DESIGN AND IMPLEMENTATION OF A VIRTUAL REALITY LABORATORY FOR MECHANICAL MAINTENANCE

A Thesis

Presented to the faculty of

the College of the College of Business and Technology

Morehead State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

---

by

Adolfo Enrique Samudio Cano

April 27, 2018

Accepted by the faculty of the College of Business and Technology, Morehead State University, in partial fulfillment of the requirements for the Master of Science degree.

_____
Dr. Jorge Alberto Ortega Moody
Director of Thesis

Master's Committee: _____, Chair
Dr. Ahmad Zargari

_____
Dr. Nilesh Joshi

_____
Date

DESIGN AND IMPLEMENTATION OF A VIRTUAL REALITY LABORATORY FOR MECHANICAL MAINTENANCE

Adolfo Enrique Samudio Cano
Morehead State University, 2018

Director of Thesis: _____
Dr. Jorge Alberto Ortega Moody

Every manufacturing facility requires preventive and corrective maintenance to their industrial equipment to maintain the performance indicators to their desired levels. One of the limitations related to maintenance is the training of new and existing personnel. New personnel are unaware of the necessary steps and skills to perform maintenance on equipment. The lack of available training equipment often requires either shutting down operations, which cost the industry money and/or putting trainees and other employees in potentially dangerous situations if mistakes are made during live operations without previous training. At the educational level, students lack the proper industrial grade training due to the lack of relevant equipment and high budget constraints. Universities often train students with obsolete equipment or lower level training stations as compared to the one's used at an

industrial level. For these reasons, Virtual Scenarios for Maintenance are proposed in this research.

The potential of laboratories relies on their ability to carry out industrial personnel training in a safer and more efficient way without the constraints of equipment availability and budget constraints. Virtual Scenarios lower the cost of maintenance, increase safety, reduce replacement costs for obsolete equipment and open options for online training. Virtual Scenarios for Maintenance open new avenues for education, since components possess the physical, electrical, and mechanical characteristics of the industry necessary to simulate real behavior without the limitations of having to acquire and maintain a physical laboratory or industrial space reserved for training. This research covers the development of one specific virtual scenario for training in Mechanical Maintenance, by virtualizing a standard DC motor and creating a Virtual Reality Scenario tutorial.

The procedure followed is one established through common video game development and evolves the methods used in the entertainment industry for educational and training purposes.

The method is defined by the following steps:

The first step of the development process starts with the design specifications. The mechanical process or machine that the virtual process in this research will recreate has to be analyzed thoroughly and understood.

Secondly, in the design and modeling step, the machine and/or mechanical process, the tools use to perform specific tasks and the surroundings have to be designed and created using 3D modeling software.

The third step focuses on the environment design where the knowledge gained in phase one, combined with the 3D modeling from step two are put together to create a realistic environment, a representation of a real world laboratory, manufacturing facility or, in this case, a maintenance workshop.

The fourth step, and one of the most important in the whole process, is the environmental and behavior programming. This step focuses on implementing C# programming to create realistic animations that allow the trainee to interact with objects like tools, machines, components and more. This programming also allows for the environment to behave realistically with factors such as weight, mass, drag, and force.

The sixth, and final step, centers on creating a tutorial, which guides the trainee through a certain procedure; in this case, the procedure to assemble and disassemble an AC motor in order to change the bearings. The tutorial includes voice commands and full interaction with VR hardware.

Accepted by: _____, Chair
Dr. Ahmad Zargari

_____
Dr. Jorge Alberto Ortega Moody

_____
Dr. Nilesh Joshi

ACKNOWLEDGEMENT

This project and the corresponding research was performed at the facilities 21st Century Center for Manufacturing Systems, belonging to the Department of Engineering & Technology Management at Morehead State University.

I would like to acknowledge and thank those individuals who aided and supported in completing this thesis. Firstly, I would like to thank my thesis director, Dr. Jorge Alberto Ortega Moody, who not only assisted and guided me through this 2-year project, but who also shared the entirety of his experience and knowledge on Engineering, Virtual Reality and Video Game Development. Without his knowledge, patience and experience this project would not have been a reality. Next, thesis committee chairman, Dr. Ahmad Zargari, and committee member, Dr. Nilesh Joshi, who provided general guidance through 3D modeling training early in my college career. I would also like to thank my friend Tyler who played a vital part in the development of this project and who spent countless hours in the 21st Century Center for Manufacturing Systems with Dr. Moody and me.

I would like to thank my parents, for without their support and guidance none of this would have been possible and my girlfriend Kelsey, who has been there with continuous encouragement throughout the entire process.

Lastly, I would like to thank the Morehead State University community for five and a half years of not just great education, but great memories, friendships and continued personal growth.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Definitions

Virtual environments or virtual scenarios are custom-made 3D virtual worlds created out of 3D models, which allow trainees to interact and immerse themselves in “worlds” that are distant, expensive, hazardous, or inaccessible. Virtual scenarios are also known as, “artificial reality”, “virtual worlds” and “synthetic environments”. Some Virtual Reality scenarios, like the ones described in this paper, offer full immersion into the virtual world through virtual reality headsets from existing companies leading the VR industry (*Alaraj, Lemole, Finkle and Yudkowsky,* 2011).

The virtual reality headset used throughout this research is the Oculus Rift by Oculus. This particular headset is a cranium-mounted device that has a stereoscopic head-mounted display, which provides separate images for each eye, sound, and motion tracking sensors. Oculus Rift also includes gaming controllers as an accessory. The Oculus Rift was the first device to introduce “real” virtual reality on a larger scale. It can also be argued that the Oculus Rift is the best on the market with the most integration for developers, which is one of the reasons it was chosen as the hardware for this research (*Kuchera, 2016).*

The most important software used throughout this research is known as Unity, which is a cross-platform game engine. A game engine is a software which allows the development of 3D environments specifically designed for video game development; otherwise known as a physics engine. Unity creates a 3D environment with real behavior through scripting and full interaction with the user as well as compatibility with Virtual Reality hardware such as Oculus Rift (*Riccitiello, 2014).*

Software development kits known as SDKs were used through this research they are typically tools that allow the creation of applications for certain software packages, they help with the communication between different technologies, and platforms.

1.2 General background

Industrial maintenance can be defined as the repair and upkeep of equipment and machines used in an industrial setting. Some general knowledge that is required to perform industrial maintenance is safety, an understanding of tools and following procedures. Using the correct tools and understanding how the tools are implemented are crucial skills for performing maintenance on industrial machines. Another important ability that the person performing maintenance should focus on is safety, including his or her own safety as well as the safety of those operating the machinery.

Industrial maintenance cannot be overlooked. It plays a vital role in the effectiveness of manufacturing success and lean manufacturing. Industrial maintenance is required to efficiently reduce waste in addition to running efficient, continuous, and safe operations. Maintenance training is central to any given manufacturing success due to its insurance that all of the equipment, new or old, is functioning efficiently and effectively (*Bureau of Labor Statistics 2016-2017).*

1.3 Purpose of the Study

Industrial maintenance training takes time. Employees must have proper general knowledge prior to performing any maintenance on specific machines and also must have

proper maintenance and safety training relevant to the specific machine the maintenance is being performed on. Both, companies and academic institutions face training limitations when teaching maintenance to students or employees.

Limitations in the industry are largely related to high costs of employee training, as training often requires shutting down operations and putting employees and trainees in potentially dangerous situations. Industrial facilities also face limitations with equipment availability. Companies need to have equipment on hand relegated solely for maintenance training or wait until a machine is damaged to teach maintenance.

Academic limitations stem from the lack of infrastructure allocated to students to conduct trials in an industrial level environment. Often times, academic institutions face budget restraints, limiting the amount and variety of machinery available for student use. Industrial facilities and academic institutions thrive on safety, so keeping employees and student's safe is paramount. Maintenance training, if not done correctly with the proper prior knowledge and the right tools and procedures, can be extremely unsafe, seeing as high temperatures, heavy machinery and electrical currents are likely involved (*Zyda, 2005).*

Due to these reasons, Virtual Reality Scenarios are proposed for maintenance training applications. Said scenarios will benefit manufacturing companies, industrial facilities, (*Brasil, Neto, Chagas, Monteiro, Souza, Bonates and Dantas*, 2011) along with academic institutions ranging from technical high schools to Universities. Virtual Reality scenarios for maintenance focuses on realistically portraying the scenario and physical characteristics found in the real world, as well as the corresponding implementation of tools and a tutorial system *(Guo, Li, Chan and Skitmore, 2012*).

Virtual Reality scenarios solve the problem of availability, cost, maintenance, constant upgrades, and safety. (*Brasil, Neto, Chagas, Monteiro, Souza, Bonates and Dantas*, 2011) Virtual reality scenarios use a Game Engine platform (in this case, Unity), CAD modeling and C# programming to implement everything needed into a VR reality headset where the trainee is fully capable of interacting with their surroundings by grabbing, touching, holding, dropping, throwing and more through existing virtual reality controllers. The VR scenarios allow for the trainee to master the procedures, through muscle memory and repetition (*Bruner, 2001*). As a procedure is practiced and repeated over time, long-term muscle memory is created for the task, eventually allowing it to be performed without conscious effort. This process decreases the need for attention and creates maximum efficiency (*Krakauer, Shadmehr, 2006).*

Trainees and employees have a fully immersive experience with no equipment availability constraints that permits them to do the procedure several times. Virtual Reality Scenarios possess no safety concerns for trainees or people that operate the machines, since the equipment is not physically present and is strictly virtualized into a computer software (*Alaraj, Lemole, Finkle and Yudkowsky,* 2011). Students and trainees can perform maintenance on machines that, without the use of virtual training, could cause severe injuries if training is not performed adequately. The virtual scenarios proposed will provide real time feedback when mistakes are made, if the procedure is not followed correctly or a safety rule is broken. This feedback will allow the trainer to understand that when maintenance is performed in live operations, outside of the VR environment, that those mistakes can cause severe injury or damage the machine.

1.4 Objectives

The main objectives of this research are as follows:

- Finding a mechanical maintenance process that would allow the creation of a Virtual Reality Tutorial
- Understand the process and the tools required to perform it efficiently
- Using 3D modeling software to create tools, environment objects and any 3D object necessary for the Virtual Environment
- The design of a realistic 3D environment that recreates a maintenance workshop, that allows the trainee to feel immersed
- Using C# programming, program the behavior of the 3D environment to have realistic behaviors, movements, interactions between trainee and Virtual World and the operations of tools.
- Create a Voice guided tutorial that will lead the trainee through the procedure and alert the trainee when mistakes have been made
- Create full interaction through Virtual Reality Hardware, specifically the Oculus Rift and Touch controllers

Chapter 2 Literature Review

2.1 VR History

Virtual Reality dates back to the 1930s. Pygmalions Spectacles, Stanley G. Weinbaum explains a goggle-based game in which individuals can watch a holographic recording of virtual stories including touch and smell. This amazing vision of the future would turn into what we think of as virtual-reality today (*Project Gutenberg, 2014).*

What we know now as Virtual Reality headsets actually started development in the 1960s.The first VR headset developed by Ivann Sutherland (Figure 1, Appendix 1) was created for military applications, specifically training purposes. These VR training applications are standard for military use nowadays (*Brockwell, 2016).*

VR Headsets began appearing in a few arcade games in the 1990s. Nintendo announced their first home VR System, called Virtual Boy (Figure 1, Appendix 1), and Sega introduced a Sega VR headset for the Sega genesis console in year 1993. These prototypes already contained LCD screens, head tracking and stereo sound (*Horowitz, 2004).*

In 2014 VR development took a large jump forward when the company Facebook, bought the Oculus VR system. At that time, it was a small company that developing a VR system and had a successful Kickstarter campaign in 2012. Oculus has fully functional VR systems for consumers to purchase and use at home, one of which is used throughout this research. Before Oculus and Facebook made a deal, VR headsets and technology were highly inaccessible and expensive for the regular consumer. At the time, systems were very technical and mostly used for Military training (*Metz, Cade, 2017).*

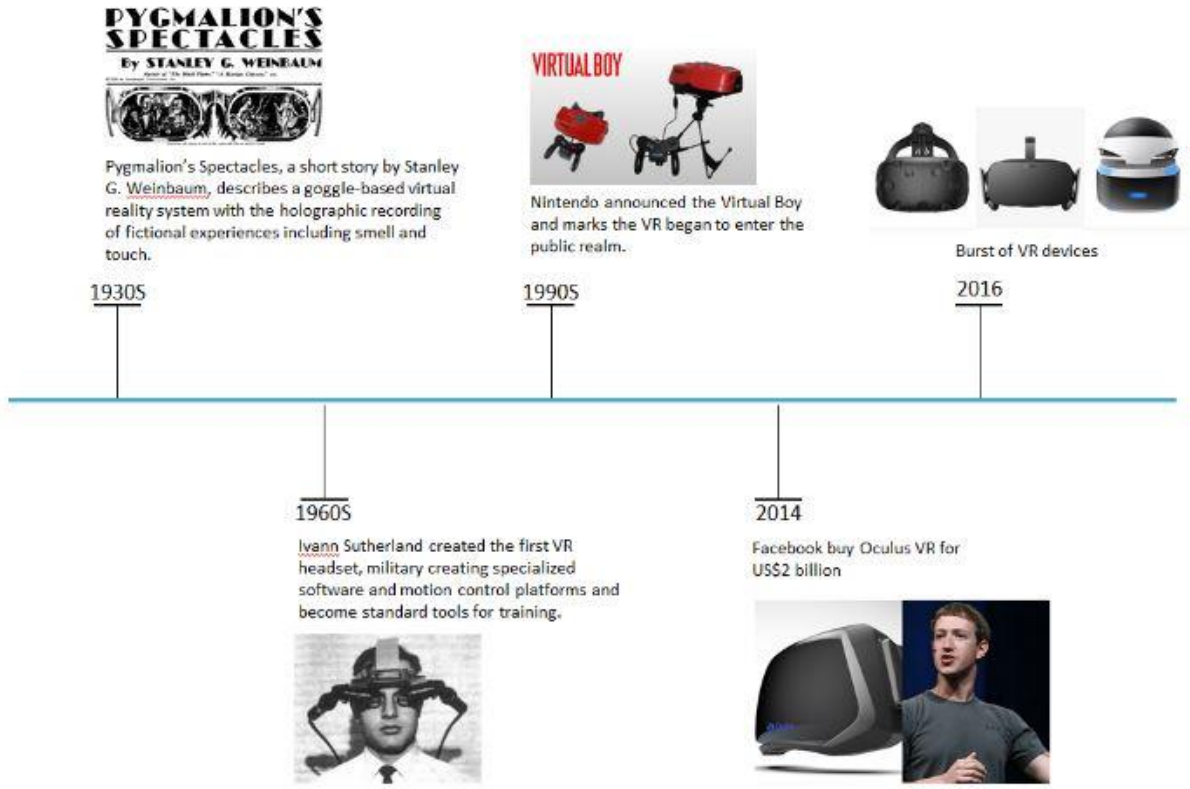


Figure 1 Virtual Reality Timeline (filmora.wondershare.com)

2.2 VR Training

Currently, it is becoming increasingly evident that educational and training institutions use different e-learning solutions. Various technologies and approaches have been combined for creation of new solutions, including learning management systems (LMS) which provide benefits for m-learning and tv-learning as well as other new training methodologies. Nevertheless, the use of virtual and augmented reality (VR/AR) technologies

are not widespread, even though the implementation of these technologies increases training effectiveness *(Cirulis, Arnis & Ginters, Egils. 2011).*

VR training has existed for over a decade, but has been mostly focused on military application, due to its high cost. Even now, VR is highly used in the military for training purposes. The army's responsibility is to provide realistic training for the individual soldier while mitigating risk. Thanks to emerging VR technology, this is becoming a reality. The Army has been using flight simulators for years, but in 2012 they opened their first fully immersive virtual simulation training system for soldiers in Fort Bragg. It is important to mention that VR training enhances training, it does not replace it (*Fort, Bragg, 2012).*

Thanks to the newest development and advancement of VR technology, mostly geared towards Videogames, VR is cheaper and more accessible than ever, not just for consumers but developers as well. This has allowed for designers to further explore the possibilities of VR, like training in the education and professional sectors.

2.3 Required Hardware

There were several hardware components required for this research to be completed efficiently. As the research advanced over the course of one year, the hardware too evolved.

2.3.1 Oculus Rift

The virtual reality headset hardware used for this research, is called Oculus Rift CV (Consumer Version) 1 (Figure 2). The Oculus Rift uses an OLED panel for each eye, each

having a resolution of 1080 x 1200. These panels have a refresh rate of 90Hz. The lenses allow for a wide field of view. The combination of the high refresh rate, global rate, global refresh and low persistence means that the user experiences none of the motion blurring or judder that is experienced on a regular scree. *(Hollister and Buckley, 2015).* Headphones are included and integrated in the headset which provide real time 3D audio. The headset has a full 6 degrees of freedom and rotational and positional tracking thanks to the Oculus Constellation tracking system (Figure2) *(Oculus.com, 2011).* This VR headset included the headset itself, an Xbox controller, a simple controller for menu navigation and Oculus Constellation system. Oculus constellation system is the headset positional tracking system, which tracks the users head and device position. It consists of an external infrared tracking sensor (*VR Focus, 2015).*

Figure 2 Oculus Rift and Oculus Sensors (oculus.com)

2.3.2 Oculus Touch Controllers

The Oculus controllers were still under development when this research began. The Oculus Rift Controllers were acquired for the research as soon as they were released for public use (Figure 3). They consist of a pair of handheld units, one for each hand, they contain analog sticks, three buttons and two triggers. The controllers are fully tracked in a 3D space by the Constellation system which allow them to be represented in the virtual environment.

Figure 3 Oculus Touch Controllers (oculus.com)

2.3.3 Computers

The Oculus Rift operating system has very specific hardware requirements for the computer utilizing it; specifying a CPA equivalent to at least Intel Core i5, minimum of 16GB of RAM, at least an NVIDA GeForce GTX 970 graphics card, an HDMI output, three USB 3.0 ports and one 2.0 USB port. A Dell desktop computer provided by Morehead State University School of Engineering and Information Systems and a Dell laptop bought by the student, specifically for this research meeting and exceeding all of these requirements, were used throughout the research.

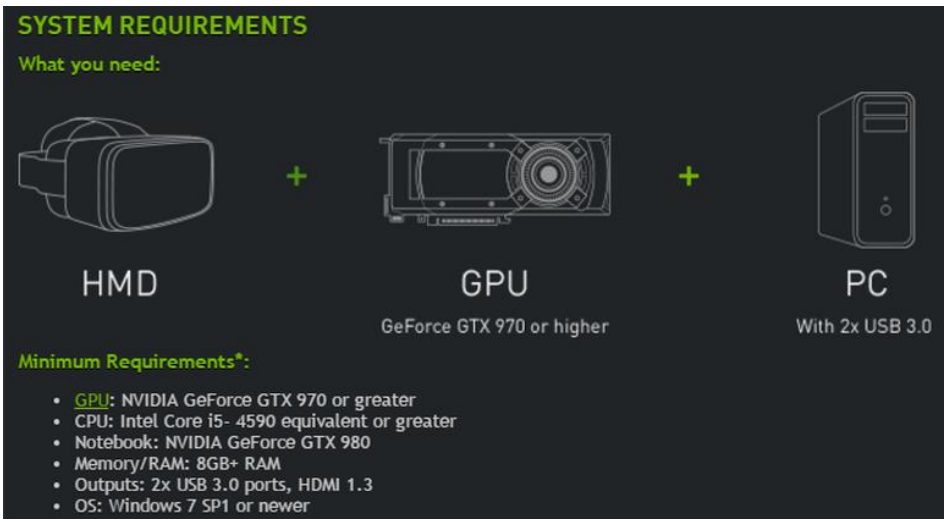


Figure 4 Oculus System Requirements (oculus.com)

2.4 Required Software

Different software were used for this research. From 3D modeling software to specific Game engines. Beneath is a list that provides the descriptions of what each were used for.

2.4.1 Unity

Unity Game Engine's latest personal (free) versions as they were released through the research; used for the creation of the 3D Environment. Unity also allows scripting using C# which is the programming language used through this research. Oculus has also developed SDKs to create easy interaction and communication between their hardware and the Game Engine, several SDKs where used. The Oculus SDKs, are built to help quickly and easily develop VR applications. Some of the Oculus SDKs used were "Oculus Utilities for Unity", "Oculus Avatar SDK" and "Oculus Platform SDK".

2.4.2 Autodesk 3DS Max

This software is a professional 3D computer graphics program specifically for making 3D animations, models, games and images. It was made and developed by Autodesk Media and Entertainment and the license used was a free student version. This software was used as a bridge between the 3D design modeling software, Solid Works, and the video Game Engine, Unity. 3D models are normally heavy files with complicated structures that are hard to render when several are put together in one same 3D environment. 3Ds Max simplified these models during the research to make them lighter and editable for the Game Engine.

2.4.3 Solid Works

3D modeling was a large part of the design and implementation of the Virtual Reality training environment. This research utilized Solid Works, a solid modeling computer aided design (CAD) and computer aided engineering (CAE) computer program. Morehead State University school of Engineering and Information Systems provided this software. In the development of a 3D environment, every object existing in the VR world was created using 3D modeling. A number of the 3D models used throughout this research were downloaded from different free cloud-based websites. 3D models not available through online cloud-based websites were created specifically for this research using Solid Works.

2.4.4 Autodesk Maya

Another software used throughout the research specifically to create an animation that allows the trainee to see how the motor used for the Virtual Scenario gets disassembled, was Autodesk Maya. Maya is a 3D computer graphics application. It is used to create interactive 3D applications, including video games, animated films, TV series and visual effects. The full free educational version was used.

Chapter 3 Methodology

The proposed methodology is composed of the stages described below:

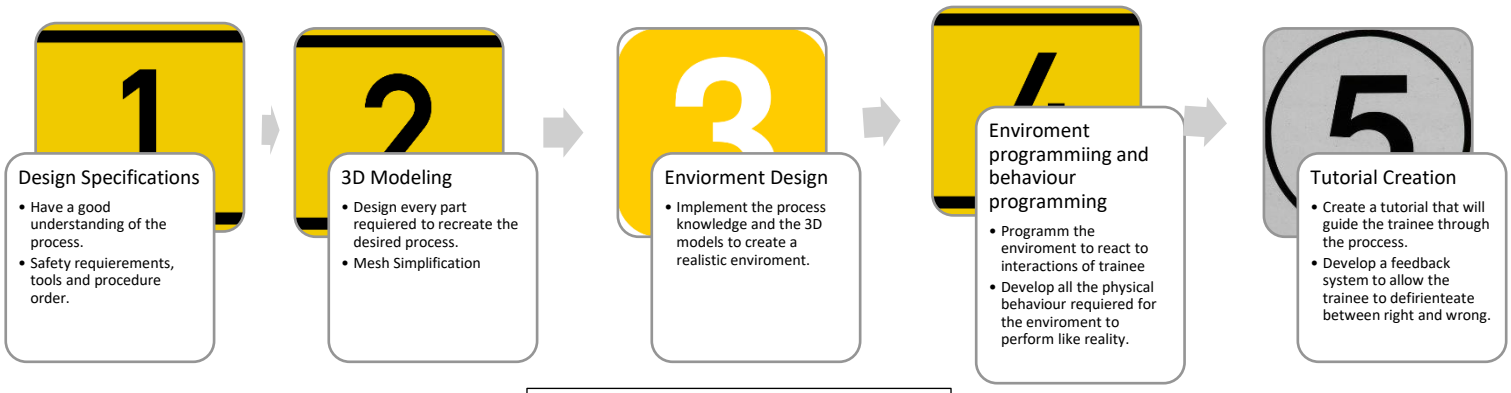


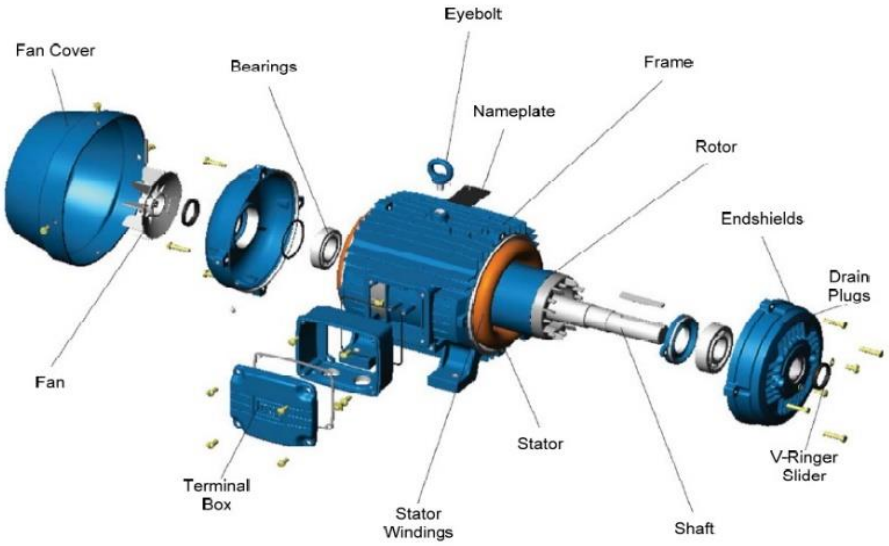
Figure 5 Methodology Diagram

The development of the Virtual Scenarios for maintenance training applications, has several different steps. Each step plays a vital role in the final implementation of the environment for training. The steps must be completed in the correct sequence for the process to work effectively and in a timely manner (Figure 5).

3.1 Design Specifications

The first step of the development process starts with the design specifications. The environment recreates a real-life application in the desired field, this research focuses on mechanical maintenance. The mechanical process or machine that the virtual process is going to recreate has to be analyzed and understood, as well as practiced with. Once the designer has a good understanding of the process, specifications and tools on whichever part of the process the tutorial will be created for, machines and physical parts needed are established

and the first step is complete. The scenario discussed in this paper focuses on the disassembly, bearing maintenance, and re-assembly of an AC motor (Figure 6).



A typical 3-phase induction motor [Courtesy of Electromotors WEG SA, Brazil]

Figure 6 Motor Components and

An AC motor is an electric motor driven by an alternating current. AC motors are commonly used in industrial settings for different applications including, but not limited to, pumps, blowers and conveyors, which is why AC electric motors are essential to numerous operations, no matter the industry. If a motor fails, it can mean costly downtime since they are essential for making sure plants are running smoothly and effectively; making their maintenance a priority. The specific process concerning the maintenance of this motor is the replacement of rusted bearing (Figure 2). The motor selected for this training includes several parts that have to be removed in order to exchange the bearings.

Parts to be removed	**Quantity**
Fan cover	1
Fan	1
Non-Drive Endshield	1
Drive Endshield	1
Rotor	1
Bearings	2

Table 1 Motor Parts to be disassemble

The parts listed in Table 1 have to be removed in the correct descending order. The real-life motor also requires the removal of bolts and nuts before the steps in Table 1, but this Virtual Scenario does not focus on small hardware.

The process of selection for the desired application that will be virtualized is extensive. There are thousands of different possibilities of manufacturing training applications that can be virtualized. For this research, the application selected had to meet certain requirements in order to be considered for selection. The requirements are listed in Table 2, below.

Requirements	**Met**
Available at MSU Engineering Department	**Yes**
Easy to 3D model, or 3D models already available	**Yes**

There is a need to perform maintenance on a regular basis for Manufacturing facilities	**Yes**
Found in most manufacturing facilities	**Yes**
Assembly consists of more than 5 individual parts	**Yes**
Special or specific tools necessary to perform maintenance	**Yes**
A specific order is required in order for the maintenance to be successful	**Yes**

As seen in the table above, the process selected for this research met all the requirements necessary for the process to be a success and possibly create an impact for Engineers, and Engineering Students. The first requirement that was dictated was to find a process that Morehead State University had available, meaning that, if necessary, the resources (tools and equipment) were available to perform the same procedure physically. The second requirement was to find a process that didn't required labor intensive 3D modeling. 3D modeling can be a very long process. Being able to create a realistic environment requires a multitude of differing parts. If the main component is already available for download or was not too complicated to model (Example: AC Motor vs Airplane), then the amount of work will decrease significantly. An already existing 3D model of an AC motor was available for download (Figure 10). Following some adjustments, this 3D model was deemed suitable for this application. Other important requirements included, the need in the industry for this specific task and making sure the task was not specific to one process or sector. In order for the virtual training environment to work, the process had to

include several components to be disassembled in the right order, adding a level of difficulty, which gave the scenario increased purpose.

The second part of the design specifications step focused on the selection of the tools needed for the specific process. In this case, disassembly of the motor to get to the old bearings will be done manually without the use of tools. The specific task and end goal of this scenario was the replacement of old bearings. For the trainee to be able to perform this task successfully in real life as well as in the virtual environment, the use of a specific tool called a bearing puller is required (Figure 7).

Figure 7 Real Bearing Puller (hillshire.ie)

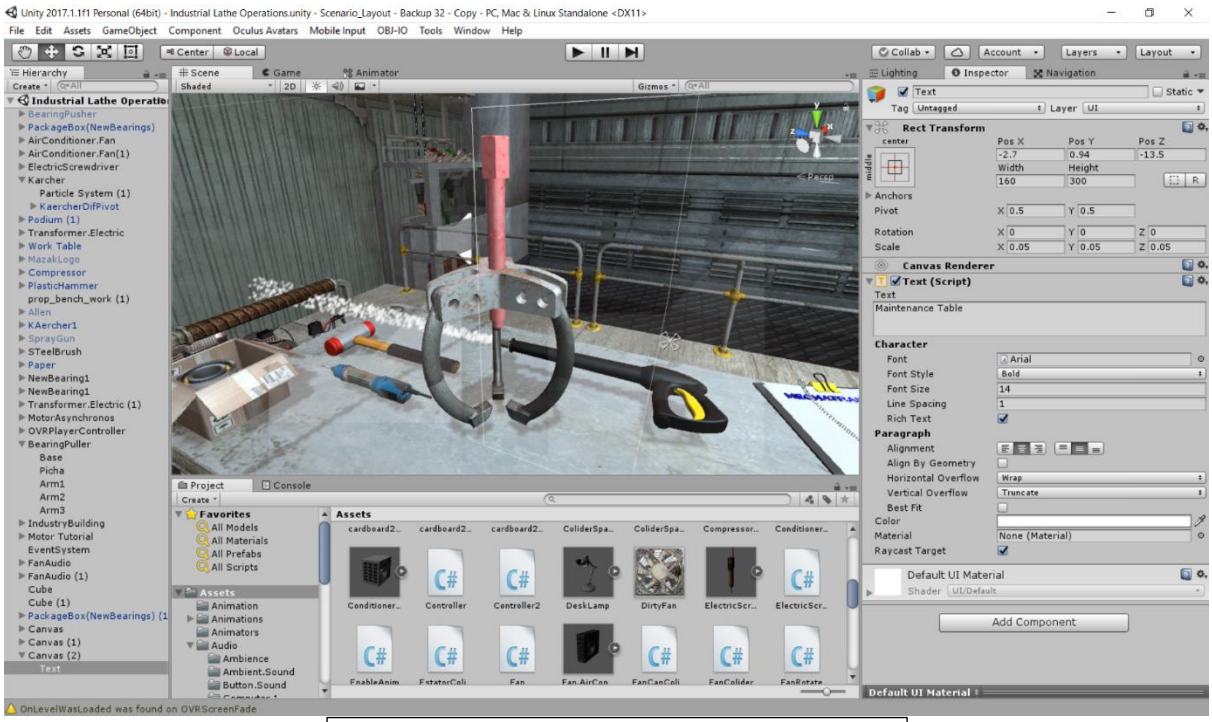


Figure 8 Bearing Puller in the Virtual Scenario

The trainee has to identify the tool and use it correctly in the virtual environment to be able to complete the procedure successfully. This will eventually teach the trainee the correct tool for a determined application. More tools will be added into the scenario for visual purposes, some of which will be operational, such as an electrical screw driver.

3.2 3D Modeling

The second part of the process starts once the parts necessary for the development of the scenarios are listed. The parts can either be provided through existing 3D cad models or can be created through 3D modeling software's such as Solid Works, Solid Edge, NX9, etc. (Figure 9).

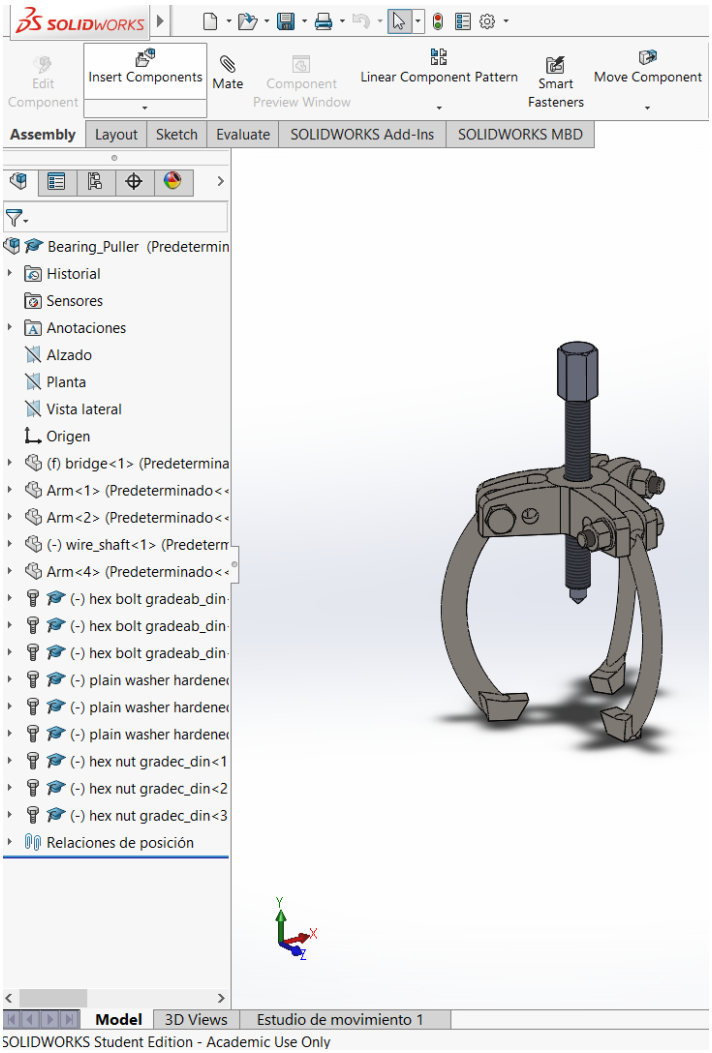


Figure 9 3D Model Bearing Puller Using Solid Works

A list, description, and an explanation of the purpose of the 3D models used for the scenario is provided below. Pictured above is a 3D model of the bearing puller in Solid Works (Figure 9) shown as an example of the 3D modeling process, as well as a figure of the bearing puller in the Virtual Environment (Figure 8) which is the result.

Table 3 List of 3D Models

	3D Models		
	Name:	**Description:**	**Behaviour:**
1	Bearing Puller	Tool used during the tutorial.	Closes and Opens to grab bearings
2	Package Box	Box that holds the new bearing.	None
3	Air Conditioner Fan	Visual	Is turning constantly providing a visual effect, it also shows vapor coming out of it.
4	Electric Screwdriver	Visual and physical purposes. Provides Ineraction	It turns when the specific botton is pressed it also has a sound effect.
5	Podium	Provides the bottos to start, stop and reset the Motro Tutorial	Activates the voice commands and starts and stops the motor tutorial bottons change color when pressed.
6	Transformer	Visual	None
7	Work Table	Visual	None
8	Compressor	Visual	None
9	Plastic Hammer	Visual and physical purposes. Provides Ineraction	The trainee can grab it and use it.
10	Bench Work	Visual	None
11	Allen	Visual and physical purposes. Provides Ineraction	The trainee can grab it and use it.
12	Kaercher	Visual and physical purposes. Provides Ineraction	The trainee can grab it and use it, visual effect of water and sound effect.
13	Spray Gun	Visual	None
14	Steel Brush	Visual	None
15	Paper	Visual	None
16	New Bearing	The ultimate goal of the scenario is to replace old bearings with this new bear	The bearing can be grabbed by the bearing puller only.
17	Old Bearing	The ultimate goal of the scenario is to replace these old bearings.	The bearing can be grabbed by the bearing puller only.
18	Industry Building	Houses the scenario	None
19	Motor Tutorial	The most important piece of equipment in the scenario, the motor is an animation that shows the steps that the motor requieres to be dissasembled when the start button on the podium is pressed.	The motor has an animation that shows the steps in a correct order for its dissasemble, the motor also displays the names of its parts when the trainee touches them.
20	Motor Assembly	This motor is the same one as the tutorial motor with the difference that there is no animation. The trainee has interaction with this motor.	The motor allows for the trainee to grab each individual part and dissasemble the motor, the motor has to be dissasemble in the correct order for the trainee to be able to interact with the different parts.
21	Rails and Stairs	Visual	None
22	Lamps	Visual	None

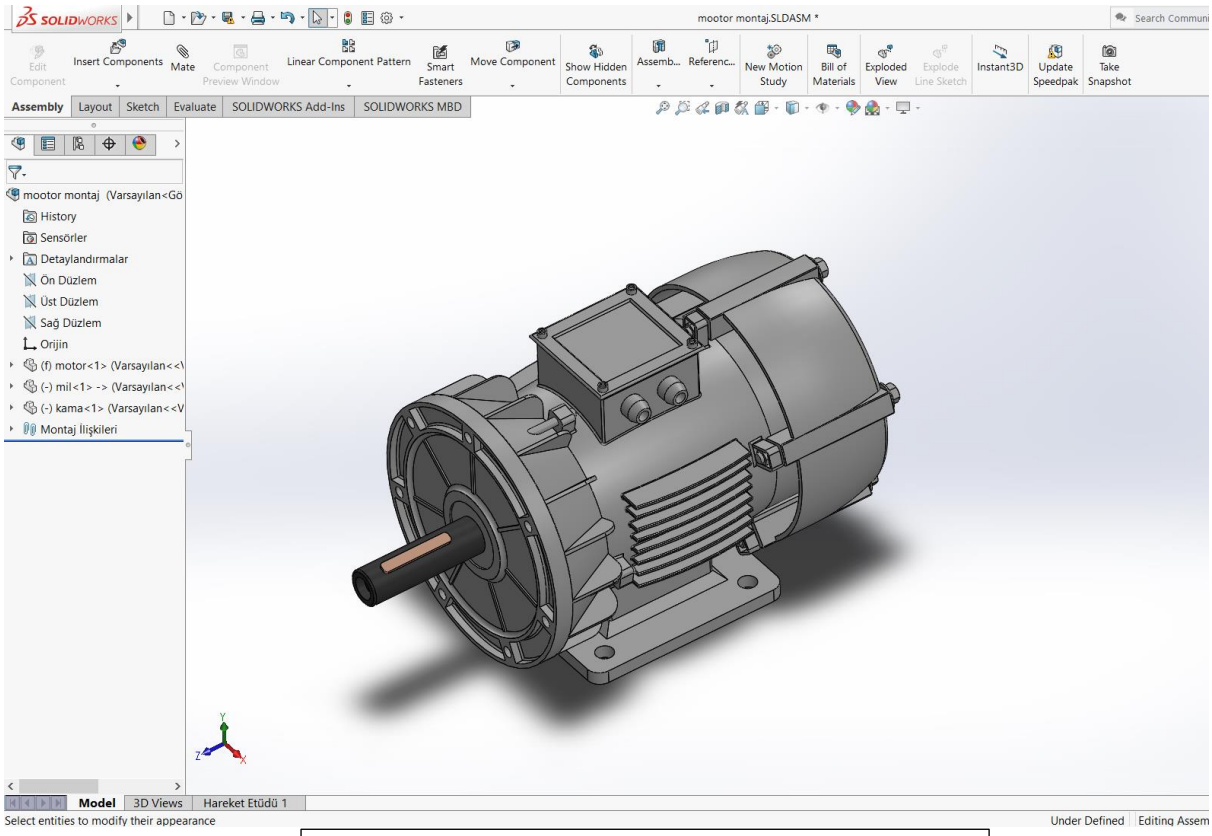


Figure 10 3D Model of AC motor using Solid Works

3.2.1 Mesh Simplification

Solid works creates heavy 3D models that are composed of thousands of different triangles, lines and shapes (Figure 11). This creates a really detailed file that has a lot of weight. When translating this file into a readable extension for the Virtual Environment such as FBX, we encounter a lot of problems. For this reason, once the 3D model of the desired part is obtained or created, the mesh is simplified by using a different software than that of 3D modeling (in this case 3D StudioMax; a software by Autodesk for modeling, animation and rendering). (3DS Max 2017) A mesh is a collection of vertices, edges and faces that describe the shape of a 3D object; a vertex being a single point (Tobler and Maierhofer, 2016) (Bruce Baumgarten, 1975). Every 3D model is composed of meshes. The more complicated the mesh, the more detailed and better the object will look in the final Virtual

Reality Scenario, but more complex meshes equal a heavier overall file. Meshes also allow for the 3D objects to be given textures and colors, giving the finished scenario a more realistic look. 3D models with a lot of meshes will create complications when importing several at once into the Game Engine creator, Unity. Each object has to be edited individually and then imported. Once the object has been exported from 3D StudioMax into Unity, the meshes cannot be altered. This means that the process between the Game Engine and 3D StudioMax must be done several times until the optimal condition is found. The "sweet spot", where the object will be detailed enough to look good in the virtual environment but light enough for Unity to manage it, must be found. Complications in the game engine creator occur because Unity must render every single object that the person wearing the Virtual Reality is looking at, as well as calculate the physical behavior of this objects in real time. The objects' mesh must be divided depending on the parts of the object that have to move or have different colors and textures. If a part of the 3D model will not be necessary for the desired tutorial application this object can either be eliminated or simplified. The Game Engine has a specific file type that it accepts, so 3D Studio Max software also allows to change the file type from models with endings like .SLDPRT or .SLDASM into an .FBX, which is the type that Unity supports. Figure 11 illustrates an example of an object with 3 differing mesh qualities. The object furthest to the right has thousands of meshes which adds a lot of detail. The object to the far left is simplified to just a few, making it easy to be imported but with a decreased quality. Figure 12 shows how the simplified mesh object located in the far left appears to be flat and does not have round corners. This figure shows the relationship between meshes and quality of a model.

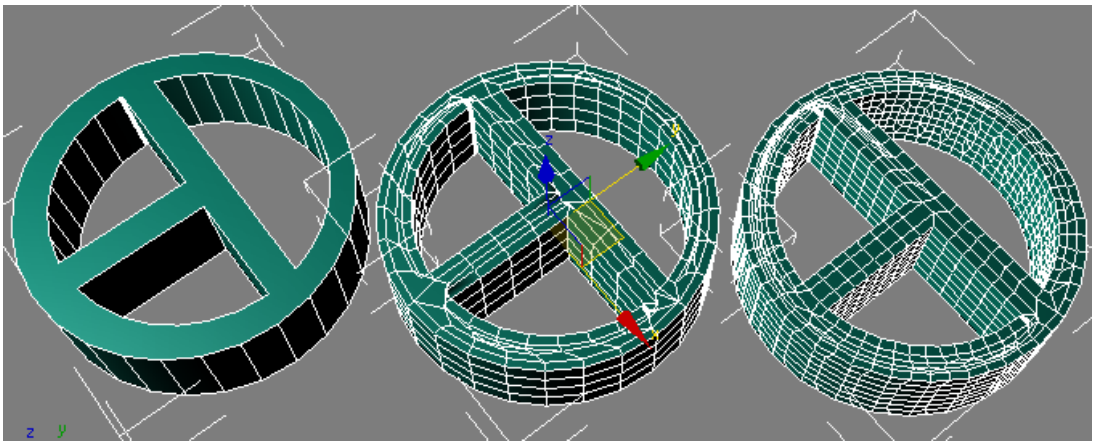


Figure 11 Mesh Comparison

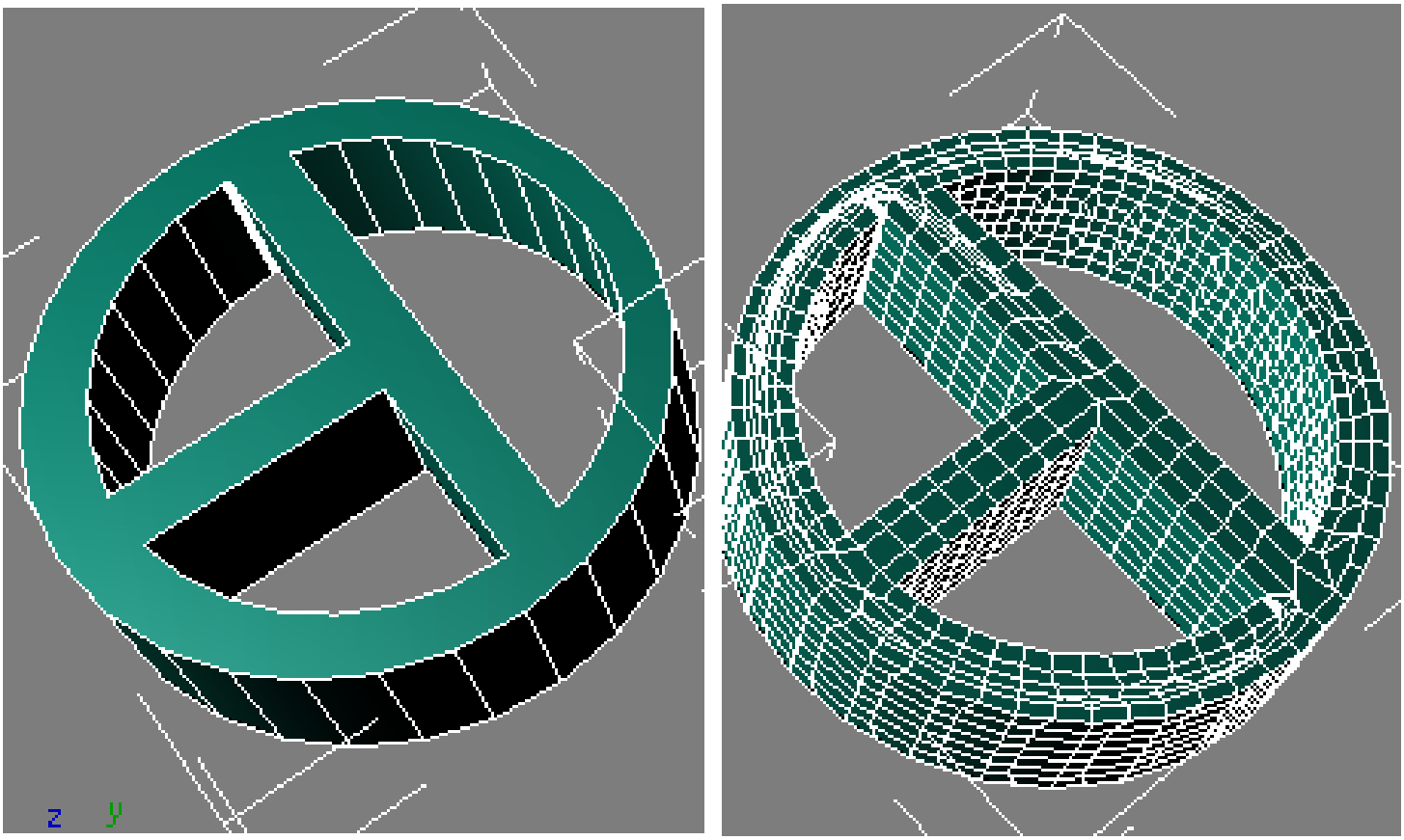


Figure 12 Mesh Comparison Close-up

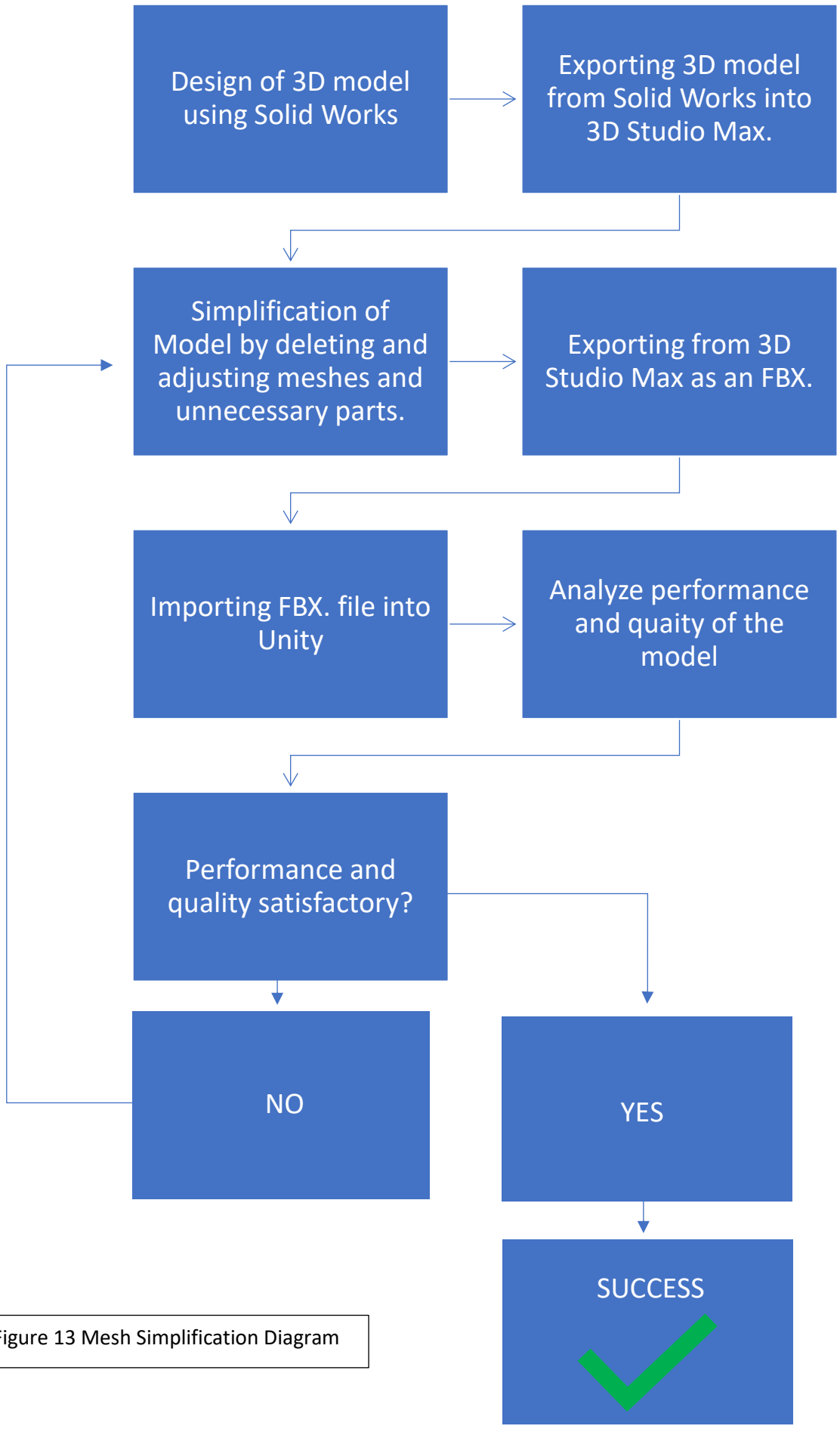


Figure 13 Mesh Simplification Diagram

Once the mesh simplification process has been successfully completed, we are able to add textures and color to the models. The type and amount of colors and textures a specific model can take depends on the format of the original file, the mesh simplification and the amount of components on our model. As an example, Figure 14 shows an electrical screw driver in 3D Studio Max. You can see how the model is divided into 3 components, each of which can have a different texture or color.

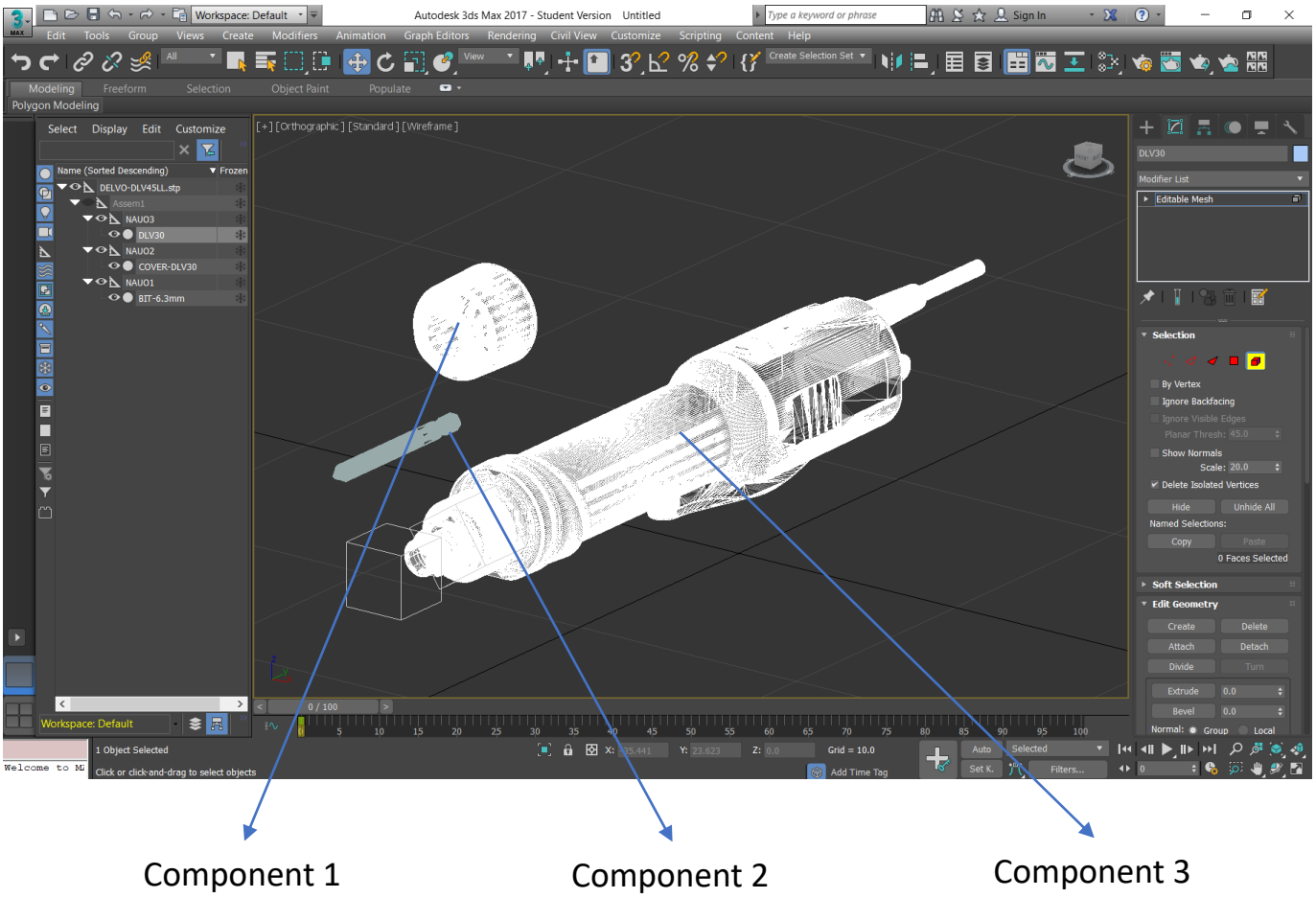


Figure 14 Mesh Simplification Example

The screw driver the component highlighted in red, Figure 15, is the body of the screw driver. Due to the complexity of the meshes, this model cannot be separated further,

which does not allow for the body of the screwdriver to have different textures or colors. A models complexity depends entirely on the way the original file was designed in a 3D modeling software. During this research, several models were downloaded from open source websites, which gave no design control over some of the components. Conversely, some reverse engineering had to be performed on such models. Every component underwent the same mesh simplification process in order to reach the beset rendering quality.

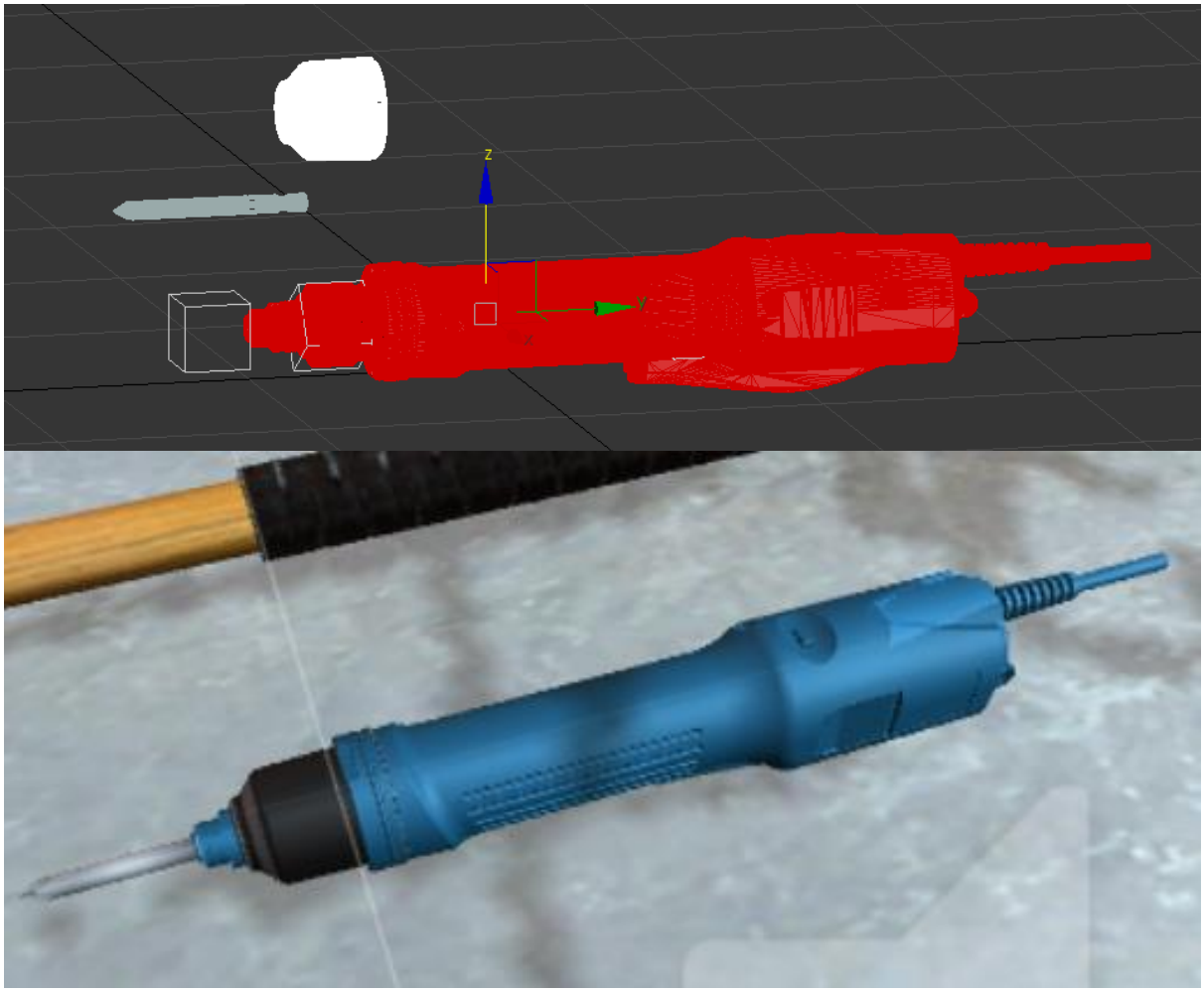


Figure 15 Mesh Simplification Example

3.3 Environment Design

3.3.1 Textures

The next step when designing the environment is adding textures. This step is a really important part of giving the realistic look and feel to the virtual environment. Textures can be added in different forms. Textures can be added as simple colors or more complicated images that give objects a material like finish. Textures work as a stamp from an image of a certain material or color, as seen in Figure 16.

Figure 16 Textures Example

The images seen above work as materials when inported into unity. They wrap around objects and give them the desired appearance. As seen Figure 17 the bearing puller has a meterial texture that creates the appearanc of rusted old metal. This is found originally as a simple close up image from the rusted metal. Then imported into unity and added to the shader tab under the objects properties, as seen on Figure 18 (Appendix 2).

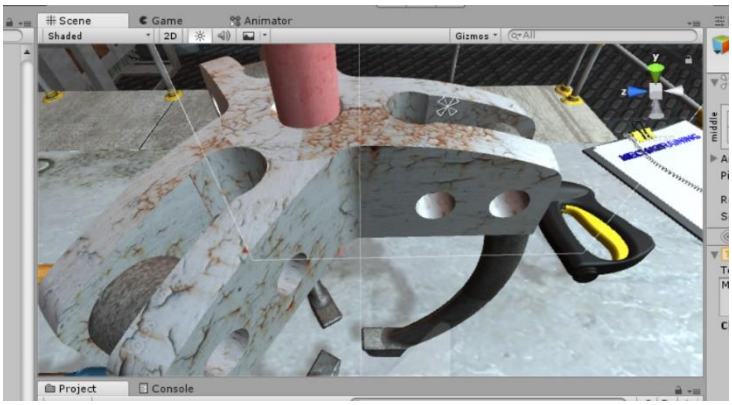


Figure 17 Virtual Scenario Texture Example

As seen on Figure 18 (Appendix 2) under the right side of the objects properties and under shader properties all the different options have to be adjusted. The options include, Tiling, Offset, Emission, Secondary Maps, etc. These options get adjusted in a trial and error method until the desired or best result is acquired. Every single object in the virtual scenario underwent this process, no objects were left without textures or colors.

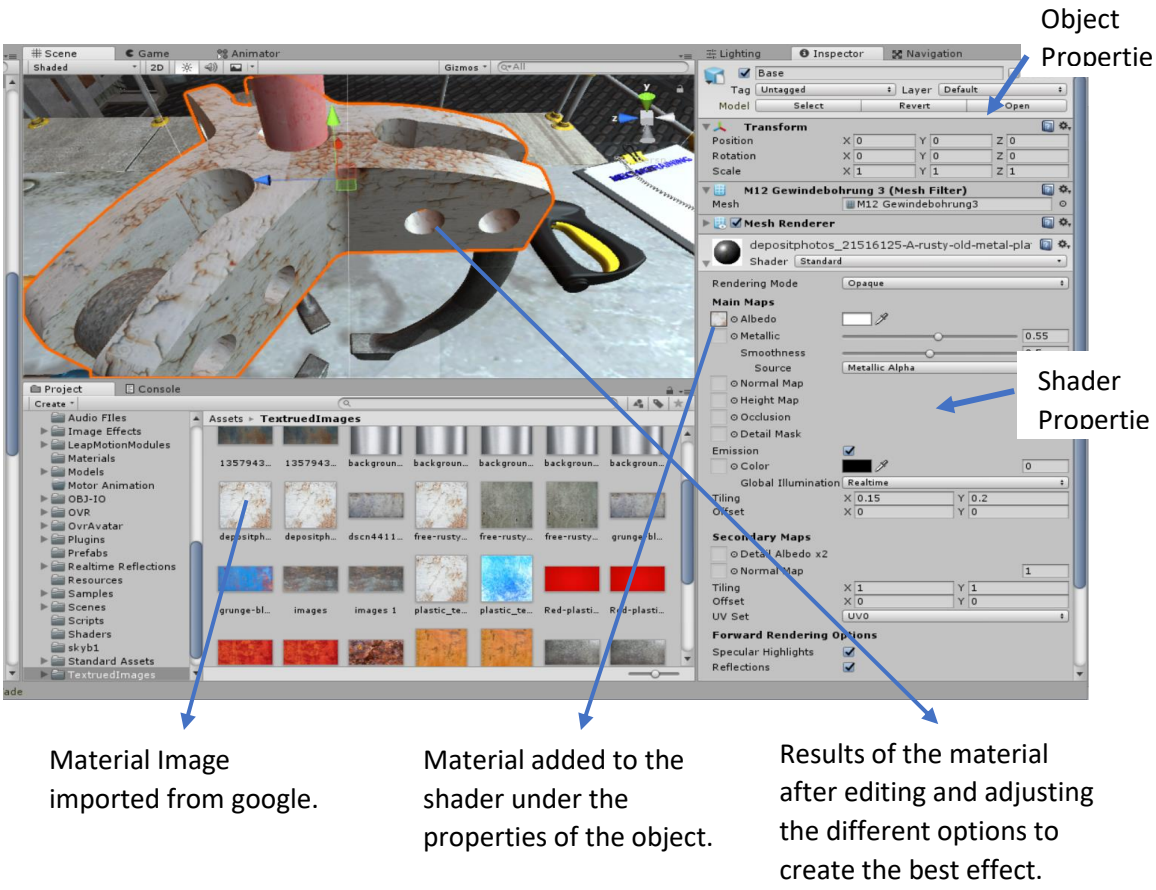


Figure 18 Textures

A different way that textures can be applied to objects to create a realistic experience, is by adding images to objects as textures. As an example, Figure 19 (Appendix 3) displays a simple cube created in Unity; this cube has no texture, color or any type of shader.

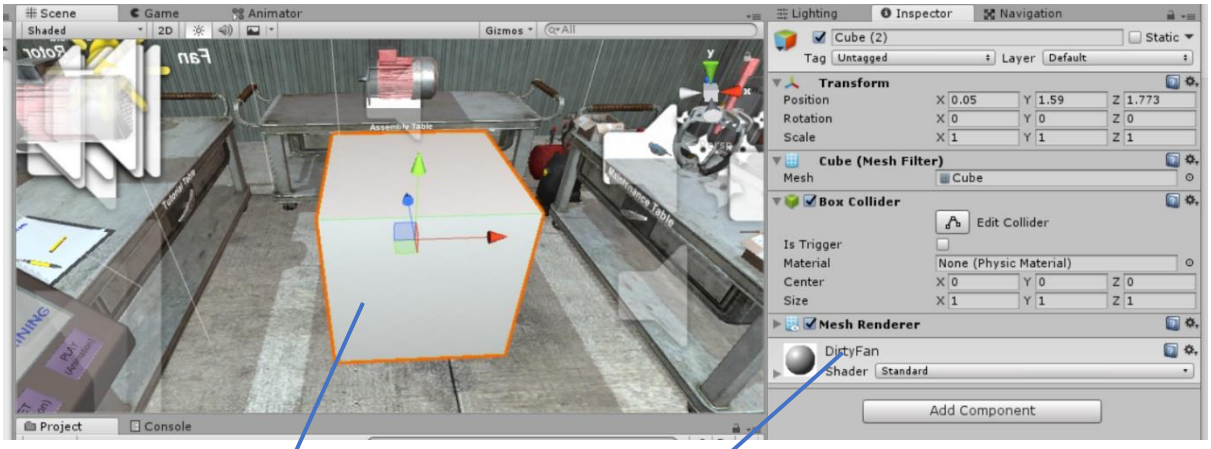


Figure 19 Textures

Figure 20 shows the image of a fan. This image gets imported into unity and used as a texture.

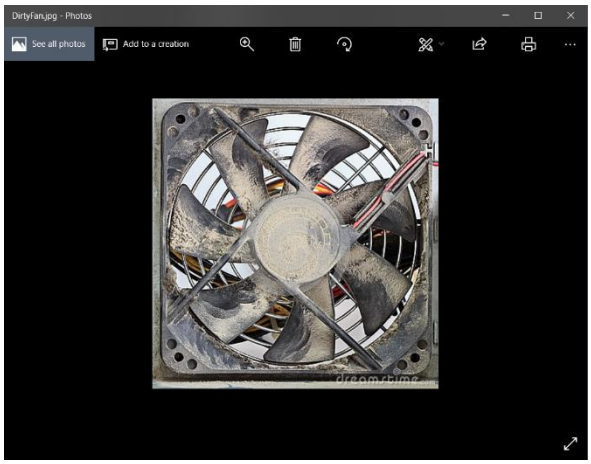
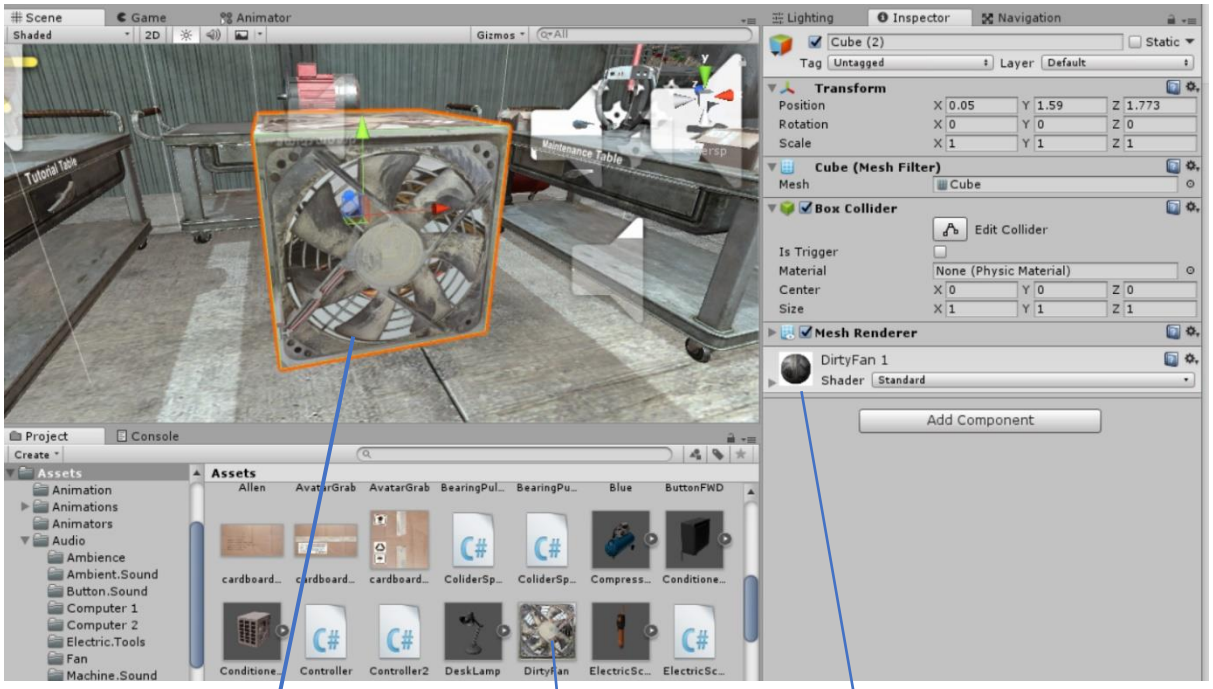


Figure 20 Textures

Figure 21 (Appendix 4) displays the cube with an image of the fan showed in Figure 20 applied as a texture, which gives the cube the appearance of being a fan.



Cube with the texture applied.

Image imported

Image used as a texture

Figure 21 Texture Results

3.3.2 Lights, Sounds and Special Effects.

Some other important things to consider when designing a Virtual Environment are secondary effects. These effects are the ones that the person using the virtual environment normally would not notice directly. They influence the environment through the background with things such as lighting, shadows, sound, tactic feedback in the controllers and smoke effects.

Unity the introduction of different types of lights, like directional lights, point lights, spotlights, and area lights (Figure 24). Every light has a different purpose and creates a

different look. Figures 22 and figure 23 compare the scene with and without light effects. Also, Unity automatically creates shadows on objects if the shadow feature is selected.

Figure 22 Bad Lighting

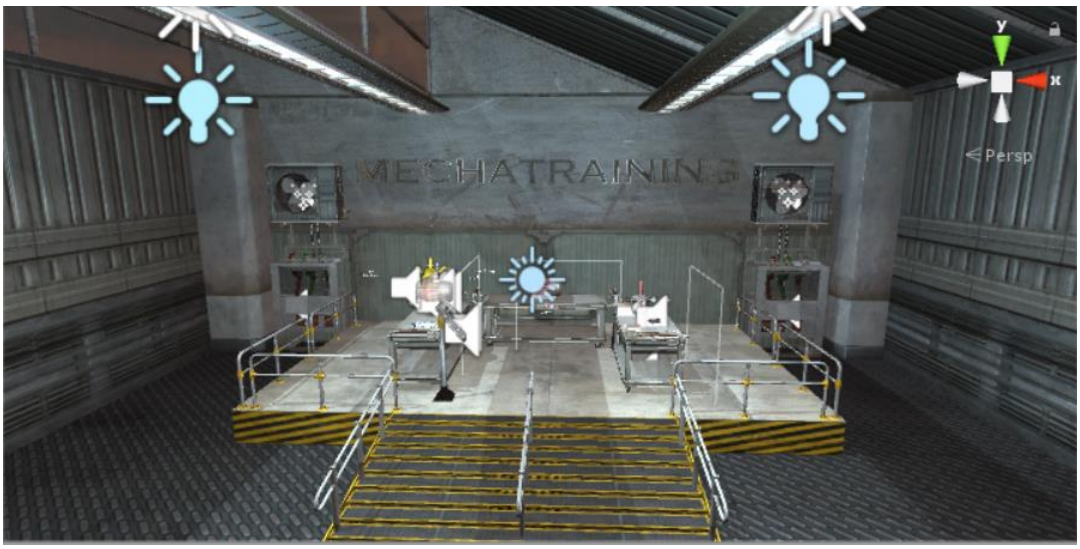


Figure 23 Good Lighting

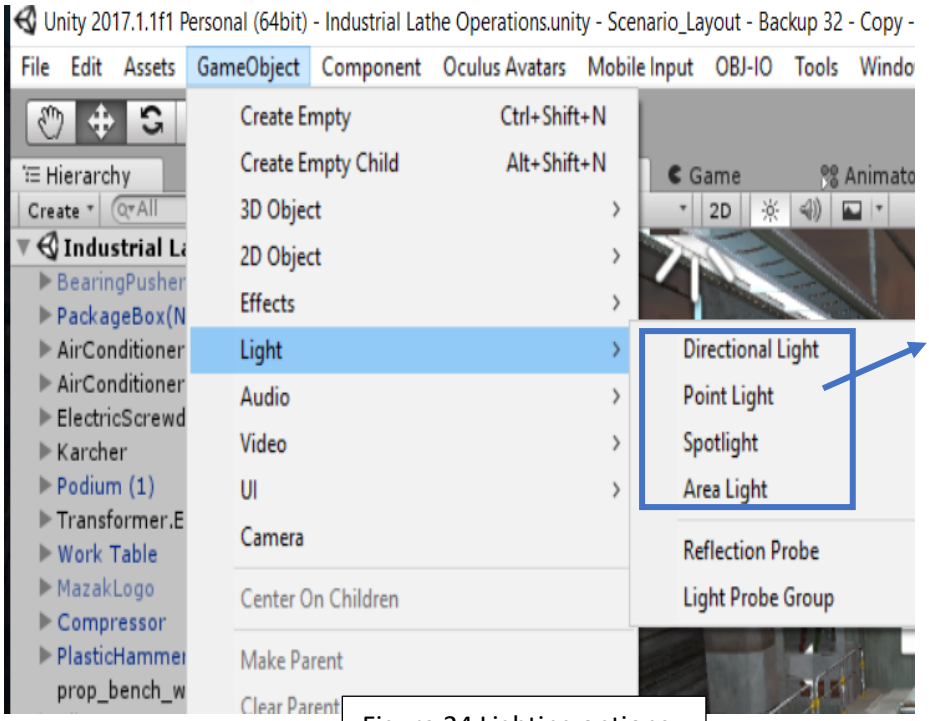


Figure 24 Lighting options

The smoke effect used in this scenario was used in conjunction with a movement script (More on scripts will follow). In Figure 25 we can observe the smoke coming out of the industrial fan. Simple effects like this provide a sense of realism that gets subconsciously noticed.

Figure 25 Smoke Effect

The sounds effects used for this scenario are air movements, machine sounds, as well as electric static. These details allow the trainee to feel immersed in the virtual world. Unity has the capability to create and add audio sources to objects (Figure 26), which also adds a realistic 360 audio effect since the sounds comes out of the determined object. If the subject gets closer to the object the sound will increase. The sounds also get translated to the earphones from the direction of the object. The sound source shown below was electrical static.

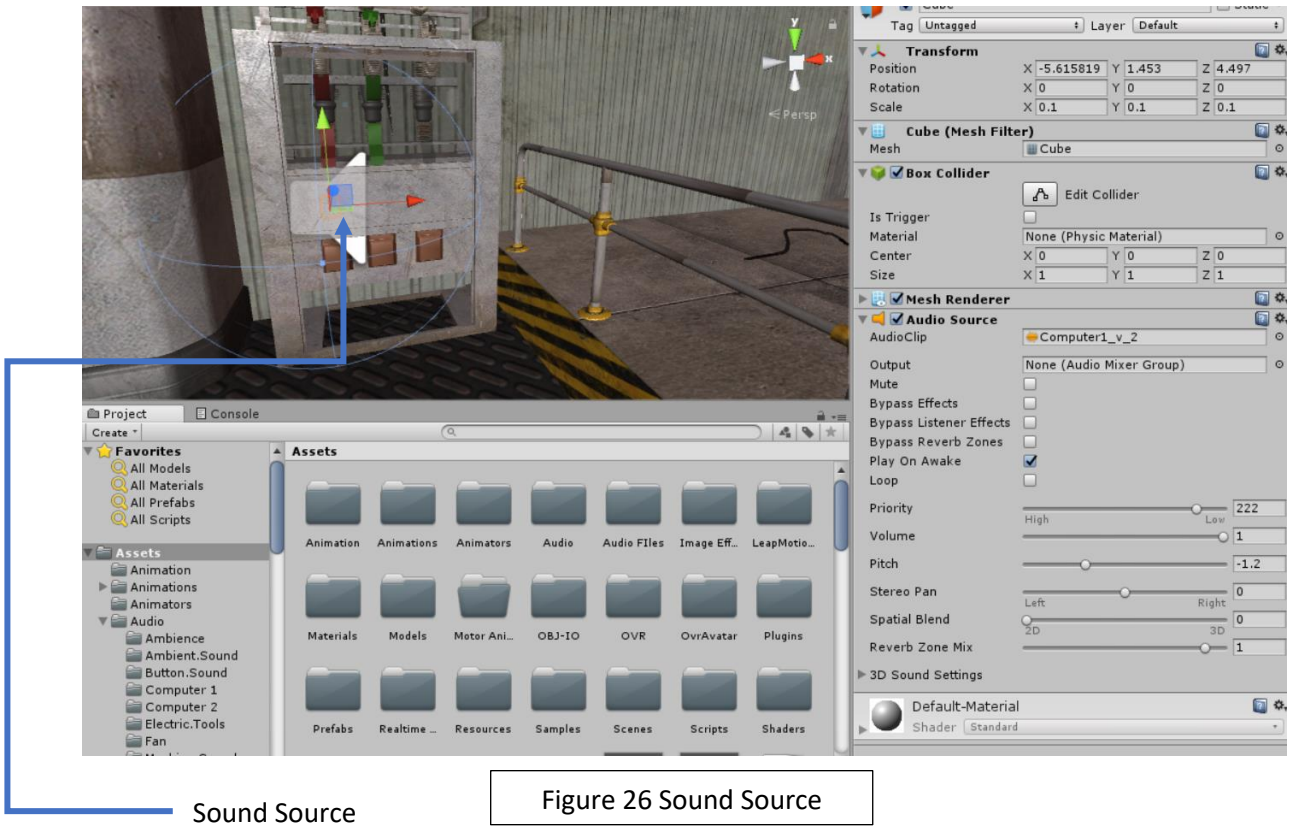


Figure 26 Sound Source

3.4 Environment Programming

3.4.1 Colliders and Rigid bodies

Now that the environment is created and looks and feels realistic, the physical behavior of the machines and/or parts that the tutorial aims to cover must be programmed. Just like in real life, Unity is capable of implementing real characteristics to objects such as mass, gravity, weight, drag, force, buoyancy and more (Figure 27).

Unity describes its physical behaviors as followed:

"To have convincing physical behavior, an object in a game must accelerate correctly and be affected by collisions, gravity and other forces. Unity's built-in physics engines provide components that handle the physical simulation for you. With just a few parameter settings, you can create objects that behave passively in a realistic way (i.e., they will be moved by collisions and falls but will not start moving by themselves). By controlling the physics from scripts, you can give an object the dynamics of a vehicle, a machine, or even a piece of fabric."

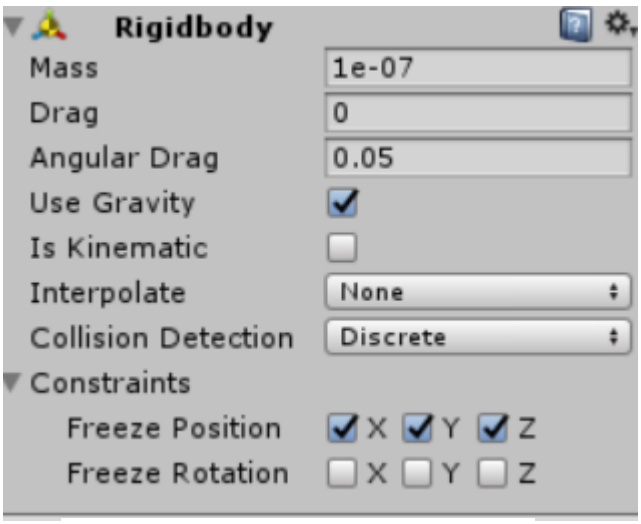


Figure 27 Rigidbody

In order for a component or object (called Game Objects in Unity) to work, a rigid body has to be added to it. With a rigid body attached, the object will respond to gravity. The rest of the characteristics are achieved through the addition of something called colliders to objects. Colliders are a region that surrounds the object which delineates the space of contact with other objects (Ortega-Moody, Sánchez-Alonso, González-Barbosa and Reyes-Morales, 2016). Colliders play a big role in the creation of virtual reality laboratories since, without them, every object would have no physical interaction with the trainee or other objects. There are different types of colliders available in Unity: Box collider, Sphere collider, Capsule collider, and Mesh collider. In Figure 30 and 31 below, we can observe the stairs in the virtual scenario where we can observe one box collider per step (Figure 27a). Having one collider per step allows for the stairs to work as a ramp. The colliders in the first-person controller which serves as the character as seen on Figure 28 (Appendix 5) reacts physically with the colliders in the stairs, as well as any other colliders in the environment for example the ground or walls.

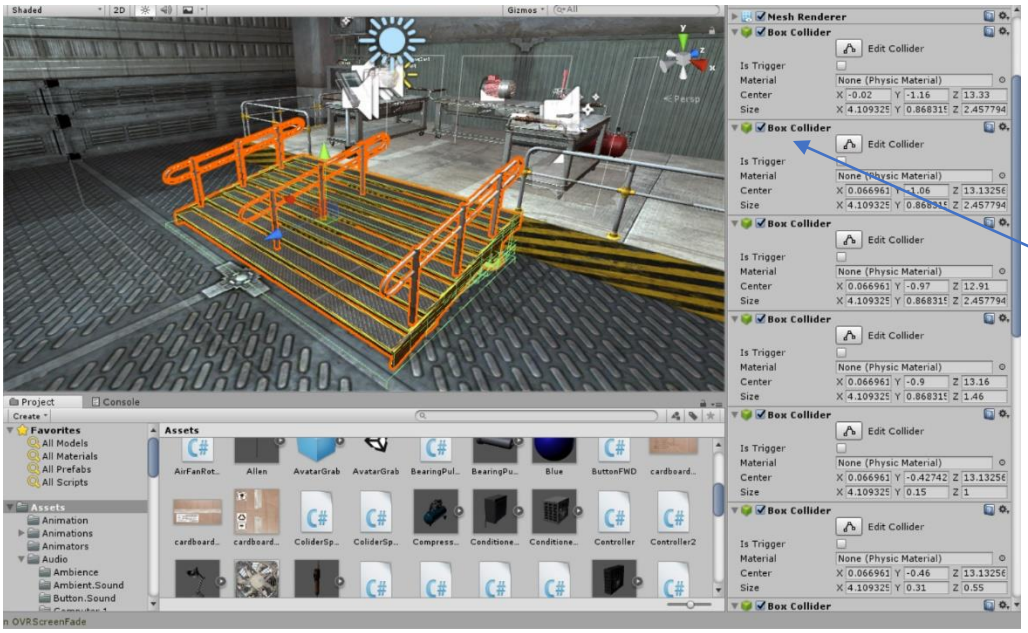
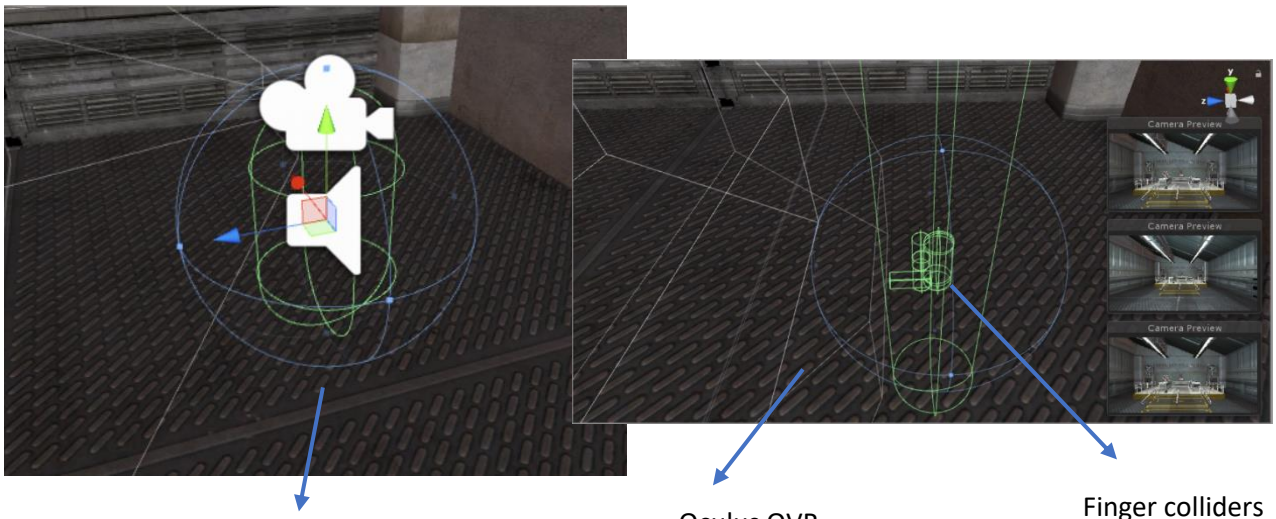


Figure 27a Box Colliders in Steps



Unity First person controller

Oculus OVR Player Controller

Finger colliders for hand interaction with objects.

Figure 28 Character Controllers

The first-person controller seen in Figure 28 (Appendix 6) is preloaded into unity this allows you to control the character through the keyboard and display the image in your computer screen. The controller and character used for the purpose of this research was one of an SDK made by Oculus specifically for Unity. This controller includes the scripts for the VR reality headset and Virtual Reality Controllers, which allow for hand motions. As seen on Figure 28 (Appendix 6) there is several colliders on the OVR plater controller, the purpose of this colliders are the hands and fingers which will allow the physical interaction with objects, throughout the scenario.

Colliders from the overall scenario

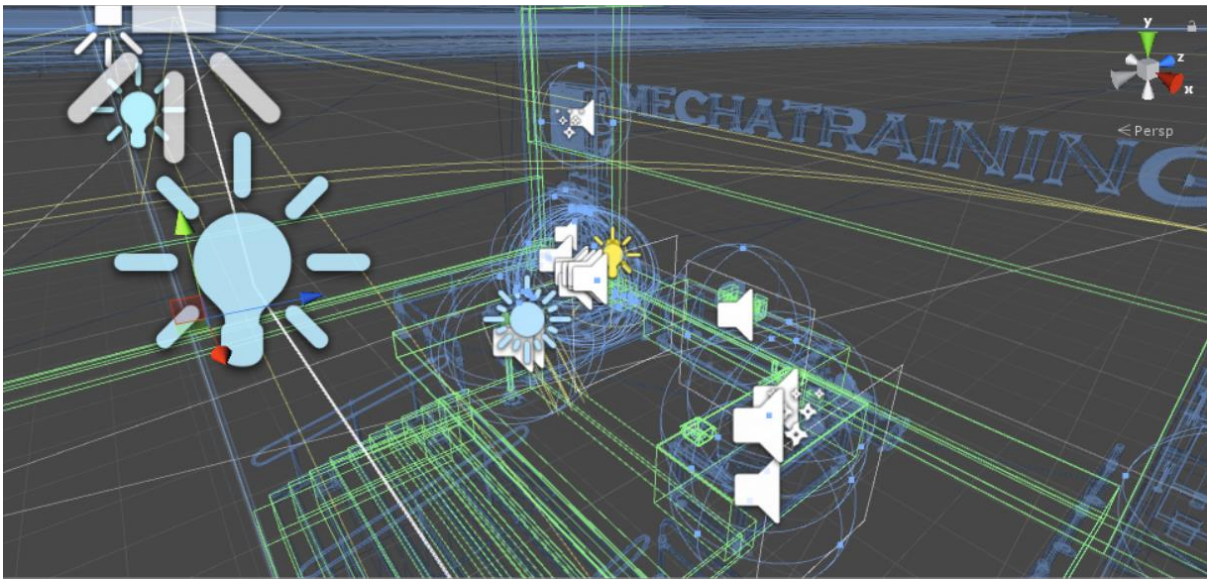


Figure 29 Colliders

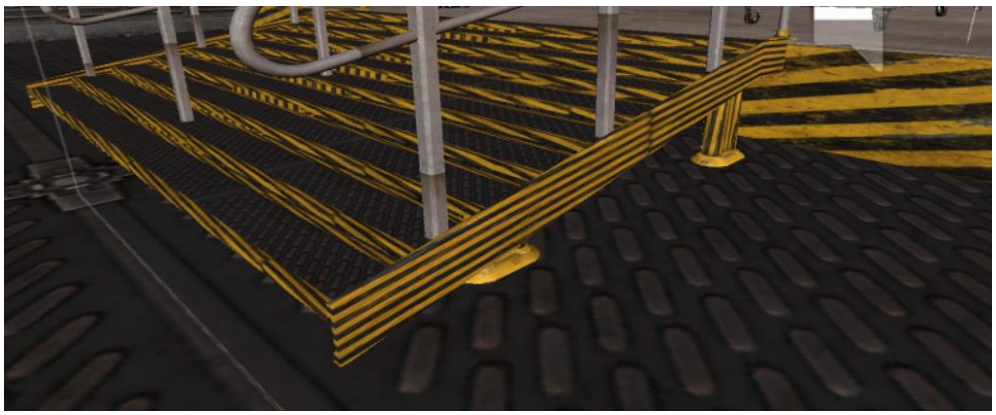
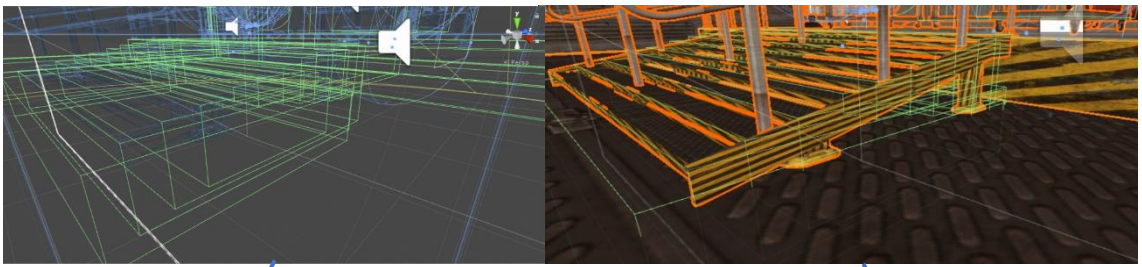


Figure 30 Virtual Scenario Stairs



Colliders from the stairs shown in wireframe mode.

Colliders from the stairs shown over the stairs.

Figure 31 Stairs Collider

3.4.2 Scripting Programming

Once all the objects have colliders and physical characteristics, their behavior related to the trainee can be programmed.

Most of the physical behavior that relates to interactions with the trainee as well as movements have to be programmed using scripting. As Unity states on their website:

> "Scripting is an essential ingredient in all games. Even the simplest game needs scripts, to respond to input from the player and arrange for events in the gameplay to happen when they should. Beyond that, scripts can be used to create graphical effects, control the physical behavior of objects"

Unity allows for two programming languages C# and Java. C# was the programming language used for creating the physical behavior of the Virtual Reality Laboratory for Mechanical Maintenance. Programmers can choose between C# or Java depending on experience or preference since both have the same functionality. Some of the functions that were programmed using C# in Unity for the Mechanical Maintenance scenario include:

- the automated reproduction of the assembly and disassembly of a motor which serves as a tutorial.
- The interaction between the trainee which allows to grab parts in the environment such as parts of the motor for disassemble (Figure 36) and assemble.
- The interaction of the trainee and the tools (Figure 35), the interaction between tools and parts for example a bearing puller and the bearings.
- The system that allows for the motor to be only assemble and disassemble in the correct order.

Through the Unity developer's window (Figure 32, Appendix 7), we can observe the motor that the trainee has to disassemble. In the right side of the window we can see some of the tools that this specific Game Object has, including a script called "Controller". This complex script allows the trainee to grab each individual part of the motor and assemble as well as disassemble it. The scrip only permits the trainee to perform these tasks in the exact order and also provides audio feedback when the action is performed correctly. The script window is composed of several variables. Unity creates these variables in the script and physically displays them in order to see the behavior and monitor the programming as it is happening in real time.

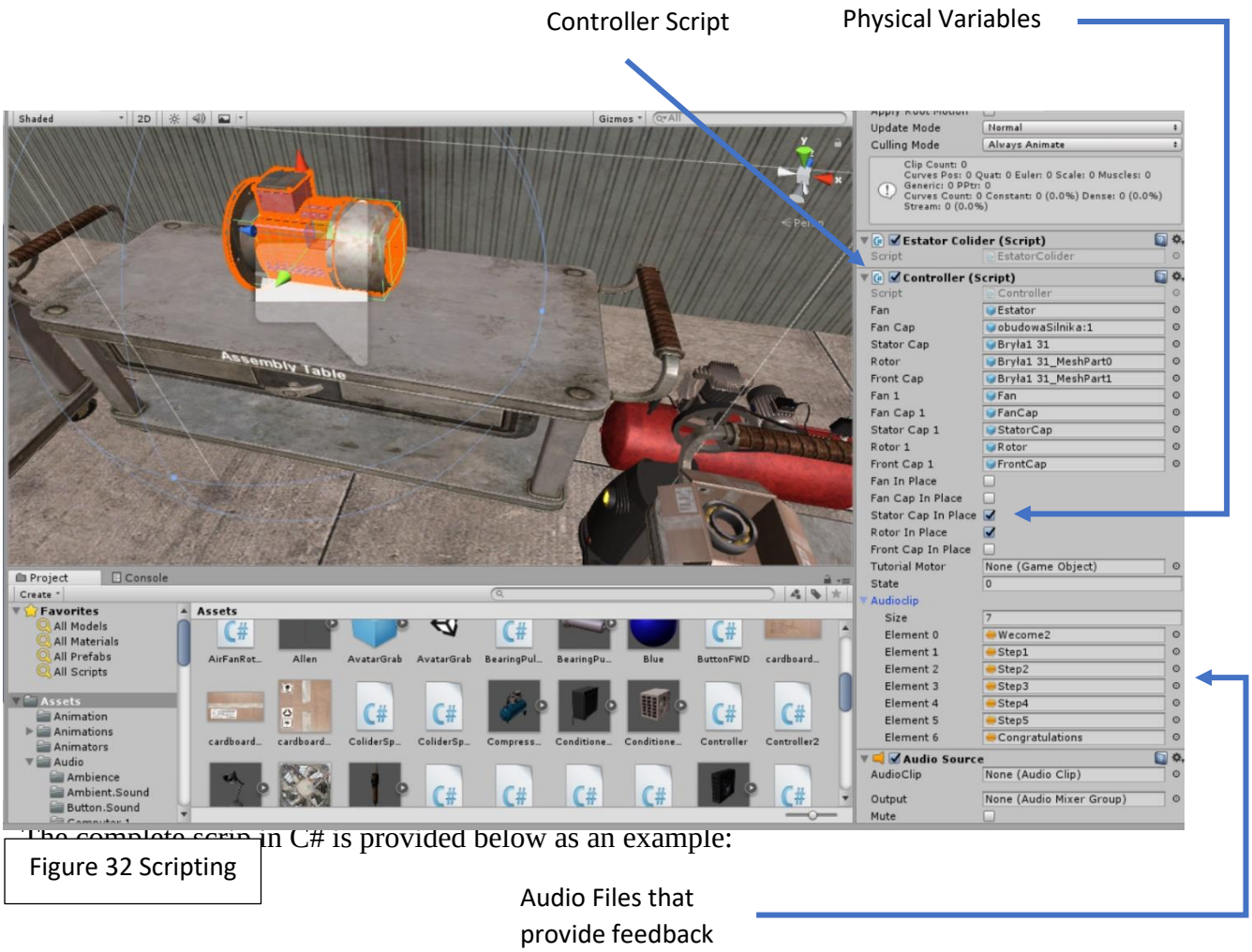


Figure 32 Scripting

The complete scrip in C# is provided below as an example:

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

public class Controller : MonoBehaviour {
    public GameObject Fan;
    public GameObject FanCap;
    public GameObject StatorCap;
    public GameObject Rotor;
    public GameObject FrontCap;

    public GameObject Fan1;
    public GameObject FanCap1;
    public GameObject StatorCap1;
    public GameObject Rotor1;
    public GameObject FrontCap1;

    public bool FanInPlace;
    public bool FanCapInPlace;
    public bool StatorCapInPlace;
    public bool RotorInPlace;
    public bool FrontCapInPlace;

    public GameObject TutorialMotor;

    public int state;

    public AudioClip[] audioclip;


    void PlaySound(int clip)
    {
        AudioSource audio = GetComponent<AudioSource>();
        audio.clip = audioclip[clip];
        audio.Play();
    }


    // Use this for initialization
    void Start () {
        state = 1;
        PlaySound(0);

    }                                                                                   apInPlace == false)

    // Update is called once per frame
    void Update () {



        FanInPlace = Fan.GetComponent<ColiderSpanPosition>().bandera;
        FanCapInPlace = FanCap.GetComponent<ColiderSpanPosition>().bandera;
        StatorCapInPlace = StatorCap.GetComponent<ColiderSpanPosition>().bandera;
        RotorInPlace = Rotor.GetComponent<ColiderSpanPosition>().bandera;
        FrontCapInPlace = FrontCap.GetComponent<ColiderSpanPosition>().bandera;


        switch (state)
        {
            case 1:


                Fan1.gameObject.transform.localPosition = new Vector3(0.08674575f, -0.02191839f, -0.135f);
                Fan1.gameObject.transform.localEulerAngles = new Vector3(-90f, 0f, 0f);
                StatorCap1.gameObject.transform.localPosition = new Vector3(0.08674575f, -0.02191839f, -0.08499998f);
                StatorCap1.gameObject.transform.localEulerAngles = new Vector3(-180f, 0f, 0f);
                Rotor1.gameObject.transform.localPosition = new Vector3(0.1407f, -0.0762f, -0.229f);
                Rotor1.gameObject.transform.localEulerAngles = new Vector3(0f, 0f, 0f);
                FrontCap1.gameObject.transform.localPosition = new Vector3(0.08674575f, -0.02191839f, 0.08499998f);
                FrontCap1.gameObject.transform.localEulerAngles = new Vector3(0f, 0f, 0f);

                if (FanInPlace == true
                    && StatorCapInPlace == true
                      && RotorInPlace == true
                      && FrontCapInPlace == true
                      && FanCapInPlace == false)
                {
                    PlaySound(2);
                    state = 2;


                }
                else
                {
                    state = 1;
                }

                break;
            case 2:


                /* Fan1.gameObject.transform.localPosition = new Vector3(0.08674575f, -0.02191839f, -0.135f);
                 Fan1.gameObject.transform.localEulerAngles = new Vector3(-90f, 0f, 0f);*/
                StatorCap1.gameObject.transform.localPosition = new Vector3(0.08674575f, -0.02191839f, -0.08499998f);
                StatorCap1.gameObject.transform.localEulerAngles = new Vector3(-180f, 0f, 0f);
                Rotor1.gameObject.transform.localPosition = new Vector3(0.1407f, -0.0762f, -0.229f);
                Rotor1.gameObject.transform.localEulerAngles = new Vector3(0f, 0f, 0f);
                FrontCap1.gameObject.transform.localPosition = new Vector3(0.08674575f, -0.02191839f, 0.08499998f);
                FrontCap1.gameObject.transform.localEulerAngles = new Vector3(0f, 0f, 0f);

                if (FanInPlace == false
                    && StatorCapInPlace == true
                    && RotorInPlace == true
                      && FrontCapInPlace == true
                     && FanCapInPlace == false)
                {
                    PlaySound(3);
                    state = 3;
```

```
                /* Fan1.gameObject.transform.localPosition = new Vector3(0.08674575f, -0.02191839f, -0.135f);
                Fan1.gameObject.transform.localEulerAngles = new Vector3(-90f, 0f, 0f);
                StatorCap1.gameObject.transform.localPosition = new Vector3(0.08674575f, -0.02191839f, -0.08499998f);
                StatorCap1.gameObject.transform.localEulerAngles = new Vector3(-180f, 0f, 0f);*/
                /*FrontCap1.gameObject.transform.localPosition = new Vector3(0.08674575f, -0.02191839f, 0.08499998f);
                FrontCap1.gameObject.transform.localEulerAngles = new Vector3(0f, 0f, 0f);*/
            Rotor1.gameObject.transform.localPosition = new Vector3(0.1407f, -0.0762f, -0.229f);
            Rotor1.gameObject.transform.localEulerAngles = new Vector3(0f, 0f, 0f);

            Fan.GetComponent<ColiderSpanPosition>().ObjectName = "Tyler";
            FanCap.GetComponent<ColiderSpanPosition>().ObjectName = "Tyler";
            StatorCap.GetComponent<ColiderSpanPosition>().ObjectName = "Tyler";
            FrontCap.GetComponent<ColiderSpanPosition>().ObjectName = "FrontCap";
            Rotor.GetComponent<ColiderSpanPosition>().ObjectName = "Rotor";

            if (Rotor.GetComponent<ColiderSpanPosition>().bandera == true
                && FrontCap.GetComponent<ColiderSpanPosition>().bandera == true
                && StatorCap.GetComponent<ColiderSpanPosition>().bandera == false
                && FanCap.GetComponent<ColiderSpanPosition>().bandera == false
                && Fan.GetComponent<ColiderSpanPosition>().bandera == false)
            {
                state = 8;
            }
            else
            {
                state = 7;
            }
            break;

            case 8: //Assembly Order

            /* Fan1.gameObject.transform.localPosition = new Vector3(0.08674575f, -0.02191839f, -0.135f);
            Fan1.gameObject.transform.localEulerAngles = new Vector3(-90f, 0f, 0f);
            StatorCap1.gameObject.transform.localPosition = new Vector3(0.08674575f, -0.02191839f, -0.08499998f);
            StatorCap1.gameObject.transform.localEulerAngles = new Vector3(-180f, 0f, 0f);*/
            FrontCap1.gameObject.transform.localPosition = new Vector3(0.08674575f, -0.02191839f, 0.08499998f);
            FrontCap1.gameObject.transform.localEulerAngles = new Vector3(0f, 0f, 0f);
            Rotor1.gameObject.transform.localPosition = new Vector3(0.1407f, -0.0762f, -0.229f);
            Rotor1.gameObject.transform.localEulerAngles = new Vector3(0f, 0f, 0f);


            Fan.GetComponent<ColiderSpanPosition>().ObjectName = "Tyler";
            FanCap.GetComponent<ColiderSpanPosition>().ObjectName = "Tyler";
            StatorCap.GetComponent<ColiderSpanPosition>().ObjectName = "StatorCap";
            FrontCap.GetComponent<ColiderSpanPosition>().ObjectName = "FrontCap";
            Rotor.GetComponent<ColiderSpanPosition>().ObjectName = "Rotor";

            if (Rotor.GetComponent<ColiderSpanPosition>().bandera == true
                && FrontCap.GetComponent<ColiderSpanPosition>().bandera == true
                && StatorCap.GetComponent<ColiderSpanPosition>().bandera == true
                && FanCap.GetComponent<ColiderSpanPosition>().bandera == false
                && Fan.GetComponent<ColiderSpanPosition>().bandera == false)
            {
                state = 9;
            }
```

```
            }
            else
            {
                state = 8;
            }

            break;

        case 9: //Assembly Order
                /* Fan1.gameObject.transform.localPosition = new Vector3(0.08674575f, -0.02191839f, -0.135f);
               Fan1.gameObject.transform.localEulerAngles = new Vector3(-90f, 0f, 0f);*/
               StatorCap1.gameObject.transform.localPosition = new Vector3(0.08674575f, -0.02191839f, -0.08499998f);
               StatorCap1.gameObject.transform.localEulerAngles = new Vector3(-180f, 0f, 0f);
            FrontCap1.gameObject.transform.localPosition = new Vector3(0.08674575f, -0.02191839f, 0.08499998f);
            FrontCap1.gameObject.transform.localEulerAngles = new Vector3(0f, 0f, 0f);
            Rotor1.gameObject.transform.localPosition = new Vector3(0.1407f, -0.0762f, -0.229f);
            Rotor1.gameObject.transform.localEulerAngles = new Vector3(0f, 0f, 0f);

            Fan.GetComponent<ColiderSpanPosition>().ObjectName = "Fan";
            FanCap.GetComponent<ColiderSpanPosition>().ObjectName = "Tyler";
            StatorCap.GetComponent<ColiderSpanPosition>().ObjectName = "StatorCap";
            FrontCap.GetComponent<ColiderSpanPosition>().ObjectName = "FrontCap";
            Rotor.GetComponent<ColiderSpanPosition>().ObjectName = "Rotor";

            if (Rotor.GetComponent<ColiderSpanPosition>().bandera == true
                && FrontCap.GetComponent<ColiderSpanPosition>().bandera == true
                && StatorCap.GetComponent<ColiderSpanPosition>().bandera == true
                && FanCap.GetComponent<ColiderSpanPosition>().bandera == false
                && Fan.GetComponent<ColiderSpanPosition>().bandera == true)
            {
                state = 10;
            }
            else
            {
                state = 9;
            }


            break;


        case 10: //Assembly Order

             Fan1.gameObject.transform.localPosition = new Vector3(0.08674575f, -0.02191839f, -0.135f);
             Fan1.gameObject.transform.localEulerAngles = new Vector3(-90f, 0f, 0f);
            StatorCap1.gameObject.transform.localPosition = new Vector3(0.08674575f, -0.02191839f, -0.08499998f);
            StatorCap1.gameObject.transform.localEulerAngles = new Vector3(-180f, 0f, 0f);
            FrontCap1.gameObject.transform.localPosition = new Vector3(0.08674575f, -0.02191839f, 0.08499998f);
            FrontCap1.gameObject.transform.localEulerAngles = new Vector3(0f, 0f, 0f);
            Rotor1.gameObject.transform.localPosition = new Vector3(0.1407f, -0.0762f, -0.229f);
            Rotor1.gameObject.transform.localEulerAngles = new Vector3(0f, 0f, 0f);
```

```
                Fan.GetComponent<ColiderSpanPosition>().ObjectName = "Fan";
                FanCap.GetComponent<ColiderSpanPosition>().ObjectName = "FanCap";
                StatorCap.GetComponent<ColiderSpanPosition>().ObjectName = "StatorCap";
                FrontCap.GetComponent<ColiderSpanPosition>().ObjectName = "FrontCap";
                Rotor.GetComponent<ColiderSpanPosition>().ObjectName = "Rotor";

                if (Rotor.GetComponent<ColiderSpanPosition>().bandera == true
                    && FrontCap.GetComponent<ColiderSpanPosition>().bandera == true
                    && StatorCap.GetComponent<ColiderSpanPosition>().bandera == true
                    && FanCap.GetComponent<ColiderSpanPosition>().bandera == true
                    && Fan.GetComponent<ColiderSpanPosition>().bandera == true)
                {
                    PlaySound(6);
                    state = 1;
                }
                else {
                    state = 10;
                }



                break;

            default:
                state = 1;
                break;
        }


    }
}

```

3.5 Tutorial Creation

The last step in the creation of the Virtual Reality Environment for Mechanical Maintenance is turning the game into a tutorial. This is done by scripting as well as by implementing voice commands. The scripting allows for the trainee to realize when mistakes have been made by implementing different colors to parts when the wrong part is grabbed or by not allowing the trainee to assemble a part when the part is not being assembled in the right order. The voice commands guide the trainee through the different steps of the mechanical maintenance process, each voice command is activated when the prior step is completed correctly. The tutorial creation also includes the scripting of the motor disassembly animation tutorial (Figure 33) which allows for the trainee to observe how the motor is disassembled. Then, the trainee can physically touch the parts which display their names (Figure 34). This teaches the trainee the components, their location within the motor and their names.

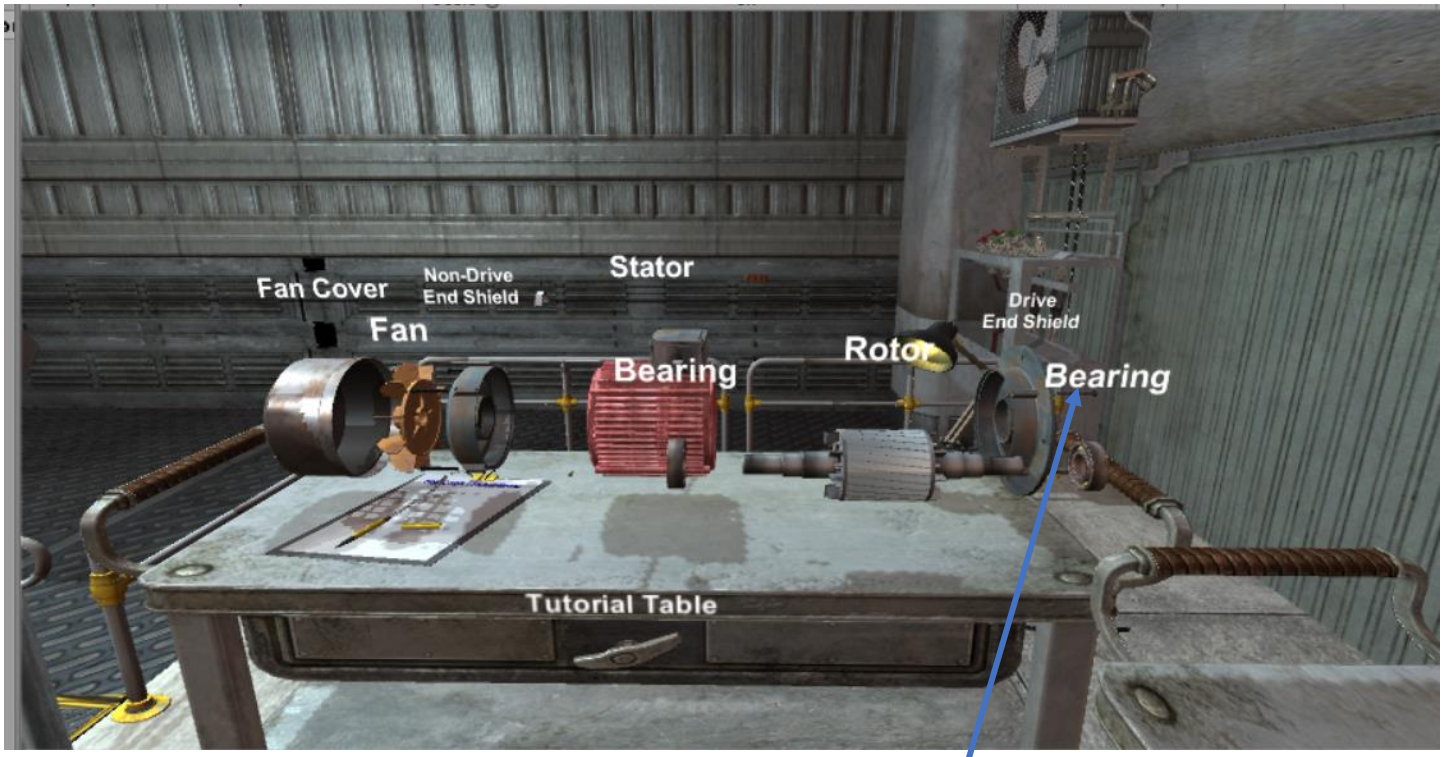
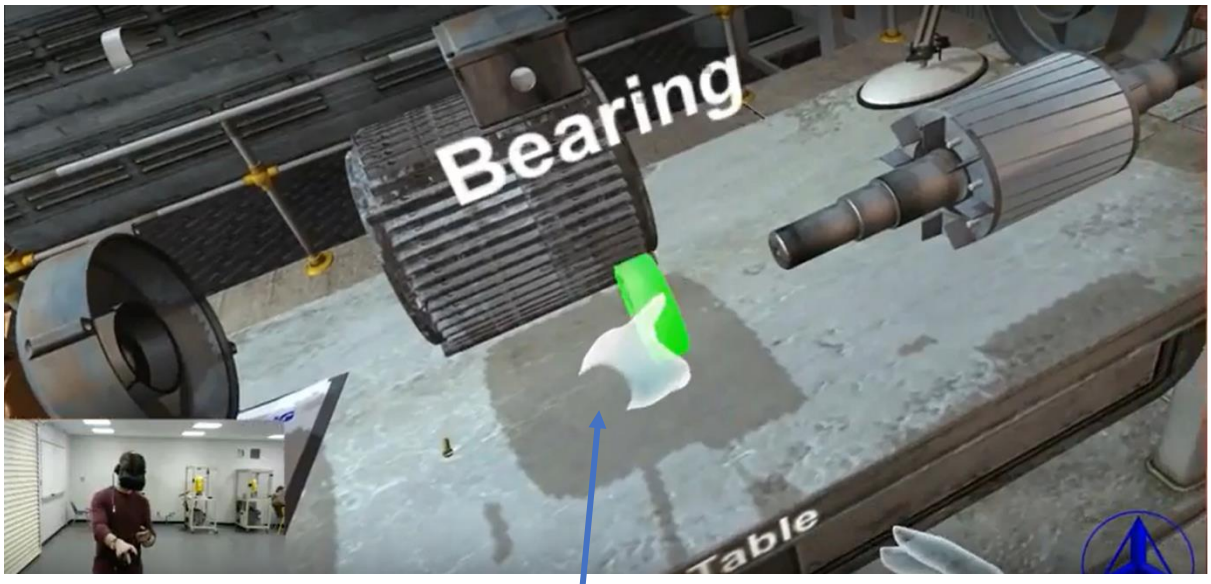


Figure 33 Motor Disassembly Tutorial

Names of the components



Student Using the Oculus Rift, Oculus Touch Controllers and controlling the VR Scenario

Color change when the trainee touches the part and name being displayed on contract.

Figure 34 Motor Disassembly Tutorial

Chapter 4 Findings, Results and Conclusions

The overall objective of this research was to create a Virtual Reality Scenario which was realistic and with the capabilities of training a person to perform a specific procedure in safer way without any constraints that regular training methods have now a day. All of the objectives of this research where met. Firstly, a mechanical maintenance process was identified. This process met all the requirements, for the creation of a successful training tool. The process was fully understood, and the tools needed to perform the mechanical maintenance in real life where identified. The realism of the scenario was achieved and even surpassed expectations through the use of textures, ambience objects, lightning and more. The environment proves to be safer without the use of any physical equipment, which also allows for the scenario to be available at all times. The behavior of the Virtual Scenario was programmed using C#, the behavior provides both functionality and realism to the virtual training scenario. The creation of a tutorial with the purpose of training a person the right order of a mechanical maintenance procedure was achieved using a 3-step process. The first step was the tutorial animation which showed the trainee an animation of the process, with the different steps of a disassembly and the parts names. The second step, allowing the trainee to perform said process with the use of physical interaction through Oculus Touch controllers which permitted the trainee to grab, touch and operate tools as well as components with ease. The third step was to implement a voice guided tutorial that guides the trainee through the training provided both instructions and feedback.

This scenario has proven to be a great instrument for education as well as for industrial training, as an added tool. Throughout the research, different groups of students, teachers, and industrial personnel had the chance to utilize the virtual scenario. The feedback

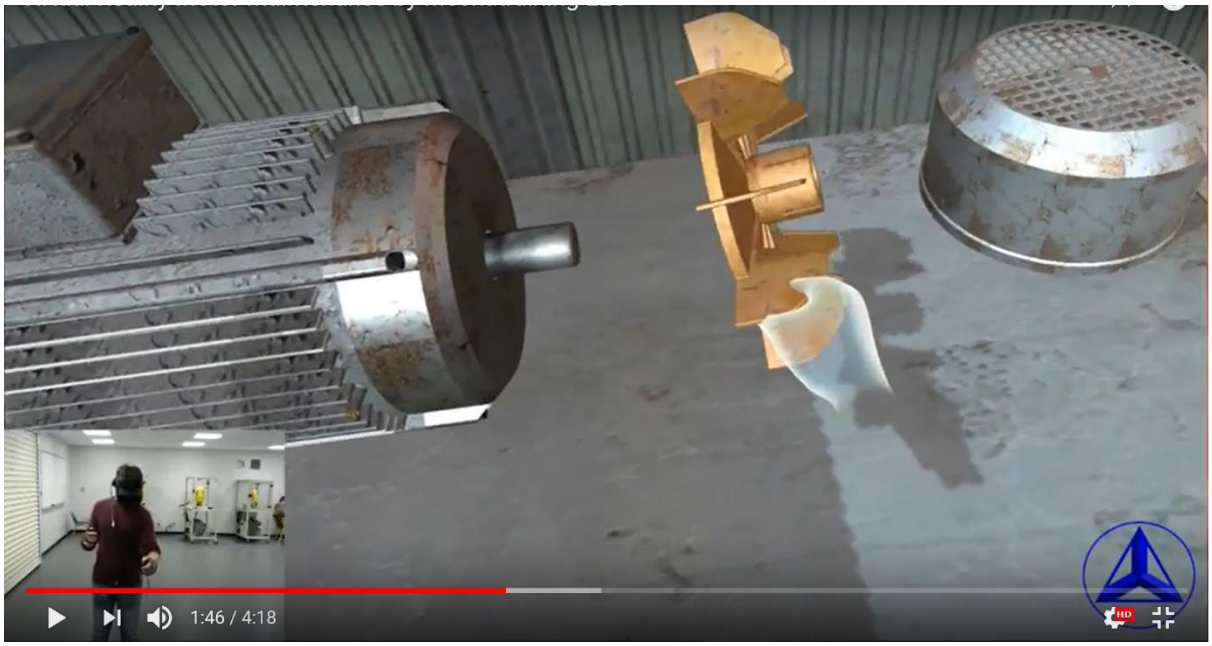
gained from the wide variety of groups at different levels was very positive. This scenario was showed at 3 different high school recruiting events and one college recruiting event where it was always the number one attraction for students as well as teachers.

Virtual Reality has been in use for several years for gaming purposes, due to the gaming market being a 100 billion dollar industry according to the 2017 global games market report. This has made numerous companies invest millions of dollars into this technology. Gaming technology, as seen with this research, can be used for educational as well as industrial training goals, giving Video Game technologies a more life-relevant purpose. This research showed that a realistic training scenario can be created. Such a scenario allows for a person to learn in a more efficient way. This training scenario does not have the constraints of equipment availability, budget restrictions and safety issues. This virtual scenario proves that we can train more people at a lower cost and in a safer way with the use of existing virtual reality technologies. This research also proves that this technology can be used and developed at a lower level than multimillion dollar corporations such as GE, Boeing and the US military (leaders in VR-AR training), allowing for educational institutions to benefit from advance educational technologies.

Below are some screenshots of the tutorial being used by students at Morehead State University and a complete video can be found at:
https://www.youtube.com/watch?v=dq2RSlslQcU&t=61s

Student Using the Oculus Rift, Oculus Touch Controllers and controlling the VR Scenario

Figure 35



Student Using the Oculus Rift, Oculus Touch Controllers and controlling the VR Scenario

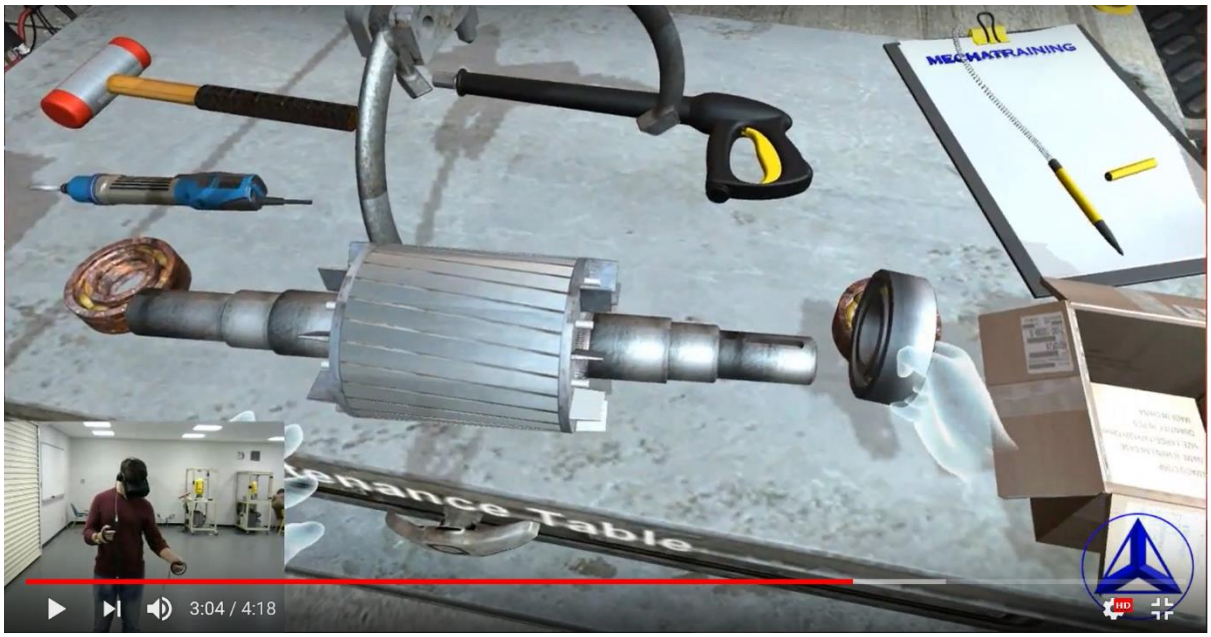
Figure 36

Student Using the Oculus Rift, Oculus Touch Controllers and controlling the VR Scenario

Figure 37

Student Using the Oculus Rift, Oculus Touch Controllers and controlling the VR Scenario

Figure 38



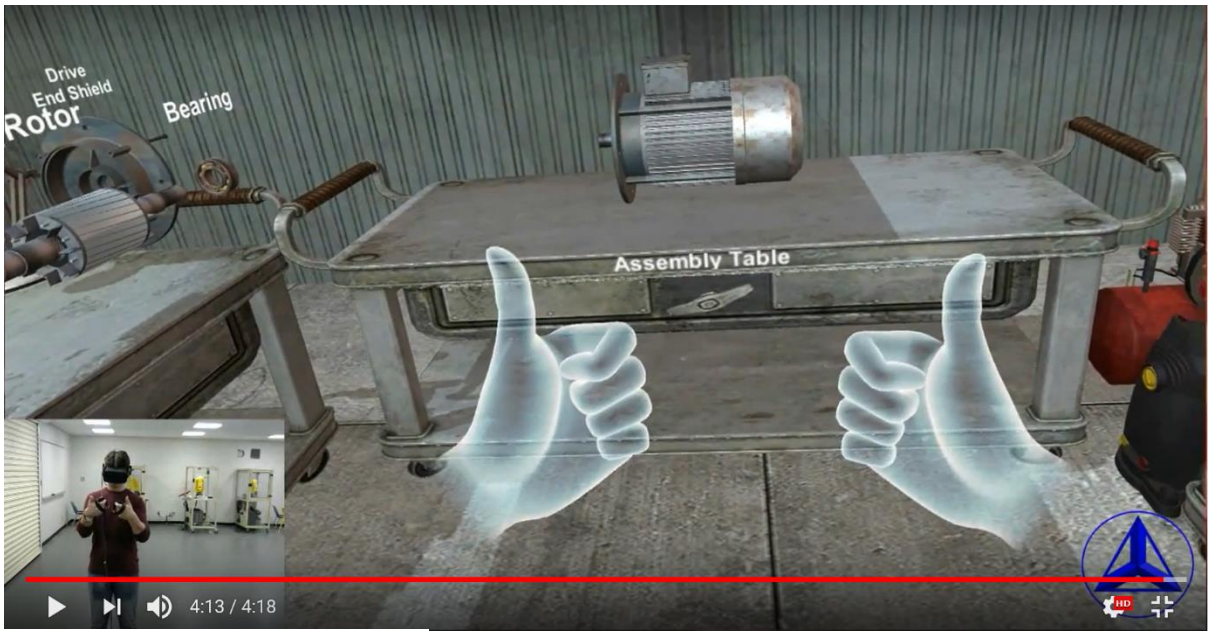
Student Using the Oculus Rift, Oculus Touch Controllers and controlling the VR Scenario

Figure 39

`

Student Using the Oculus Rift, Oculus Touch Controllers and controlling the VR Scenario

Figure 40



Student Using the Oculus Rift, Oculus Touch Controllers and controlling the VR Scenario

Figure 41

References

A Alaraj, MG Lemole, JH Finkle, R Yudkowsky… - 2011

Virtual reality training in neurosurgery: Review of current status and future applications

A.M. Hussaan and K. Sehaba, "Adaptive Serious Game for Rehabilitation of Persons with Cognitive Disabilities," in Advanced Learning Technologies (ICALT), 2013 IEEE 13th International Conference on, Beijing, 2013, pp. 65-69.

Brockwell, Holy "Pygmalion's Spectacles". Project Gutenberg. Retrieved 21 September 2014.

(3 April 2016).

Bruce Baumgart, Winged-Edge Polyhedron Representation for Computer Vision. National Computer Conference, May 1975. http://www.baumgart.org/winged-edge/winged-edge.html

Bruner, Robert F., Repetition is the First Principle of All Learning (August 17, 2001). Available at SSRN: https://ssrn.com/abstract=224340

Bureau of Labor Statistics, U.S. Department of Labor, *Occupational Outlook Handbook*, *2016-17 Edition*, Industrial Machinery Mechanics, Machinery Maintenance Workers, and Millwrights, on the Internet at https://www.bls.gov/ooh/installation-maintenance-and-repair/industrial-machinery-mechanics-and-maintenance-workers-and-millwrights.htm

"Facebook to buy Oculus virtual reality firm for $2B". Associated Press. March 25, 2014.

Retrieved March 27, 2014

"Forgotten genius: the man who made a working VR machine in 1957". Tech Radar. Retrieved 7 March 2017.

H. Guo, H. Li, G. Chan and M. Skitmore. (2012, Sept.). Using game technologies to improve the safety of construction plant operations. Accident Analysis & Prevention, 48, pp. 204-213.

Hloska, k. 2. (2014). VC of Mechatronic Systems with the use of Simulation. *Virtual Commissioning.*

Hollister, Sean and Buckley, Sean. "Here's The Final Oculus Rift, Coming In Early 2016". Gizmodo. Gawker Media. Retrieved June 17, 2015.

Horowitz, Ken (December 28, 2004). "Sega VR: Great Idea or Wishful Thinking?". Sega-16. Archived from the original on 2010-01-14. Retrieved 21 August 2010.

I.S. Brasil, F.M.M. Neto, J.F.S. Chagas, R. Monteiro, D.F.L. Souza, M.F. Bonates and A. Dantas, "An intelligent and persistent browser based game for oil drilling operators training," in Serious Games and Applications for Health (SeGAH), 2011 IEEE 1st International Conference on, Braga, 2011, pp. 1-9.

*Krakauer, J.W.; Shadmehr, R. (2006). "Consolidation of motor memory". Trends in Neurosciences. **29**: 58–64. doi:10.1016/j.tins.2005.10.003.)*

Kuchera, Ben (15 January 2016). "The complete guide to virtual reality in 2016 (so far)". Polygon

Lee, C. G., & Park, S. C. (2014, June 2). Survey on the virtual commissioning of manufacturing systems. *Journal of Computational Design and Engineering, I*(3)

Metz, Cade. "Facebook Buys VR Startup Oculus for $2 Billion". WIRED. Retrieved 13 March 2017.

M. Zyda. (2005, Sept.). From visual simulation to virtual reality to games. Computer, 38(9), pp. 25-32.

Ortega-Moody, J. A., Sánchez-Alonso, R. E., González-Barbosa, J. J., & Reyes-Morales, G. (2016, February). Virtual Laboratories for Training in Industrial Robotics. *IEEE Latin America Transactions, XIV*(2), 665-672.

Quantitative and qualitative effects of repetition on learning from technical text.
Bromage, Bruce K.; Mayer, Richard E.
Journal of Educational Psychology, Vol 78(4), Aug 1986, 271-278.
http://dx.doi.org/10.1037/0022-0663.78.4.271

Riccitiello, John (October 23, 2014). "John Riccitiello sets out to identify the engine of growth for Unity Technologies (interview)". VentureBeat (Interview). Interview with Dean Takahashi. Retrieved January 18, 2015.)

R.J. Lancaster. (2014, Mar.). Serious Game Simulation as a Teaching Strategy in Pharmacology. Clinical Simulation in Nursing, 10(3), pp. 129-137.

Rouse, M. (2015). Virtual Reality. *A look at the cutting-edge tech and the enterprise.*

R.S. Torres and F.L.S Nunes, “Applying Entertaining Aspects of Serious Game in Medical Training: Systematic Review and Implementation,” in Virtual Reality (SVR), 2011 XIII Symposium on, Uberlandia, 2011, pp. 18-27.

Smith, Colin, On Vertex-Vertex Meshes and Their Use in Geometric and Biological Modeling, http://algorithmicbotany.org/papers/smithco.dis2006.pdf

"The Oculus Rift, Oculus Touch, and VR Games at E3". oculus.com. June 11, 2015.

"Palmer Luckey Explains Oculus Rift's Constellation Tracking and Fabric - VRFocus".

Tobler & Maierhofer, A Mesh Data Structure for Rendering and Subdivision. 2006.

("What is a Game Engine?". GameCareerGuide.com. Retrieved 2013-11-24.)

www.army.mil/article/84728/DSTS__First_immersive_virtual_training_system_fielded FORT BRAGG, N.C. (Aug. 1, 2012))

VRFocus. Retrieved June 17, 2015.

2018 Unity Technologies. Publication: 2017.4-002A. Built: 2018-04-04. https://docs.unity3d.com/Manual/PhysicsSection.html

3ds Max | 3D Modeling, Animation & Rendering Software | Autodesk. (n.d.). from https://www.autodesk.com/products/3ds-max/overview

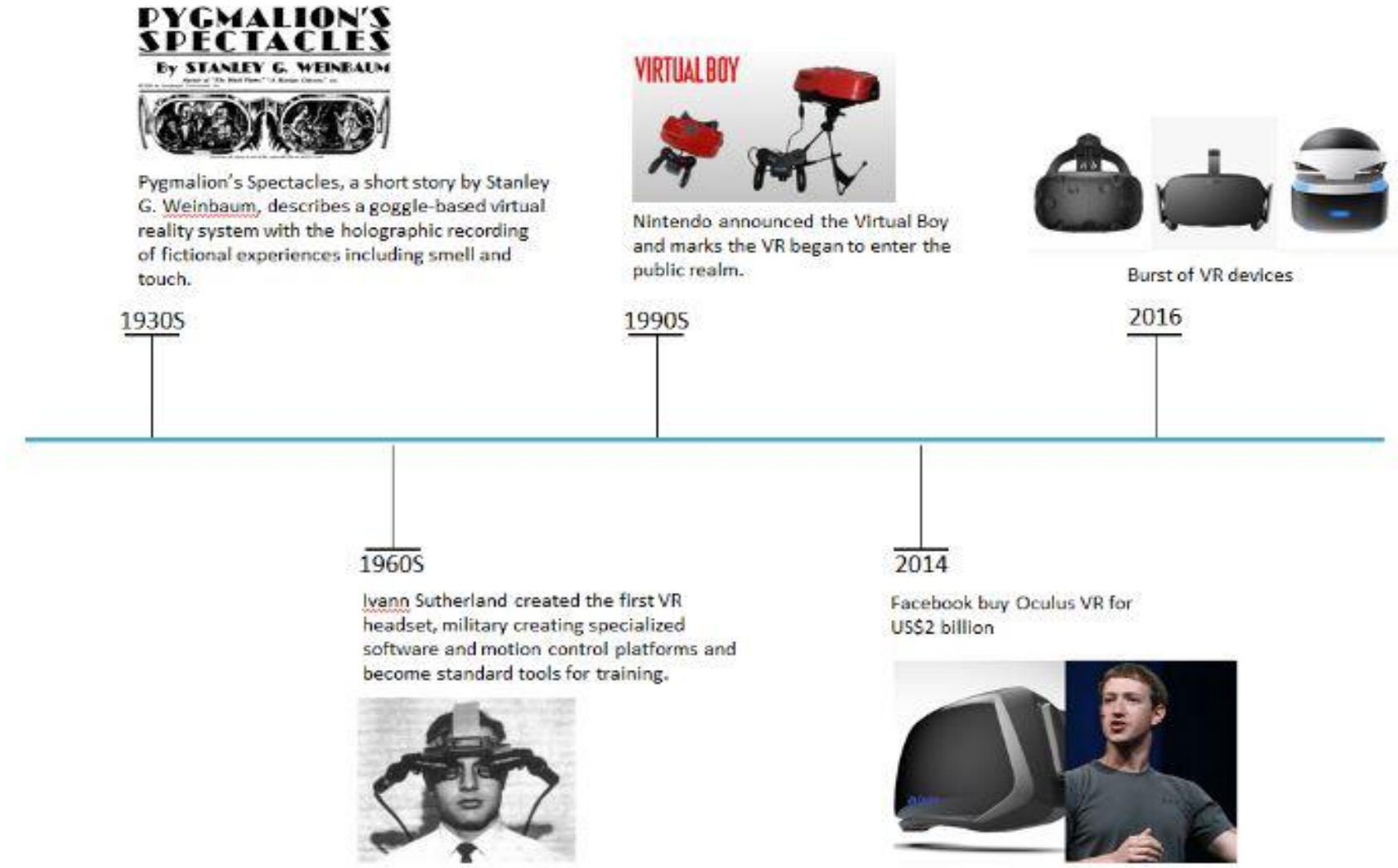


Figure 1 Virtual Reality Timeline (filmora.wondershare.com)

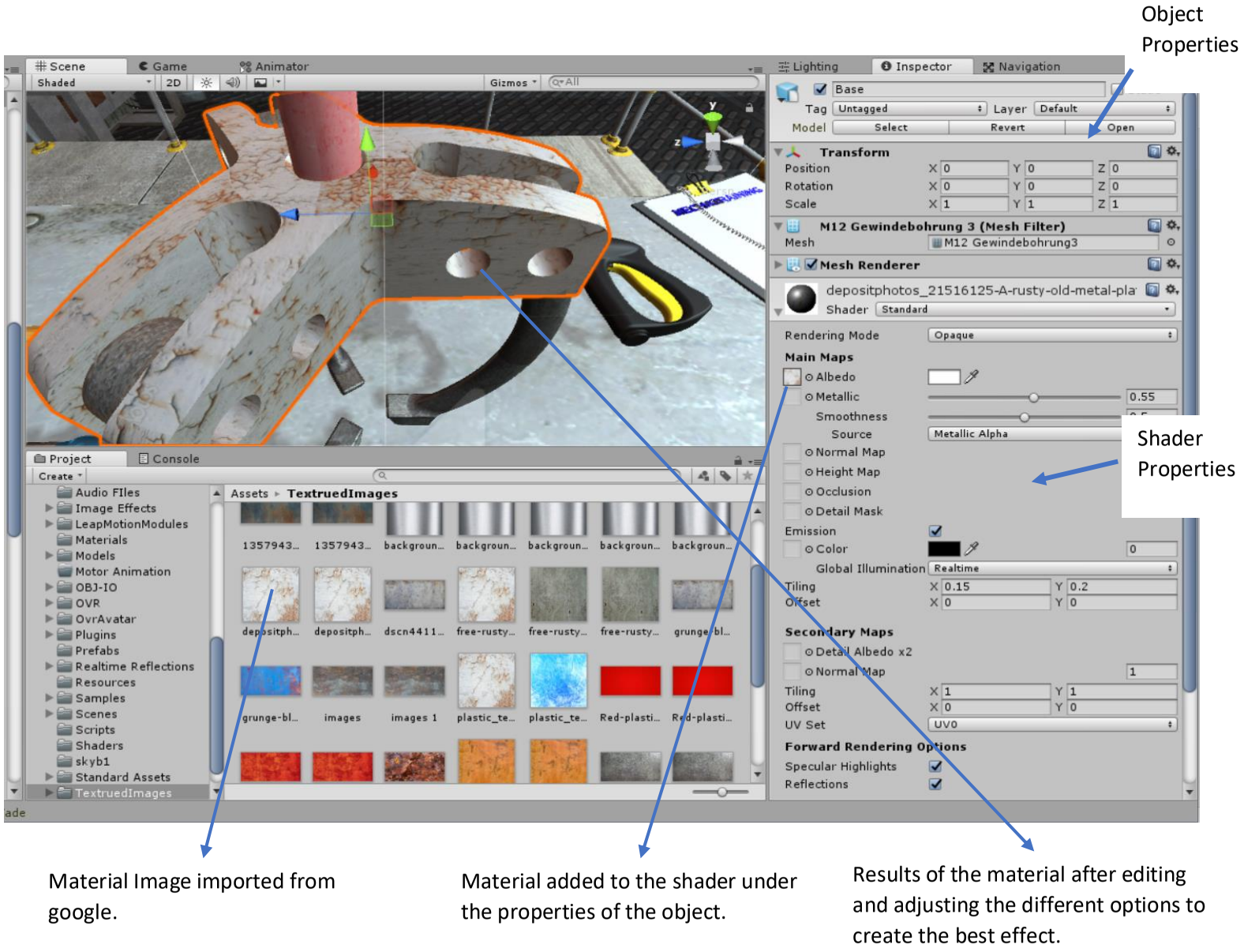


Figure 18 Textures

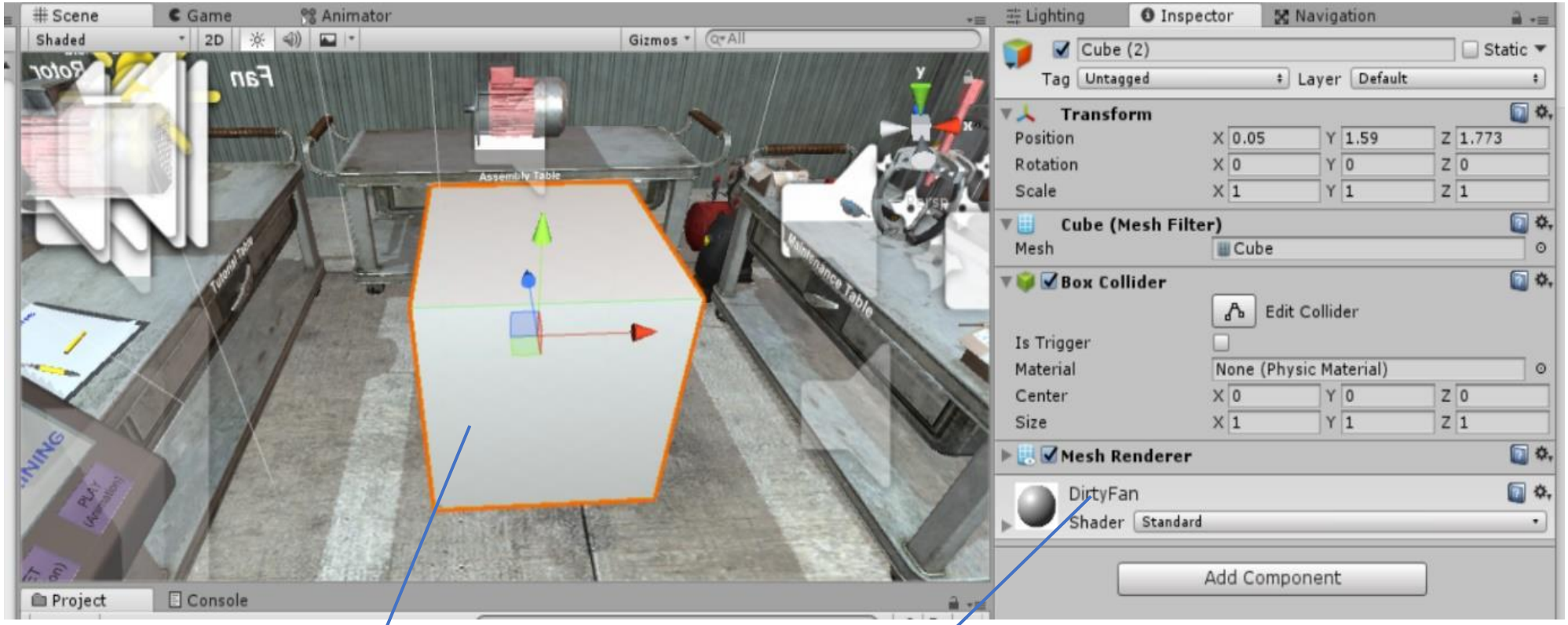
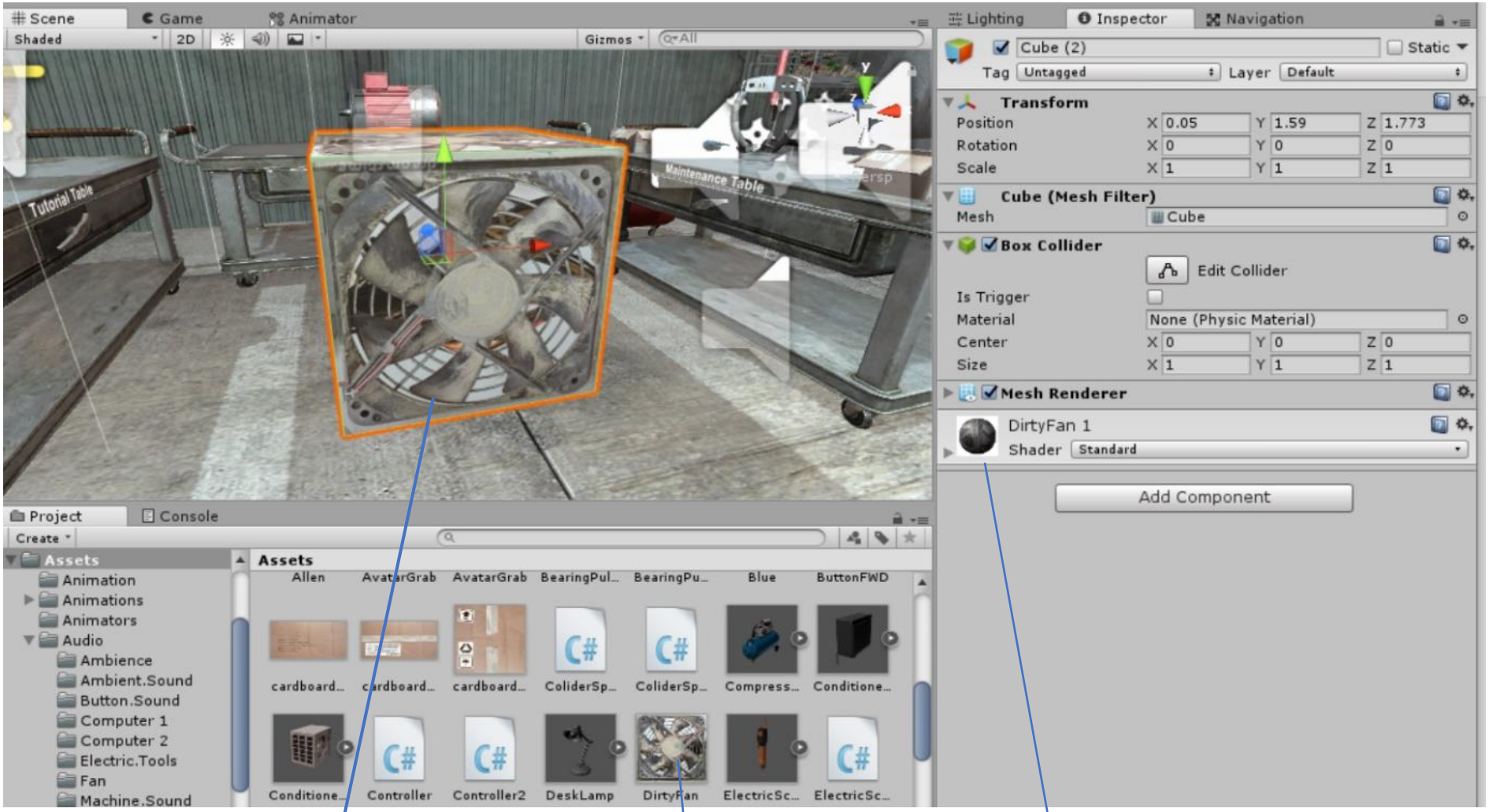


Figure 19 Textures



Cube with the texture applied.

Image imported

Image used as a texture

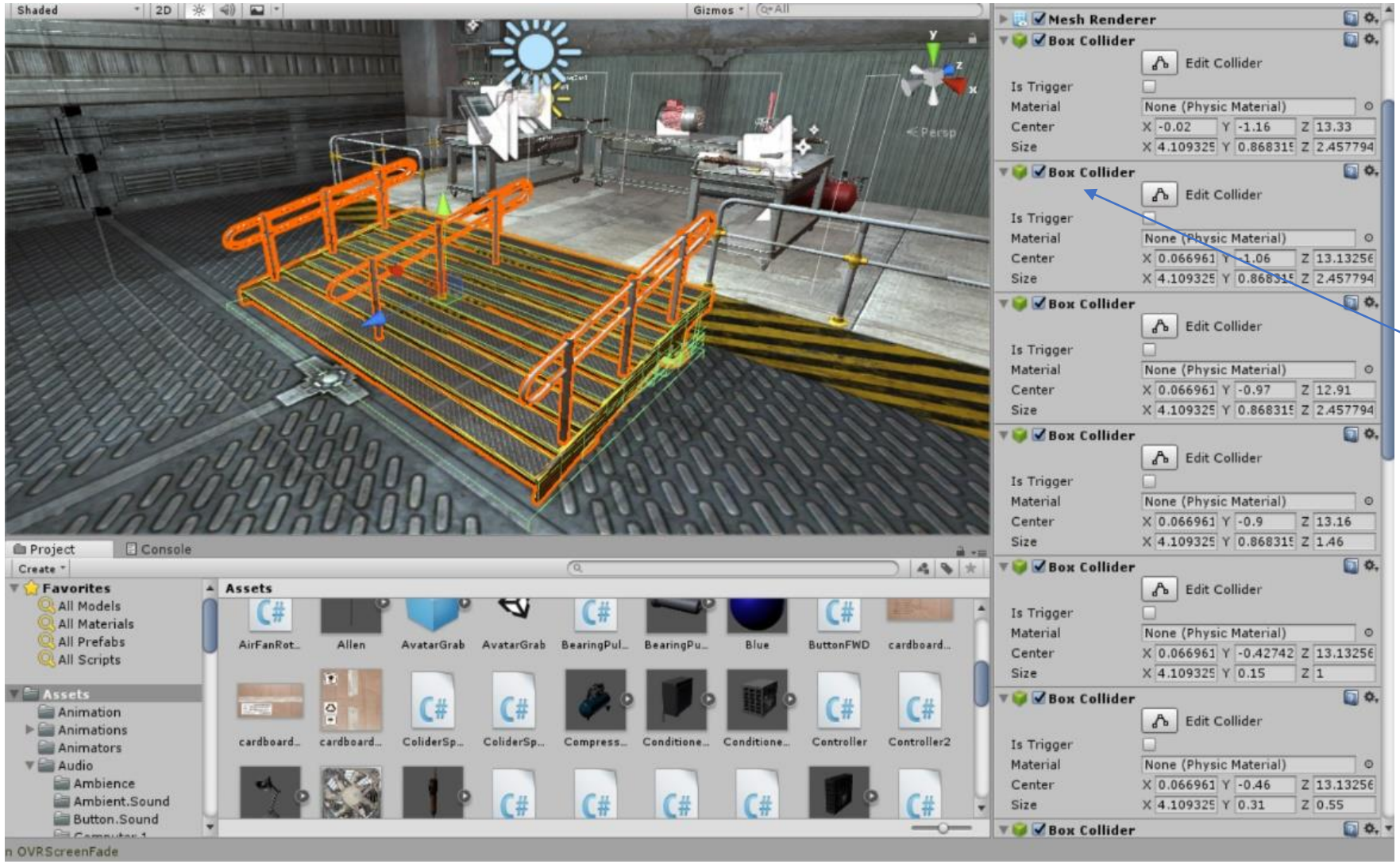
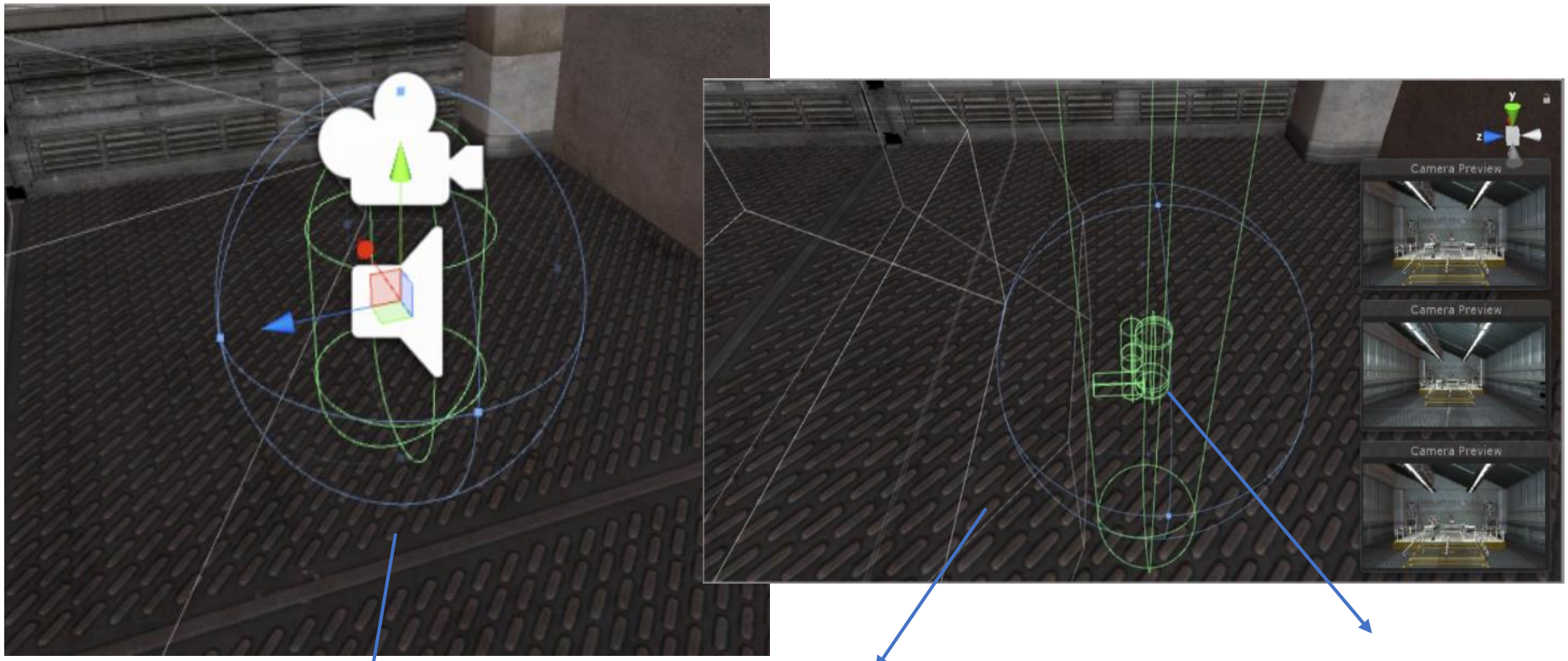


Figure 27a Box Colliders in Steps



Unity First person controller

Oculus OVR Player Controller

Finger colliders for hand interaction with objects.

Figure 28 Character Controllers

Controller Script

Physical Variables

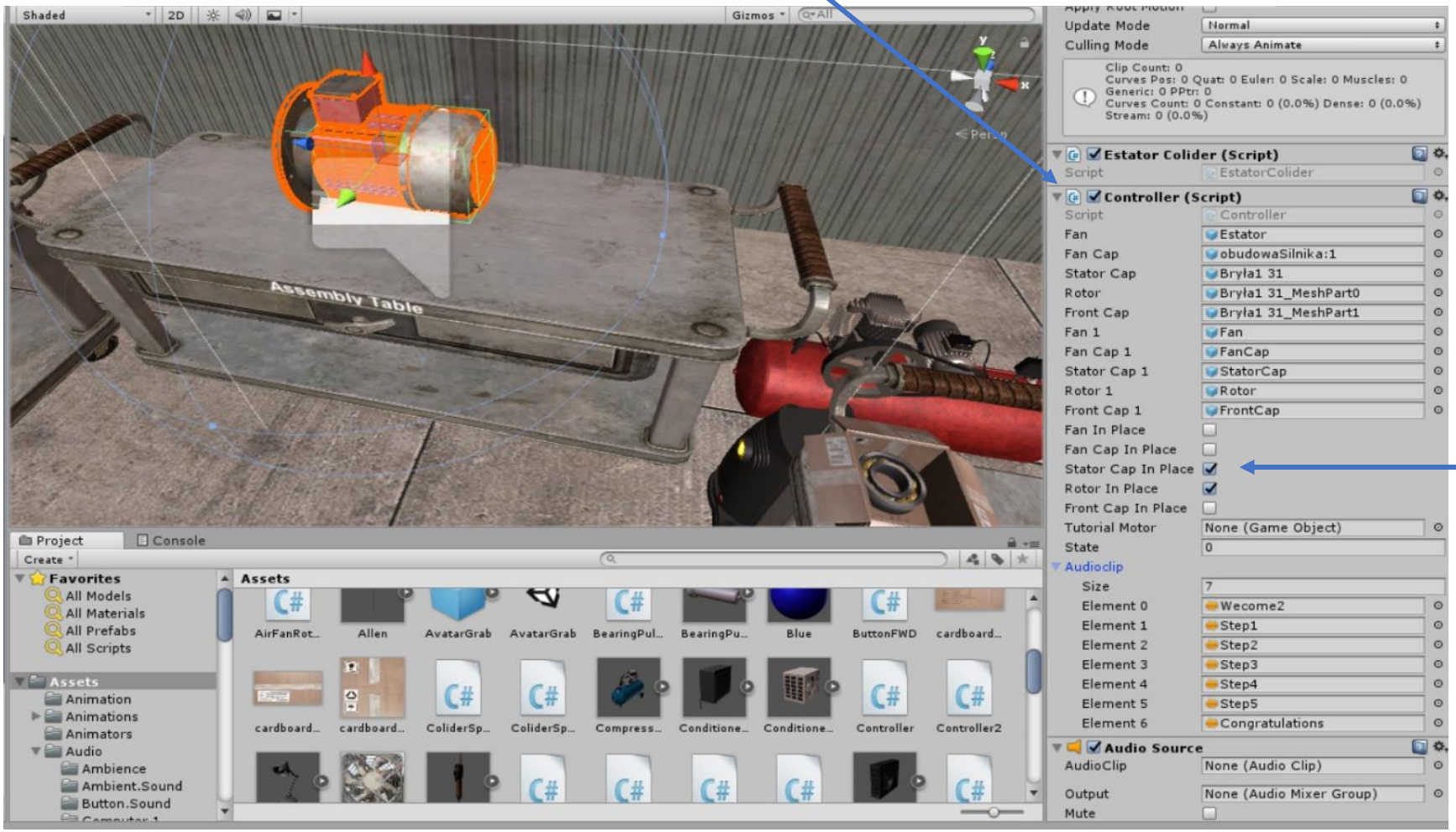


Figure 32 Scripting

Audio Files that provide feedback