh, Safford, Gessner, Gordon & Ewy, 1999).

FUNCTIONAL AND CLINICAL SIGNIFICANCE OF THE PELVIS

Patients who sustain trauma to the pelvis resulting in complex and unstable fractures, not only suffer damage to the bony pelvis, but also to surrounding soft tissues as well. These fractures can cause hemorrhaging from the fracture sites and lead to long term impairment of normal locomotive functions (Palmer & Bircher, 1997). In severe cases, pelvis fractures can be fatal (Halawi, 2015).

A serious pelvic injury can have a lasting effect on the lifestyle and comfort of the patient. Structurally, the human pelvis is designed for bipedal movement and locomotion. This is a defining characteristics in the development of the human pelvis (Lovejoy, 1988; Smithsonian National Museum of Natural History, 2018). In addition to this function, the pelvis is also optimized for carrying and transmitting weight. Weight-bearing is most significant during gait activities and sitting (OpenStax, 2013), and notably absent when a person is lying recumbent such as when sleeping (AfterTrauma, n.d.).

In 2017, according to the United States Department of Labor, the average person spends 8.8 hours per day sleeping (AfterTrauma, n.d.). The remainder of the day appears to be spent in other activities most of which involve an upright body position, and thus weight-transmission through the pelvis (United States Department of Labor, 2018). Therefore, it can be estimated that the average person spends approximately two thirds of each day, about 16 hours, in activities where the pelvis is weight-bearing. Thus, if the pelvis sustains trauma and inadequately heals or is poorly repaired during surgery, the lasting effects can have a significantly negative impact on the individual's quality of life. This is one reason only highly experienced surgeons perform ORIF pelvic surgeries. The impact of complications can be devastating, and residents simply do not have the experience to adequately ensure successful outcomes. In fact, in 2009, it is reported that not only do residents expose patients to injury due to skill and knowledge-based mistakes, their mistakes can and have led to patient death (Rodriquez-Paz, Kennedy, Salas, Wu, Sexton, Hunt & Pronovost, 2009).

UNSTABLE PELVIC FRACTURE EPIDEMIOLOGY AND RESIDENCY TRAINING

In the year 2000, the global estimate of all skeletal fractures was 9 million (Johnell & Kanis, 2006). Pelvic fractures account only for 1-3% of that number (Mardanpour & Rahbar, 2013, p. 77); in addition, only 10% of pelvic fractures are considered unstable and therefore qualify for ORIF intervention (White, Hsu, & Holcomb, 2009). These statistics would yield between 9,000 to 27,000 global unstable pelvic

fracture cases per year.

As of 2013, there were approximately 150 orthopaedic residency programs in the U.S. producing approximately 620 graduates annually (Wolf & Britton, 2013). During residency, trainees are limited to a total of 12 months of specialty training in areas such as trauma surgery (The American Board of Surgery 2019). Even if these U.S. orthopaedic residents had access to the caseload of the Americas, each student would still only have access to between 2.3 and 6.8 cases.

Thus, the average resident will experience a volume of ORIF-qualifying pelvic fracture cases lower than 2.3 to 6.8 during residency. This limited exposure to specialty cases is one of the reasons why residents seek further training. In fact, approximately 12% of those 2013 graduates went on to pursue fellowships in orthopaedic trauma to further develop their skills (Horst, Choo, Bharucha & Vail, 2015; Daniels & DiGiovanni, 2014).

2D TRAINING MATERIAL IN ORTHOPAEDIC TRAINING

2D templating using acetate templates, tracing paper, and radiographs is commonly used in orthopaedic training and pre-operative planning. Although 3D templating software is available, the use of 2D techniques remains the standard (Carter, Stovall & Young, 1995; Ruedi, Buckley & Moran, 2007). However, templating does not allow for appropriate ORIF skill-transfer into the operating room (OR). The expectation of using these 2D materials is that the individual is able to transfer knowledge gained from a 2D source and apply it to a 3D situation. An experienced surgeon would have honed this ability over years of practice, but a resident who has received limited opportunities to do so would likely encounter challenges. Barr (2010) describes the phenomenon in this way:

Mapping a memory encoded from a 2D image onto a 3D object presented at a later time relies on a representation of the object that can enable translation between dimensions. The need for a translation between 2D images and 3D objects presents significant perceptual challenges (p. 132).

POTENTIAL OF RADIOLOGICAL IMAGING

Treatment of fractures requires radiographic examination for assessment and surgical planning. However, current technology in segmentation of volumetric data also allows the imagery to be used for surgical training (John, 2007). These images have the potential to be converted into virtual cases and

used as training material, thus allowing the surgeon to “practice” with a simulation many times before operating on the real patient (Bloice, Simonic & Holzinger, 2013). Not only would this be beneficial in pre-operative planning, residents from all over the world would have access to a vast collection of training cases and be able to practice repeatedly on virtual tissue. In this format, residents would have access to real cases and infinite opportunities to improve their skills using virtual models without endangering real life patients (Triola, Feldman, Kalet, Zabar, Kachur, Gillespie, Anderson, Griesser & Lipkin, 2006).

EFFECTIVENESS AND BENEFIT OF VIRTUAL TRAINING

In 1997, it was estimated that the cost of training 1,014 general surgery residents in the U.S. was \$53 million dollars; this was largely due to time spent in the OR to gain proficiencies. It has been suggested that virtual surgery modules should be used to lower this cost (Bridges & Diamond, 1999). Several studies revealed that virtual simulation has not only proven to abate the cost of the OR, it has also proven to improve surgical performance and reduce rates of error (Seymour, Gallagher, Roman, O’Brien, Bansal, Anderson & Satava, 2002; Gurusamy, Aggarwal, Palanivelu & Davidson, 2008). These studies indicated the importance of virtual training. They concluded that surgical trainees who received training in virtual simulation modules outperformed their peers who were trained in traditional methods. In addition, those that were trained in virtual simulation modules reduced their rates of error and operating time far more than their peers who were trained in traditional methods.

3D VISUALIZATION FOR 3D IMPLEMENTATION

A number of studies have concluded that students and clinicians of varying experience and skill levels prefer and benefit from 3D visualization over traditional 2D methods of teaching complex anatomy (Noguera, Jiminez, & Osuna-Perez, 2013; Beermann, Tetzlaff, Bruckner, Schoebinger, Muller-Stitch, Gutt, Meinzer, Kadmon, & Fischer, 2010; Estevez, Lindgren & Bergethon, 2010; Brown, Hamilton, & Denison, 2012; Tanagho, Andriole, Paradis, Madison, Sandhu, Varela, & Benway, 2012; Pujol, Baldwin, Nassiri, Kikinis, & Shaffer, 2016).

The studies tested the effectiveness of learning surgical and nonsurgical anatomy using 3D as opposed to 2D visualization. In conducting the studies, a variety of 3D visual communication media, subject matters and subjects were employed. Findings suggested that 3D visualization:

- Led to improved performance, greater speed, and fewer errors on difficult tasks.
- Allowed for increased understanding anatomical shapes and spatial relationships.
- Is preferred over 2D visualization or traditional methods of teaching.
- Was a viable potential supplement when teaching complex anatomy.
- Better prepares users for visualization of 3D anatomy.
- Led to better learning outcomes in teaching anatomy.
- Was recommended by participants for future use.
- Provided an advantage to those who employed it.
- Improved the understanding of complex spaces.
- Aided in the understanding of anatomy.
- Led to higher anatomy quiz scores.
- Greatly enhances proficiency.

Although these studies did not involve the pelvis and ORIF surgery per se, they indicated that despite the experience level of the subject (student or clinician) and medical subject matter, 3D visualization was advantageous in the context of understanding 3D medical subject matter.

MATERIALS AND METHODS

Early Stage Medical Software Accelerator Program

Since the proposed solution to the problem fell within the domain of digital healthcare, the startup was developed in collaboration with the Johns Hopkins Medicine Technology Innovation Center's (TIC) early stage medical software accelerator program, Hexcite. Hexcite provided a platform in which the solution could be thoroughly explored through extensive background and market research.

The program lasted 16 weeks and followed a "shark-tank" business development model, in that business pitches were conducted weekly and critiqued by a variety of judges. Throughout the program, key steps were taken to develop and refine the business solution. These steps are listed below, in chronological order:

- Establish and solidify a team through contractual agreement.
- Name the startup; the name selected for this project was FxAR, a combination of the abbreviation and acronym for fracture and Augmented Reality.
- Define the problem, solution, and market opportunities.
- Develop a business plan using a business model canvas: Utilizing LaunchPad Central, to identify: value proposition, customer segments, key partners, activities, resources, customer relationship, distribution channel, cost structure, and revenue stream.
- Develop "user stories".
- Design and refine business hypotheses through stakeholder interviews.
- Develop technology evaluation plan (a one year study at Johns Hopkins Hospital (JHH)).
- Develop a commercialization plan.
- Create preliminary wireframes.
- Identify key competitors.
- Define the Minimum Viable Product (MVP).
- Identify deployment platform.
- Perform final pitch at business competition.

Tasks were distributed amongst the team which consisted of a clinical lead, Greg Osgood, M.D, a senior design lead, David Rini, MFA, and a junior design lead (the writer). Elaborate market research was

conducted throughout the process to develop and refine the business solution and model. The program concluded in the form of a business competition at the 2018 Johns Hopkins Digital Health Day. The clinical lead, Greg Osgood M.D., pitched the proposal, and the team received the top award.

BUSINESS SOLUTION

It was decided that the business solution to the problem was to deliver a product to the target audience (described below), which would be mobile based, contain a suite of 3D simulated skeletal fracture cases, allow the practice of ORIF procedures using Digital Imaging and Communications in Medicine (DICOM) derived data, and augment the training received in the operating room.

BRIEF SUMMARY OF THE BUSINESS MODEL

1. Identifying the target audience

Orthopaedic surgical residents were identified as the primary users of the business solution. As of 2018, there were 358 in the US (National Resident Matching Program, 2018, 2017). As expressed above, residents are the most affected by the challenge of learning complex pelvic reconstructive surgery and have the greatest need for the business solution. The secondary users are orthopaedic surgical faculty who specialize in trauma surgery and are responsible for teaching ORIF cases to residents.

2. Identifying market opportunity and target customers

The global market for medical simulation devices in 2017 was \$1.2 Billion USD, and was estimated to double by 2022 (Hayhurst, 2018). To the authors knowledge there was no other existing tablet-based simulation device which allows users to manually perform ORIF in virtual 3D. Thus, the uniqueness of the business solution, in addition to the need discovered, suggested that a significant portion of the medical simulation device market could be appropriated.

Although the primary users of this product would be orthopaedic residents it was decided that the financial customers would likely be residency programs in the United States, of which there were 202 (Accreditation council for Graduate Medical Education, 2019).

3. Determining partnerships

Potential business partnerships were also considered during Hexcite Research conducted by Gerald Donahoe and Guy King, F.S.A., M.A.A.A. estimated that 150 billion dollars is spent each year in the U.S.

alone on medical devices (Okike, O’Toole, Pollak, Bishop, McAndrew, Mehta, Cross, Garrigues, Harris, & Lebrun, 2014). This identified advertising opportunities for medical device manufacturers. Several device manufacturing companies were considered as potential partners during this process based on global market share. In addition, due to the medical content of the product, secondary partnerships were considered with orthopaedic training associations.

4. Identifying key competitors

Extensive research was conducted to identify potential competitors within the scope of the startup’s market space, product features, and product content. Once competitors were identified and scrutinized, a refined list was developed (Figure 2). This list displayed features that were within the scope of each platform. It should be noted that competitors utilized a variety of computer operating platforms, or in the case of one competitor, 3D Systems, a combination of platforms. Only one competitor, however, utilized a mobile platform to deliver its product.

© 2019 Vondel Mahon





	Expanding case library	Feedback alerts	Crowd-sourced content	Case approach variation	Real-time guided & non-guided modes
	✓	✗	✗	✗	✗
	✗	✓	✗	✗	✗
	✓	✓	✗	✗	✗
	✓	✓	✓	✓	✓

Figure 2. Competitor matrix showing refined list of product features.

5. Conducting user interviews

Over thirty user interviews were conducted with physicians and technologists at Johns Hopkins Hospital (JHH), AO Trauma Foundation, One AO, and the Mixed Augmented and Virtual Reality Conference (MAVRiC). Together with CAMPs initial aims, the stakeholder feedback assisted in the

development of the product design at the early stages of project. Stakeholders were asked what actions they took when preparing for a case which required 3D understanding of anatomy. One mentioned studying the x-ray and visualizing the procedure in his head after reading articles. Another mentioned drawing on a bone model and mentally rehearsing the case to plan out the procedure.

The responses from the interviews highlighted a definite need and genuine interest for the proposed product. The stakeholders expressed that if the proposed product existed, it would be a tool that they would find particularly useful. In addition, several potential users suggested other features that they would find useful in such a tool.

6. Identifying the program's technical specifications

Although an AR solution was considered prior to Hexcite, it was decided that the thesis product would be delivered via a mobile device. This decision was made due to the accessibility a mobile device would provide to the user. Thus, a tablet device was chosen, specifically an Apple iPad Pro (12.9-inch). The Apple iPad Pro, specifically, was selected due to the screen resolution and screen to fingertip-width ratio, which was critical for the user experience due to the manual movement expected while using a touch screen device.

7. Identifying Minimum Viable Product (MVP)

The Minimum Viable Product (MVP) is a product designed with the minimum features required to function; these features nevertheless solve the underlying problem. The MVP is considered to be an optimal first-stage-of-development product allowing developers to iterate and gradually test the value of a product while assuming minimum risk (Rancic Moogk, 2012). This methodology was implemented through a process learned in Hexcite, which enabled the refinement of a gross list of user-desired features. Once the gross list was completed, a condensed list of MVP features was established.

After the MVP was determined, the first iteration of this product was designed to focus on features targeting the spatial, geometrical, and depth challenges that orthopedic residents experienced during training for ORIF of unstable pelvic fractures. Further, it was decided that an Acetabular Associated Anterior Column Posterior Hemitransverse fracture would be used as the proof of concept. This fracture type was chosen due to the complexity of the fracture and the spatial and geometrical understanding, which would be required to perform ORIF.

Creative Software and Units of Measurement

A variety of software applications were used to develop this interactive surgical training tool, including Zbrush® 4R8, Adobe Illustrator® CC, Adobe Photoshop® CC, Maxon Cinema 4D® R20, Visual Studio® 2017, and Unity® 3D 2018. The metric system of measurement was used in all modeling software except for Zbrush, which does not have a standard system of measurement. The decision to use the metric system was based on three factors: 1) The surgical hardware used in this interactive surgical training tool was measured in millimeters (mm), 2) metric units of measurement can be easily converted by dividing or multiplying by factors of 10, and 3) all assets were designed to be imported into Unity, which uses meters as the default measurement unit. Once assets were brought into Unity their true measurement values were represented in meters. Thus, the unit of measurement was set to centimeters (cm) for Adobe Illustrator, Adobe Photoshop, and Maxon Cinema, and assets were set to their appropriate values in cm.

Creating the User Interface (UI) Elements

Both Adobe Illustrator and Adobe Photoshop were used to create assets for the UI elements. These assets were 2D images intended to be used as parts of the UI and included button icons, a scrollbar, rule lines, and window and UI borders. These were created in a light grey color to limit the adjustability to two color properties, value and saturation, once brought into Unity. Assets solely created using Adobe Photoshop were immediately saved as .png files; however, assets created using Adobe Illustrator had to be first saved as .ai files, then opened in Adobe Photoshop and saved as .png files. These assets were used as design elements for all menu windows, buttons, and scene borders.

Creating the Virtual Bone Models

DATA ACQUISITION

The Computer Aided Medical Procedures laboratory (CAMP) provided an .obj file of a segmented Acetabular Associated Anterior Column Posterior Hemitransverse fracture. The selected dataset was an anonymized CT scan provided by CAMP. Although this fracture type was ideal for the simulation, the resolution of the segmented file was low and therefore needed to be increased. To accomplish this, a set of DICOM-derived high-resolution skeletal models, in the form of .obj files, were obtained from the U.S.

National Institutes of Health National Library of Medicine Lister Hill National Center for Biomedical Communications (NIH-NLM-LHNCBC). These models were part of NLM3D, an Open-Source Library of Medical-Imaging-derived 3D Polygon Models for use in Public Health, Medical and STEM Applications. Figure 3 shows a side by side comparison of the models at the pelvic region. Once the models were obtained a high-resolution model of the selected fracture type was created. In total 10 .obj files were received from NIH-NLM-LHNCBC. These included vertebrae L1 through L5, sacrum, right and left femur, and right and left hip bone.

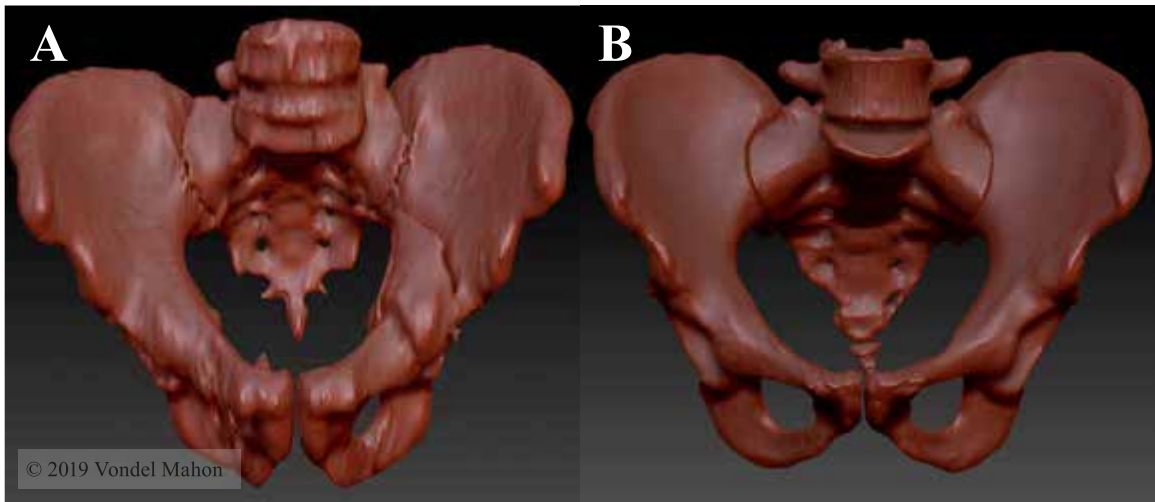


Figure 3. Bone models retrieved. (A) CAMP model, (B) NIH-NLM-LHNCBC model.

FRACTURE CREATION

The obj. file received from CAMP was imported into ZBrush, as a “subtool”, along with the high-resolution obj. files received from NIH-NLM-LHNCBC. These were brought into one Zbrush “tool” by using the “append” function. In Zbrush, a “tool” is a palette (repository) that holds 3D models, and each 3D model within that “tool” is classified as a “subtool”. Since more than one “tool” was used in the development process, the “tool” that was used to create the high-resolution model of the said fracture type will be referred to as the “main tool”.

Next, the high-resolution skeletal model was superimposed onto the low-resolution pelvis model (Figure 4). This was done by visually registering the Anterior Superior Iliac Spine (ASIS), Anterior Inferior Iliac Spine (AIIS), Symphysis Pubis, and Sacral Promontory of the low-resolution pelvis model to

those of the high-resolution pelvis model. Although the models were from different sources and therefore varied in size and detail, this method allowed the fracture type of the low-resolution pelvis model to be approximated and generally applied to the high-resolution pelvis model.

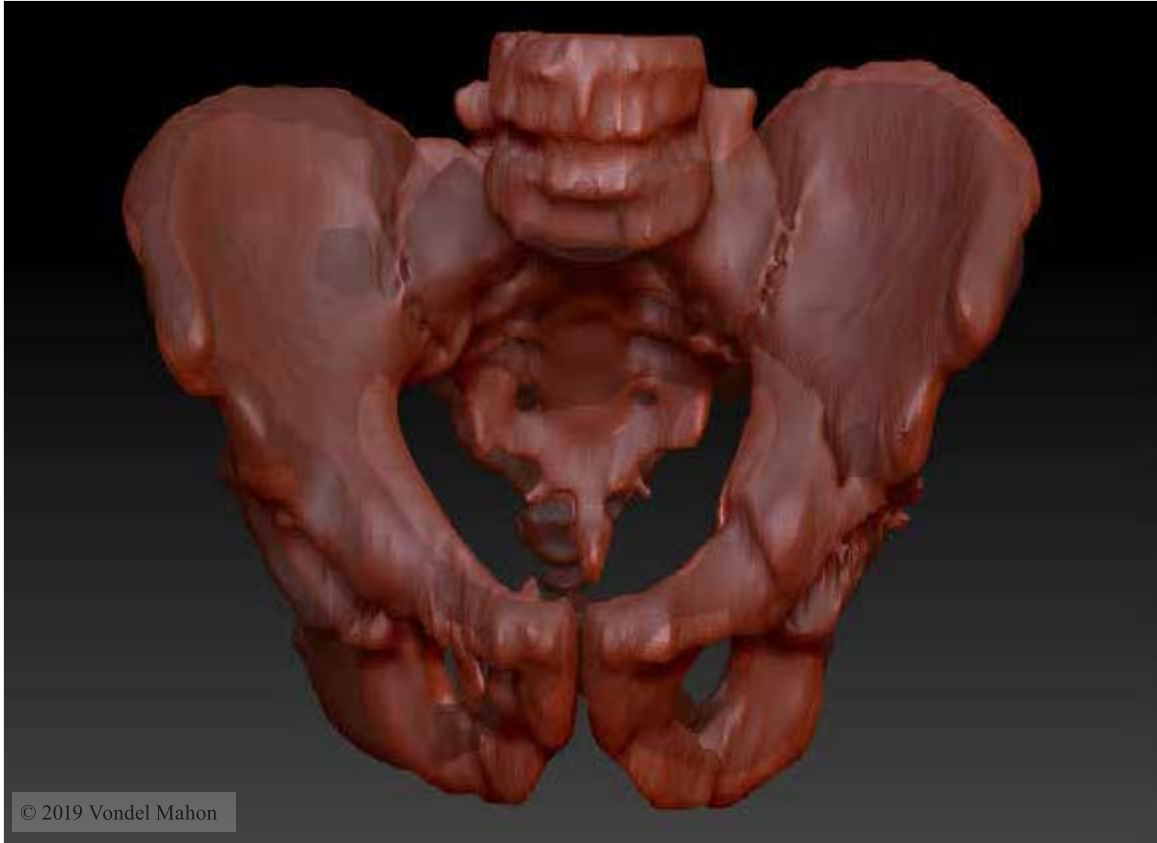


Figure 4. *Superimposition of the high and low resolution models.*

APPROXIMATION AND RECREATION OF THE FRACTURE

The process of recreating the fracture on the high-resolution pelvis model began with the selection of a Cube3D, which is a PolyMesh3D model composed of polygons. This was selected because Zbrush only allows sculpting on PolyMesh3D models. This Cube3D was then appended to the “main tool”. The position of the Cune3D “subtool” was then moved to the high acetabular fracture fragment’s geometrical center point, then sculpted to match the surface contour (topography) of the fractured sides only. This process created an impression of the high acetabular fracture on the Cube3D. In the process of sculpting the fractured sides, it was necessary to produce smooth edges where adjacent sides of the fracture

fragment met. If the edges were jagged and more true to the real fracture geometry, a substantial amount of volume would be lost during the “ZRemeshing” process, which is explained below.

Once the general surface contour was completed, the areas of exposed cortical and cancellous bone were then sculpted (Figure 5). Upon completion, the sculpted Cube3D model was moved to the corresponding location on the high-resolution model (Figure 6). Doing so allowed the surface contour of the high acetabular fracture fragment to be profiled on the high-resolution model. “Live Boolean” was then selected from the ZBrush interface to allow activation of the subtraction, addition or intersect operators, which can only be applied to “subtools”. Furthermore, the high-resolution model was selected as the “start group” by pressing the arrow button on its “subtool” icon. The sculpted Cube3D model was then placed under the “start group” and its boolean operator was set to “intersect.” Once this was done, “Make Boolean Mesh” was selected from the “subtool” menu. This created the first fracture fragment as a new “tool.” The new “tool” was then appended into the “main tool,” and the boolean was “subtracted” from the high-resolution pelvis model. This process was repeated until all fracture fragments were created, leaving behind the intact version of the left hip bone, which became a separate subtool. This process thus resulted in the creation of 5 models (Figure 7), resulting in 14 total high-resolution models.

The fracture fragments were created in this systematic fashion in order to retain the original volume of the high-resolution model (Figure 8) prior to “Zremeshing”. The “Zremeshing process” deducts volume,

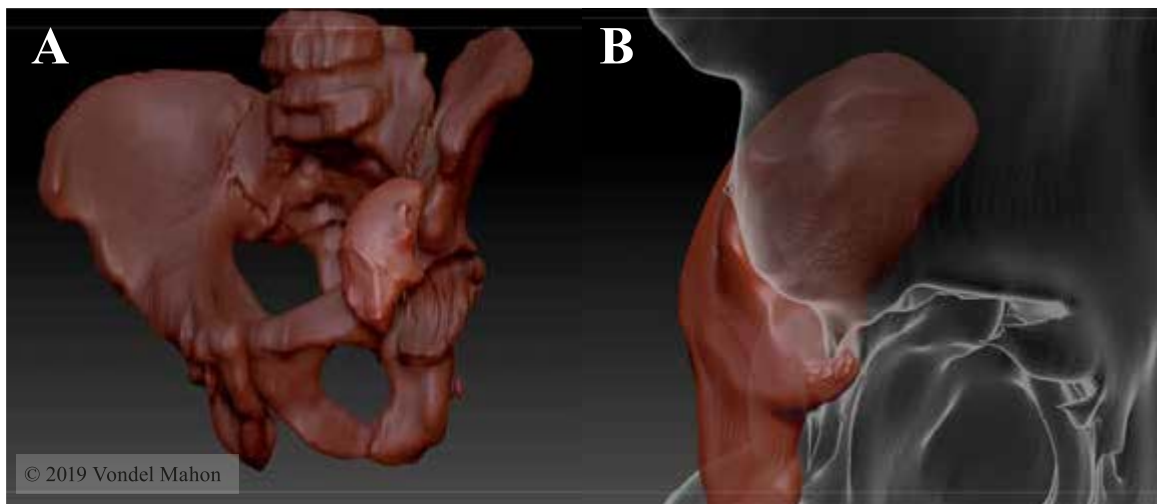


Figure 5. *Sculpted cortical and cancellous bone on fracture fragment. (A) Sculpted Cube3D model enveloping the fracture fragment. (B) Close-up of the exposed cancellous and cortical bone areas.*

so in keeping the original volume of the high-resolution model, the loss yielded would be insignificant. Although restoring the original volume of a fractured bone is not always possible in actual procedures due to debridement, it was decided that maintaining the appearance of the original volume better accomplishes the teaching goal of the interactive surgical simulator, namely to understand the spatial dynamics of the pelvis in the context of ORIF surgery.



Figure 6. *Cube3D transferred to high-resolution model.*

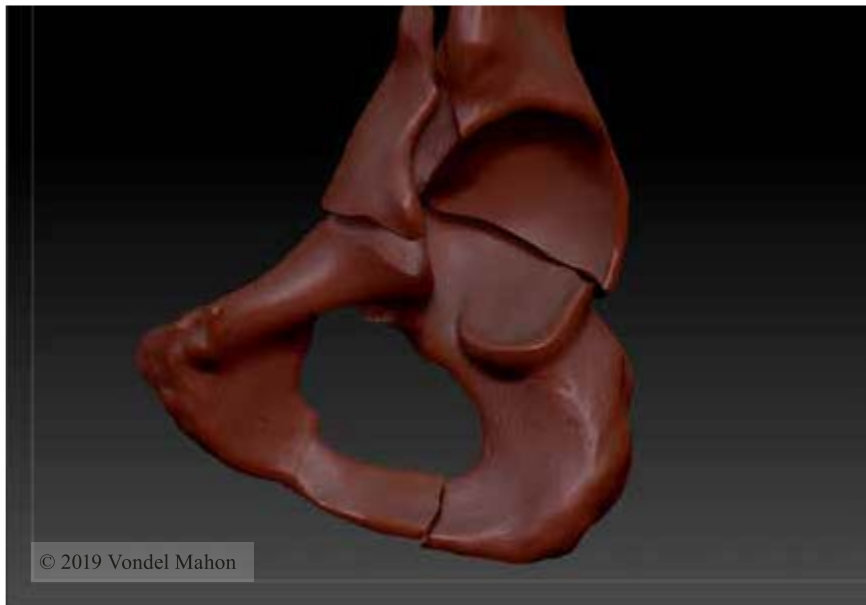


Figure 7. *Four fracture fragments created (models have been displaced for demonstration).*

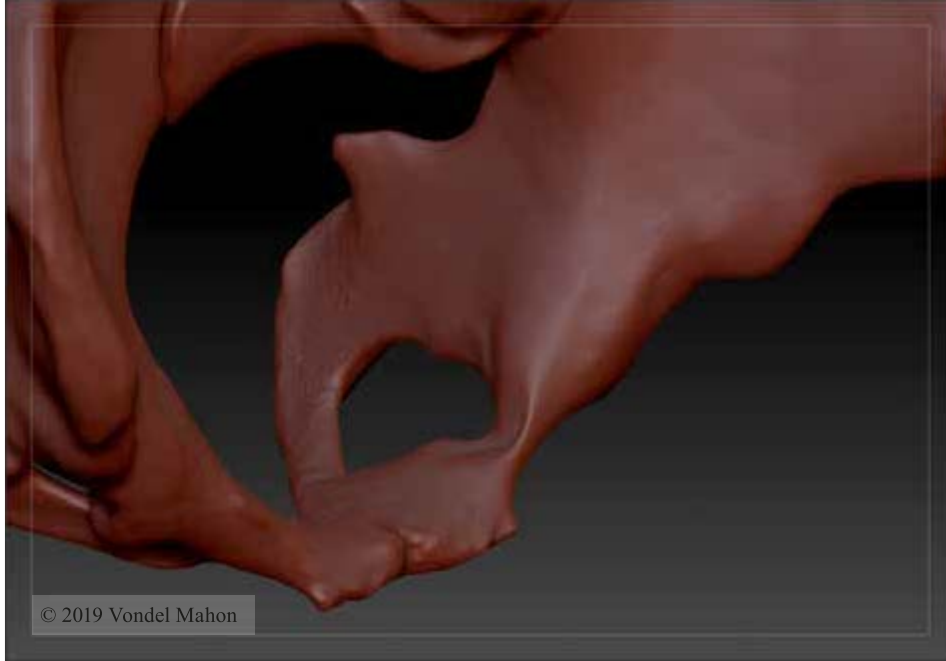


Figure 8. *Fractures made seamlessly. Fracture lines in mesh cannot be seen due to this method.*

CREATING THE LOW RES SKELETAL MODEL FROM THE HIGH RES MODEL

Once all fracture fragments were created on the high-resolution pelvis, the other components of the high-resolution dataset such as the vertebra, femurs, sacrum, etc., were duplicated and then “ZRemeshed.” In this process, a low-resolution version of each of the high-resolution models was created. This was done to limit the model’s computational size, thus optimizing performance of simulation on the mobile device.

After duplicating the high-resolution models, the following sequence of steps were applied to each: 1) The model was “ZRemeshed” with the threshold set between .5 to 1.0, 2) it was subdivided five times, 3) the duplicated model was placed beneath its high-resolution counterpart on the subtool palette, 4) the visibility of the duplicated model and its counterpart was turned on, and 5) “ProjectAll” was selected from the subtool menu.

Subdividing the models five times created five stored levels of detail, all of which are accessible using the “geometry” menu and could be exported as separate .obj files. Subdividing the model five times allowed a significant amount of the detail from the high-resolution model to be copied onto the new models. The “ProjectAll” command applied detail from the high-resolution models to the low resolution versions that were duplicated and “ZRemeshed” previously. Once the above steps were completed, the

original high-resolution models were no longer needed.

A UV map was then generated from the lowest-resolution state of each newly created model; UV maps are 2D image-representations of a 3D model's coordinates in 3D space. They were necessary for extracting and applying texture-map information to and from 3D models. Generating the UV map from the lowest-resolution state was done because this state of the model was intended to be exported out of Zbrush for use in Unity. Once the UV map was created, the models were reverted back to the highest-resolution state, then painted in color (Figure 9). The highest-resolution state was used for painting so that high-resolution color maps, referred to as texture maps in Zbrush, could be generated and applied to the low-resolution models in Unity. After the maps were generated they were exported and saved as .psd files, and the low-resolution states (Figure 10) of each newly created model were exported and saved as .obj files.



Figure 9. *Painted high-resolution model.*

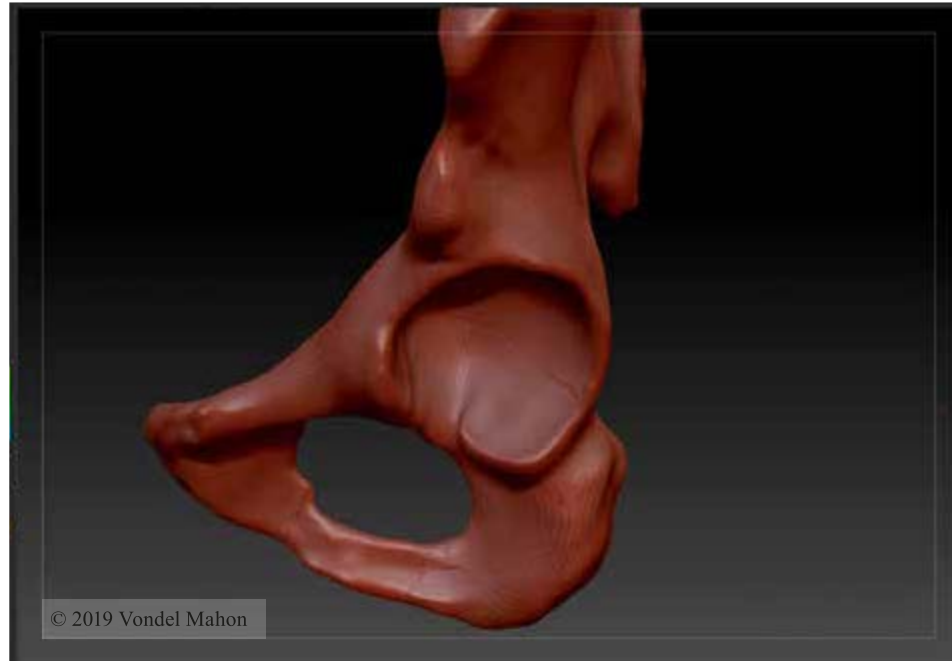


Figure 10. *Low resolution state. Fracture lines can be seen.*

Creating the Virtual Instrument Models

Instruments used by the Johns Hopkins Hospital Department of Surgery were used as reference to model the virtual instruments. They were created with accurate measurements so that once they were brought into the simulation, they would be at the correct relative scale to the skeletal model. Modeling of the surgical instruments and hardware was completed in Cinema 4D R20 and exported as .obj files.

Creating Plates

Plate creation was initiated in Adobe Illustrator. First, the unit of measurement was changed to centimeters (cm) to ensure accurate dimensions. Once this was done, the basic shape of the plate was created using the rectangle tool. The rectangle was then modified using the direct selection tool. Shapes for the flank invaginations were created using the ellipse tool and then merged together using the pathfinder “unite” operator to create a single flank invagination shape. The united flank invagination shape was then duplicated as needed and the resulting shapes were placed on the flank areas of the rectangle shape. Shapes for the screw holes were then created using the ellipse tool. Once the shape was established, the ellipses were also duplicated and placed in the proper locations on the rectangle shape. All flank invagination shapes and screw hole shapes were then grouped together and moved below the

rectangle shape using the layers menu. The rectangle shape and grouped shapes were selected together and the “minus back” operator on the pathfinder menu was activated to produce a 2D plate shape with holes. Finally, the file was saved as an Adobe Illustrator “Illustrator 8” file.

Once all 2D plate shapes were created in this fashion, the plates were “merged” (imported) into Cinema 4D, and the resulting spline was extruded using the “extrude” deformer to create the appropriate height of the plate. This process enabled the creation of a basic 3D plate shape. The exterior surface details of the plate were then modeled by first making the shape editable which allowed the surface mesh of the plate to be manipulated at any vertex point. To begin producing the complex geometry of the plate, the “caps” properties of the extrude deformer were used. The start and end caps were set to “Fillet Cap” and the radius of the start cap was set to 0.2 cm, while the radius for the end cap was set to 0.1cm. These settings were made to allow the front-facing side of the plate to be greater in thickness than the back-facing side. Once this was done the selection mode was set to “points,” and all front facing vertices were selected and scaled inward to emphasize the thickness of the front of the plate. The selection mode was then set to “loop selection” and the front-facing and back-facing regions of the screw holes were selected and scaled outwards to create the beveled appearance of the screw holes. The pivot/axis point was then placed at the center of each plate to allow a natural rotation of the models during the simulation (Figure 11).

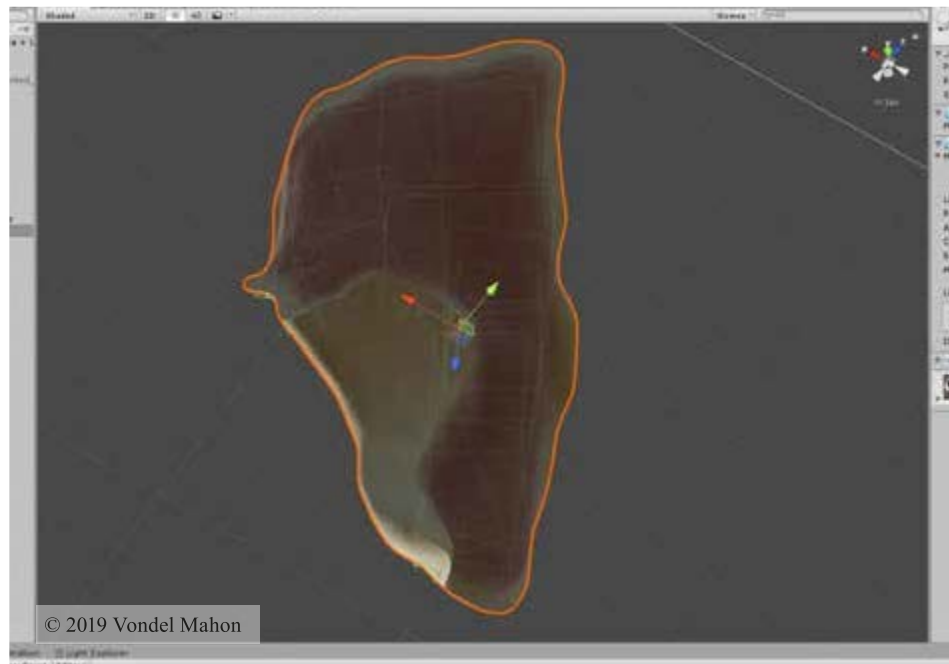


Figure 11. *Pivot/axis point centered. Text not intended to be read.*

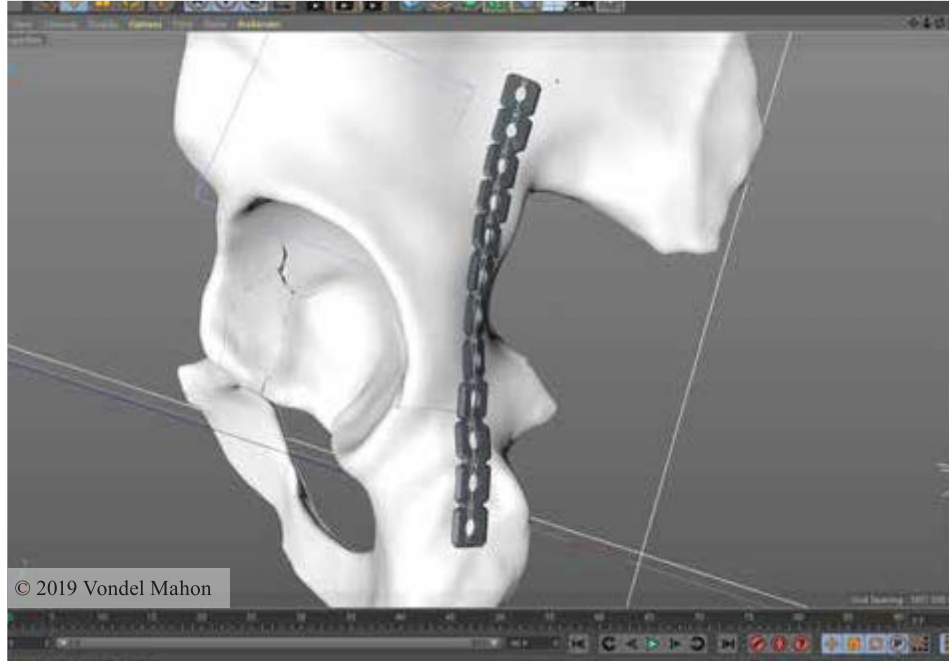


Figure 12. *Rigged plate. Text not intended to be read.*

Finally, the plates were rigged using the “joint tool,” which allowed them to be bent and contoured to the pelvis model (Figure 12). The location and final contoured shape of the plates were determined for the Acetabular Associated Anterior Column Posterior Hemitransverse fracture based on references and guidance received from Greg Osgood M.D.

CREATING SCREWS

Screws were created in Cinema 4D by using primitives such as a cylinder and a sphere to form the basic shape of the screw. To create the screw thread, a 4-side spline was swept along a helix spline using a “Sweep” generator, which converted the splines into a 3D shape. The object properties of the “Sweep” generator were edited based on the length and diameter of the screw. The pivot point was then transposed to the head of the screw so that during the simulation, this would serve as the center of rotation and users would be able to better orient the screw. Five screws of varying lengths were modelled using this process.

Creating the IOS application

IMPORT INTO UNITY

The models were imported into Unity. The axis/pivot points of the virtual instruments did not need to be modified since this was done during the modeling process. However, the axis/pivot points of the

skeletal models did need to be repositioned since it was not done previously in Zbrush. This was the case because all skeletal models were within one “tool” in ZBrush, and modifying their individual pivot/axis points at that time would have changed their relative positions to one another. To modify the axis/pivot point, an empty gameobject was created for each model, then placed at its center. Each model was then made a child of the empty gameobject, and was named appropriately.

UNITY PROJECT SET-UP AND UNITY REMOTE 5

A new project was created in Unity with build settings set to IOS, which allows the interactive surgical training tool to be built as an IOS application. Unity Remote 5 was installed and opened on the Apple iPad, which was tethered to the Personal Computer running Unity. Subsequently, the Unity project settings menu was opened, and the Unity Remote module was set to iPad Pro 12.9” 2nd gen Wi-Fi (2abf0e). This was done to allow testing of the application throughout the development process. Testing the application was done by selecting the play button on the Unity interface.

IMPORTING 3D MODELS

To import 3D models into Unity, a folder was created for each 3D model within the Unity assets folder. Then .obj files and all associated files generated during the modeling phase were placed into the folders. Once in the folders, Unity automatically created a material and associated texture maps for the models. In addition, Unity also automatically generated metafiles associated with each asset, which are used by Unity to properly refer to each asset and should not be deleted.

IMPORTING UI ELEMENTS

The UI elements which were saved as .png files were imported in the same fashion as the 3D models, except, all files were placed in a single folder in the assets folder. Once imported, the “texture type” of the .png files was changed to “sprite (2D and IU).” This converted the .png files into a “sprite,” allowing Unity to recognize them as 2D textures that can be edited as necessary using the Unity Sprite Editor.

SCENE DEVELOPMENT

After all assets were imported, two scenes were created. The first was named “Menu” and contained the menu system for accessing simulation cases. The second was named “case 001,” and was used to develop the Acetabular Associated Anterior Column Posterior Hemitransverse ORIF simulation. To develop these scenes, gameobjects were placed into the scene’s “hierarchy” panel.

The UI for both scenes were developed by first placing a UI canvas into the scene hierarchy. A UI canvas is a 2D transparent object to which UI elements and components can be added. The creation of the UI canvas automatically generated an “Event System” gameobject, which contained a script that controls all UI elements within the UI Canvas. A Camera was then brought into the scene and set as the “render Camera”. This allowed the entirety of the canvas to be seen at all times. Static parts of the UI were developed by creating UI images. The .png files which were imported into Unity were then designated as the “Source image” of those UI images. Menu windows were developed by creating “UI Panels”. Since the menu windows were intended to be interactive, “UI Buttons” were added to each UI panel to allow the user to navigate between other menu windows or panels. The UI panels were also made to animate between either an open state or a closed state based on UI button action. UI Buttons were created to allow navigation between panels, scenes, and windows. Buttons that effect windows alter the closed and open states of the window once the button is activated. Scripting, discussed in detail below, was used to control the function of the buttons and windows. Once all UI elements were generated, they were laid out on the canvas.

MENU SCENE DEVELOPMENT

The menu scene was designed to allow the user to quickly navigate to and fro particular cases of interest and make quick decisions based on the content of each case. It is simply a collection of UI panels designed to function as windows that store content. Each window is made up of sprites and UI buttons that either represent an extension of the content or provide an option to exit the window. Through the menu scene, the user is able to navigate to different simulations. However, only the Acetabular Associated Anterior Column Posterior Hemitransverse simulation was made available for selection in this iteration of the project. The reader can refer to above for a detailed description on how UI panels were designed.

SIMULATION SCENE DEVELOPMENT

After the UI was created for the simulation scene, the interactivity of the 3D models was programmed. First the imported skeletal models were added into the scene “hierarchy” panel from the assets folder. This enabled the models to be placed into the 3D space of the scene. Once in the scene, materials were automatically generated for each component of the model, and the color and normal maps were applied to the material. Due to the IOS build settings, Unity did not allow enhancements of normal

maps; therefore, the full details of the models could not be seen in the application. However, despite this, the quality of the models were sufficient due in large part to the color maps.

Once the models were added to the scene their axis/pivot points were set so that the models could later be moved and rotated effectively by the user. Particular attention was paid to estimating the object's center of mass so that user manipulation felt natural and realistic. This was done by creating an empty gameobject, placing it at the center of the model, then making the model a child of that object. This allowed the model to be manipulated using the coordinates and rotation of that gameobject. This was done for all skeletal models.

Once axis/pivot points were established, a 3D collider was added as a component of the parent object. The collider was then made larger than the mesh of each model and set to "is trigger". Making the collider larger than the mesh allowed the user to easily select the model during a simulation. In addition, setting the collider to "is trigger", allowed the virtual fracture fragments to receive touch input while not allowing it to exhibit physical reactions. When the collider is not set to "is trigger", an invisible force field is created around the mesh, based on the shape of the collider, which repels it from other colliders, thus separating all of the objects in the scene.

Next, 14 collections of gameobjects were made; one collection for each skeletal model. Each collection was then made the child of the corresponding skeletal model. A 3D collider component was added to each gameobject, and then the colliders were modified to match the internal contour of each skeletal model (Figure 13). This was done to all the virtual fracture fragments so that realistic physics would be experienced by the user during the simulation.

To enable realistic dynamics, "rigidbody" components were added to the parent gameobject of each skeletal model. Furthermore, the "rigidbody's" mass, drag and angular drag values were set to one million each to prevent the virtual fracture fragments and instruments from floating in the virtual environment during the simulation. In addition, the "use gravity" and "is kinematic" parameters were set to "false" to help further stabilize the elements within the virtual environment; these settings are shown in figure 14.

Lastly, another gameobject was created and centered on the entire skeletal model, and was made the parent of all the individual component models. The main gameobject was named "skeleton". This hierarchy allowed the entire model to be later targeted by the main camera using scripts.

A camera was created to serve as the scene's main camera of the simulation. Because it served this

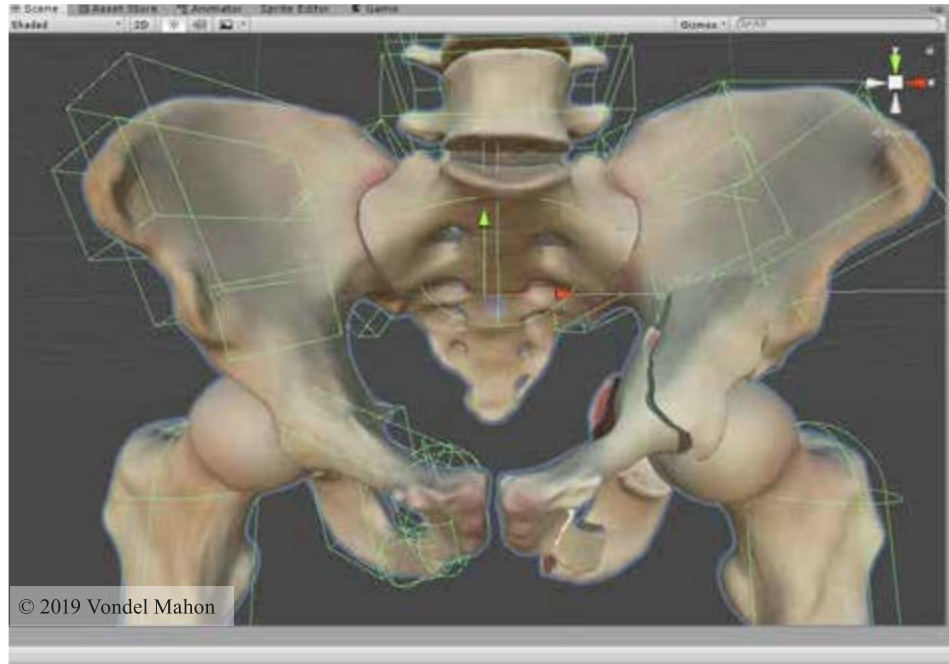


Figure 13. *Collider modifications. Text not intended to be read.*

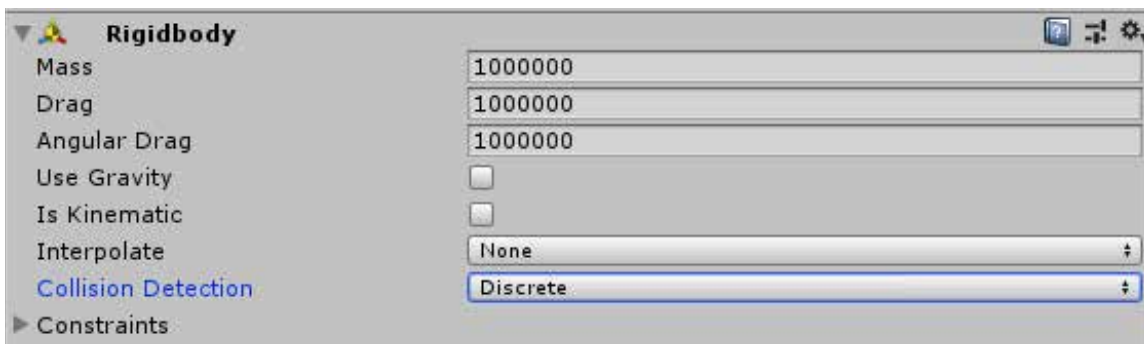


Figure 14. *Rigid body parameter settings.*

function it was tagged as the “main camera” on the inspector panel. The camera was then repositioned in the 3D scene to establish the initial view the user encounters when starting the simulation. The “3D orbit Fracture Selection” and “Pan Rotate” scripts were then added as components of the main camera. These scripts are discussed below.

Script Functions and Usage

Scripting assets were retrieved from the Unity asset store and customized to produce the intended actions for the interactive learning simulation. Moreover, all coding was done in C# programming language using VisualStudio 2017. Once installed, VisualStudio can be accessed through Unity by

double clicking on any script. All scripts were then placed as a component of a gameobject to access their functions.

Touch Gestures

Touch gestures were created using the Fingers asset package retrieved from the Unity asset store. This package provided a suite of six gesture types; tap, pan, swipe, scale, rotate, and longPress. In addition, the asset package allowed any of the touch gestures to be assigned other functions as well such as select, target, spawn, destroy and reset. In this iterations of the project, the pan, rotate, and tap gestures were used. The application of these gestures can be found in the scripts below.

3D Orbit Fracture Selection (Appendix A) and Pan Orbit Scripts (Appendix B)

The 3D Orbit Fracture Selection and Pan Orbit scripts operate as one unit and have four primary functions: 1) they allow the user to optically rotate the entire skeletal model in 360 degrees, by swiping his/her finger across the touch screen surface; 2) they allow the rotate and zoom functions to be disabled via a UI button permitting the user to move the virtual instruments and fracture fragments in 3D space without interruption from unwanted camera movements; 3) they allow the virtual fracture fragments to be selected and rotated and moved individually; 4) they allow the user to go back and forth from skeleton selection mode to fracture fragment selection mode.

To apply these functions, variables and tap gestures were first created within the 3D Orbit Fracture Selection script. The variables allowed each virtual fracture fragment to be uniquely identified, while the gestures allowed the virtual fracture fragments to be selected and manipulated. Once these were coded, empty gameobject fields on the camera's inspector panel were automatically generated for each virtual fracture fragment model. In addition, an empty gameobject field was also generated for the skeleton as well (Figure 15). The virtual fracture fragments and skeleton gameobjects were then dragged into the designated fields allowing the script to identify which object is selected when the tap gesture is executed. Once a tap gesture is executed on a virtual fracture fragment, the Pan Orbit script is activated and changes the target of the main camera from the current target to that object.

During the simulation, the skeleton is set as the default orbit target, until a tap gesture is executed on an individual virtual fracture fragment. This allows the user to freely inspect the entire skeleton, prior to engaging in a reduction exercise. When the user selects a virtual fracture fragment, the main camera

targets it. However the user also has the option of double tapping to revert back to the default gameobject and again focus on the entire skeleton.

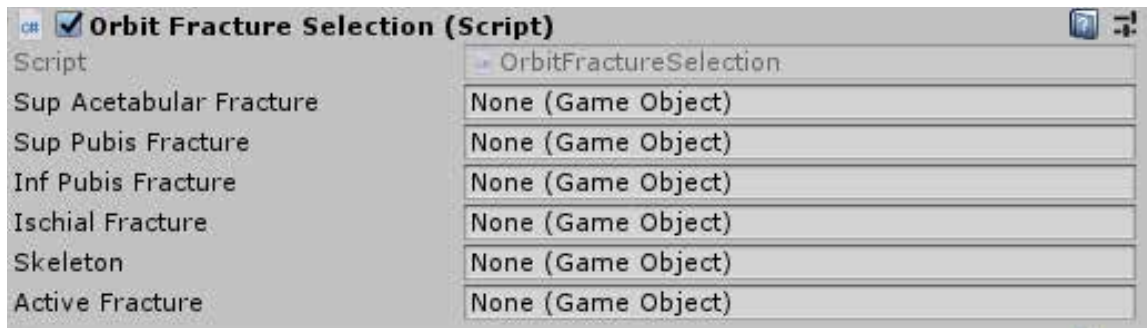


Figure 15. Empty fields on main camera's inspector panel.

Pan Rotate (Appendix C)

The Pan Rotate script allowed virtual fracture fragments and virtual instruments to be manually manipulated during the simulation. A one-finger gesture allows the user to pan and change the position of the models in 3D space, while a two-finger gesture permits rotation of the models. This allows the user to reduce and fix the virtual fracture fragments during the simulation. Both manipulations were applied with respect to the plane of the main camera. To activate the function of this script it was added as a component to each virtual fracture fragment and surgical instrument.

Window Manager Script (Appendix D)

The Window Manager script manages the way UI panels function and was utilized on all windows in this application. In this script open and closed states of the UI windows were described as C# functions. Also described was the visual appearance of each state using the Unity animator and animation panels; animation states were created using the animation panel, and the animation controller, which describes how the animations will transition, was made using the animator panel. Once these descriptions were made, the states could then be recruited through the use of UI buttons.

First, an empty gameobject was created and placed on the scene hierarchy panel. Then the Window Manager script was added to it as a component. This makes the C# functions of the script accessible to any UI button. This empty gameobject was then named "Window Manager". In order for the UI window to be animated when the user presses a UI button, an animator component was added to the button, then the

animator controller was added as the controller of the animation. With that done, the “Window Manager” gameobject was designated as the event handler of the button thus allowing any UI button to now access both the open and closed animation states.

Snap in Place (Appendix E) and Lerp Script (Appendix F)

The Snap in Place and Lerp scripts function as one unit, thus they will be described together. After coding the functionality, both scripts were added as components of the fragments. These scripts allowed the virtual fracture fragments to “snap” into place once the user transforms the position of the virtual fracture fragment near its anatomically correct position.

The snap script monitors the position of an object and is triggered in this case when a fragment is within 5 mm of its normal position. Once triggered, the lerp script is activated which then moves the virtual fracture fragment into anatomical position. Once lerp occurs, the virtual fracture fragment is locked into that position, and can no longer be manipulated. The 5 mm value was determined to balance the degree of struggle verse the preciseness of reduction. Based on trials performed by the author, it was observed that users would become frustrated if the tolerance was set too low and required overly precise or “air-tight” reductions. This would be detrimental to the program’s effectiveness and deviate from the learning objectives of the interactive surgical training tool application. To use this function, the normal, non-fractured version of the pelvis was brought into Unity, and the transpose positions of each virtual fracture fragment model was used as a reference for both the snap and lerp scripts.

Spawn Object Script (Appendix G)

The Spawn Object script allows objects to appear on command; it was used to allow the user to populate the scene with virtual instruments. An empty gameobject was created and named “Spawn Manager” and the script was added as a component of it. Within the script, public arrays were created to allow multiple empty fields to be displayed on the inspector panel of the “Spawn Manager” gameobject. The arrays were for spawn locations, the prefab which would be spawned (discussed below), and the clone which would appear in the simulation scene. A total of eight fields for each array were created. Eight empty gameobjects were then created as the spawn locations and spaced 2 cm apart on the x-axis to avoid overlap.

The instrument models which were previously imported into Unity, were dragged from the assets folder into the scene hierarchy panel, then dragged back into the assets folder. This action created prefabricated versions (a.k.a prefabs) of those models within the assets folder. The prefabs were then dragged from the assets folder into the empty array fields designated for the prefabricated models, and the instrument models in the scene hierarchy panel were then deleted. In addition, these same prefabs were dragged into the empty array fields designated for the clone models.

To spawn the virtual instruments, a function was made for each virtual instrument within the spawn object script. Then a UI button was created for each virtual instrument and programmed to spawn that virtual instrument when pressed.

Destroy All Tagged Script (Appendix H)

This script was created to remove all virtual plates or virtual all screws from the simulation scene. Each prefabricated virtual instrument was given a tag. Screws were tagged as “screw” and plates were tagged as “plate”. This allowed discrimination when deleting either of the gameobjects. A UI button was created to remove all gameobjects with the tag “plate”. Another UI button was created to remove/destroy all gameobjects with the tag “screw”. These buttons were placed as icons on the control panel of the simulation and are easily accessible to the user permitting them to remove the hardware and restart the fixation exercise.

OR Timer Script (Appendix I)

The OR Timer script was created to time simulations and provide user feedback and performance data on how long it takes them to complete a case. The timer is placed discreetly on the UI to not distract the user during the simulation.

To run the timer a UI text gameobject was created on the scene hierarchy and the script was added as a component. In the text field on the UI text gameobject’s inspector panel, 00:00:00 was entered. The first two digits represent hours, the second represents minutes, and the last two digits represent seconds. As soon as the user presses the “begin case” button, the script initiates the timer.

Enable Disable Object Script (Appendix J)

The Enable Disable script was created to hide the instructions presented to the user at the beginning of the simulation. The script was applied to the “begin case” button, so that once the user chooses to start the case, the script will remove the instructions from view.

Change Material Script (Appendix K)

The Change Material script allows the user to see the virtual bone model in an x-ray material. This function is accessible to the user at any time during the simulation to provide real-time positional feedback and promote an understanding of the anatomy and injury in 3D space. Having this function during the simulation will allow the user to examine screw trajectories in 3D within the virtual skeleton and make adjustments as needed.

Once the script was coded, it was added to all bone models on the gameobject which carries the mesh render component. Two empty fields were then populated on the inspector panel of that gameobject, for normal bone color material, and for the x-ray material. A UI button was then created to toggle between the two materials.

Load Scene Script (Appendix L)

The Load Scene script allows navigation between different scenes. Only two scenes were created in the IOS application. The first carries the main menu system and case selection, the second carries the reduction fixation simulation. Once scenes were created, they were then added to the project build settings, and assigned a specific whole number, beginning at zero allowing them to be identified and targeted through button action. The first scene was assigned zero, and the second scene was assigned one. The button which will allow the user to navigate to the first scene was placed in the second scene, in the form of the “home” icon. The button which will allow the user to navigate to the second scene was placed in the first scene, in the form of the “begin” button, which is located on the case 0001’s UI.

RESULTS AND DISCUSSION

The primary function of this interactive surgical training tool is to address the spatial challenges orthopedic residents face when approaching ORIF procedures to treat an unstable pelvis. In this iteration of the project, a simulation of an ORIF procedure to reduce and fixate an Acetabular Associated Anterior Column Posterior Hemitransverse fracture was developed as a proof of concept. In addition, a case selection menu system was developed to allow access to this simulation and others that will be developed in the future. In this process functional segments and non-functional segments were created.

The cognitive load was an important consideration when developing the interactive surgical training tool. Thus, only essential content was presented to the user in the case selection menu system. This would allow the user to spend less time and effort choosing a case, and more time practicing the simulation.

The visual aesthetics were also considered. The user interface (UI) was designed to complement commonly seen interfaces in the operating room. This allows for a level of familiarity for the user and decreases the learning curve. The UI was also designed with a low saturation grey to allow the user to focus on the task at hand and not be distracted by UI elements. In addition, to emphasize the concept of space, 2D UI windows were made to move in 3D space so that the user unknowingly become more familiar with that concept.

Case Selection Menu System

At the beginning of the application, the user will encounter the first scene which first displays the Main Menu window (Figure 16). This first window also provides the user with the options of changing the settings, exiting the application, or navigating to skeletal regions. These items are in the form of buttons and are described as follows:

- Skeletal regions - This menu item allows the user to venture into the surgically relevant skeletal regions of the body.
- Settings - This is a non-functional feature and will be available in future iterations.
- Exit - Exit allows the user to exit the application.

Once the user selects the skeletal regions, a Skeletal Regions window will become visible allowing the user to choose from 13 major surgically relevant skeletal regions (Figure 17); in this iteration, only the pelvis can be explored. Once the user selects the pelvis, a Pelvis window appears giving users access to



Figure 16. *Main Menu Window.*



Figure 18. *Pelvis Window.*



Figure 17. *Skeletal Regions Window.*

surgically significant regions of the pelvis; in this iteration only the Acetabulum can be explored (Figure 18).

Once the user selects the Acetabulum, a new window will be displayed, containing a variety of fracture cases that occur at the acetabulum. The Acetabular Associated Anterior Column Posterior Hemitransverse fracture case will be available for selection. Though the Acetabular Associated Anterior Column Posterior Hemitransverse fracture represents one of 10 acetabular fracture types, which is classified based on the Letournel classification system, future iterations of this interactive surgical training tool will include all other acetabular fracture types. The interface of this window was designed to host a growing number of cases that the user can scroll through; figure 19 shows how these are laid out.

The descriptions of the cases were kept to a minimum to make browsing easier. This design allows the user to quickly navigate through multiple cases and make a selection based on four criteria: 1) the fracture type, 2) the age and sex, 3) a short clinical description of the injury, and 4) the score received after completing the simulation (will be available in future iterations) In addition to these selection criteria,

each case was assigned a case number that the user can refer to when revisiting that case; in this iteration of the interactive surgical training tool, only the Acetabular Associated Anterior Column Posterior Hemitransverse fracture is available as a case selection.

Once the user selects the fracture type, which in this case is labeled as case 0001, another window will emerge containing a detailed description describing the cause, symptoms, nature of the injury, and observations derived from medical examinations (Figure 20). These details will allow the user to have an informed approach when performing the case. Once the user decides to advance, he/she can select the begin button which initiates the simulation.

Simulation Scene

Upon beginning the simulation the user is provided with instructions (Figure 21) that explain the



Figure 19. Case Selection

touch gestures controls used to manipulate the virtual instruments and fractures and the functions of each UI element. Once the user finishes reading the instructions, the begin button should be pressed and the timer will initiate. Recording the simulated operating time allows the user to monitor and track the time it takes to perform the virtual procedure or different tasks. The user can then explore the virtual skeletal anatomy in 360° and fully appreciate the gross anatomy and the geometry of the pelvis and all associated fractures in 3D space.



Figure 20. Case 0001.

When the user decides to focus on a particular fracture, the line-of-sight is changed by switching the camera's target to that fracture. Then the user will be able to reduce the fracture or move the fragments into natural anatomical position using touch gestures. Once the user reduces the fractures within a clinically acceptable range (one that will allow proper bone healing), the fracture will "lock" into place indicating that the fracture has been successfully reduced and prepared for fixation (Virkus & Marota, n.d.). The user will also have the option to experience realistic fracture interactions while performing reductions; this is



Figure 21. *UI instruction panel.*

due to the use of rigid bodies and colliders described in the Materials and Methods section of this thesis. Once reductions are complete, the user will manually perform fixations using a suite of virtual surgical hardware lag screws and plates of different specifications. The user will do so by selecting the instrument icon located on the user interface (UI), which opens a UI Instrument Selection window (Figure 22). Once instruments are selected, they are spawned into view, and they can be used to perform fixation of the pelvis. Once this is completed the user can generate volumetric x-rays to inspect the results. The ability to see the results as an x-ray is a key feature of this program because it will reinforce the users understanding of the anatomy and reduction and fixation process. The x-rays are only applied to the virtual bones (Figure 23), leaving the virtual hardware opaque, so that the user can effectively inspect screw trajectories, plate placements, and make necessary modifications.

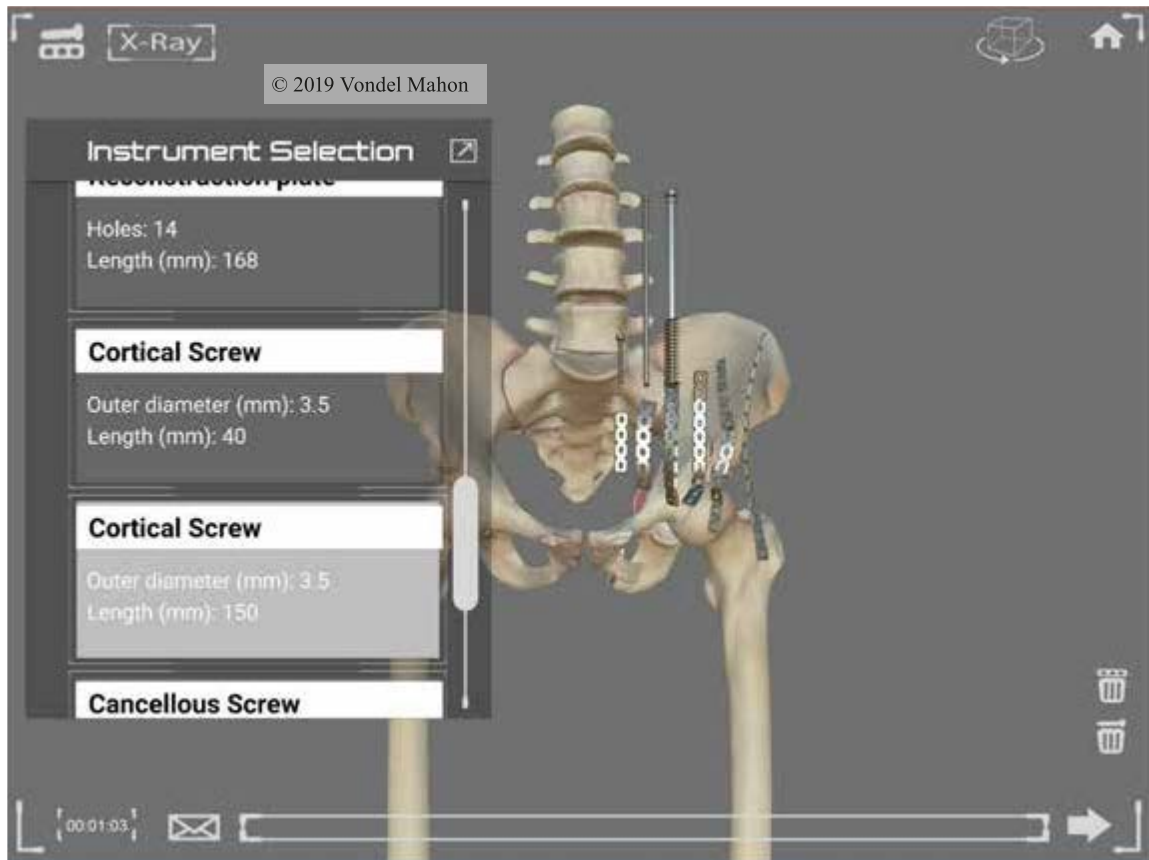


Figure 22. *Instrument Selection Window.*

Conclusion

Open Reduction Internal Fixation (ORIF) of unstable pelvic fractures is surgically complex and requires the work of highly skilled and experienced orthopedic surgeons. To develop the appropriate skills, surgeons complete five years of intense orthopaedic training during a residency program. The expectation of residency training is that the skills, competencies and experience of the resident-surgeon are honed to a level needed to perform complex surgeries. However, the training system currently used in orthopaedic residency programs is outdated and often does not provide sufficient opportunities needed to master complex skills -- particularly those that require surgeons to translate 2D information to a 3D context. Residents often require further training to perform ORIF procedures, not having the right skills and training can ultimately place the lives of patients in danger.

By creating a modern, convenient, and accessible interactive training device that will allow orthopaedic residents to understand spatial dynamics in the context of ORIF procedures, have unlimited

access to orthopaedic cases, and perform procedures in virtual space, orthopaedic residents can gain the skills, competencies, and experience than enable them to preserve lives.

Future Research on the Interactive Surgical Training Tool

The research conducted in the project is a work in progress. It is part of a larger study by participants within the Johns Hopkins University Department School of Medicine and the Johns Hopkins University School of Engineering. Based on the statement of the problem, and the purpose of the research, an interactive surgical training application was created. This application is scheduled to be tested in the latter part of 2019. It will be tested on 81 participants including orthopedic surgery residents and fellows

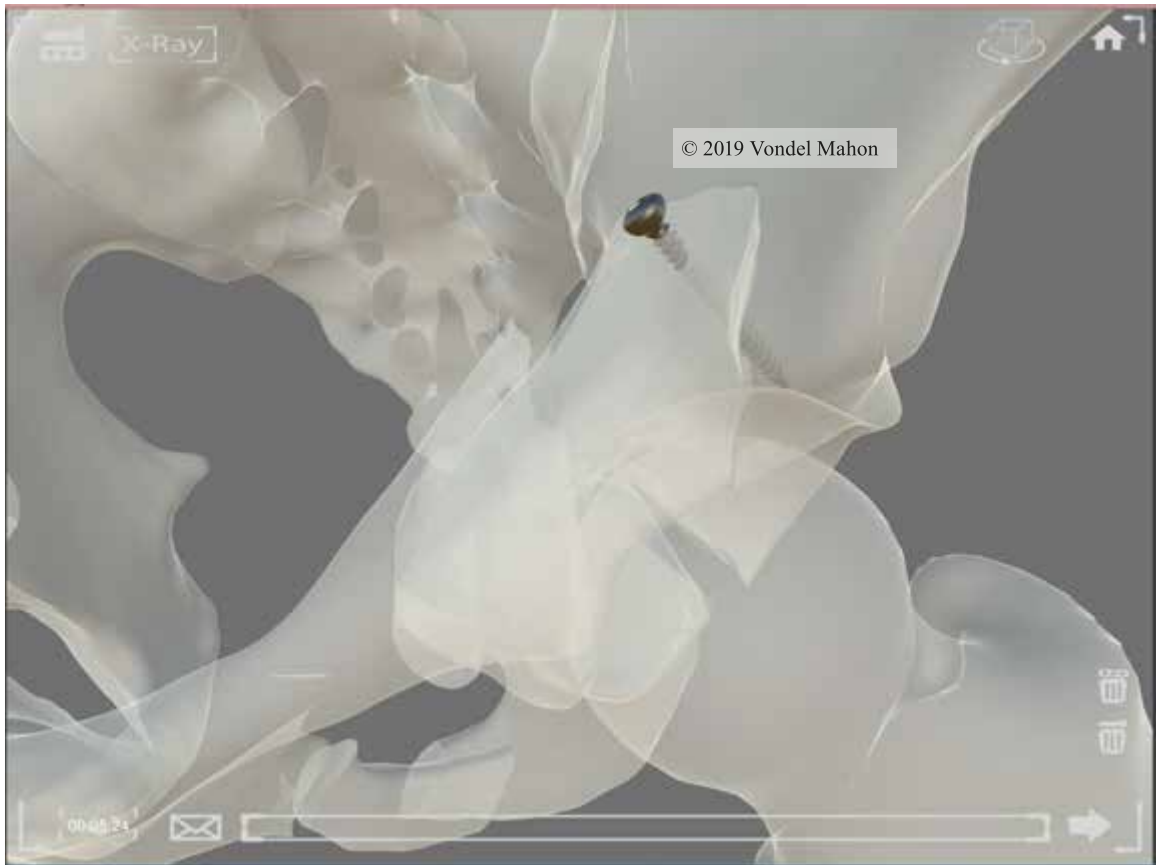


Figure 23. *X-ray only applied to the bony model, allowing hardware and anatomy to be evaluated. Text not intended to be read.*

attached to The Johns Hopkins University School of Medicine. The application has the potential to save lives since participants will be gaining more skills, knowledge, and experiences by interactive procedures as if in real time. All of these added skills, knowledge, and experiences can be easily transferred into the operating room during traumatic pelvic surgery. The application has the potential to be used at any orthopaedic residency globally once it is fully tested and developed. It can be implemented in residency programs and medical schools to give students a jump-start on traumatic pelvic surgery even before they set foot in the operating room.

After the initially testing of the app, it is recommended that further testing be conducted at other residency programs at other universities. Additionally, it is recommended that the application be tested on veteran surgeons as well so that they too can profit from this new approach as well as provide valuable feedback. The application should be further developed to enhance its effectiveness then implemented. If it is not implemented, residents will continue to use the same traditional methods and still have spatial challenges, in the context of ORIF of the pelvis. They will continue to make errors and lives could be lost due to deficiencies in skill, experience, and proper training.

Access to Assets Resulting from this thesis

Images and assets resulting from this thesis can be partially found at www.mahonillustrations.com. The author may also be reached through the Johns Hopkins University School of Medicine Department of Art as Applied to Medicine at medicalart.johnshopkins.edu.

APPENDICES

Appendix A: 3D Orbit Fracture Selection

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace DigitalRubyShared
{
    public class OrbitFractureSelection : MonoBehaviour
    {
        public GameObject SupAcetabularFracture;
        public GameObject SupPubisFracture;
        public GameObject InfPubisFracture;
        public GameObject IschialFracture;
        public GameObject Skeleton;
        public GameObject ActiveFracture;
        private PanOrbit orbit;

        private void Start()
        {
            ActiveFracture.GetComponent<PanRotate>().enabled = true;
            orbit = GetComponent<PanOrbit>();
            orbit.OrbitTargetTapped += Orbit_OrbitTargetTapped;
            TapGestureRecognizer doubleTapGesture = new TapGestureRecognizer();
            doubleTapGesture.NumberOfTapsRequired = 2;
            doubleTapGesture.StateUpdated += DoubleTapGesture;
            FingersScript.Instance.AddGesture(doubleTapGesture);
            TapGestureRecognizer tap = new TapGestureRecognizer();
            tap.StateUpdated += Tap_Updated_Fracture1;
            tap.PlatformSpecificView = SupAcetabularFracture;
            tap.RequireGestureRecognizerToFail = doubleTapGesture;
            FingersScript.Instance.AddGesture(tap);
            TapGestureRecognizer tap2 = new TapGestureRecognizer();
            tap2.StateUpdated += Tap_Updated_Fracture2;
            tap2.PlatformSpecificView = SupPubisFracture;
            tap2.RequireGestureRecognizerToFail = doubleTapGesture;
            FingersScript.Instance.AddGesture(tap2);
            TapGestureRecognizer tap3 = new TapGestureRecognizer();
            tap3.StateUpdated += Tap_Updated_Fracture3;
            tap3.PlatformSpecificView = InfPubisFracture;
            tap3.RequireGestureRecognizerToFail = doubleTapGesture;
            FingersScript.Instance.AddGesture(tap3);
            TapGestureRecognizer tap4 = new TapGestureRecognizer();
            tap4.StateUpdated += Tap_Updated_Fracture4;
            tap4.PlatformSpecificView = IschialFracture;
            tap4.RequireGestureRecognizerToFail = doubleTapGesture;
            FingersScript.Instance.AddGesture(tap4);
        }

        private void Tap_Updated_Fracture1(DigitalRubyShared.GestureRecognizer gesture)
        {
```

```

        if (gesture.State == GestureRecognizerState.Ended)
        {
            ActiveFracture.GetComponent<PanRotate>().enabled = false;
            ActiveFracture = SupAcetabularFracture;
            orbit.OrbitTarget = ActiveFracture.transform;
            orbit.Orbiter.transform.LookAt(orbit.OrbitTarget.transform);
            Debug.LogFormat("Tap gesture executed on Sup Acetabular Fracture at {0},{1}", gesture.
FocusX, gesture.FocusY);
        }
    }

    private void Tap_Updated_Fracture2(DigitalRubyShared.GestureRecognizer gesture)
    {
        if (gesture.State == GestureRecognizerState.Ended)
        {
            ActiveFracture.GetComponent<PanRotate>().enabled = false;
            ActiveFracture = SupPubisFracture;
            orbit.OrbitTarget = ActiveFracture.transform;
            orbit.Orbiter.transform.LookAt(orbit.OrbitTarget.transform);
            Debug.LogFormat("Tap gesture executed on Sup Pubis Fracture at {0},{1}", gesture.FocusX,
gesture.FocusY);
        }
    }

    private void Tap_Updated_Fracture3(DigitalRubyShared.GestureRecognizer gesture)
    {
        if (gesture.State == GestureRecognizerState.Ended)
        {
            ActiveFracture.GetComponent<PanRotate>().enabled = false;
            ActiveFracture = InfPubisFracture;
            orbit.OrbitTarget = ActiveFracture.transform;
            orbit.Orbiter.transform.LookAt(orbit.OrbitTarget.transform);
            Debug.LogFormat("Tap gesture executed on Inf Pubis Fracture at {0},{1}", gesture.FocusX,
gesture.FocusY);
        }
    }

    private void Tap_Updated_Fracture4(DigitalRubyShared.GestureRecognizer gesture)
    {
        if (gesture.State == GestureRecognizerState.Ended)
        {
            ActiveFracture.GetComponent<PanRotate>().enabled = false;
            ActiveFracture = IschialFracture;
            orbit.OrbitTarget = ActiveFracture.transform;
            orbit.Orbiter.transform.LookAt(orbit.OrbitTarget.transform);
            Debug.LogFormat("Tap gesture executed on Ischial Fracture at {0},{1}", gesture.FocusX,
gesture.FocusY);
        }
    }

    private void DoubleTapGesture(DigitalRubyShared.GestureRecognizer gesture)
    {
        if (gesture.State == GestureRecognizerState.Ended)
        {
            ActiveFracture.GetComponent<PanRotate>().enabled = false;
            ActiveFracture = null;            orbit.OrbitTarget = Skeleton.transform;
            orbit.Orbiter.transform.LookAt(orbit.OrbitTarget.transform);
        }
    }

```

```

        this.GetComponent<PanOrbit>().enabled = true;
        Debug.LogFormat("Double tap gesture executed on not a fracture at {0},{1}", gesture.FocusX,
gesture.FocusY);
    }
}

public void ToggleOrbitAndZoom()
{
    this.GetComponent<PanOrbit>().enabled = !this.GetComponent<PanOrbit>().enabled;

    if (this.GetComponent<PanOrbit>().enabled == true)
    {
        ActiveFracture.GetComponent<PanRotate>().enabled = false;
    } else
    {
        ActiveFracture.GetComponent<PanRotate>().enabled = true;
    }
}

private void Orbit_OrbitTargetTapped()
{
    Debug.Log("Skeleton tapped");
}

private void Update()
{
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            UnityEngine.SceneManagement.SceneManager.LoadScene(0);
        }
    }
}
}

```

Appendix B: Pan Orbit

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace DigitalRubyShared
{
    /// <summary>
    /// Allows orbiting a target using a pan gesture to drag up and down or left and right
    /// to orbit, and pinch to zoom in and out
    /// </summary>
    [AddComponentMenu("Fingers Gestures/Component/Pan Orbit", 2)]
    public class PanOrbit : MonoBehaviour
    {
        [Tooltip("The transform to orbit around.")]
        public Transform OrbitTarget;
        [Tooltip("The object to orbit around OrbitTarget.")]
        public Transform Orbiter;
        [Tooltip("The minimum distance to move to the orbit target, 0 for no minimum.")]
        [Range(0.1f, 100.0f)]
        public float MinimumDistance = 5.0f;
    }
}

```

```

[Tooltip("The maximum distance to move away from the orbit target, 0 for no maximum.")]
[Range(0.1f, 1000.0f)]
public float MaximumDistance = 1000.0f;
[Tooltip("The zoom speed")]
[Range(0.01f, 100.0f)]
public float ZoomSpeed = 20.0f;
[Tooltip("The speed at which the orbiter looks at the orbit target is it has panned away from looking
directly at the orbit target.")]
[Range(0.0f, 10.0f)]
public float ZoomLookAtSpeed = 1.0f;
[Tooltip("The threshold in units before zooming begins to happen. Start distance must change this
much in order to start the gesture.")]
public float ZoomThresholdUnits = 0.15f;
[Tooltip("The speed (degrees per second) at which to orbit using x delta pan gesture values. Negative
or positive values will cause orbit in the opposite direction.")]
[Range(-100.0f, 100.0f)]
public float OrbitXSpeed = -30.0f;
[Tooltip("The maximum degrees to orbit on the x axis from the starting x rotation. 0 for no limit. Set
OrbitXSpeed to 0 to disable x orbit.")]
[Range(0.0f, 360.0f)]
public float OrbitXMaxDegrees = 0.0f;
[Tooltip("Whether the orbit on the x axis is a pan (move sideways) instead of an orbit.")]
public PanOrbitMovementType XAxisMovementType = PanOrbitMovementType.Orbit;
[Tooltip("Speed if OrbitXPan is true")]
[Range(0.1f, 10.0f)]
public float OrbitXPanSpeed = 1.0f;
[Tooltip("Set a movement limit from orbit target if OrbitXPan is true. 0 for no limit.")]
public float OrbitXPanLimit = 100.0f;
[Tooltip("The speed (degrees per second) at which to orbit using y delta pan gesture values. Negative
or positive values will cause orbit in the opposite direction.")]
[Range(-100.0f, 100.0f)]
public float OrbitYSpeed = -30.0f;
[Tooltip("The maximum degrees to orbit on the y axis from the starting y rotation. 0 for no limit. Set
OrbitYSpeed to 0 to disable y orbit.")]
[Range(0.0f, 360.0f)]
public float OrbitYMaxDegrees = 0.0f;
[Tooltip("Whether the orbit on the y axis is a pan (move sideways) instead of an orbit.")]
public PanOrbitMovementType YAxisMovementType = PanOrbitMovementType.Orbit;
[Tooltip("Speed if OrbitYPan is true.")]
[Range(0.1f, 10.0f)]
public float OrbitYPanSpeed = 1.0f;
[Tooltip("Set a movement limit from orbit target if OrbitYPan is true. 0 for no limit.")]
public float OrbitYPanLimit = 100.0f;
[Tooltip("Whether to allow orbit while zooming.")]
public bool AllowOrbitWhileZooming = true;
private bool allowOrbitWhileZooming;
[Tooltip("Whether to allow orbit and/or pan on both axis at the same time or to only pick the axis
with the greatest movement.")]
public bool AllowMovementOnBothAxisSimultaneously = true;
private int lockedAxis = 0; // 0 = none, 1 = x, 2 = y
[Tooltip("How much the velocity of the orbit will cause additional orbit after the gesture stops. 1 for
no inertia (orbits forever) or 0 for immediate stop.")]
[Range(0.0f, 1.0f)]
public float OrbitInertia = 0.925f;
[Tooltip("The max size for the orbit or pan. An x,y or z value larger than this away from orbit target
will be clamped in. Set to 0 for no limit.")]
public Vector3 OrbitMaximumSize;

```

[Tooltip(“Whether the pan and rotate orbit gestures must start on the orbit target to orbit. The tap gesture always requires that it be on the orbit target.”)]

```
public bool RequireOrbitGesturesToStartOnTarget;
/// <summary>
/// Types of movement
/// </summary>
public enum PanOrbitMovementType
{
    /// <summary>
    /// Orbit only
    /// </summary>
    Orbit,
    /// <summary>
    /// Pan only
    /// </summary>
    Pan,
    /// <summary>
    /// One touch orbit, two touch pan
    /// </summary>
    OrbitWithTwoFingerPan
}

/// <summary>
/// Scale gesture to zoom in and out
/// </summary>
public ScaleGestureRecognizer ScaleGesture { get; private set; }
/// <summary>
/// Pan gesture to orbit
/// </summary>
public PanGestureRecognizer PanGesture { get; private set; }
/// <summary>
/// Tap gesture to tap on orbit target
/// </summary>
public TapGestureRecognizer TapGesture { get; private set; }
private float xDegrees;
private float yDegrees;
private Vector2 panVelocity;
private float zoomSpeed;
public event System.Action OrbitTargetTapped;

private void OnEnable()
{
    // create a scale gesture to zoom orbiter in and out
    ScaleGesture = new ScaleGestureRecognizer();
    ScaleGesture.StateUpdated += ScaleGesture_Updated;
    ScaleGesture.ThresholdUnits = ZoomThresholdUnits;
    // pan gesture
    PanGesture = new PanGestureRecognizer();
    PanGesture.MaximumNumberOfTouchesToTrack = 2;
    PanGesture.StateUpdated += PanGesture_Updated;
    // create a tap gesture that only executes on the target, note that this requires a physics ray caster on
    the camera

    TapGesture = new TapGestureRecognizer();
    TapGesture.StateUpdated += TapGesture_Updated;
    TapGesture.PlatformSpecificView = OrbitTarget.gameObject;
    if (RequireOrbitGesturesToStartOnTarget)
```

```

    {
        ScaleGesture.PlatformSpecificView = OrbitTarget.gameObject;
        PanGesture.PlatformSpecificView = OrbitTarget.gameObject;
    }
    // point oribiter at target
    Orbiter.transform.LookAt(OrbitTarget.transform);
    FingersScript.Instance.AddGesture(ScaleGesture);
    FingersScript.Instance.AddGesture(PanGesture);
    FingersScript.Instance.AddGesture(TapGesture);
}

private void OnDisable()
{
    if (FingersScript.HasInstance)
    {
        FingersScript.Instance.RemoveGesture(ScaleGesture);
        FingersScript.Instance.RemoveGesture(PanGesture);
        FingersScript.Instance.RemoveGesture(TapGesture);
    }
}

private void LateUpdate()
{
    if (allowOrbitWhileZooming != AllowOrbitWhileZooming)
    {
        allowOrbitWhileZooming = AllowOrbitWhileZooming;
        if (allowOrbitWhileZooming)
        {
            ScaleGesture.AllowSimultaneousExecution(PanGesture);
        }
        else
        {
            ScaleGesture.DisallowSimultaneousExecution(PanGesture);
        }
    }
    Vector3 startPos = Orbiter.transform.position;
    UpdateOrbit(panVelocity.x, panVelocity.y);
    UpdateZoom();
    ClampDistance(startPos);
    panVelocity *= OrbitInertia;
    zoomSpeed *= OrbitInertia;
}

private bool IntersectRaySphere(Vector3 rayOrigin, Vector3 rayDir, Vector3 sphereCenter, float
sphereRadius, out float distanceToSphere, out Vector3 intersectPos)
{
    Vector3 m = rayOrigin - sphereCenter;
    float b = Vector3.Dot(m, rayDir);
    float c = Vector3.Dot(m, m) - (sphereRadius * sphereRadius);
    if (c > 0.0f && b > 0.0f)
    {
        distanceToSphere = 0.0f;
        intersectPos = Vector3.zero;
        return false;
    }
    float discr = (b * b) - c;
    if (discr < 0.0f)

```

```

    {
        distanceToSphere = 0.0f;
        intersectPos = Vector3.zero;
        return false;
    }
    distanceToSphere = Mathf.Max(0.0f, -b - Mathf.Sqrt(discr));
    intersectPos = rayOrigin + (distanceToSphere * rayDir);
    return true;
}

public void ClampDistance(Vector3 startPos)
{
    Vector3 orbitPos = Orbiter.transform.position;
    if ((startPos != orbitPos) && (MinimumDistance > 0.0f || MaximumDistance > 0.0f))
    {
        Vector3 targetPos = OrbitTarget.transform.position;
        Vector3 dirFromTarget = (orbitPos - targetPos).normalized;
        Vector3 intersectPos;
        float distanceToSphere;
        if (MinimumDistance > 0.0f && IntersectRaySphere(startPos, (orbitPos - startPos).normalized,
targetPos, MinimumDistance, out distanceToSphere, out intersectPos) &&
        distanceToSphere <= 0.0f)
        {
            Orbiter.transform.position = targetPos + (dirFromTarget * (MinimumDistance * (1.0f + Mathf.
Epsilon)));
            panVelocity = Vector3.zero;
            zoomSpeed = 0.0f;
        }
        else
        {
            float distance = Vector3.Distance(targetPos, orbitPos);
            float newDistance = Mathf.Clamp(distance, MinimumDistance, MaximumDistance);
            if (newDistance != distance)
            {
                Orbiter.transform.position = targetPos + (dirFromTarget * newDistance);
                panVelocity = Vector3.zero;
                zoomSpeed = 0.0f;
            }
        }
    }
}

private void UpdateZoom()
{
    if (zoomSpeed >= -0.01f && zoomSpeed <= 0.01f)
    {
        zoomSpeed = 0.0f;
        return;
    }
    Vector3 lookAtDir = (OrbitTarget.transform.position - Orbiter.transform.position).normalized;
    Quaternion lookAtRotation = Quaternion.LookRotation(lookAtDir, Orbiter.transform.up);
    Quaternion currentRotation = Orbiter.transform.rotation;
    Orbiter.transform.rotation = Quaternion.Lerp(currentRotation, lookAtRotation,
ZoomLookAtSpeed * Time.deltaTime);
    Orbiter.transform.position += (Orbiter.transform.forward * zoomSpeed * Time.deltaTime);
}

```



```

private void PerformPan(Vector3 pan, float limit)
{
    Vector3 pos = Orbiter.transform.position;
    Orbiter.Translate(pan, Space.Self);
    if (limit > 0.0f)
    {
        float distance = Vector3.Distance(Orbiter.transform.position, OrbitTarget.transform.position);
        if (distance > limit)
        {
            Orbiter.transform.position = pos;
        }
    }
}

private void UpdateOrbit(float xVelocity, float yVelocity)
{
    if (OrbitXSpeed != 0.0f && yVelocity != 0.0f)
    {
        if (YAxisMovementType == PanOrbitMovementType.Pan || (YAxisMovementType ==
PanOrbitMovementType.OrbitWithTwoFingerPan && PanGesture.CurrentTrackedTouches.Count > 1))
        {
            PerformPan(new Vector3(0.0f, yVelocity * (OrbitYPanSpeed) * Time.deltaTime, 0.0f),
OrbitYPanLimit);
        }
        else
        {
            float addAngle = yVelocity * OrbitXSpeed * Time.deltaTime;
            if (OrbitXMaxDegrees > 0.0f)
            {
                float newDegrees = xDegrees + addAngle;
                if (newDegrees > OrbitXMaxDegrees)
                {
                    addAngle = OrbitXMaxDegrees - xDegrees;
                }
                else if (newDegrees < -OrbitXMaxDegrees)
                {
                    addAngle = -OrbitXMaxDegrees - xDegrees;
                }
            }
            xDegrees += addAngle;

            Orbiter.RotateAround(OrbitTarget.transform.position, Orbiter.transform.right, addAngle);
        }
    }
    if (OrbitYSpeed != 0.0f && xVelocity != 0.0f)
    {
        if (XAxisMovementType == PanOrbitMovementType.Pan || (XAxisMovementType ==
PanOrbitMovementType.OrbitWithTwoFingerPan && PanGesture.CurrentTrackedTouches.Count > 1))
        {
            PerformPan(new Vector3(xVelocity * (OrbitXPanSpeed) * Time.deltaTime, 0.0f, 0.0f),
OrbitXPanLimit);
        }
        else
        {
            float addAngle = xVelocity * OrbitYSpeed * Time.deltaTime;

```

```

        if (OrbitYMaxDegrees > 0.0f)
        {
            float newDegrees = yDegrees + addAngle;
            if (newDegrees > OrbitYMaxDegrees)
            {
                addAngle = OrbitYMaxDegrees - yDegrees;
            }
            else if (newDegrees < -OrbitYMaxDegrees)
            {
                addAngle = -OrbitYMaxDegrees - yDegrees;
            }
        }
        yDegrees += addAngle;
        Orbiter.RotateAround(OrbitTarget.transform.position, Vector3.up, addAngle);
    }
}

private void TapGesture_Updated(DigitalRubyShared.GestureRecognizer gesture)
{
    if (gesture.State == GestureRecognizerState.Ended)
    {
        if (OrbitTargetTapped != null)
        {
            OrbitTargetTapped.Invoke();
        }
    }
}

private void PanGesture_Updated(DigitalRubyShared.GestureRecognizer gesture)
{
    if (gesture.State != GestureRecognizerState.Executing)
    {
        if (gesture.State == GestureRecognizerState.Ended)
        {
            lockedAxis = 0;
            if (OrbitInertia > 0.0f)
            {
                panVelocity = new Vector2(gesture.VelocityX * 0.01f, gesture.VelocityY * 0.01f);
                if (OrbitXSpeed == 0.0f)
                {
                    panVelocity.x = 0.0f;
                }
                if (OrbitYSpeed == 0.0f)
                {
                    panVelocity.y = 0.0f;
                }
            }
        }
        else if (gesture.State == GestureRecognizerState.Began)
        {
            panVelocity = Vector2.zero;
        }
        return;
    }
    else
    {

```

```

float xVelocity = gesture.DeltaX;
float yVelocity = gesture.DeltaY;
if (PanGestureHasEnoughMovementOnOneAxis(ref xVelocity, ref yVelocity))
{
    UpdateOrbit(xVelocity, yVelocity);
}
}

private void ScaleGesture_Updated(DigitalRubyShared.GestureRecognizer gesture)
{
    if (gesture.State != GestureRecognizerState.Executing)
    {
        return;
    }

    if (ScaleGesture.ScaleMultiplier > 1.0f)
    {
        zoomSpeed += (ScaleGesture.ScaleMultiplier * ZoomSpeed);
    }
    else if (ScaleGesture.ScaleMultiplier < 1.0f)
    {
        zoomSpeed -= ((1.0f / ScaleGesture.ScaleMultiplier) * ZoomSpeed);
    }
}

private bool PanGestureHasEnoughMovementOnOneAxis(ref float xVelocity, ref float yVelocity)
{
    if (AllowMovementOnBothAxisSimultaneously)
    {
        return true;
    }

    float unitsX = Mathf.Abs(DeviceInfo.PixelsToUnits(PanGesture.DistanceX));
    float unitsY = Mathf.Abs(DeviceInfo.PixelsToUnits(PanGesture.DistanceY));
    if (lockedAxis == 0 && unitsX <= PanGesture.ThresholdUnits && unitsY <= PanGesture.
ThresholdUnits)
    {
        return false;
    }
    else if (lockedAxis == 1 || (lockedAxis == 0 && unitsX > unitsY * 3.0f))
    {
        lockedAxis = 1;
        yVelocity = 0.0f;
        return true;
    }
    else if (lockedAxis == 2 || (lockedAxis == 0 && unitsY > unitsX * 3.0f))
    {
        lockedAxis = 2;
        xVelocity = 0.0f;
        return true;
    }
    return false;
}
}
}
}

```

Appendix C: Pan Rotate

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;
using UnityEngine.UI;

namespace DigitalRubyShared
{
    /// <summary>
    /// Allows two finger pan, scale and rotate on a game object
    /// </summary>
    [AddComponentMenu("Fingers Gestures/Component/Pan-Rotate-Scale", 4)]
    public class FingersPanRotateScript : MonoBehaviour
    {
        [Tooltip("The camera to use to convert screen coordinates to world coordinates. Defaults to Camera.
main.")]
        public Camera Camera;
        [Tooltip("Whether to bring the object to the front when a gesture executes on it")]
        public bool BringToFront = true;
        [Tooltip("Whether the gestures in this script can execute simultaneously with all other gestures.")]
        public bool AllowExecutionWithAllGestures;
        [Tooltip("The mode to execute in, can be require over game object or allow on any game object.")]
        public GestureRecognizerComponentScriptBase.GestureObjectMode Mode =
GestureRecognizerComponentScriptBase.GestureObjectMode.RequireIntersectWithGameObject;
        [Tooltip("The minimum and maximum scale. 0 for no limits. 1 for no scaling.")]
        public Vector2 MinMaxScale;
        [Tooltip("Whether to add a double tap to reset the transform of the game object this script is on.")]
        public bool DoubleTapToReset;
        /// <summary>
        /// Allow moving the target
        /// </summary>
        public PanGestureRecognizer PanGesture { get; private set; }
        /// <summary>
        /// Allow rotating the target
        /// </summary>
        public RotateGestureRecognizer RotateGesture { get; private set; }
        /// <summary>
        /// The double tap gesture or null if DoubleTapToReset was false when this script started up
        /// </summary>
        public TapGestureRecognizer DoubleTapGesture { get; private set; }
        private Rigidbody2D rigidBody2D;
        private Rigidbody rigidBody;
        private SpriteRenderer spriteRenderer;
        private CanvasRenderer canvasRenderer;
        private Transform _transform;
        private int startSortOrder;
        private float panZ;
        private Vector3 panOffset;
        private struct SavedState
        {
            public Vector3 Scale;
            public Quaternion Rotation;
        }
        private readonly Dictionary<Transform, SavedState> savedStates = new Dictionary<Transform,
SavedState>();
    }
}
```

```

    private static readonly List<RaycastResult> captureRaycastResults = new List<RaycastResult>();
    public static GameObject StartOrResetGesture(DigitalRubyShared.GestureRecognizer
r, bool bringToFront, Camera camera, GameObject obj, SpriteRenderer spriteRenderer,
GestureRecognizerComponentScriptBase.GestureObjectMode mode)
    {
        GameObject result = null;
        if (r.State == GestureRecognizerState.Began)
        {
            if ((result = GestureIntersectsObject(r, camera, obj, mode)) != null)
            {
                SpriteRenderer _spriteRenderer;
                if (bringToFront && (_spriteRenderer = result.GetComponent<SpriteRenderer>()) != null)
                {
                    _spriteRenderer.sortingOrder = 1000;
                }
            }
            else
            {
                r.Reset();
            }
        }
        return result;
    }

    private static int RaycastResultCompare(RaycastResult r1, RaycastResult r2)
    {
        SpriteRenderer rend1 = r1.gameObject.GetComponent<SpriteRenderer>();
        if (rend1 != null)
        {
            SpriteRenderer rend2 = r2.gameObject.GetComponent<SpriteRenderer>();
            if (rend2 != null)
            {
                int comp = rend2.sortingLayerID.CompareTo(rend1.sortingLayerID);
                if (comp == 0)
                {
                    comp = rend2.sortingOrder.CompareTo(rend1.sortingOrder);
                }
            }
            return comp;
        }
        return r2.gameObject.transform.GetSiblingIndex().CompareTo(r1.gameObject.transform.
GetSiblingIndex());
    }

    private static GameObject GestureIntersectsObject(DigitalRubyShared.GestureRecognizer r, Camera
camera, GameObject obj, GestureRecognizerComponentScriptBase.GestureObjectMode mode)
    {
        captureRaycastResults.Clear();
        PointerEventData p = new PointerEventData(EventSystem.current);
        p.Reset();
        p.position = new Vector2(r.FocusX, r.FocusY);
        p.clickCount = 1;
        EventSystem.current.RaycastAll(p, captureRaycastResults);
        captureRaycastResults.Sort(RaycastResultCompare);

        foreach (RaycastResult result in captureRaycastResults)
        {

```

```

        if (result.gameObject == obj)
        {
            return result.gameObject;
        }
        else if (result.gameObject.GetComponent<Collider>() != null ||
            result.gameObject.GetComponent<Collider2D>() != null ||
            result.gameObject.GetComponent<FingersPanRotateScaleComponentScript>() != null)
        {
            if (mode == GestureRecognizerComponentScriptBase.GestureObjectMode.
AllowOnAnyGameObjectViaRaycast)
            {
                return result.gameObject;
            }

            // blocked by a collider or another gesture, bail
            break;
        }
    }
    return null;
}

private void PanGestureUpdated(DigitalRubyShared.GestureRecognizer r)
{
    GameObject obj = StartOrResetGesture(r, BringToFront, Camera, gameObject, spriteRenderer,
Mode);
    if (r.State == GestureRecognizerState.Began)
    {
        SetStartState(r, obj, false);
    }
    else if (r.State == GestureRecognizerState.Executing && _transform != null)
    {
        if (PanGesture.ReceivedAdditionalTouches)
        {
            panZ = Camera.WorldToScreenPoint(_transform.position).z;
            panOffset = _transform.position - Camera.ScreenToWorldPoint(new Vector3(r.FocusX,
r.FocusY, panZ));
        }
        Vector3 gestureScreenPoint = new Vector3(r.FocusX, r.FocusY, panZ);
        Vector3 gestureWorldPoint = Camera.ScreenToWorldPoint(gestureScreenPoint) + panOffset;
        if (rigidBody != null)
        {
            rigidBody.MovePosition(gestureWorldPoint);
        }
        else if (rigidBody2D != null)
        {
            rigidBody2D.MovePosition(gestureWorldPoint);
        }
        else if (canvasRenderer != null)
        {
            _transform.position = gestureScreenPoint;
        }
        else
        {
            _transform.position = gestureWorldPoint;
        }
    }
    else if (r.State == GestureRecognizerState.Ended)

```

```

    {
        if (spriteRenderer != null && BringToFront)
        {
            spriteRenderer.sortingOrder = startSortOrder;
        }
        ClearStartState();
    }
}

private void RotateGestureUpdated(DigitalRubyShared.GestureRecognizer r)
{
    GameObject obj = StartOrResetGesture(r, BringToFront, Camera, gameObject, spriteRenderer,
Mode);
    if (r.State == GestureRecognizerState.Began)
    {
        SetStartState(r, obj, false);
    }
    else if (r.State == GestureRecognizerState.Executing && _transform != null)
    {
        if (rigidBody != null)
        {
            float angle = RotateGesture.RotationDegreesDelta;
            Quaternion rotation = Quaternion.AngleAxis(angle, Camera.transform.forward);
            rigidBody.MoveRotation(rigidBody.rotation * rotation);
        }
        else if (rigidBody2D != null)
        {
            rigidBody2D.MoveRotation(rigidBody2D.rotation + RotateGesture.RotationDegreesDelta);
        }
        else if (canvasRenderer != null)
        {
            _transform.Rotate(Vector3.forward, RotateGesture.RotationDegreesDelta, Space.Self);
        }
        else
        {
            _transform.Rotate(Camera.transform.forward, RotateGesture.RotationDegreesDelta, Space.
Self);
        }
    }
    else if (r.State == GestureRecognizerState.Ended)
    {
        ClearStartState();
    }
}

private void DoubleTapGestureUpdated(DigitalRubyShared.GestureRecognizer r)
{
    if (r.State == GestureRecognizerState.Ended)
    {
        GameObject obj = GestureIntersectsObject(r, Camera, gameObject, Mode);
        SavedState state;
        if (obj != null && savedStates.TryGetValue(obj.transform, out state))
        {
            obj.transform.rotation = state.Rotation;
            obj.transform.localScale = state.Scale;
            savedStates.Remove(obj.transform);
        }
    }
}

```

```

    }
}

private void ClearStartState()
{
    if (Mode != GestureRecognizerComponentScriptBase.GestureObjectMode.
AllowOnAnyGameObjectViaRaycast)
    {
        return;
    }
    else if (PanGesture.State != GestureRecognizerState.Executing &&
        RotateGesture.State != GestureRecognizerState.Executing)

    {
        rigidBody2D = null;
        rigidBody = null;
        spriteRenderer = null;
        canvasRenderer = null;
        _transform = null;
    }
}

private bool SetStartState(DigitalRubyShared.GestureRecognizer gesture, GameObject obj, bool
force)
{
    if (!force && Mode != GestureRecognizerComponentScriptBase.GestureObjectMode.
AllowOnAnyGameObjectViaRaycast)
    {
        return false;
    }
    else if (obj == null)
    {
        ClearStartState();
        return false;
    }
    else if (_transform == null)
    {
        rigidBody2D = obj.GetComponent<Rigidbody2D>();
        rigidBody = obj.GetComponent<Rigidbody>();
        spriteRenderer = obj.GetComponent<SpriteRenderer>();
        canvasRenderer = obj.GetComponent<CanvasRenderer>();
        if (spriteRenderer != null)
        {
            startSortOrder = spriteRenderer.sortingOrder;
        }
        _transform = (rigidBody == null ? (rigidBody2D == null ? obj.transform : rigidBody2D.
transform) : rigidBody.transform);
        if (DoubleTapToReset && !savedStates.ContainsKey(_transform))
        {
            savedStates[_transform] = new SavedState { Rotation = _transform.rotation, Scale = _
transform.localScale };
        }
    }
    else if (_transform != obj.transform)
    {
        if (gesture != null)

```



```

        {
            gesture.Reset();
        }
        return false;
    }
    return true;
}

private void OnEnable()
{
    this.Camera = (this.Camera == null ? Camera.main : this.Camera);
    PanGesture = new PanGestureRecognizer { MaximumNumberOfTouchesToTrack = 2,
ThresholdUnits = 0.01f };
    PanGesture.StateUpdated += PanGestureUpdated;
    RotateGesture = new RotateGestureRecognizer();
    RotateGesture.StateUpdated += RotateGestureUpdated;
    if (Mode != GestureRecognizerComponentScriptBase.GestureObjectMode.
AllowOnAnyGameObjectViaRaycast)
    {
        SetStartState(null, gameObject, true);
    }
    if (DoubleTapToReset)
    {
        DoubleTapGesture = new TapGestureRecognizer { NumberOfTapsRequired = 2 };
        DoubleTapGesture.StateUpdated += DoubleTapGestureUpdated;
    }
    if (AllowExecutionWithAllGestures)
    {
        PanGesture.AllowSimultaneousExecutionWithAllGestures();
        RotateGesture.AllowSimultaneousExecutionWithAllGestures();
        if (DoubleTapGesture != null)
        {
            DoubleTapGesture.AllowSimultaneousExecutionWithAllGestures();
        }
    }
    else
    {
        PanGesture.AllowSimultaneousExecution(RotateGesture);
        if (DoubleTapGesture != null)
        {
            DoubleTapGesture.AllowSimultaneousExecution(RotateGesture);
            DoubleTapGesture.AllowSimultaneousExecution(PanGesture);
        }
    }
    if (Mode == GestureRecognizerComponentScriptBase.GestureObjectMode.
RequireIntersectWithGameObject)
    {
        RotateGesture.PlatformSpecificView = gameObject;
        PanGesture.PlatformSpecificView = gameObject;
        if (DoubleTapGesture != null)
        {
            DoubleTapGesture.PlatformSpecificView = gameObject;
        }
    }
    FingersScript.Instance.AddGesture(PanGesture);
    FingersScript.Instance.AddGesture(RotateGesture);
    FingersScript.Instance.AddGesture(DoubleTapGesture);
}

```

```

    }

    private void OnDisable()
    {
        if (FingersScript.HasInstance)
        {
            FingersScript.Instance.RemoveGesture(PanGesture);
            FingersScript.Instance.RemoveGesture(RotateGesture);
            FingersScript.Instance.RemoveGesture(DoubleTapGesture);
        }
    }
}
}
}

```

Appendix D: Window Manager Script

```

using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;
using System.Collections;
using System.Collections.Generic;
public class WindowManager : MonoBehaviour {

    public Animator initiallyOpen;
    private int m_OpenParameterId;
    private Animator m_Open;
    private GameObject m_PreviouslySelected;
    const string k_OpenTransitionName = "Open";
    const string k_ClosedStateName = "Closed";
    public void OnEnable()
    {
        m_OpenParameterId = Animator.StringToHash(k_OpenTransitionName);
        if (initiallyOpen == null)
            return;
        OpenPanel(initiallyOpen);
    }

    public void OpenPanel (Animator anim)
    {
        if (m_Open == anim)
            return;
        anim.gameObject.SetActive(true);
        var newPreviouslySelected = EventSystem.current.currentSelectedGameObject;
        anim.transform.SetAsLastSibling();
        CloseCurrent();
        m_PreviouslySelected = newPreviouslySelected;
        m_Open = anim;
        m_Open.SetBool(m_OpenParameterId, true);
        GameObject go = FindFirstEnabledSelectable(anim.gameObject);
        SetSelected(go);
    }

    static GameObject FindFirstEnabledSelectable (GameObject gameObject)
    {
        GameObject go = null;
        var selectables = gameObject.GetComponentsInChildren<Selectable> (true);
    }
}

```

```

        foreach (var selectable in selectables) {
            if (selectable.IsActive () && selectable.IsInteractable ()) {
                go = selectable.gameObject;
                break;
            }
        }
        return go;
    }

    public void CloseCurrent()
    {
        if (m_Open == null)
            return;
        m_Open.SetBool(m_OpenParameterId, false);
        SetSelected(m_PreviouslySelected);
        StartCoroutine(DisablePanelDeleyed(m_Open));
        m_Open = null;
    }

    IEnumerator DisablePanelDeleyed(Animator anim)
    {
        bool closedStateReached = false;
        bool wantToClose = true;
        while (!closedStateReached && wantToClose)
        {
            if (!anim.IsInTransition(0))
                closedStateReached = anim.GetCurrentAnimatorStateInfo(0).IsName(k_
ClosedStateName);

            wantToClose = !anim.GetBool(m_OpenParameterId);

            yield return new WaitForEndOfFrame();
        }

        if (wantToClose)
            anim.gameObject.SetActive(false);
    }

    private void SetSelected(GameObject go)
    {
        EventSystem.current.SetSelectedGameObject(go);
    }
}

```

Appendix E: Snap in Place

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

namespace DigitalRubyShared
{
    public class SnapInPlace : MonoBehaviour
    {
        public Transform targetTransform;
        public Vector3 currentPosition;
    }
}

```

```

public Vector3 currentRotation;
public float xPosDiff;
public float yPosDiff;
public float zPosDiff;
public float xRotDiff;
public float yRotDiff;
public float zRotDiff;

void Update()
{
    currentPosition = this.transform.position;
    currentRotation = this.transform.rotation.eulerAngles;
    xPosDiff = Mathf.Abs(targetTransform.position.x) - Mathf.Abs(currentPosition.x);
    yPosDiff = Mathf.Abs(targetTransform.position.y) - Mathf.Abs(currentPosition.y);
    zPosDiff = Mathf.Abs(targetTransform.position.z) - Mathf.Abs(currentPosition.z);
    xRotDiff = Mathf.DeltaAngle(targetTransform.localEulerAngles.x, currentRotation.x);
    yRotDiff = Mathf.DeltaAngle(targetTransform.localEulerAngles.y, currentRotation.y);
    zRotDiff = Mathf.DeltaAngle(targetTransform.localEulerAngles.z, currentRotation.z);
    if (Mathf.Abs(xPosDiff) < 0.5 && Mathf.Abs(yPosDiff) < 0.5 && Mathf.Abs(zPosDiff) < 0.5 &&
    Mathf.Abs(xRotDiff) < 0.5 && Mathf.Abs(yRotDiff) < 0.5 && Mathf.Abs(zRotDiff) < 0.5)
    {
        GetComponent<LerpFracture>().enabled = true;
    }
}
}
}
}

```

Appendix F: Lerp Script

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class LerpFracture : MonoBehaviour
{
    public Transform target;
    private Vector3 startPosition;
    private Quaternion startRotation;
    public float speed = 1.0F;
    private float startTime;
    private float distance;

    void Start()
    {
        startTime = Time.time;
        startPosition = transform.position;
        startRotation = transform.rotation;
        distance = Vector3.Distance(startPosition, target.position);
    }

    void Update()
    {
        float distCovered = (Time.time - startTime) * speed;
        float fracJourney = distCovered / distance;
        transform.position = Vector3.Lerp(startPosition, target.position, fracJourney);
        transform.rotation = Quaternion.Lerp(startRotation, target.rotation, fracJourney);
    }
}

```

```
}  
}
```

Appendix G: Spawn Object Script

```
using System;  
using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class SpawnObject : MonoBehaviour  
{  
  
    public Transform[] spawnlocations;  
    public GameObject[] whatToSpawnPrefab;  
    public GameObject[] whatToSpawnClone;  
  
    public void spawnZero()  
    {  
        whatToSpawnClone[0] = Instantiate(whatToSpawnPrefab[0], spawnlocations[0].transform.position,  
Quaternion.Euler(0, 0, 0)) as GameObject;  
    }  
  
    public void spawnOne()  
    {  
        whatToSpawnClone[1] = Instantiate(whatToSpawnPrefab[1], spawnlocations[1].transform.position,  
Quaternion.Euler(0, 0, 0)) as GameObject;  
    }  
  
    public void spawnTwo()  
    {  
        whatToSpawnClone[2] = Instantiate(whatToSpawnPrefab[2], spawnlocations[2].transform.position,  
Quaternion.Euler(0, 0, 0)) as GameObject;  
    }  
  
    public void spawnThree()  
    {  
        whatToSpawnClone[3] = Instantiate(whatToSpawnPrefab[3], spawnlocations[3].transform.position,  
Quaternion.Euler(0, 0, 0)) as GameObject;  
    }  
  
    public void spawnFour()  
    {  
        whatToSpawnClone[4] = Instantiate(whatToSpawnPrefab[4], spawnlocations[4].transform.position,  
Quaternion.Euler(0, 0, 0)) as GameObject;  
    }  
  
    public void spawnFive()  
    {  
        whatToSpawnClone[5] = Instantiate(whatToSpawnPrefab[5], spawnlocations[5].transform.position,  
Quaternion.Euler(0, 0, 0)) as GameObject;  
    }  
  
    public void spawnSix()  
    {  
        whatToSpawnClone[6] = Instantiate(whatToSpawnPrefab[6], spawnlocations[6].transform.position,
```

```

Quaternion.Euler(0, 0, 0) as GameObject;
}

public void spawnSeven()
{
    whatToSpawnClone[7] = Instantiate(whatToSpawnPrefab[7], spawnlocations[7].transform.position,
Quaternion.Euler(0, 0, 0) as GameObject;
}

public void spawnEight()
{
    whatToSpawnClone[8] = Instantiate(whatToSpawnPrefab[8], spawnlocations[8].transform.position,
Quaternion.Euler(0, 0, 0) as GameObject;
}

}

```

Appendix H: Destroy All Tagged Script

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DestroyAllTagged : MonoBehaviour
{
    public void DestroyInstrument(string tag)
    {
        GameObject[] gameObjects = GameObject.FindGameObjectsWithTag(tag);
        foreach (GameObject target in gameObjects)
        {
            GameObject.Destroy(target);
        }
    }
}

```

Appendix I: OR Timer Script

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class ORTimer : MonoBehaviour
{
    Text text;
    float theTime;
    public float speed = 1;
    bool playing;

    void Start()
    {

```

```

    text = GetComponent<Text>();
}

void Update()
{
    if (playing == true)
    {
        theTime += Time.deltaTime * speed;
        string hours = Mathf.Floor((theTime % 216000) / 3600).ToString("00");
        string minutes = Mathf.Floor((theTime % 3600) / 60).ToString("00");
        string seconds = (theTime % 60).ToString("00");
        text.text = hours + ":" + minutes + ":" + seconds;
    }
}

public void ClickPlay()
{
    playing = true;
}

public void ClickStop()
{
    playing = false;
}
}

```

Appendix J: Enable Disable Object Script

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EnableDisableObject : MonoBehaviour
{
    public GameObject objectToDisable;

    public void DisableObject()
    {
        objectToDisable.SetActive(false);
    }

    public void EnableObject()
    {
        objectToDisable.SetActive(true);
    }
}

```

Appendix K: Change Material Script

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

```

```

public class ChangeMaterialScript : MonoBehaviour
{
    public Material Bone;
    public Material XRay;
    public bool FirstMaterial = true;
    public bool SecondMaterial = false;
    private Renderer rend;

    void Start()
    {
        rend = GetComponent<Renderer>();
        rend.enabled = true;
        rend.material = Bone;
    }

    public void ToggleTexture()
    {
        if (FirstMaterial == true)
        {
            FirstMaterial = false;
            rend.material = XRay;
        }
        else
        {
            FirstMaterial = true;
            rend.material = Bone;
        }
    }
}

```

Appendix L: Load Scene Script

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class LoadScene : MonoBehaviour
{
    public void LoadScene(int level)
    {
        Application.LoadLevel(level);
    }
}

```


REFERENCES

- Accreditation Council for Graduate Medical Education. (2019). *List of programs by specialty*. Retrieved from <https://apps.acgme.org/ads/Public/Reports/Report/1>.
- AfterTrauma. (n.d.). *Types of pelvic injury*. The pelvis. Retrieved from www.aftertrauma.org/injuries-to-the-pelvis/injuries-to-the-pelvis.
- Barr, R. (2010). Transfer of learning between 2D and 3D sources during infancy: Informing theory and practice. *Developmental Review*, 30(2), 128-154.
- Beermann, J., Tetzlaff, R., Bruckner, T., Schoebinger, M., Muller-Stitch, B. P., Gutt, C. N., Meinzer, H. P., Kadmon, M. & Fischer, L. (2010). Three-dimensional visualization improves understanding of surgical liver anatomy. *Medical Education*, 44(9), 936-940.
- Bloice, M. D., Simonic, K. M., & Holzinger, A. (2013). *On the usage of health records for the design of virtual patients: A systematic review*. Retrieved from <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC38466661/>
- Bridges, M., & Diamond, DL. (1999). The financial impact of teaching surgical residence in the operating room. *American Journal of surgery*, 177(1), 28-32.
- Brown, P. M., Hamilton, N. M., & Denison, A. R. (2012). A novel 3D stereoscopic anatomy tutorial. *The Clinical Teacher*, 9(1), 50-53.
- Carter, L. W., Stovall, D. O., & Young, T. (1995). Determination of accuracy of preoperative templating of noncemented femoral prostheses. *The Journal of Arthroplasty*, 10(4), 507-513.
- Daniels, A. H., & DiGiovanni, C. W. (2014). Is Subspeciality fellowship training emerging as a necessary component of contemporary orthopaedic surgery education? *Journal of Graduate Medical Education*, 6(2), 218-221.
- de Montbrun, S. L., & Macrae, H. (2012). Simulation in surgical education. *Clinics in colon and rectal*

surgery, 25(3), 156-65.

Estevez, M. E., Lindgren, K. A., & Bergethon, P. R. (2010). A novel three-dimensional tool for teaching human neuroanatomy. *Anatomical Science Education*, 3(6), 309-317.

Gurusamy, K., Aggarwal, R., Palanivelu, L. P., & Davidson, B. R. (2008). Systematic review of randomized controlled trials on the effectiveness of virtual reality training for laparoscopic surgery. *British Journal of Surgery Society Ltd*, 95(9), 1088-1097.

Halawi, M. J. (2015). Pelvic ring injuries: Emergency assessment and management. *Journal of Clinical Orthopaedics and Trauma*, 6(4), 252-258.

Hayhurst, C. (2018). Reality, augmented: Modeling and simulation bring new vision to healthcare. *Biomedical Instrumentation & Technology*, 52(5), 340-348.

Horst, P. K., Choo, K., Bharucha, N., & Vail, T. P. (2015). Graduates of orthopaedic residency training are increasingly subspecialized: A review of the American board of orthopaedic surgery part II database. *The Journal of Bone and Joint Surgery*, 97(10), 869-875.

Issenberg, S. B., McGaghie, W. C., Hart, I. R., Mayer, J. W., Felner, J. M., Petrusa, E. R., Waugh, R. A., Brown, D. D., Safford, R. R., Gessner, I. H., Gordon, D. L., & Ewy, G. A. (1999). Simulation technology for health care professional skills training and assessment. *Journal of American Medical Association*, 282(9), 861-866.

John, N. W. (2007). The impact of Web3D technologies on medical education and training. *Computers & Education*, 49(1), 19-31.

Johnell, O., & Kanis, J. A. (2006). An estimate of the worldwide prevalence and disability associated with osteoporotic fractures. *Osteoporosis International*, 17(12), 1726-1733

LeCompte, M., Stewart, M., Harris, T., Rivers, G., Guth, C., Ehrenfeld, J., Sexton, K., & Terhune, K. (2019). See one, do one, teach one: A randomized controlled study evaluating the benefit of

- autonomy in surgical education. *The American Journal of Surgery*, 217(2), 281-287.
- Lovejoy, C. O. (1988). Evolution of human walking. *Scientific American*, 259(5), 118-125.
- Mardanpour, K., & Rahbar, M. (2013). The outcome of surgically treated traumatic unstable Fractures by open reduction and internal fixation. *Journal of injury & violence research*, 5(2), 77- 83.
- National Resident Matching Program. (2018). *Charting outcomes in the match: U.S. allopathic seniors. Characteristics of U.S. allopathic seniors who matched to their preferred specialty in the 2018 main residency match*. Retrieved from <http://www.nrmp.org/wp-content/uploads/2018/06/Charting-Outcomes-in-the-Match-2018-Seniors.pdf>.
- National Resident Matching Program. (2017). *Results and data. 2017 main residency match*. Retrieved from <https://www.nrmp.org/wp-content/uploads/2017/06/Main-Match-Results-and-Data-2017.pdf>.
- Noguera, J. M., Jimenez, J. J., & Osuna-Perez. (2013). Development and evaluation of a 3D mobile application for learning manual therapy in the physiotherapy laboratory. *Computers & Education*, 33, 96-108.
- Okike, K., O'Toole, R. V., Pollak, A. N., Bishop, J. A., McAndrew, C. M., Mehta, S., Cross, W. W., Garrigues, G. E., Harris, M. B., & Lebrun, C. T. (2014). Survey finds few orthopedic surgeons know the costs of the devices they implant. *Health Affairs*, 33(1), 103-109.
- OpenStax. (2013). *Anatomy & Physiology*. Retrieved from <http://opentextbc.ca/anatomyandphysiology/>
- Palmer, S., Fairbank, A. C., & Bircher, M. (1997). Surgical complications and implications of external fixation of the pelvic fractures. *Injury*, 28(9-10), 649-653.
- Polavarapu, H., Kulaylat, A. N., Sun, S., & Hamed, O. (2013). *100 years of surgical education: The past, present, and future*. Retrieved from bulletin.facs.org/2013/07/100-years-of-surgical-education/
- Pujol, S., Baldwin, M., Nassiri, J., Kikinis, R., & Shaffer, K. (2016). Using 3D modeling techniques to enhance teaching of difficult anatomical concepts. *Academic Radiology*, 23(4), 507-516.

- Rassie, K. (2017). The apprenticeship model of clinical medical education: time for structural change. *The New Zealand Medical Journal*, 130(1461). Retrieved from <https://www.nzma.org.nz/journal/read-the-journal/all-issues/2010-2019/2017/vol-130-no-1461-1-september-2017/7348>
- Rancic Moogk, D. (2012). Minimum viable product and the importance of experimentation in technology Startups. *Technology Innovation Management Review*, 2(3), 23-26.
- Rodriquez-Paz, J. M., Kennedy, M., Salas, E., Wu, A. W., Sexton, J. B., Hunt, E. A., & Pronovost, P. J. (2009). *Beyond "see one, do one, teach one": Toward a different training paradigm*. Retrieved from <https://qualitysafety.bmj.com/content/18/1/63>
- Ruedi, T. P., Buckley, R. E., & Moran, C. G. (2007). *AO principles of fracture management*. NY: Thieme Medical.
- Seymour, N. E., Gallagher, A. G., Roman, S. A., O'Brien, M. K., Bansal, V. K., Anderson, D. K., & Satava, R. M. (2002). Virtual reality training improves operating room performance. Results of randomized, double-blinded study. *Annals of Surgery*, 236(4), 458-464.
- Smithsonian National Museum of Natural History. (2018). Human characteristics: *Walking upright*. Retrieved from humanorigins.si.edu/human-characteristics/walking-upright
- Tanagho, Y. S., Andriole, G. L., Paradis, A. G., Madison, K. M., Sandhu, G. S., Varela, J. E. & Benway, B. M. (2012). 2D versus 3D visualization: Impact on laparoscopic proficiency using the fundamentals of Laparoscopic surgery skill set. *Journal of Laparoendoscopic & Advanced Surgical Techniques*, 22(9).
- The American Board of Surgery. (2019). *Training and certification*. Retrieved from www.absurgery.org/default.jsp?certgsqe_training
- Triola, M., Feldman, H., Kalet, A. L. Zabar, S., Kachur, E. K., Gillespie, C., Anderson, M., Griesser, C., & Lipkin, M. (2006). A randomized trial of teaching clinical skills using virtual and live standardized patients. *Journal of General Internal Medicine*. 21(5), 424-429.

United States Department of Labor, (2018). *American time use survey – 2017 results*. Retrieved from <https://www.bls.gov/news.release/pdf/atus.pdf>

Virkus, W. W., & Marota, M. (n.d.). *Which fractures require anatomic reduction?* Retrieved from <https://www.healio.com/orthopedics/curbside-consultation/%7B54c0de94-6247-42a6-9798-343e1a141a61%7D/which-fractures-require->

Weinstock, P. [Ted]. (2017, April 11). *Lifelike simulations that make real-life surgery safer* [Video File]. Retrieved from <http://www.youtube.com/watch?v=TeGr86q06c>

White, C. E., Hsu, J. R., & Holcomb, J. B. (2009). Haemodynamically unstable pelvic fractures. *Injury*, 40(10), 1023-1030.

Wolf, B. R., & Britton, C. L. (2013). How orthopaedic residents perceive educational resources. *The Iowa Orthopaedic Journal*, 33, 185-190.

VITA

Vondel Shadrach Edwin Mahon was born on July 26th, 1989 in St. Georges, Grenada. Being surrounded by tropical wildlife, celestial landscapes, and the Caribbean Sea, he adopted a love and appreciation for nature and life, and was forever mesmerized by it. He spent his time during childhood studying subjects through drawing as a way to conceptualize and demystify them.

Upon learning about a field in which these passions were combined, he decided to pursue a career in Medical Illustration. His goal is to enrich the lives of others, by uncovering life's mysteries through the use of visual communication media. In 2014, Vondel received his Bachelor of Science degree in Medical Illustration from the City University New York (CUNY). He graduated summa cum laude, through CUNY's Baccalaureate for Unique and Interdisciplinary Studies program. To fulfill his course requirements, he attended 3 colleges: New York City College of Technology, Brooklyn College, and Medgar Evers College (MEC).

During his time at the City University New York (CUNY), he was an honor student, an award winning student-athlete, a tutor in Biology, Mathematics, Chemistry, and Human Anatomy and Physiology, an undergraduate research assistant studying antibiotic producing soil bacteria, a curator's assistant at Medgar Evers College (MEC) Department of Mass Communications, and a student representative for the MEC's School of Science Health and Technology.

After attending CUNY, he spent two years at the Art Students League of New York to continue his studies in Fine Art and Human Anatomy.

Vondel is scheduled to receive his Master of Arts degree in Medical and Biological Illustration in May of 2019. Upon completing his master's degree from Johns Hopkins University, he plans to use novel technologies in visual communication media to teach and disseminate the marvels of science and medicine to the world.