5-2021

Effects of Bounding Tree Complexity on Immersion in a Virtual Environment

Jonathan Caputo

Effects of Bounding Tree Complexity on Immersion in a Virtual Environment

By

Jonathan Caputo

* * * * * * * * *

Submitted in partial fulfillment

of the requirements for

Honors in the Department of Computer Science

UNION COLLEGE

March 24, 2021

Abstract

CAPUTO, JONATHAN Effects of Bounding Tree Complexity on Immersion in a Virtual Environment. Department of Computer Science, March 24, 2021.

ADVISOR: Matthew Anderson

Bounding boxes are volumetric spaces in a 3D environment defined by a set of points that denote the area of collision for a given object. Bounding trees are known as a collection of bounding boxes. Simple bounding trees contain few bounding boxes and complex bounding trees contain many. Our research aims to find the relationship between the complexity of a bounding tree and a user's immersion in a 3D virtual environment. We created an algorithm that used the bounding trees to check for when an object collides with another. Using this algorithm, we ran a user study where users would be set in a 3D environment and experience the effects of the bounding tree of an object. In the end, we found that there is a possible relationship between the two, in the form of an N-shaped curve. This curve shows that as bounding trees get become more complex, user immersion rises to a point, then falls, then rises once again. In addition, we found that there were suggested significant differences in user immersion between less complex bounding trees and more complex ones.

Contents

List of Figures

List of Tables

1 Introduction

Nearly all modern 3D applications must account for object-object interactions. The process of determining these interactions is known as collision detection. This is an important aspect of any 3D experience, since it aids in the realism of the application, which improves the user's immersion in it [6]. One way of implementing collision detection is using bounding boxes. These "boxes" surround an object and denote its area of collision [1]. An example of a bounding box can be seen in Figure 1.

Bounding boxes can be utilized in many ways, one of which includes using a "bounding tree." A bounding tree is a collection of bounding boxes which all surround a single object [14]. The shape of a bounding tree can be either simple or complex. Both kinds of bounding tree have their benefits and downsides: simple bounding trees can lead to unrealistic collisions and complex bounding trees may be unnecessary if less complex bounding trees can provide just as realistic collisions. My research aims to find the best balance between simple and complex bounding trees by conducting a user study that evaluates the user experience, using different complexity bounding trees. Based on the user experience, I found which complexity of bounding tree works best for users and how complex a bounding tree can be before being unnecessarily complex.

1.1 Background

This section discusses what collision detection is, the common methods of implementing it, and the user experience in an application. At the end of this section, I discuss what aspects of each topic are focused on in my research.

1.1.1 Collision Detection

Collision detection is the process of determining whether or not two objects collide at a given time [1]. Collision response is the process of determining the reaction that occurs when the two objects overlap, also known as interact, with one another [1]. These two processes have many uses in 3D applications. One of the most important applications of these processes is physics simulations. Physics simulations are software that model and simulate physical phenomena [10]. Every physics simulation contains a scene, the space in which the application takes place. The main purpose of collision detection and collision response is to be the model for physical interactions, which happen often in the scene of physics simulations [10]. Collision response is especially important to these simulations, as it relies on physical factors of the environment [1]. However, collision detection is equally important. There are many approaches to detecting collisions in 3D

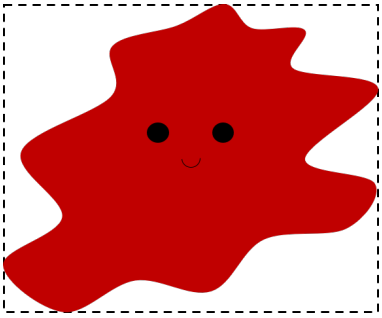

Figure 1: A bounding box (dotted line) surrounding a red figure.

environments, but every method falls under two general cases: a posteriori methods or a priori methods.

A posteriori methods are the discrete way of solving collision detection, meaning that collisions are checked for at every time interval, or "frame," of an application. For example, when the scene updates, the collision detection method checks to see if any objects are overlapping [1]. If two objects overlap, the collision detection method says that they are colliding. From there, the method produces a response to said collision. If not, the scene updates again until there is a collision. The name of this type of method comes from the Latin word for "after the fact." This aligns with how these methods operate; they detect a collision once it has already happened. Compared to a priori methods, a posteriori methods are easy to implement, since a posteriori methods only need to use geometry to see if two objects are overlapping [1]. However, this type of method can also miss collisions entirely. Consider two objects, a and b. Further consider that a is travelling fast enough that at time interval i it is on one side of b and at time interval $i + 1$, it is on the other side. This means that in the time interval from i to $i + 1$, the two objects never overlapped. Thus, a travels through b without a collision being detected, hence the method being considered discrete [1]. Thus, while comparatively easy to implement, it is not as accurate. An example of how a posteriori methods can miss collisions can be seen in Figure 2.

A priori methods are the continuous way of detecting collisions. Rather than checking for collisions at each time interval, a priori methods predict the trajectory and position of each object. Then, they calculate the time in which the two objects would collide, if they do [1]. The name of this type of method comes from the Latin word for "before the fact." This aligns with how these methods operate; they detect a collision by calculating it before it happens. These predictions rely on an array of factors from the environment and the objects, such as gravity, the speed of the objects, wind resistance, etc. Due to this, these calculations are precise [1]. However, due to the same factors, they are much more complicated than a posteriori methods. Unlike a priori methods, a posteriori methods do not need to worry about variables other than position. This makes a priori methods overall more difficult to implement.

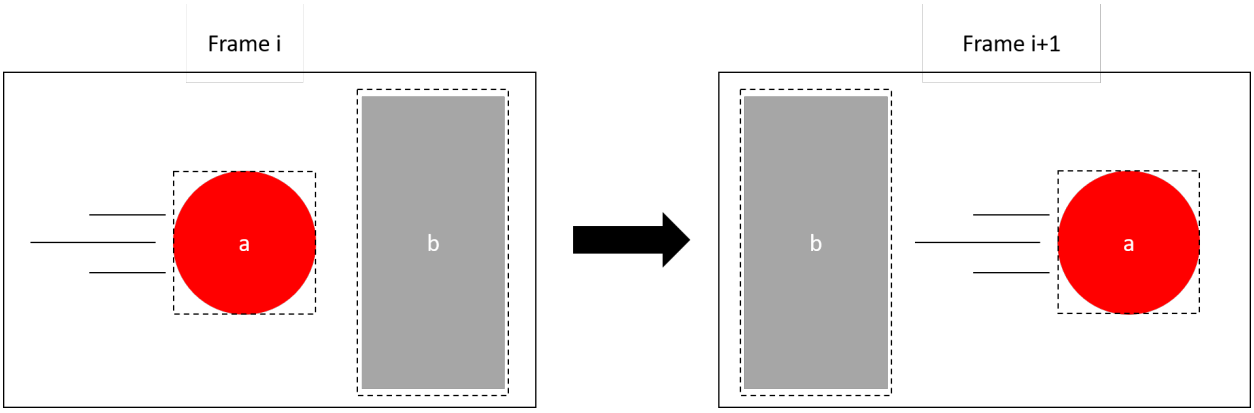

Figure 2: A visual example of how a posteriori methods can completely miss collisions.

There are many algorithms that are used for collision detection. Many of these implementations are a posteriori, as they are easier to implement. Below are a few well-known ones.

1.1.2 Different Implementations

Temporal coherence Temporal coherence is an a posteriori method for collision detection that uses the results of previous frames in order to find the result of the current frame [13]. This relies on the fact that objects change a predictable amount between frames and that previous frames have "results," such as the object's position and speed, that are similar to the current frame [13]. For example, consider two objects travelling in a 3D application during a specific frame, where the objects were not touching on the previous frame. As long as the positions of the objects do not change greatly between the frames, it can be concluded that the objects also do not touch in the current frame [13]. This method is generally used to find a pair of objects to investigate whether or not they are going to collide soon.

Bounding Box The bounding box method is a simple, a posteriori way of implementing collision detection. This involves creating a rectangular prism around an object that acts as an area in which a collision can be detected for said object [11]. When two bounding boxes are overlapping, it is said that the two objects are colliding. The bounding boxes are normally represented as a set of points that would form a rectangular prism if connected, meaning they are not actual objects in the scene [6]. This method is efficient, as the bounding box is less geometrically complex than the object it surrounds.

The area that denotes collision can take many other shapes such as spheres and pyramids, but rectangular prisms are the most common for 3D applications [1]. Different shapes can provide different benefits. For example, there are also "bounding spheres" that use spheres to surround an object instead of rectangular

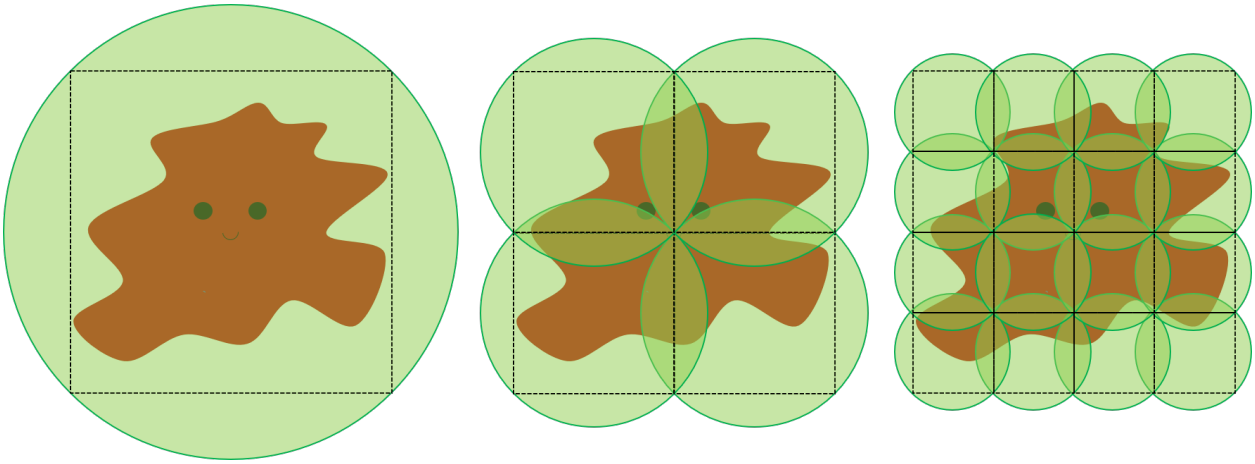

Figure 3: Three levels of bounding tree complexity, each made up of bounding spheres and shown in two dimensions. The most simple (left) is made up of a single bounding sphere, while the most complex (right) is made up of sixteen.

prisms. All that is needed for computation using bounding spheres is the sphere's radius and the position of the sphere's center. This is simpler than bounding boxes, which need to keep track of either: the positions of the eight corners, the positions of two opposite corners, or the position of one corner and the height, width, and depth of the rectangular prism. No matter the case, the geometry necessary for computation is easier with spheres. However, cubes can more accurately cover an object's surface area due to their shape [1]. Regardless, any shape used for bounding box collision detection works well. In video games, these are usually referred to as the "hitboxes" or "hurtboxes."

Generally, bounding boxes can be a single box that surrounds an object, or multiple, smaller boxes surrounding sections of the object in order to more accurately cover its surface area [6]. The collection of many smaller bounding boxes surrounding an object is referred to as a "bounding tree" in this research. The origin of this term is discussed in Section 2.3.1. A bounding tree is defined by the number of overall bounding boxes it contains, n. The number of bounding boxes n can be described by a term c where $n = 8^c$. Essentially, c is a measurement of the bounding tree's complexity, where as c rises, the shape of the bounding tree becomes closer to the object's geometry. A visual example of different bounding tree complexities can be seen in Figure 3.

Pairwise Testing Combining temporal coherence and bounding boxes, pairwise testing uses temporal coherence to find two objects that are going to collide soon and then creates bounding boxes to test for collisions [9]. All 3D objects are made up of small "primitives", which are 2D triangles that are formed by three points connected by three lines. This method creates a search tree based on the volume of the

bounding box for each primitive [9]. The positions of the primitives on each object are then checked for any collisions between them. Thus, this process cuts down on the area to check for collisions.

1.1.3 User Experience

The user experience of 3D applications is another important aspect of my research project. Collisions can occur and be responded to, but if they are not realistic, then they are not effective. From *Designing Immersive Video Games Using 3DUI Technologies*, the definition of a 3D user interface is "an interface that involves human-computer interaction in which the user performs tasks in three dimensions" [3]. The 3D user interface is integral to the user experience, as it is the user's gateway to the environment in which they reside. There are two important aspects of the user experience that are related to any user interface and are relevant to this research: immersion and interaction.

The general idea behind immersion is "the sense of being there [literally inside the 3D application]" [12]. The measurable way of determining is immersion is officially defined as "the objective level of fidelity of the sensory stimuli produced by a technological system" [5]. This essentially means the level to which the user feels as though they are apart of their environment. This would account for areas of the 3D application such as "resolution, field of view, and stereoscopy (creating the illusion of depth)" [5].

Another important aspect of the user experience is user interaction. In 3D environments, user interaction is the interconnection of actions between the user and the environment and how they react to one another [5]. For example, in the baseball mode of Wii Sports, the environment responds to the user swinging their remote by showing the user's avatar swing their bat. Conversely, the user responds to the opponent avatar's pitch by swinging their remote.

In general, user immersion and user interaction are key to studying the user experience. For example, Pantel and Wolf studied how user immersion is impacted by delay in multiplayer video games [7]. They had users play a racing game several times where the users experienced an increasing amount of synthesized delay. They found that a delay of 50ms was when the user's immersion was broken [7]. In the past, many areas of research either combine the two aspects or separate them to learn something about the user experience. For example, McMahan et al. separated the two in order to learn the benefits of user immersion in 3D applications alone [5]. They did so by varying two components that affect immersion between users and also varying the interaction technique used by subjects, testing three overall. This allowed them to see if the components or if the interaction technique affected the tasks that the users performed [5]. On the other hand, many other studies such as Takatalo et al. combine the two aspects to gain an understanding of user experience as a whole [12]. Takatalo et al. did so by varying only one component of user immer-

sion between users, while keeping the interaction technique constant. This allowed them to see how the interaction technique worked in tandem with each different variation of the immersion component [12].

1.1.4 This Research

Of all of the proposed collision detection methods, by far the most simple and most well-known is the bounding box method. The simplicity of the model allows for easy collision detection without harming the computation time. In fact, this is the reason that so many modern 3D applications, mainly video games, use bounding boxes [6]. Not only can they be used on their own, but they can also be combined with other methods in order to maximize efficiency, such as in pairwise testing [9]. However, since bounding boxes on their own are far more common, my research focuses on them alone. The main focus in this research for bounding boxes is bounding trees, specifically with the idea of them being simple vs. complex. In addition, my research focuses on studying bounding trees that consist of bounding spheres, for reasons that are discussed in Section 2.3.1.

On the other hand, user interaction and its effects on immersion are the main user experience focuses of this research. This research aims to combine user interaction and user immersion in order to learn something about the user experience as a whole. This is done similar to Takatalo et al. where the interaction technique is kept constant, while varying an aspect of the environment that may affect immersion [12]. The interaction technique being used involves the user moving around an object in a 3D environment. In addition, the varied component affecting immersion is the difference in bounding tree complexity.

1.2 Research Question

My research focuses on the idea of bounding trees and their relation to the user experience. Specifically, I am studying the relationship between bounding tree complexity and user experience to find out how the complexity of a bounding tree can impact the immersion of a modelled environment in order to understand the trade-off between the realism and complexity of a 3D environment. The question I ask in my research is the following: How does the complexity of a bounding tree for an object in a 3D virtual environment affect the user experience of the environment? In the end, I answer this question by running a between-subjects, user-based experiment, where one variable differs among all users.

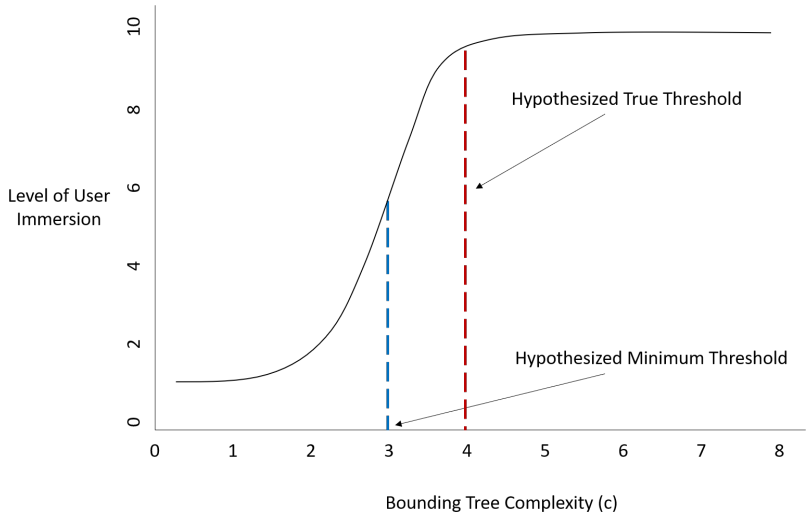

Figure 4: A graph of the hypothesis for the research question. The most ideal bounding tree is hypothesized to be at the minimum threshold, shown by the blue dotted line. Alternatively, the most ideal bounding tree is hypothesized to be at the true threshold, shown by the red dotted line.

1.3 Hypothesis

I hypothesize that there is a minimum threshold between bounding tree complexity and user experience, where the collisions are realistic and the complexity of the bounding tree is not redundant. The bounding tree complexity at this minimum threshold is considered to be the most optimal. This is because simple bounding trees are much more likely to detect collisions that look unrealistic. As seen in figures above, bounding spheres can contain "empty space," an area within the bounding sphere that is not a part of the object [1]. When another object collides with this empty space, collisions still register. Thus, a collision happens, but the two objects do not visually collide. This would be most common in simple bounding trees, since they contain the most empty space. In addition, the most complex bounding trees are more likely to either lag the application due to their complex nature and numerous calculations, or be redundant.

In terms of redundancy, consider a bounding tree of complexity c. If a bounding tree of complexity $c-1$ can garner the same kind of user experience and immersion, then the bounding tree of complexity $c-1$ is better to use. There is an increased amount of space being used with more complex bounding trees, so using less complex ones is more efficient in terms of space and time. For this reason, I hypothesize that as a bounding tree gets more complex, the level of user immersion eventually levels off.

For the bounding tree complexity at the minimum threshold, t, the difference in immersion between $t-1$ and t is greater than the difference in immersion between $t+1$ and t. Essentially, this minimum threshold is at the turning point of the hypothesized relationship between bounding tree complexity and

user immersion. A graph of the hypothesis and the mentioned minimum threshold can be seen in Figure 4.

Alternate Hypothesis For my alternate hypothesis, I hypothesize that there is a true threshold in the relationship of bounding tree complexity and user immersion. At this point, seen in Figure 4, further bounding tree complexities have slightly higher levels of user immersion. As mentioned before, these complexities would be redundant and therefore the most optimal bounding tree complexity would instead be at this true threshold.

2 Methods

This section describes how my research was conducted. The experimental design and implementation of the 3D environment are discussed. By the end, the overall set-up of the experiment and its parts should be clear.

2.1 Experimental Outline

The experiment was conducted as follows:

1. A user and I met in a Zoom call where I informed them how the study was to be conducted. I had them sign a consent form saying they are okay with being a participant. Then, I asked them preliminary, demographic questions.
2. The user was put into a simple version of the environment in order to get a good grasp of the controls.
3. The user brought up the actual version of the 3D environment on their device.
4. The user performed a task in which they would hit colored spheres throughout the environment by moving around and using a stick. I told the user to avoid the complex 3D object in the middle of the environment while doing so. They also had 60 seconds to complete the task.
5. Once done with the task, I asked them 3 questions pertaining to the task they performed.
6. The user did this for 5 different versions of the environment, where each version had a different complex object in the middle.

2.2 Experiment

The conducted experiment was a between-subjects study, where each user tested one condition of the independent variable: bounding tree complexity. The study was conducted over the online video chat application, Zoom, due to COVID-19 restrictions. Users for the study were gathered via a Union College campus-wide email and participation in the study was completely voluntary. The email outlined several important factors of the study. Mainly, they would need a laptop or desktop, a mouse or a touch-pad that allowed for two-finger scrolling, the ability to use Zoom with their camera and microphones on, and that they would be paid for their participation. Students could also sign-up for any available time slot via a Google Calendar that was provided in the email. I kept a record of the students who signed up so that, upon signing up, I would email each new participant with more specifics for the study. This way, I would be able to keep contact with my participants, while also making expectations of the study clear.

Pilot Studies Before conducting the actual studies, I ran three "pilot studies." These were conducted exactly like a real study, following the outline in Section 2.1. However, the main intent of these studies was to find any show-stopping flaws in the experimental set-up. After conducting all three of the pilot studies, I found the following flaws. In the second pilot study, the user did not have a mouse. This reinforced the fact that I should not limit users to only using a mouse to be able to perform the study, which was the original plan. This was unrealistic to ask, since many people use their touch-pad in place of a mouse. To resolve this, I added more controls that made sure that the users had many options for controls. Also, when talking about the task, the user was not as focused on how much they hit the object. This prompted me to change the wording of the task to focus the fact that they should avoid the object. The third pilot study participant helped show that I needed to fix one of the set-ups of the environment, due to buggy behavior in its initialization. With these changes in place, I conducted the full study.

Initial Set-Up The specifics for how the study was conducted are as follows. First, I created a 3D environment that currently exists on a Union College web-page. This environment contains two main components: a large, complex 3D object in the middle of the scene and a stick. This stick is an object that the user would have control of, allowing them to move it around in the environment. By moving it and swinging it, they would be able to "hit" anything they swung at.

In addition, the environment contains small, colored spheres, which are referred to as "lights." When first created, the lights were scattered throughout the environment in predetermined positions. The positions were mostly arbitrary, though there was some method to the lights near the 3D object. In those cases,

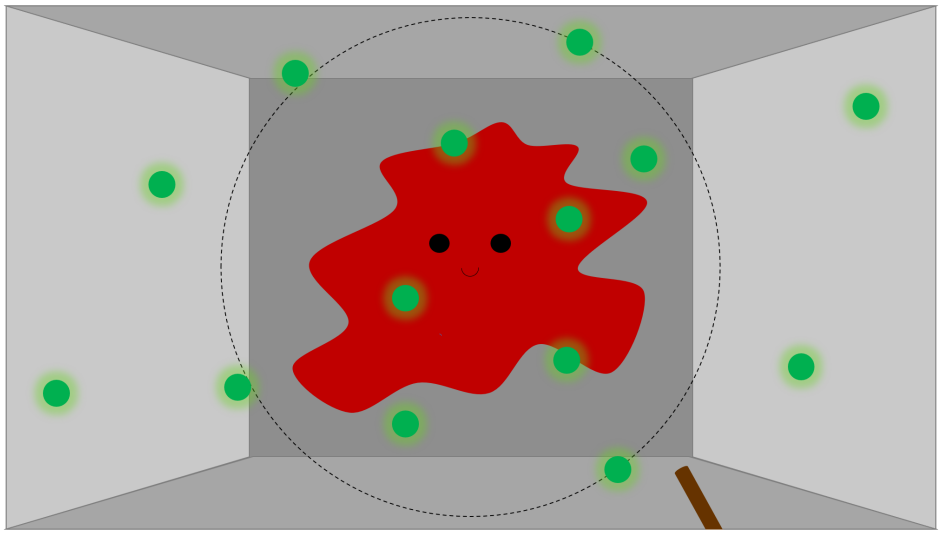

Figure 5: A basic diagram of the room that users were placed in for this experiment. The gray walls and floors are the bounds of the room. The red blob in the middle represents some 3D object. The black outline represents the bounding tree for the object, in this case, a simple one. The green dots represent the lights in the environment that the user was promoted to interact with. Finally, the brown cylinder in the corner represents the "stick" that the users used to interact with the lights.

I wanted to make sure that each user hit the object at least once for all complexities. Thus, these lights were positioned such that they were within the bounding tree for certain complexities. For example, one such light may have been reachable for bounding tree complexity 3, but would require the user to hit the object for complexity 2.

All other components in the environment that have not been mentioned were for decoration. These decorative components were used to help disguise the true nature of the study [8]. This general setup for the environment was used for all users. A diagram for the experimental set-up can be seen in Figure 5.

Conducting the Study For these studies, a script was written in order to keep consistency among what was being told to users during their session. To start the study, each user and I met in a Zoom call. Then, I explained what they were going to be doing and how long the experiment would take. They then filled out a consent form to make sure that they understood what it meant to be taking part in the study. Next, I asked the user some very quick demographic questions that would be used for my data. These questions pertained to their academic year at Union College, their experience with video games, and what operating system they were using. Each user was then put into the environment by opening the web link to it, which I provided.

Before getting into the testable versions of the environment, or the "rooms" as they were referred to, the

user opened a link to the room without a 3D object in the middle. This room allowed the user to familiarize themselves with the controls. In order to do so, I asked them to input each of the controls one at a time. Then, I allowed them to use the controls more freely until they felt comfortable with them.

Once ready, I sent the user a link to the next room, which contained a black box in the middle and a single light. The main goal of the room was to get the user to understand the set-up of the application, which was as follows: any light in the room would disappear when hit by the user's stick. Any other object in the room, the box in this case, would turn to a bright green color when hit by the stick to indicate that they had hit the object. Both of these facts were explained to the user and I let them test it out for themselves.

Data Collection Once done with the initial set-up, I prepared the user for the actual study. I provided them a link to the first room they would go to. This room would have 10 total lights scattered around the room, with a giant, complex object in the middle. When in the application, I told the user to wait before moving around. Then, I gave them the following one task and allowed them to move around once I was done reading it:

"Please remove all of the lights from the room within 60 seconds. While doing so, do not touch the [object] in the middle with the stick. When you are ready to start, please press the "T" key and proceed with the task."

This is where the main user interaction portion of my research came into play: the ability to move around the environment and interact with objects in it in different ways. For example, removing the lights involved getting the stick close enough to "hit" them. They did so by using their keyboard and mouse or touch-pad for a set amount of time. Regarding the lights, each of them had a simple bounding tree that was used for detecting the hits. However, some of the lights were within the bounding tree of the object in the middle. In that case, if the user attempted to hit the light, they would also inadvertently interact with the object, causing it to change color as mentioned before.

While the users were given 60 seconds to complete the task, I let them continue after time was up if they wanted to, since the timer was just a decorative component. For my personal notes, I would take note of each user's success with the task, being a "Yes" or a "No." If the 60 seconds had passed and they still chose to remove all of the lights, I simply noted their success with the task as a "Timeout." During the study, the timer counted down from 60, since I believed that a countdown timer was the most effective way of making the environment seem like a game. However, in hindsight, a stop-watch timer would have been just as effective, while also keeping an exact count on the user's time. If I could do this study again, I would make the timer work that way.

Upon completion of the task, I asked the users three questions on their experience in the environment:

1. How did you feel about your success with removing all lights while avoiding the [object]?
2. I noticed the [object] in the middle turned a different color while you were removing the lights. What was your reaction/thoughts about that? Did it seem surprising or did it meet expectations/feel natural? On a scale of 1-10, where a 1 does not meet expectations at all, and a 10 completely meets expectations, to what extent did hitting with the [object] meet your expectations?
3. Did you notice any slowdown in the application as you moved around? Did the slowdown affect your experience?

These questions were mostly used to gage the user's experience and immersion in the application. The most important question was the second, as it directly addressed the collisions that the users had witnessed. Once answered, the user would move on to the next room. The users did the above process for five versions of the environment, where all that differed was the object in the middle and the placement of the lights. Once done with all five rooms, I paid the users for their time in the study and let them go on their way.

Goal of Tasks The main goal of these tasks was to make the user interact with the bounding tree of an object without intentionally doing so. By doing this, the user would be more immersed in the environment, as they would be focused on the task at hand. Thus, I could see how the unintentional interactions impacted their immersion. This meant that the less the users knew about the study, the better. Had they known about what was being tested, they would have had a predetermined assumption of when they might interact with the object in the middle. This would have resulted in a bias in the data and the answers of these users would not be reliable.

Thus, to lessen these initial assumptions, I decided to disguise the true nature of the study by framing it as a game. This was further done when sending out the campus-wide email to collect participants, as the subject line was "Game-Related Application Study." In order to disguise the study further, I added the aforementioned timer. From my point of view, the time it took to complete the task was not relevant to answering the question of the effect of bounding tree complexity on user immersion. If anything, time-related measurements could have been how long until the first time the user interacted with the object and the rate at which they hit the object. However, since both could be affected by many other factors, I did not find these measurements to be relevant. Thus, the timer and the time taken to complete the task were used to further enforce the idea that the application was a game.

Handling Validity Threats Since this is a user study, there were several factors that needed to be taken into account in terms of the validity of the data. Factors that can affect the data in any sort of way are known as "validity threats." [8]. For example, one such threat is known as an ordering threat. This states that participants learn and adapt over the duration of the study [8]. For example, consider the order of the rooms that the users tested. By the final room, the users could have better adapted to the environment. This could make their answers skew one way or the other, based on how they adapted. Thus, if all users tested room 5 last, then room 5 would be affected by this bias across all users. In order to reduce the bias of this threat, the order of the rooms tested was randomized for each user. For example, one user may test room 1 and then room 2, but another may test room 2 and then room 1.

In addition, for each user, I pseudo-randomized the bounding tree complexity for the 3D objects. The process was as follows: I randomized every user's bounding tree complexity. Then, I would balance the number of participants for each complexity. For example, if I had 6 users and was testing 6 different bounding tree complexities, I would want each bounding tree complexity tested once. Let's say 3 users were given a complexity level of 2 and 0 users were given a complexity level of 4 and 5. I would then randomly choose two of the users testing a complexity 3 and randomly assign them either a complexity of 4 or 5. I made sure that each step of this process involved randomization in order to further reduce bias, even if slightly.

Other Experimenatal Factors The bounding tree of the stick, the shape of the complex objects, and the placement of the lights were variables that were held constant across all users. The only necessary tool or hardware for this experiment was the device that the users used in the study. This required that the user had a desktop or laptop and a mouse or touch-pad.

In terms of bounding tree complexity, I decided to test a total of six different complexities, which were all c where $0 \leq c \leq 5$. This means that each user tested anywhere from a bounding tree complexity 0 to a complexity 5. This range was chosen since it gives the full range of complexities. On one end, there is a bounding tree with lots of empty space and on the other is a bounding tree that essentially covers the surface area of the object. This distinction is visually shown in Section 2.3.5.

Finally, users in the study were paid for their participation. The amount was based on how long the user took part in the study, using \$10.00 per hour as a baseline. For example, if a user took 45 minutes, then they would be paid \$7.50. However, I planned to pay each participant at least \$5.00, so any participant that was in the study for less than 30 minutes was still paid \$5.00.

2.3 Implementation

In order to accomplish the goals of this research, I needed to implement a few necessities that would allow me to run my experiment. First, I had to create the 3D environment that the user was to be placed in. The environment was created in JavaScript using the ThreeJS library. JavaScript was chosen due to its ease of front-end visualization. ThreeJS was chosen to due its library of objects that can be created in a 3D environment. Both were also chosen due to my familiarity with them. I initially implemented the actual scene itself, including the walls, ceiling, lighting, etc. I created the in-game "player," which consists of the stick, the camera, and the controls used to interact with the environment. Users would be able to move around using the arrow keys or the WASD keys. They would be able to tilt the camera using their mouse or touch-pad and move the stick back and forth via the mouse scroll wheel or two-finger scrolling on the touch-pad. Due to the set-up of the controls, the aforementioned issue of a posteriori methods potentially missing collisions was resolved. This is because I purposely set the speed for moving the camera and the stick such that one object could not move fast enough to pass through another without a collision being detected.

Finally, I created an interface for the controls menu, giving users an opportunity to view the controls while conducting the study. In addition, I added three counters that relate to the application: the number of times the complex object was hit by the user's stick, the time left in the application, and the number of lights left in the environment. These counters were used so that I would be able to take measurements. In addition, both the controls menu and the measurement counters were used to further disguise the true nature of the study, as discussed in Section 2.2. The environment, the controls menu, and the measurement counters can be seen in Figure 6.

The current setup for the environment and controls was not always the intent of this research. Originally, the users would be placed in an environment in Virtual Reality. This would make the controls for the interaction with objects more natural, as users would physically move the controller around to move the stick. They would also be able to move around the room by walking, rather than having to use the arrow keys. However, given COVID-19, I would not reasonably be able to conduct such an experiment. The required setup would need to ensure rigorous cleanliness and safety, which was too much of a cost and outweighed the benefit of a natural control scheme.

Regarding other implementation, I designed and completed the two most important algorithms needed for collision detection: creating a bounding tree for an object and detecting collisions between bounding trees. Both algorithms are inspired by the previous works of Tzafestas et al. that offered a collision detection

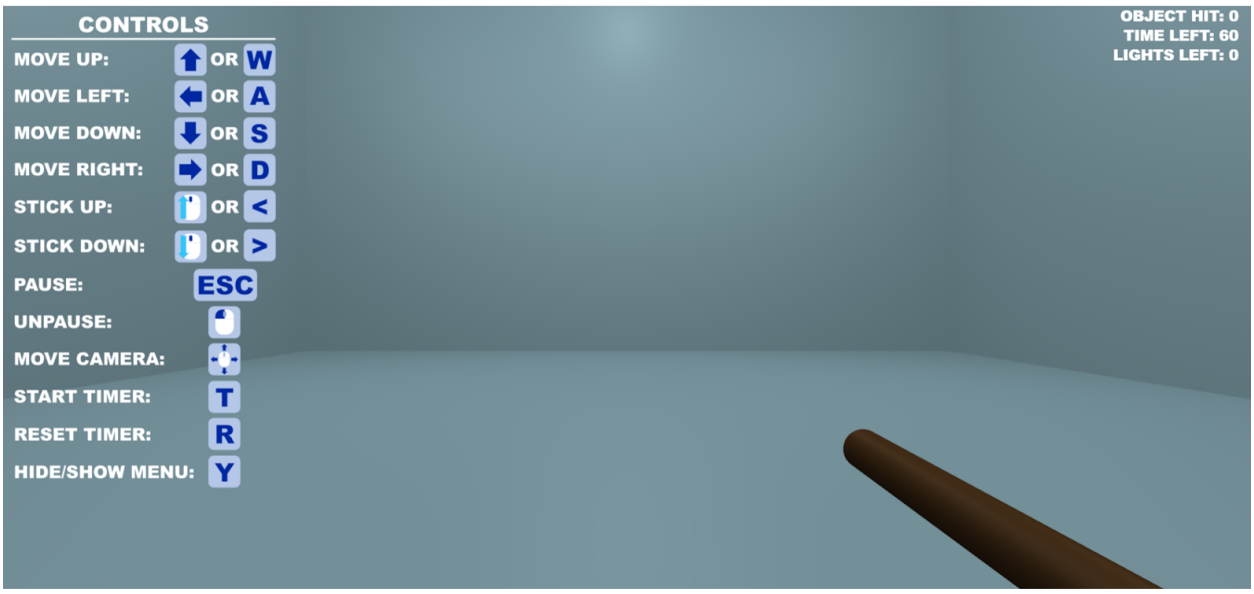

Figure 6: The 3D environment in its basic state. The UI for the controls can be seen in the top left and the measurements counters can be seen in the top right.

algorithm using a collection of bounding spheres [14].

2.3.1 Bounding Tree Creation

The algorithm for creating a bounding tree, $createBoundingTree(c)$, works as follows: given a 3D object in the scene and a complexity for the bounding tree c, create the object's bounding tree. As mentioned before, bounding spheres were chosen over bounding boxes. These were chosen due to the similarity of the algorithm described by Tzafestas et al [14]. In addition, calculations for overlapping spheres are simpler than calculations of overlapping cubes. Just like Tzafestas et al., I represented the bounding tree using an Octree, a tree-like data structure in which every node has eight children. This implementation choice is where the name "bounding tree" comes from.

Bounding Tree Structure The bounding tree is structured as follows. First, the bounding box for the object is calculated. In this case, the bounding box takes the shape of a cube. Then, a sphere that inscribes the cube is calculated and used as the root of the Octree. The bounding cube's center is used as the sphere's center. The radius of the sphere is calculated using the distance from the center to any corner of the cube. The eight children of the node are created with the original bounding cube in mind. The original cube is split into eight, equally sized, equally distant pieces. The center and radius of the child spheres are calculated the same way, but with their newly partitioned space. This means that all of the children are contained within

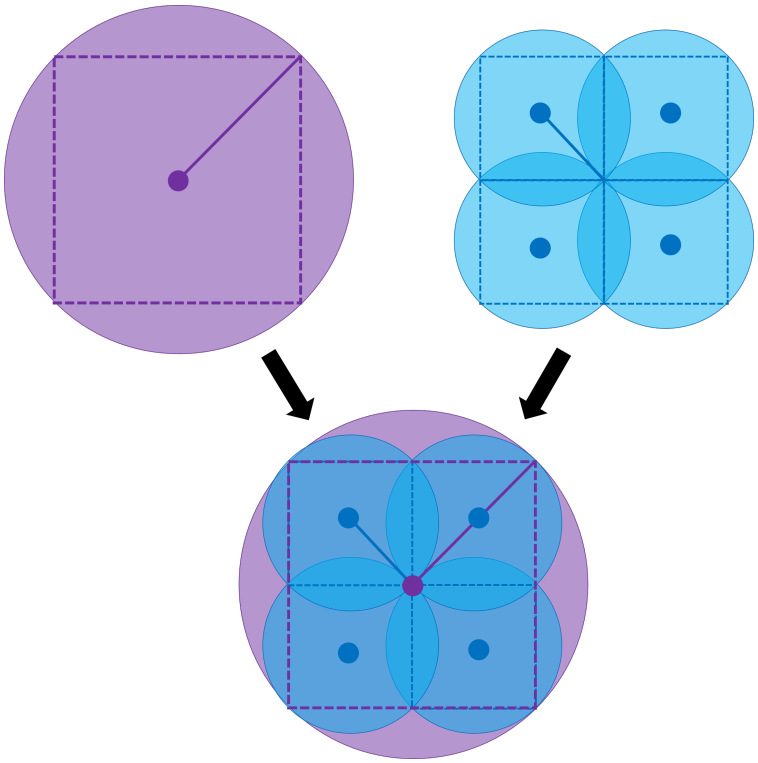

Figure 7: A visualization of the Octree for the bounding tree. Purple represents the parent node and its attributes, while blue represents the children nodes.

the parent node, have the same radius as one another, and are equally distant from the center of the parent. This process is done until the desired depth of the Octree is reached. In this case, that depth would be the complexity c of the bounding tree. An example of the structure of the Octree in 2D can be seen in Figure 7.

In terms of complex bounding spheres, there was one other detail that needed to be taken into account. Given how the bounding spheres are created, smaller children of the bounding tree eventually may not contain any part of the object. Thus, for any sphere that does not contain the object, it is given the classification of "NONE." This essentially means that the sphere is empty space. For the purpose of the aforementioned algorithm, any node of type "NONE" does not have any children created for it. There is no point in dividing a node that is just empty space if the goal of the bounding tree is to accurately cover the object. This classification is also important for the collision detection algorithm.

The bounding tree's general complexity is defined by the level of its leaf nodes; if the root is a leaf node, then the bounding tree is said to be simple, meaning $c = 0$. If the leaf nodes are five levels down in the Octree, then the bounding tree is said to be more complex, meaning $c = 5$. This was how different complexities of bounding tree were created, used, and analyzed.

Difficulties in Implementation When it came to implementation, there were many big challenges. One of them was figuring out the geometry for properly spacing the children from the center of the parent. The next biggest challenge was checking to see if any part of the object is contained within a sphere. Any object in a 3D application is made up of many vertices, which are points in space defined by their position. The vertices are connected by edges, which are lines between the vertices. A set of three vertices and edges is known as a face, also known as a primitive as mentioned before [1]. If a sphere contains any vertex, edge, or face of the object, then it is not given the type "NONE," meaning that the node representing the sphere can have children. The geometry involved with this process was tricky to figure out and took a lot of thought. However, I was eventually able to get it working, but it was certainly an impediment for a while.

2.3.2 Collision Detection Algorithm

After the bounding tree creation algorithm, I created a collision detection algorithm, $collide(this, other)$. This algorithm takes two objects, $this$ and $other$, and checks whether or not their bounding trees overlap. The general idea for how the algorithm works is as follows:

1. Check to see if two nodes of two different bounding trees are overlapping.
2. If so, check the children of each node that has children to see if they overlap.
3. Recurse on steps 1-2 until leaf nodes are reached for both Octrees.
4. If two leaf nodes overlap, the objects are said to be colliding. Otherwise, they are not.

Since bounding trees are considered a posteriori, the environment runs this algorithm at every frame, checking for a collision during said frame. In addition, the algorithm does not consider collisions with empty space, which is why spheres of type "NONE" are important. The full algorithm can be found below in Figure 8.

Difficulties in Implementation There were also many difficulties with implementing $collide(this, other)$. Originally, the algorithm was implemented as a pair-wise testing method, where *every* combination of nodes was checked. This was done by traversing one Octree down to the leaf nodes and finding which are overlapping with the nodes of the other Octree. In their research, Tzafestas et al. provide a good conceptualization of the algorithm for Octrees that use bounding spheres. However, no concrete algorithm was given in the paper [14]. Thus, I had to design the algorithm myself based on my interpretation of their conceptualization. Unfortunately, the solution I came up with was not efficient. As stated before, users are able

```
COLLIDE(this, other)
 1  if this or other does not contain geometry
 2      return False
 3  if this and other do not overlap
 4      return False
 5  else
 6      if this is a leaf and other is a leaf
 7          return True
 8      elseif this is not a leaf and other is a leaf
 9          return Collide-With-Children(other, this)
10      elseif this is a leaf and other is not a leaf
11          return Collide-With-Children(this, other)
12      else
13          for each child, v, in children of this
14              if Collide-With-Children(v, other)
15                  return True
16          return False

COLLIDE-WITH-CHILDREN(this, other)
1  for each child, v, in children of other
2      if Collide(this, v)
3          return True
4  return False
```

Figure 8: The algorithm used for checking collisions between two bounding trees.

to detect delay of 50ms or more [7]. Therefore, if the collision detection is inefficient, then users can notice the delay in detection. Due to this, the original design of the algorithm was scrapped. The algorithm would instead recurse on both Octrees at the same time, as Tzafestas et al. had originally suggested. This would mean that nodes on one Octree are only checked against nodes at the same depth of the other Octree. This happens until leaf nodes are reached on one Octree, at which point the leaf nodes are checked against all nodes at lower depths on the other Octree. Combinations of nodes would still need to be checked against each other, but the only pairs of nodes that would be recursed on would be nodes that are overlapping. This cuts down on the sheer amount of recursion that my original algorithm proposed. In the end, this issue came down to my misunderstanding of the algorithm proposed by Tzafestas et al.

The second difficulty came from the implementation of how positions were stored. Originally, the position of a node was statically stored within the node itself. However, this proved to be a challenge since the user's stick, which has a bounding tree itself, moves around. This means that the stored positions had to be updated. One benefit of ThreeJS that I had forgotten about was that it can track the positions of objects in the scene at any given frame, by using a simple function. In addition, the positions of the nodes are

only ever needed for collision detection when the application is running. Thus, the algorithm was changed to calculate the position of a node during the collision detection process. This was easier than having to update statically stored positions.

Once the difficulties were handled, I was able to get the algorithm working. From there, I was able to condense the code itself, making it approximately $\frac{1}{3}$ of its original size. The $collide(this, other)$ algorithm in Figure 8 is a testament to how I was able to condense my code.

2.3.3 Creation of Complex 3D Models

Once done with the above implementation, I needed to have complex models that would be placed in the middle of each room. In the end, I decided to create models of five objects with very familiar shapes: a grey milk bottle, a dark orange hourglass, a cyan train, a pink teapot, and a brown chair. These objects were chosen based on their shape, since each has a differently shaped bounding tree at higher complexities. The milk bottle is more compact and convex, whereas an hourglass has concave areas and a train has many parts to it. The reason for the colors of the objects are discussed later in this section.

These objects were created by using models for shapes supplied by ThreeJS. First, the model would be constructed using many smaller shapes, such as a rectangular prism, a pyramid, or a sphere. From there, the individual shapes would be positioned such that they took the shape of an object, such as an hourglass or a teapot. Once correctly positioned, I used an algorithm that I created, $combineGeometry(objects, color, scale)$ which takes a set of 3D models, $objects$, and combines them into one single model of a specified color, $color$, and size, $scale$. This was done by taking all of the vertices, edges, and faces of each object and creating a single model from all of that graphical data. The reason for this combination of models was due to my implementation of $createBoundingTree(c)$. It only supported the creation of a bounding tree for a single 3D model, as that is how bounding trees should operate. Thus, if I wanted to create a shape from smaller models, I would need to combine them, hence the need for $combineGeometry(objects, color, scale)$. A diagram of the process for model combination is shown in Figure 9. In addition, each of the models used in the application can be seen in Figure 10.

2.3.4 Collision Reaction

Finally, with the collision detection algorithm written and the models created, I would then need to implement the appropriate collision reaction. This was a key factor in my study, as this would indicate to the user that a collision had happened. A user cannot respond to a realistic or unrealistic collision if they do

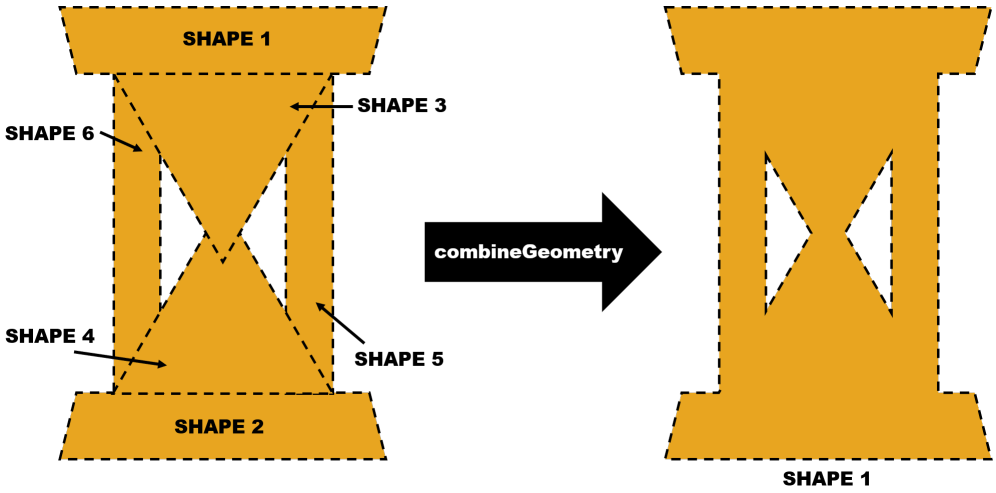

Figure 9: The process of combining many models into one single model. Each model is initially created and then deconstructed into its vertices, edges, and faces. This collection of graphical data for each object is then used to create a single geometry object.

not know that one had occurred. Thus, the collision reaction had to be obvious and able to clearly communicate to the user that a collision had happened. For this, I chose to change the color of the object when hit, as outlined in Section 2.2. A color change from the previously mentioned duller colors of the models to a bright green color is very noticeable. This is why the previously mentioned colors were chosen for each model. However, this choice impacts the accessibility of the application to color-blind individuals. To my knowledge, none of my participants were color-blind, since I indicated there being a color change, but no participants mentioned this being an issue. Regardless, this would have been a potential source of error if any of them were. For implementation, since bounding trees are a posteriori, all I needed to do was check for a collision each frame. If a collision happened, I just needed to change the color of the model using a simple function provided by ThreeJS.

2.3.5 Finalized 3D Environment

The current version of the 3D environment that the user was placed in, with all of the aforementioned points in this section implemented, can be seen in Figure 11. In addition, Figure 11 shows a comparison of all bounding tree complexities. In the middle of the scene is the hourglass model with its bounding tree visualized in the bottom images. Here, the full range of complexities mentioned in Section 2.2 can be seen. A complexity 0 shows a great amount of empty space around the hourglass, whereas a complexity 5 shows the surface area nearly covered. In the bottom-right corner of each image is the stick that the user used, along with its bounding tree. Currently, only the leaf nodes of the bounding tree are being shown,

Figure 10: The 3D models used for the environment. From top-left to bottom-right: a milk bottle, an hourglass, a chair, a teapot, and a train.

since they are the most relevant for collision detection. This was used as a visual aid for myself to see whether or not the collision detection algorithm was working by checking the visual accuracy. In addition, the visualization of the nodes was not present in the final study.

Additionally, the nodes are colored for my visual aid for implementing the bounding tree creation algorithm. The color of each sphere differs depending on what part of the object the node contains, if anything at all. If it contains nothing, the color is red. In Figure 11, nodes that are red are not shown since they unnecessarily take up space in the scene and are not considered for collisions. The colors are given as follows: Given some object in the environment, z,

1. If node contains a vertex of z, color is green.
2. Else if node contains an edge of z, color is orange.
3. Else if node contains a face of z, color is blue.
4. Else, color is red.

Finally, the lights used in the study can be seen along with their bounding tree. The bounding tree generation algorithm, $createBoundingTree(c)$, was used for creating the lights' bounding trees. This resulted

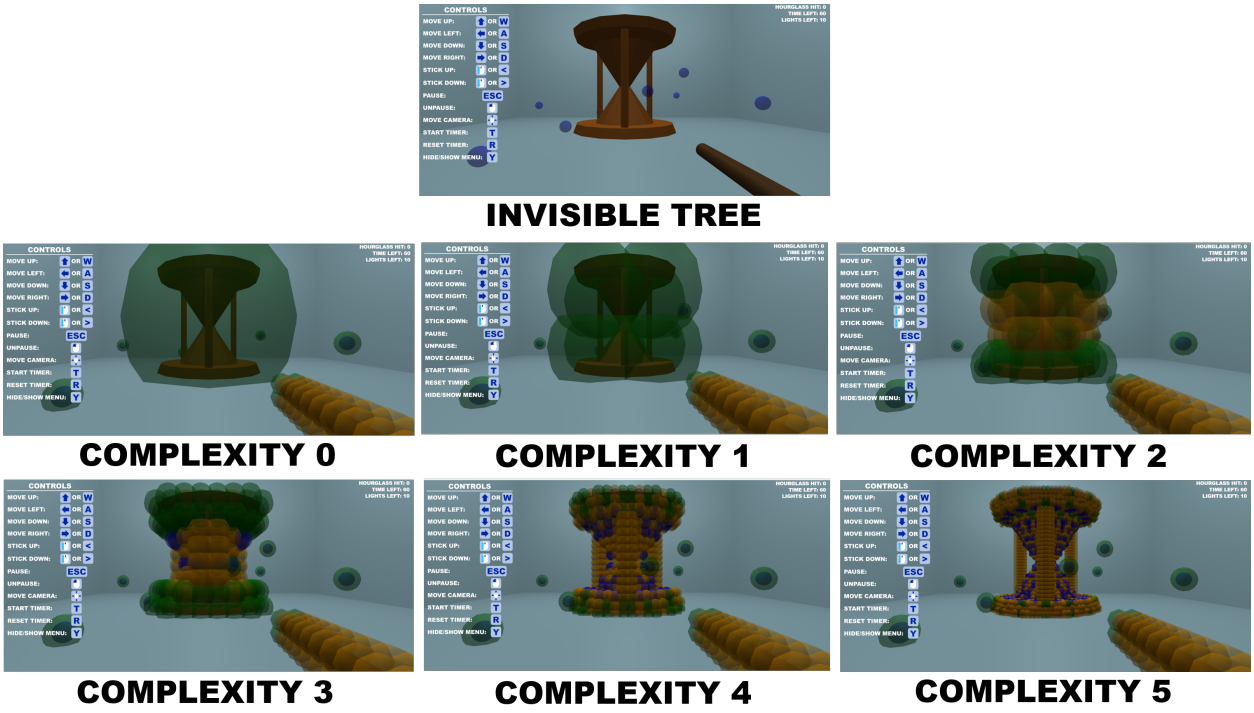

Figure 11: A comparison of the 3D environment in its final state. One image (top) has the bounding tree hidden from view while the other images show the bounding tree, each labeled with their bounding tree complexity. The top image is what the user saw during the study.

in each light having bounding trees that are slightly bigger than themselves. The cause of this is the first step of $createBoundingTree(c)$, where a bounding box is created around the shape and the root node of the bounding tree inscribes that box. The created bounding box is always be bigger than the sphere itself and thus so is the root node of the bounding tree, as seen in Figure 11.

In conclusion this section outlines the accomplishments I made in preparation for the user study. I finished implementing an environment for the user to be placed in. I created a simple control scheme that allows the user to move around the environment. I designed and implemented two algorithms for bounding tree creation and collision detection. I created the models needed for the study and used a model color change to act as the collision reaction. These accomplishments are integral to the functioning of this research and are important to have correctly implemented.

3 Results

This section aims to discuss the data gathered from the study, focusing mainly on the measurements that help answer the research question. By the end, the relevant measured data collected should be clear.

User Number	Sex	Union Year	Game Experience	OS
1	Male	Senior	6	Windows
2	Male	Junior	7	Windows
3	Female	Senior	1	Mac
4	Male	Freshman	5.5	Mac
5	Male	Junior	10	Mac
6	Female	Freshman	1	Windows
7	Female	Sophomore	0	Mac
8	Male	Freshman	6	Mac
9	Female	Sophomore	1	Mac
10	Male	Freshman	4.5	Windows
11	Female	Senior	1	Windows
12	Female	Junior	2	Mac
13	Male	Junior	5	Mac
14	Female	Sophomore	3	Mac
15	Male	Freshman	6.5	Mac
16	Female	Sophomore	1.5	Mac
17	Female	Sophomore	1	Windows
18	Female	Sophomore	2	Mac
19	Female	Sophomore	0	Mac
20	Female	Senior	2	Windows
21	Female	Senior	10	Mac
22	Female	Sophomore	2	Mac
23	Female	Freshman	8.5	Mac
24	Male	Junior	6	Mac
25	Male	Junior	7.5	Mac

Table 1: Demographics of users in the study.

3.1 Demographics and Sample Size

For my study, I used solely Union College students as subjects. I was able to gather a total of 25 users. Six of the users tested were first years, eight were sophomores, six were juniors, and five were seniors. Fifteen of the users were females and the other 10 users were males. Users were also asked about their experience with video games using a 0-10 scale, with 0 meaning the user had little to no experience and a 10 meaning the user had plentiful experience with video games. When asked about this, 13 users answered with a 3 or lower, 9 answered between a 4 and a 7, and 3 answered with an 8 or higher. Finally, a total of 18 users were using a Mac during the study, while the other 7 were using a Windows machine. The demographics of the users can be seen in Table 1.

Small Sample Size As outlined in Section 2.2, each user had a complexity of bounding tree for their objects and tested 5 total objects each. This amounted to 20 data points per complexity, since 4 users tested each complexity. The exception was bounding tree complexity 2, where 5 users tested it, amounting to 25 data points in total. This specific complexity having one extra user was due to random chance.

Unfortunately, this was not enough for me to make conclusions for statistically significant differences. I used an online calculator for computing optimal sample size, which took in two values as input: a confidence interval and a standard error. A confidence interval is the level of confidence that the measured value falls within a certain range of values [2]. A standard error is the measurement of statistical accuracy of an estimate, being the confidence interval in this case [2]. In any ideal study, a confidence interval of 95% and a standard error of 5% to 10% makes statistical significance analyses possible [2]. In general, this combination of factors would say that one could be 95% confident that the actual measured mean was within 5% of the true mean for the measured variable.

For computation, I tried different confidence intervals and standard errors in order to see what would produce 20 as the optimal sample size, since this was the number of data points per complexity. What I found was that a confidence interval of 95% with a standard error of 22% would equate to a sample size of 20. I also found that a confidence interval of 62% and a standard error of 10% would produce the same sample size.

For example, let's say that I want to measure the mean immersion level for a bounding tree complexity. Then, I would be able to say that I am 95% confident that the actual mean for the immersion level was within 22% of the measured mean. At a lower confidence level, this would also show that I am 62% confident that the actual mean for the immersion level was within 10% of the measured mean. Either way, I would not be able to reliably make any statistical conclusions from these confidence intervals and standard errors.

Optimal Sample Size Using the optimal confidence interval of 95% and standard error of 5%, I found that the ideal sample size would be 385 data points per bounding tree complexity. If each user tests 5 different data points per their complexity and there are 6 total bounding tree complexities, I would need a total of $385 * 6 = 2310/5 = 462$ users. This would mean that if I wanted to make statistically significant conclusions for differences in user immersion levels between complexities, I would have needed a total of 462 users for my study. The lack of this optimal sample size would further indicate that no statistically significant conclusions can truly be drawn from my data.

3.2 Level of User Immersion

As the study was conducted, the main focus for measurements was the level of the user's immersion. This was measured by the second post-room question from Section 2.2, which asked users to rate how hitting the object met their expectations on a scale of 1-10. Post-study, the results of this question were plotted against the complexities that were tested. This was done to be able to find a trend in the relationship

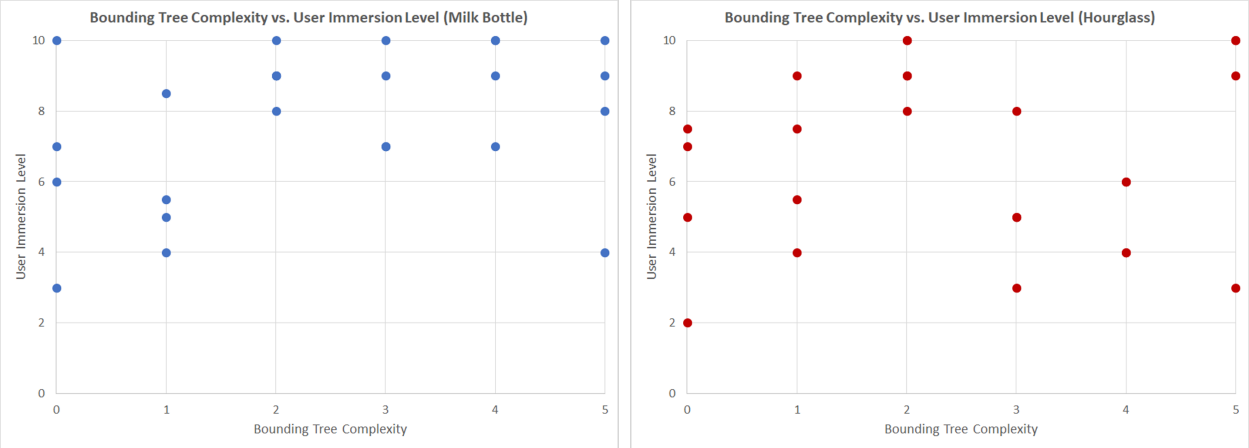

Figure 12: Individual plots for each object, comparing the relationship between bounding tree complexity and user immersion level. From left to right: milk bottle, hourglass.

between bounding tree complexity and user immersion level. This plot was created for each individual object, where the points on the plot represented the user immersion level in the room with the specified object at the given bounding tree complexity. This was done to find the general trend in bounding tree complexity vs. user immersion for each object. For every object, each complexity was tested by 4 users, aside from complexity 2 that was tested by 5. This means that for each plot, the total number of data points is 25. The individual plots for each object can be found in Figure 12 and Figure 13.

Then, these individual plots were combined into one plot to see the overall trend between all of the models. Since there were 5 tested objects and each object had 25 data points, the aggregated plot would then have a total of 125 data points. The aggregated plot for all of the objects can be found in Figure 14. As a note for Figure 12, Figure 13, and Figure 14, some points may overlap on the plot, making it appear as though some complexities were tested less than others. Finally, a plot was created using the average user immersion level at each bounding tree complexity. The standard deviation was included in the form of error bars and a line connecting the data points was included as well. This helps in more accurately seeing the trend in the average user immersion level. The plot containing the mean user immersion levels can be found in Figure 15.

Trends in Individual Plots From the plots in Figure 12 and Figure 13, one can see that the trends are suggestive of an N-shaped curve, where the data increases to maximum, then falls to a minimum, and continues in an upward trend from there. This N-shaped curve is suggestive in the plots for the train, teapot, and chair shown in Figure 13. In these cases, the user immersion level increases up to a complexity 2, decreases at complexity 3, and rises again from there. This matches the idea of an N-shaped plot. On

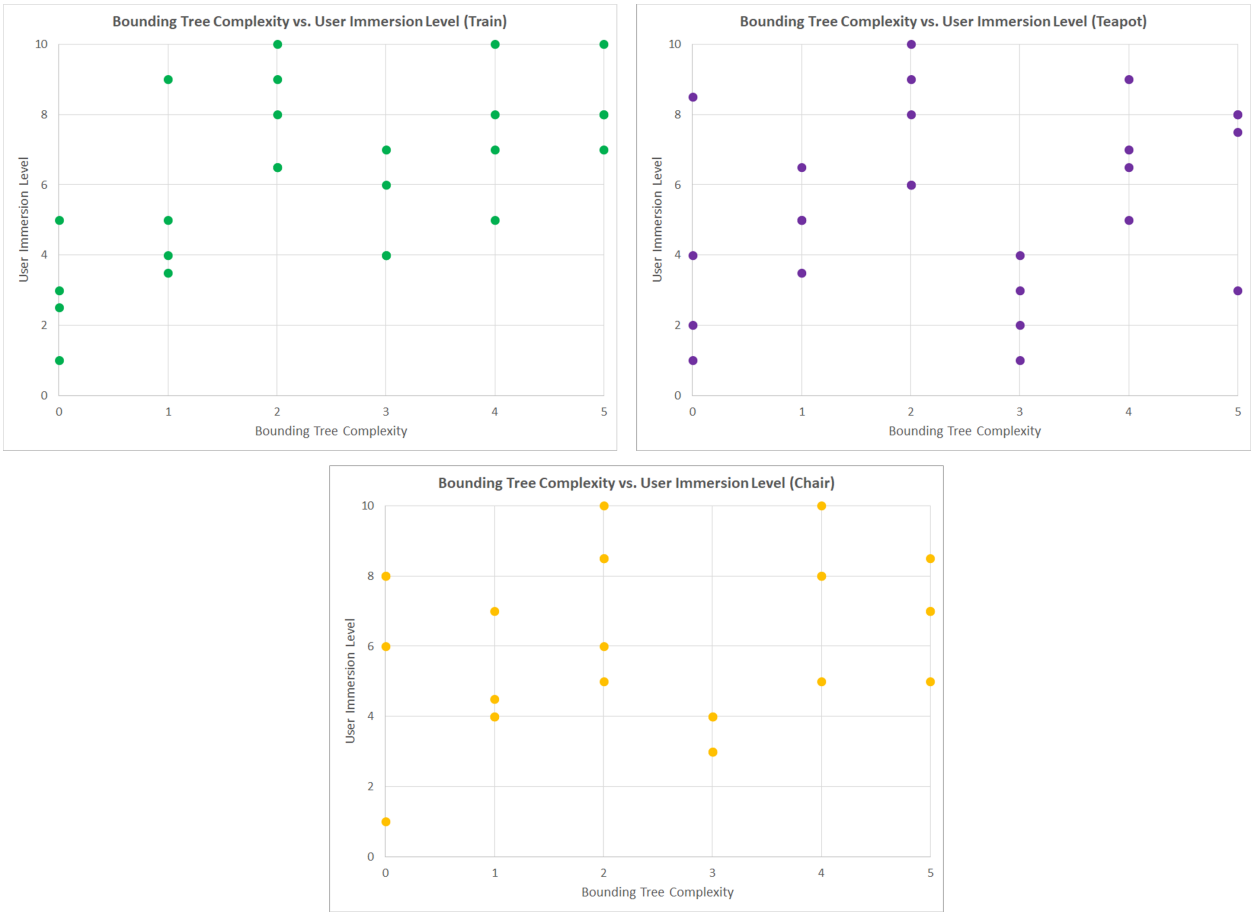

Figure 13: Individual plots for each object, comparing the relationship between bounding tree complexity and user immersion level. From left to right, top to bottom: train, teapot, chair.

the other hand, the plot for the milk bottle shows a curve that is vaguely reminiscent of a logarithmic curve. Finally, the plot for the hourglass was also similar to the N-shaped curve, with a minimum at either complexity 3 or 4. In addition, this plot contained one outlier for complexity 5 where the user immersion level was much lower than the surrounding points.

Trends in Aggregated Plots Moving to the aggregate plot and the plot of the averages, one can see similar trends to those described by the individual plots of the objects. The plot of the averages in Figure 15, with the line connecting each point, shows a much more obvious N-shaped plot, in which the maximum is at complexity 2 and the immediate minimum to its right is complexity 3. However, both plots show other related data that is relevant to the study. Overall, a complexity of 0 and 3 produced collisions that were inconsistent with users, having a low average immersion level, but a high standard deviation. However, complexity 1 tends to have a slightly higher average level of user immersion and with a lower standard

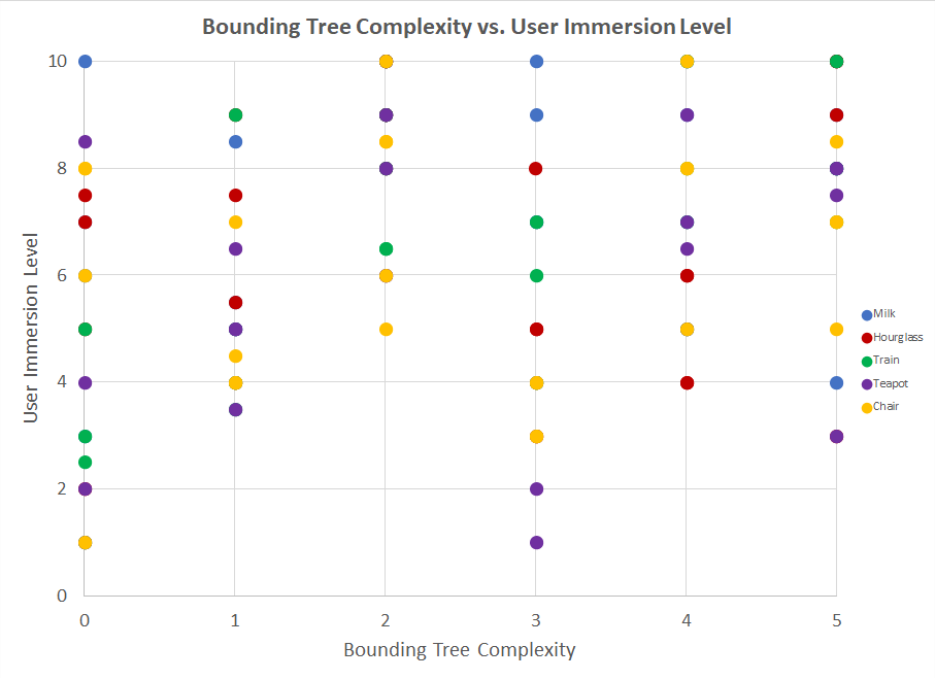

Figure 14: Aggregate plot for all objects, comparing the relationship between bounding tree complexity and user immersion level.

Complexity	Mean	SD	N
BTC 0	4.875	2.832	20
BTC 1	5.500	1.806	20
BTC 2	8.320	1.506	25
BTC 3	4.950	2.395	20
BTC 4	7.225	2.029	20
BTC 5	7.500	2.194	20

Table 2: Descriptive Statistics for user immersion level by each bounding tree complexity.

deviation. In addition, a complexity of 2, 4, and 5 all tend to cluster around an even higher level of user immersion, with little difference between a complexity 4 and 5.

In addition to the plots, I computed the averages and standard deviations for each bounding tree complexity. These values with their corresponding complexities can be found in Table 2. For this table and any future tables, the abbreviation "BTC" stands for bounding tree complexity. In general, complexities 0 and 3 had the lowest average immersion level and the highest standard deviation, whereas complexity 2 had the highest average immersion level and the lowest standard deviation.

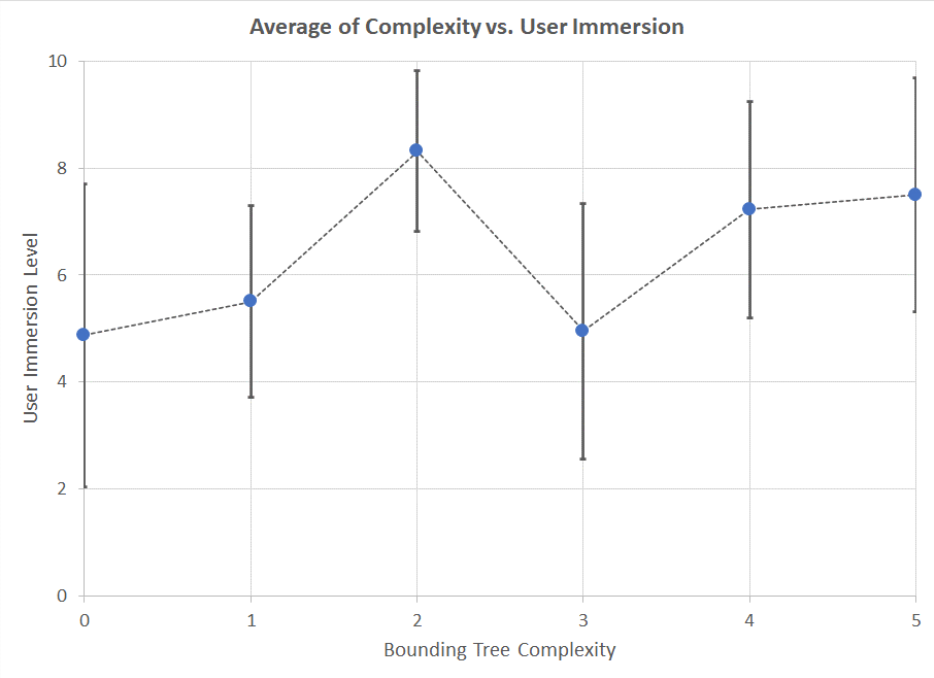

Figure 15: Plot for the mean and standard deviation for user immersion level compared to bounding tree complexity.

3.3 Tests of Significance

Once done with initial data, I ran tests for significant differences in the means of the level of user immersion between all of the complexities. This was a necessary step, as it would be important to know if any of the complexities had significantly different user immersion levels from the others. Before getting into any of the data, I had to recognize what was discussed in Section 3.1. Since my sample size was not big enough, I could not make any claims of statistical significance. Instead, I could claim that there is a suggestion of statistical significance, but further research would need to be conducted to back up that claim. Thus, the results posed from here on out must be taken with a grain of salt.

First, I had to define my variables in terms of whether or not they were categorical or continuous. The independent variable for my study of bounding tree complexity is categorical, since the data is discrete in nature. In addition, the dependent variable for my study of user immersion level, measured on a scale of 1-10, is continuous. All of the users opted to either answer in integers or a rational number half-way between two integers, such as 6.5. Given that I have a categorical independent variable and a continuous dependent variable, I would need to run an ANOVA test of significance in order to test for a significant difference in the means of the user immersion levels [8].

Before running any ANOVA tests, I needed to make sure whether or not I could assume there was a sig-

	Levene's Test	**ANOVA Test**
P-Value	0.019	$<$0.001

Table 3: Resulting p-value for ANOVA test for difference in means and Levene's test for equality of standard deviations for user immersion levels.

Mean Difference for Post Hoc	BTC 0	BTC 1	BTC 2	BTC 3	BTC 4	BTC 5
BTC 0		-0.625	-3.445	-0.075	-2.350	-2.625
BTC 1			-2.820	0.550	-1.725	-2.000
BTC 2				3.370	1.095	0.820
BTC 3					-2.275	-2.550
BTC 4						-0.275
BTC 5						

Table 4: Resulting difference in means from Post Hoc comparisons between bounding tree complexity a homogeneity correction, since the standard deviations of the complexities are not equal.

nificant difference in the standard deviations of the distributions of the complexities. If so, all further tests would need to be ran using a homogeneity correction, which simply corrects for this difference in standard deviations [8]. The outcome for the test for the equality of standard deviations, known as the "Levene's Test," can be seen in Table 3. The result of this test gives a p-value of 0.019. Since the p-value is less than 0.05, I can say that there is a suggestion of a significant difference in the standard deviations of the measured populations. In this case, that would be the level of user immersion for each of the complexities. For the purposes of my study and results, since I only had the data I collected, I had to work with suggestion of a significant difference to mean that I should use homogeneity corrections in any further statistical analyses.

From there, I was able to run an ANOVA test to see if there was a significant difference between the means of the user immersion levels for all of the complexities. This test used a homogeneity correction in order to account for the significant difference in the standard deviations of the complexities. The outcome of this ANOVA test can be seen in Table 3, with it reporting a p-value of less than 0.001.

Finally, I decided to see if there were pair-wise differences in the means between all of the complexities. For a pair-wise test for significant differences in the means, I would need to use a Post Hoc comparison test [8]. In addition, I would still need to use a homogeneity correction to run the test. This test would produce the mean differences and the p-values for each pair of complexities. Due to the homogeneity correction, a high magnitude for the mean difference does not completely correlate to a low p-value in the Post Hoc test. The Post Hoc test for testing each complexity against one another can be viewed in Table 4 and Table 5. Table 4 shows the mean difference between pairs of complexities and Table 5 shows the resulting p-value from the Post Hoc test.

P-Value for Post Hoc	BTC 0	BTC 1	BTC 2	BTC 3	BTC 4	BTC 5
BTC 0		0.959	<0.001	1.000	0.050	0.026
BTC 1			<0.001	0.962	0.073	0.036
BTC 2				<0.001	0.358	0.712
BTC 3					0.028	0.014
BTC 4						0.998
BTC 5						X

Table 5: Resulting P-values from Post Hoc comparisons between bounding tree complexity using a homogeneity correction, since the standard deviations of the complexities are not equal. The cells highlighted in green are pairs of bounding tree complexities where the difference in means could be statistically significant.

4 Discussion

This section aims to discuss the results of the study on a more analytical level.

4.1 Plots and Descriptive Statistics

The current data would suggest that there could be an upward correlation between user immersion and bounding tree complexity. This is most evident for the milk bottle and train, where the lower bounding tree complexities produced lower levels of user immersion and both having a mostly upward trend. However, all models showed an upward trend from complexity 0 to 2, with a dip in user immersion at a complexity 3, aside from the milk bottle. This ties into the N-shaped curve described above. However the reason potential reason for the dip at complexity 3 is discussed in Section 4.2.

This would all indicate that the proposed hypothesis is incorrect, as the shape of the plot does not match the hypothesized shape. Instead, the data could suggest that there is an upward trend until about a level 2. However, it does not level off, since the next highest complexity level lowers the overall immersion level. This would not indicate a minimum threshold, but rather a maximum value in the middle of the data, matching the N-shape. From this shape, it can be determined that as bounding tree complexity is increased, user immersion is also increased to a point, but eventually becomes worse after the peak and gets better from there.

4.2 Tests for Significance

After looking at the plots, it would also be beneficial to analyze the statistical tests for significance. As one can see from Table 3, the p-value of the test came out to be less than 0.001, even with the homogeneity correction. This would again suggest a possible significant difference between the means of user immersion

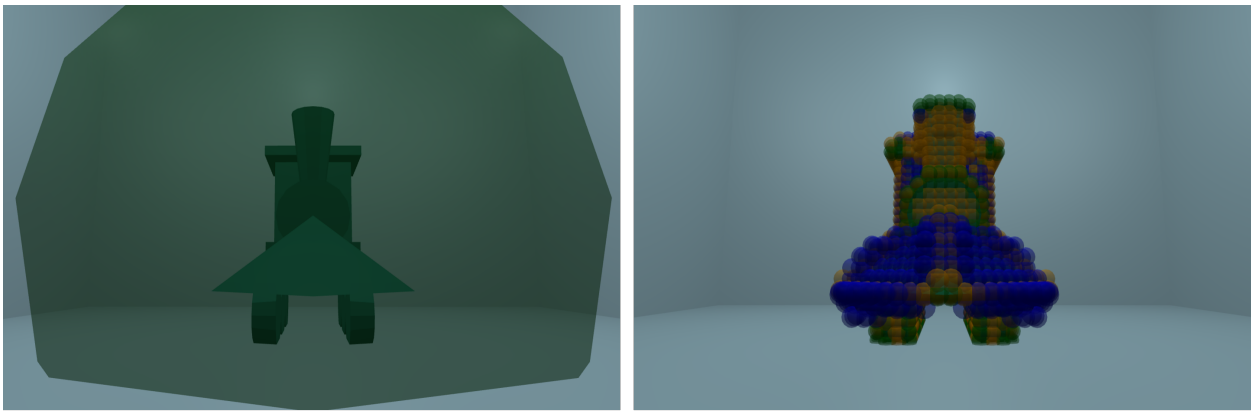

Figure 16: A comparison of bounding tree complexities 0 and 5, using the train object. The complexity 0 has much more empty space within the bounding tree than the complexity 5 does.

for all bounding tree complexities. If this held true, then it would mean that at least one of the bounding tree complexities produced, on average, significantly different levels of user immersion than the others. This outcome is likely due to the difference in the shape and precision of the bounding trees. Even without the Post Hoc tests, one could use Table 2 to speculate on this possibility. Consider the complexity with the second highest average level of user immersion, complexity 5, and the complexity with the lowest average level of user immersion, complexity 0. The shape of the bounding tree for a bounding tree complexity 0 is much less precise, since its volume of collision is much bigger and has much more empty space. Thus, users are more likely to hit the object when they are not expecting to, versus a complexity 5. The difference in the sizes of the bounding tree and the amount of empty space between the two complexities can be seen in Figure 16. From this example, it is already clear that if there was a significant difference in the means of the user immersion, that it could be due to the independent variable of bounding tree complexity.

Post Hoc Tests As one can see, there are a few suggestions of significant difference in the Post Hoc tests. The three most obvious are for the complexity pairs of (0,2), (1,2), and (2,3), since the p-value provided for each is less than 0.001. As for other pairs of complexities, (0,4), (0,5), (1,5), (3,4), and (3,5) all suggest a possible significant difference in means, all having p-values of less than 0.05. This would mean that for all of these pairs of bounding tree complexity, there could be a significant difference in the average level of user immersion between the two. This would result in one bounding tree being more optimal than the other based on the direction of the mean difference. For these pairs, if there is a negative mean difference in Table 4, then the bounding tree complexity in the column of the matrix is the more optimal. Similarly, a

positive mean difference means the bounding tree complexity in the row of the matrix is the more optimal. However, since the sample size is not large enough, I cannot make any hard conclusions on significant differences. If this were the case, it would suggest that bounding tree complexities 2 and 5 are significantly different from 0, 1, and 3. In addition, complexity 4 would also be significantly different from 0 and 3. For differences between the extremes, being 0 or 1 versus 4 or 5, this could be due to the aforementioned idea that the size of the bounding tree and amount of empty space are factors in this difference. This argument could also apply to complexity 2 in comparison to 0 and 1, since the difference in bounding tree size between 1 and 2 is quite big. A bounding tree complexity of 1 has at most 8 leaf nodes, whereas complexity 2 has at most $8^2 = 64$ leaf nodes.

Uncanny Valley and Expectation of Complexity 3 The main point of interest from the Post Hoc tests would be bounding tree complexity 3. The above argument would not be able to be applied; the immersion level at this complexity was much lower than the adjacent complexities with a much higher standard deviation. This dip in immersion at complexity 3 could be because complexity levels 0-2 are obviously shocking in their range of collision, which catches the user off guard immediately. However at level 3, the range of collision is more subtle since it is smaller. Therefore, the users could have been caught off guard when they were more focused, rather than when the users were simply walking around, which was more likely to happen at lower complexities. Thus, immersion would be broken more so being focused than being caught off guard from the start. This would also explain why a complexity 3 has such a low average immersion level, yet a high standard deviation. The standard deviation would be accounted by the difference in how focused users were when in the application: more focus means immersion was broken harder, less focus means immersion was broken less. It is likely that users had a different level of focus when performing tasks, hence the large difference among the data points in Figure 14, the high standard deviation reported in Table 2, and the suggested significant differences in means reported in Table 5. In addition, this difference in focus would only affect complexity level 3.

While it may be more precise than previous complexities, complexity 3 still has enough empty space to account for surprising collisions. Bounding tree complexities of 4 and 5 however have much less empty space. In this case, users would also be focused when "accidentally" hitting the object, but they would be more likely to expect hitting the object since the volume of the whole bounding tree is much tighter around the object. This is backed up by the answers given by users that tested a complexity 4 or 5. Many of them had indicated that when they hit the object, they were "pretty much expecting that [they] were going to hit the [object]." Comments like this were much less frequent in users that tested a complexity 3. This

would indicate a kind of "uncanny valley" [4] with bounding tree complexity 3, where the collisions are just subtle enough that users are more focused when hitting the object, but still have enough empty space to be unrealistic. This would cause complexity 3 to break immersion harder than the adjacent complexities, hence the uncanny valley.

In addition, there is something to be said about "expectation." As the user goes through the application, it is likely that their expectations for the collisions is raised. So the earlier the collision, the lower the expectations. Thus, a bounding tree complexity 3 has a small enough volume that the user would take longer to first hit the object, thus resulting in a higher expectation of the volume for collision. However, the volume of the bounding tree would be just big enough to still catch users off-guard and look unrealistic. Essentially, a bounding tree complexity 3 is just good enough to raise expectations until first hit, but the size of the bounding tree is still big enough to be noticeable and break immersion.

Conclusions from Discussion With all of the data together, it is possible to gather that there is an N-shaped curve in the relationship between bounding tree complexity and user immersion. In this curve, there could be significant differences between the average immersion level produced by certain pairs of complexities, with there being a possible significant difference between all of them. This curve and the significant differences in user immersion could be explained by the shape and size of the bounding trees. It would also be explained by the circumstances surrounding bounding tree complexity 3 that lead it to be an "uncanny valley" for immersion [4]. These circumstances would amount to a bounding tree complexity 3 having enough empty space to be realistic, while also breaking immersion harder than lower complexities, due to how much more subtle the collisions are compared to those lower complexities.

4.3 Validity Threats

Maturity Threat In terms of the data, there were two possible sources of validity threats. For one, users tended to learn more about the application as they went on. In some cases. the users would have similar comments on their surprise of the collisions between the first and last room, but the last room would have a higher score. This was likely because the user was more used to how unrealistic the collision would be and in many cases, they would note that they supposed that that was how the application worked. This would indicate a maturity threat.

Spatial Awareness Issues The other potential threat was that some users would indicate having issues in spatial awareness. This was likely due to the walls, ceiling, and floor having no texture and equal

lighting. Another potential source of this issue could be that the lights were transparent which, in hindsight, was a poor aesthetic choice that I had made towards the beginning. It is likely that this messed with the stereoscopy and depth perception of the user, thus impacting the user's immersion, as discussed in Section 1.1.3. This could have made collisions seem worse or better than they were, since users could not accurately get a read on their position in the application.

Infallible Strategy Confusion However, there was one more potential validity threat that was more subtle than the other two, being a confounding variable. For this confounding variable, consider the following. The way the application is set up with the stick and the lights is reminiscent of a game of pool, where the user could "line up the shot" for each of the lights and the could simply walk into them. This was not an intentional design choice of mine, but the comparison is quite obvious. In fact, during the study I had noted that every single user had implemented this strategy at least once. This is due to the fact that this strategy of "lining up the shot" and walking forward is infallible; in the mind of the user, if the shot is lined up and they just walk forward, they should hit the light. In many cases, this resulted in the users hitting the object first, which did confuse them. This was evident in the responses given by users to the question on their surprise of hitting the object. At least 6 users noted that they were confused by the fact that they "hit the object before the light." This is what is to be expected from the application. However, one must ask the question: what were they confused by? It is certainly possible that the confusion came from the complexity of the bounding tree and the empty space it provided. However, it is entirely valid to assume that the user was confused by the fact that the infallible strategy of "lining up the shot" had failed. Though not necessarily conscious, this strategy is at least subconsciously foolproof, evident by the fact that every user implemented it. However, when a foolproof strategy proves the user to be a fool, this creates a cognitive dissonance. Thus, both options for surprise are entirely valid and it is impossible to tell which was the true reason behind the surprise from the data and notes I had collected. This would be a confounding threat as the strategy used to remove lights was not a factor to be thought of as threatening prior to conducting the study. Thus, this factor could also threaten the validity of the data collected.

5 Conclusions and Future Work

In conclusion, I ran a between-subjects user study in order to test the relationship between bounding tree complexity and user immersion. The results I obtained would suggest that there is a possible relationship between the two, where a complexity level 2 would be a maximum and a complexity 3 would be signifi-

cantly lower than that. This could suggest an "uncanny valley" for complexity 3, making it not optimal. The results would also suggest that there could be a significant difference in the average user immersion level produced by bounding tree complexities 0, 1, 3 and 2, 4, 5.

This research aimed to explore the balance between simplicity and complexity of bounding trees by having users indirectly interact with objects of varying complexities of bounding tree. This was done in order to understand the effects that bounding tree complexity had on user immersion. This balance is a useful area of research, as it allows for realistic-looking collisions and responses while keeping a complexity that either does not effect performance or is not redundantly complex. Knowing this balance could help any software developer looking to start a career in 3D applications, as they could use this as a baseline for creating bounding trees that sacrifice realism for performance, and vice versa.

For future work of this research, I believe that I could have changed the setup of the bounds of the room to have texture to avoid the threat of lack of spatial awareness to the data. I would have also made the lights completely opaque to avoid that as a potential issue. In addition, I believe that I could have had a better way of testing the user's immersion level than asking them non-leading questions about it. Finally, had COVID-19 not been an issue, I could have tested my study in VR instead of on a website, as the user would be more immersed in a VR application. This is because they would actually be placed in the environment, rather than controlling a character in it. This could have made for a much different study, one that I would love to see conducted some day.

6 Acknowledgements

I would like to thank Matthew Anderson for his constant help in this project, whether it be for small questions about setup or larger questions involving implementation. I would like to give a specific thanks for pointing out my misunderstandings in the design of the two algorithms that defined my thesis for the first half of this research. I would also like to thank the Union College Student Research Grant Committee for approving my funds so that I would be able to conduct the study and pay my participants. In addition, I would like to thank Matthew McClosky for helping get my code on a Union College web-page for users to be able to access. I would not have been able to conduct my study without this set-up. Finally, I would like to thank everyone who supported me in any way along the way, whether it be participants or those who listened to my ideas for the project. All of you helped more than you know.

References

[1] Christer Ericson. *Real-time collision detection*. CRC Press, 2004.

[2] Avijit Hazra. "Using the confidence interval confidently". In: *Journal of thoracic disease* 9.10 (2017), p. 4125.

[3] Arun K. Kulshreshth and Joseph J. LaViola. "3D User Interface Technologies and Games". In: *Designing Immersive Video Games Using 3DUI Technologies: Improving the Gamer's User Experience*. Cham: Springer International Publishing, 2018, pp. 1–7. ISBN: 978-3-319-77953-9. DOI: 10.1007/978-3-319-77953-9_1. URL: https://doi.org/10.1007/978-3-319-77953-9_1.

[4] William Lidwell, Kritina Holden, and Jill Butler. *Universal principles of design, revised and updated: 125 ways to enhance usability, influence perception, increase appeal, make better design decisions, and teach through design*. Rockport Pub, 2010.

[5] Ryan P McMahan et al. "Separating the effects of level of immersion and 3D interaction techniques". In: *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*. 2006, pp. 108–111.

[6] Ramakrishnan Mukundan. "Collision Detection". In: *Advanced Methods in Computer Graphics: With examples in OpenGL*. London: Springer London, 2012, pp. 231–276. ISBN: 978-1-4471-2340-8. DOI: 10.1007/978-1-4471-2340-8_9. URL: https://doi.org/10.1007/978-1-4471-2340-8_9.

[7] Lothar Pantel and Lars C Wolf. "On the impact of delay on real-time multiplayer games". In: *Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*. 2002, pp. 23–29.

[8] Jennifer Preece, Helen Sharp, and Yvonne Rogers. *Interaction Design: Beyond Human-Computer Interaction*. John Wiley & Sons, 2015.

[9] Matheus Henrique Junqueira Saldanha and Paulo Sérgio Lopes de Souza. "High performance algorithms for counting collisions and pairwise interactions". In: *International Conference on Computational Science*. Springer. 2019, pp. 182–196.

[10] Aditya Ravi Shankar. "Physics Engine Basics". In: *Pro HTML5 Games*. Springer, 2012, pp. 39–64.

[11] Nilson Souto. *Video Game Physics Tutorial - Part II: Collision Detection for Solid Objects*. Mar. 2015. URL: https://www.toptal.com/game/video-game-physics-part-ii-collision-detection-for-solid-objects.

[12] Jari Takatalo et al. “User experience in 3D stereoscopic games”. In: *Media Psychology* 14.4 (2011), pp. 387–414.

[13] Oren Tropp et al. “Temporal coherence in bounding volume hierarchies for collision detection”. In: *International Journal of Shape Modeling* 12.02 (2006), pp. 159–178.

[14] Costas Tzafestas and Philippe Coiffet. “Real-time collision detection using spherical octrees: virtual reality application”. In: *Proceedings 5th IEEE International Workshop on Robot and Human Communication. RO-MAN'96 TSUKUBA*. IEEE. 1996, pp. 500–506.