Simulating Echolocation Through the Use of Slow-light Radiosity

By Daniel John Ellis - UP940148

Supervised by Dr. Jacek Kopecký

School of Computing

Final Year Engineering Project

PJE40

May 6, 2022

Abstract

Echolocation is a heavily used navigational technique utilised by many different creatures. Despite being so widely used, there are very few visual resources that demonstrate how those that use echolocation perceive the world around them. This project follows the development of a web-based application which aims to serve as a visual aid to help people better understand the concept of echolocation, and how it can be used to explore one's surroundings. This is done by using a modified version of the radiosity global illumination algorithm in which the speed of light is slowed down to the speed of sound. This report details how the system is designed and how 3D scenes were created to imitate the flight of a bat in the wild. As an educational tool, the success of the system is still undetermined, and the testing of the system revealed some performance issues that will need addressing before it can be deployed as a working product. There are also some recommendations for future improvements that can be made to the system which were not implemented here due to time constraints.

Consent to share

I consent for this project to be archived by the University Library and potentially used as an example project for future students.

Table of Contents

List of Tables

List of Figures

List of Code Listings

Acknowledgements

Thank you to:

My supervisor, Dr. Jacek Kopecký, for providing constant support and guidance throughout the course of this project, and my moderator, Mark Venn, who provided support and encouragement when this project was demonstrated to him.

My friends and family for putting up with me whilst I've been working on this project, especially my housemates: Kirsty, Tom, Lily, and Misia.

My course leaders, Dr. Matt Dennis, Dr. Jacek Kopecký, and Dr. Rich Boakes for always providing support in all areas of academic life.

All of the anonymous participants who provided feedback on the software artefact, which gave useful insight and helped to fuel my evaluation.

Chapter 1

Introduction

> "Because we rely largely on vision to perceive the world, we find it difficult to comprehend the challenges faced by organisms that use other senses for perception."
> Jones (2005)

As stated by Jones (2005), we use our vision as a main method of perceiving the world around us. Given that over a thousand species around the world, including bats, toothed whales, and some people, use echolocation to navigate their surroundings and find food (Langley, 2021). Developing an understanding of how this navigational system works will help us to (A) better understand the habits of creatures that use it, and (B) potentially improve accessibility for blind people that utilise it.

Echolocation has been studied by countless people since the term was first used by Griffin (1944). The principles of echolocation are also used in many modern systems such as sonar scanning and fetal ultrasound scans (JGuerra, 2019). However despite the in-depth studies around the field, I'm unable to find any resources that visualise how echolocation might be perceived by the creatures that use it.

Because of the lack of resources that can be found to visualise this intricate system, this project aims to fill this gap by using a slowed down global illumination algorithm called radiosity to simulate how the information from

echolocation might be perceived by a bat in flight. The system will be built upon the existing *Slow-light Radiosity* system created by Kopecký and Mattone (2020) which implements a slowed down version of the radiosity algorithm in an array of 3D environments.

1.1 Aim

The primary aim of this project is to create a web application that will allow users to view a simulation of how a bat might perceive the information it receives from echolocation, with a set of YouTube videos of the scene/s implemented to act as an accompaniment to the system. The basis behind this project is that it could be used in an educational environment to provide a better understanding of how echolocation works.

1.2 Objectives

The objectives that will be needed to meet the project aims are:

1. Collect information about how the existing system operates.
2. Develop an understanding of the radiosity algorithm.
3. Review literature relating to:
    - The characteristics of sound waves.
    - The principles of a bat's echolocation system.
4. Construct a scene of a forest to act as the bat's habitat.
    (a) Successfully import tree models into the scene.
    (b) Use poisson disc sampling to generate a natural looking distribution of trees.
5. Devise a way of imitating echolocation pulses along a bat's path.
    (a) Create a flight path for the bat and a camera that follows that path.

(b) Find a way of setting initial exitance at multiple stages in the simulation.

(c) Create a means of hiding light sources from the viewer.

6. Continuously test features of the system.
7. Create videos of the scene/s and post to YouTube.
8. Gather user feedback regarding the usability and usefulness of the system.
9. Evaluate successes, failings, and limitations of the project.
10. State ideas and improvements for future development.

1.3 Report structure

Table 1.1 shows an overview of this report's structure along with the topics that will be discussed. Chapters 1-3 detail the problem topic and the research surrounding it. Chapters 4-8 cover the details about how the software artefact was created. Finally chapters 9 & 10 bring the project to a close by evaluating the artefact and the methods used to develop it, and then suggesting future modifications that could be made.

#	Chapter	Description
1	Introduction	Initial overview of the project as well as it's aims and objectives
2	Background	An detailed explanation of how the radiosity algorithm works
3	Literature Review	Review of available literature relevant to the project's topic
4	Project Management and Methodology	Discussion and justification of this project's chosen methodology
5	Requirements	Specification of the requirements laid out for the project
6	Design	Explanation of the design choices made throughout and their justifications
7	Implementation	Detailed overview of the steps taken to implement the chosen design
8	Testing	Statement of the testing carried out and discussion of results
9	Evaluation	Discussion of the project's success, and critique of the steps taken throughout
10	Conclusions	The project's final notes and future considerations

Table 1.1: Report structure.

Chapter 2

Background

This chapter will go over some background information about how the radiosity algorithm works that will help provide a better understanding for the rest of the report.

2.1 How light works

For the purposes of this project, when talking about light, it is in reference to the range electromagnetic radiation that exists between Ultraviolet (UV) light, and Infrared (IR) light, better known as "visible light".

If you imagine a dark room, with a directed light such as a torch in the centre of it, pointed upwards, and no other light sources in the room. When the torch is turned off, the room is pitch black, and nothing can be seen. Now imagine turning the torch on. Light flows from the torch, up towards the ceiling, and then that light is reflected from the ceiling, around the room and towards the floor (Fig. 2.1).

This is how we perceive light, and it all happens near instantaneously, at the speed of light. The amount of light reflected off each surface in the room is dependent on each surface's reflectance, and in the real world we have surfaces like mirrors which can direct the light that hits them ("Mirrors", n.d.).

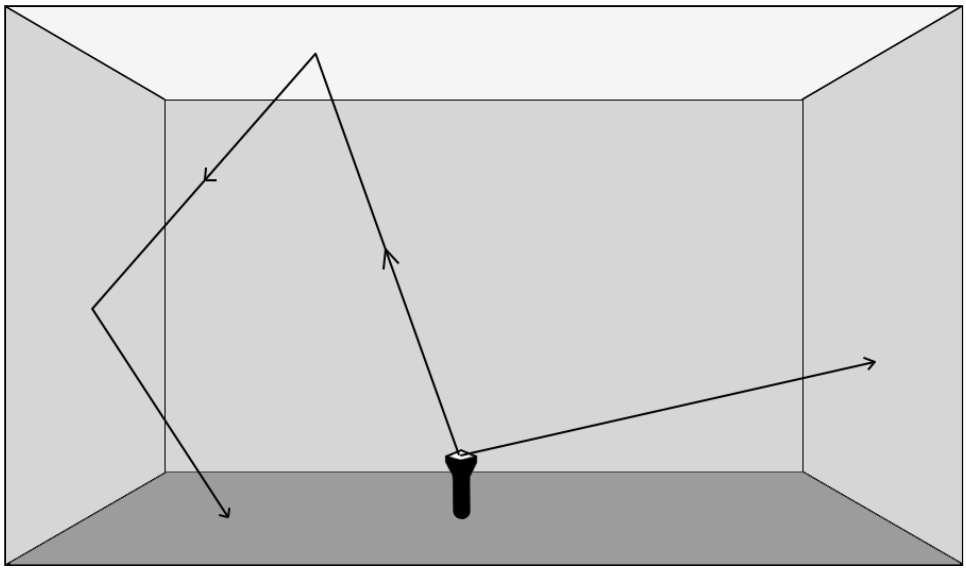


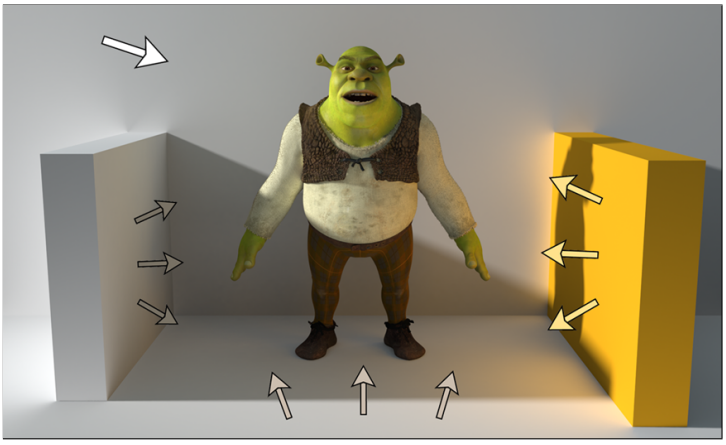
Figure 2.1: Light flow in a room with a single torch.

2.2 Radiosity

Radiosity, as a concept is far too detailed to explain in full depth within this report, with Ashdown (1994) taking some 500 pages to explain it in one book. Hopefully by the end of this section, you, the reader, will have a better understanding of the basic principles underlying the radiosity algorithm that will be used in this project.

The radiosity algorithm was first created as a mathematical tool that could render the inside of an empty box (Ashdown, 1994, p. 1; Goral et al., 1984). Since it's creation, the radiosity algorithm has been deployed in many commercially available products such as 3ds Max ("Modeling Global Illumination with Radiosity", 2018), and the Enlighten game engine (jun.yoshino, 2021).

(a) Local illumination.



(b) Global illumination.

Figure 2.2: Shrek scene when lit with local and global illumination.
Source: Tabellion (2010)

2.2.1 What it does

The radiosity algorithm is a global illumination algorithm, which means it models light in a scene where the light reflected by a surface A is dependent on the light coming directly from a light source, as well as the light reflected from a surface B towards A (Fig. 2.2b). This is different to a local illumination algorithm such as the Phong illumination model, which does not take into account the light reflected from a surface B towards A, but only considers light coming from the source directly towards the surface (Fig. 2.2a).

The two primary forms of reflection in 3D graphics are specular and diffuse; ambient reflection is also used, however ambient light is a crude model of diffuse which does not consider the distance to, or angle of incident of, the light in the scene (BBeck1, 2016, para. 2; "3D Common Properties", n.d.). Specular reflection is the type of reflection observed in most standard mirrors, where the light hitting the surface will be perfectly reflected, which is to say that the angle of incidence is the same as the angle of reflection. Diffuse reflection contrasts this, which means that the light which hits the surface will be scattered in many different directions.

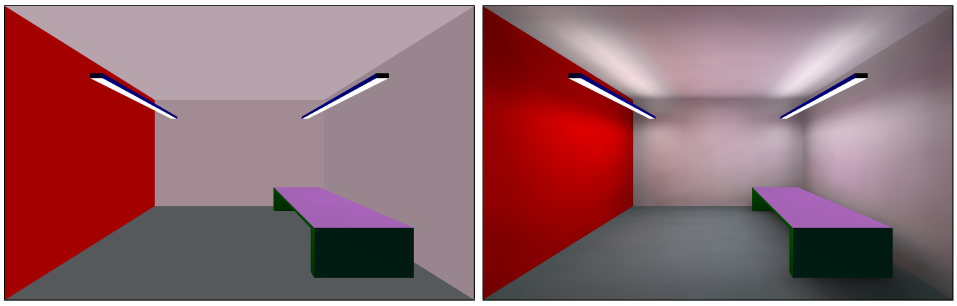
All surfaces being lit using the radiosity algorithm are modelled with Lambertian reflection, which is a specific form of diffuse reflection where the light that hits the surface will be scattered equally in all directions (ideal diffuse) (Ashdown, 1994, p. 8-9). Because all surfaces are modelled with Lambertian reflection, the radiosity algorithm is not useful for lighting scenes with shiny/glossy objects such as mirrors. Modelling light on these surfaces would require some algorithm that utilises specular reflections such as ray-tracing (NVIDIA, n.d.) or path-tracing (Caulfield, 2022).

2.2.2 How it works

Every surface in a scene is subdivided into a mesh of patches. These patches act as smaller surfaces which allow light to be displayed with a higher level of detail (Fig. 2.3).

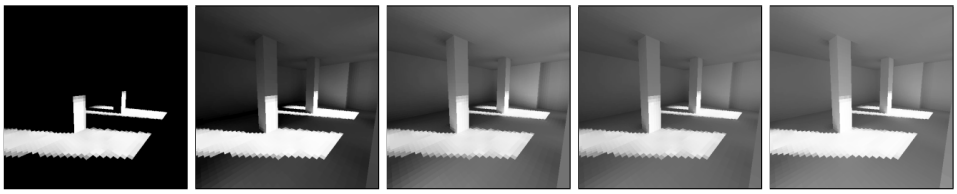
Every surface (and every patch by proxy) has two main properties: *reflectance*, which is a measure of the amount of light that the surface will reflect; and *emittance*, which describes the light emitted by the surface initially (initial radiant exitance). Both of these attributes are commonly stored using distinct RGB channels.

To begin with, every patch E_i has a form factor calculated between itself and every other patch E_j. This form factor F_{ij}, in layman's terms, represents the proportion of E_i's view, that contains E_j. Because all surfaces exhibit Lambertian reflection, the proportion of E_i's view that contains E_j will be the same as the proportion of light emitted from E_i that E_j receives.



(a) No subdivision. **(b)** Subdivided 20 times.

Figure 2.3: Simple room scene when room is not subdivided, and when subdivided 20 times.
Source: Kopecký and Mattone (2020)

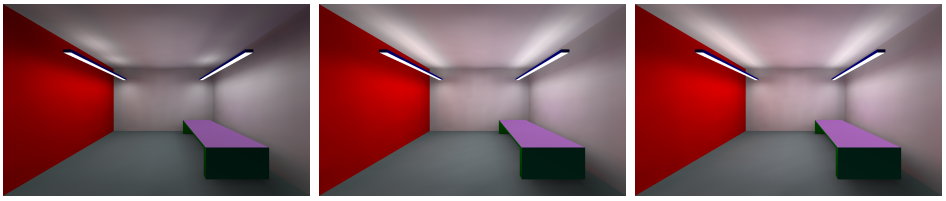


(a) Iteration 1. **(b)** Iteration 2. **(c)** Iteration 3. **(d)** Iteration 4. **(e)** Iteration 16.

Figure 2.4: Simple black and white scene after different iterations of the radiosity algorithm.
Source: Elias (2000)

Radiosity works iteratively. In every iteration, every patch that can see any light source receives some of the light that is being emitted by all the sources that it can see. When a patch receives light from a source, it becomes a light source itself, and then it can be used as a light source for other patches in future iterations.

This iterative process will continue on, with patches reflecting light to one another as many times as required (Fig. 2.4). In static scenes, with constant light sources that don't change in any way, this iterative process will continue to make small changes to the light in the scene until the changes become so insignificant that the lighting appears to have stabilised completely (Fig. 2.5).



(a) Iteration 30. **(b)** Iteration 60. **(c)** Iteration 90.

Figure 2.5: Radiosity changes becoming less noticeable after different iterations. **Source:** Kopecký and Mattone (2020)

2.2.3 Useful terms

Some useful terms used when talking about radiosity are named and explained below.

Term	Explanation
Exitance	The amount of light emitted at a given time.
Emittance	The "initial radiant exitance". The amount of light initially emitted from a surface.
Reflectance	The amount of light reflected by a surface.
Form factor	The proportion of light emitted by one patch which is received by another patch.

Table 2.1: Useful radiosity terms.

Chapter 3

Literature Review

3.1 How sound works

Sound is a longitudinal wave that exists in some medium, and these waves cause fluctuations in pressure that cause vibrations, which our ears detect, and our brains perceive them as sound as we know it (Young et al., 2012, p. 510).

When a sound wave meets the interface of two media, a portion of the wave will be reflected, whilst another portion of the wave will permeate through the new medium. The amount of the wave that gets reflected is determined by the diversity (impedance ratio) of the two media. For example, if the sound wave travels through the air, and then reaches a concrete barrier, because of the difference between concrete and air, most of the sound wave will be reflected, and very little will be absorbed ("Reflection, Refraction, and Diffraction", n.d.).

As sound travels through any medium, some of the sound is lost due to absorption. This is because sound is a vibration of the atoms that make up a material, and due to molecular collisions that occur within that material, some the sound wave's kinetic energy is converted to heat energy (Kurtus, n.d.). Some materials have been designed specifically to absorb sound waves, as a form of sound insulation. The design of these materials has to take

into account multiple factors that affect the sound absorption coefficient of the material, such as fibre size, thickness, porosity, and density (Seddeq, 2009).

3.1.1 Acoustic attenuation and light absorption

Acoustic attenuation is the measure of a sound wave's energy loss as a function of depth of a material (Eq. 3.1) (Mischi et al., 2014, p. 364)

$$I(z) = I_t e^{-\alpha z} \tag{3.1}$$

Where I_t is the transmitted acoustic intensity, α is the attenuation coefficient, and z is the covered distance.

Acoustic attenuation is incredibly similar to Lambert's law of absorption for light waves, which is (Mazda, 1993, Sect. 7.10.2.1)

$$I = I_o e^{-Kx} \tag{3.2}$$

Where I and I_o are the resulting and initial intensity respectively, K is the absorption coefficient, and x is the distance covered.

As can be seen in figure 3.1, the absorption coefficient of light in different materials trends downwards as the wavelength of the light increases. As wavelength is inversely proportional to frequency (Eq. 3.3), it can be said that the absorption coefficient of light trends downwards as the light frequency decreases.

$$Frequency_{(Hz)} = \frac{Wave\ Velocity_{(ms^{-1})}}{Wavelength_{(m)}} \tag{3.3}$$

Figure 3.2 shows the sound absorption (acoustic attenuation) coefficients of different materials for different frequencies of sound. The general trend of this relation is that the sound absorption coefficient increases as the frequency of the sound wave increases, save for a few outlying cases such as Epoxy resin from 0 to $\approx$ 700Hz.

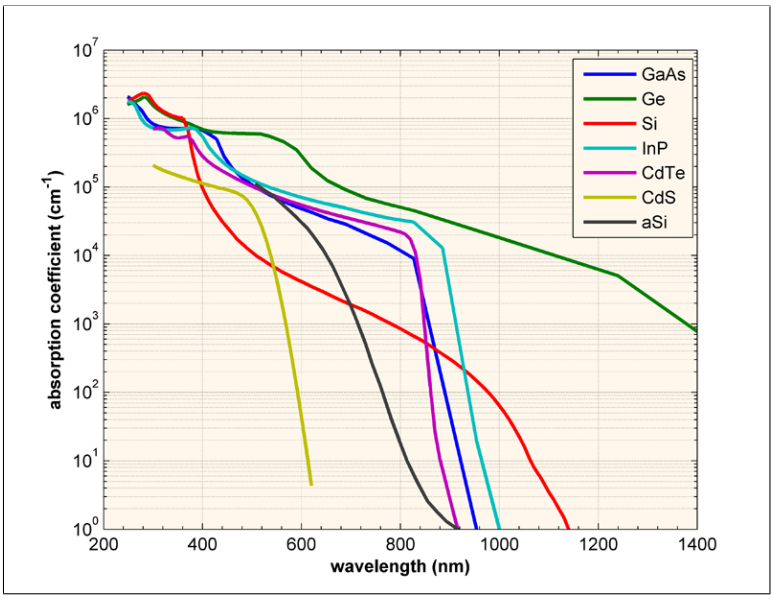


Figure 3.1: The absorption coefficient, α, in a variety of semiconductor materials at 300K as a function of the vacuum wavelength of light.
Source: Honsberg and Bowden (2019)

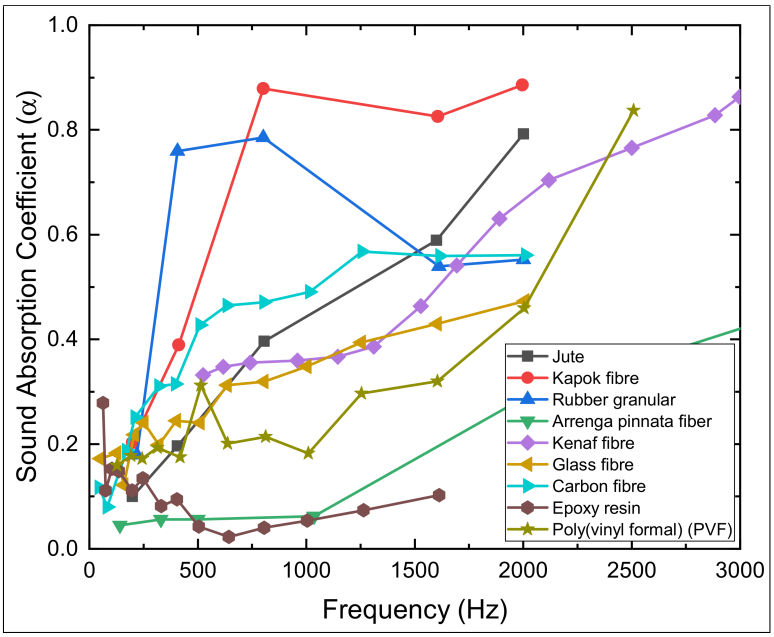


Figure 3.2: Sound absorption coefficients of various traditional materials.
Source: Kumar and Lee (2019, Fig. 1)

Given the similarity between these attenuation and absorption rates, whilst there are many other factors that may impact the rate at which a wave loses it's intensity as it travels through a medium, a general assumption can be made to say that as the frequency of a wave increases, so does the absorption/attenuation coefficient applied to its rate of decay, and as such, waves with a higher frequency decay faster when travelling through a medium than waves with a lower frequency.

3.2 Echolocation

Echolocation, otherwise known as bio sonar, is a process used by animals to sense their environment when visual input isn't an option, for example if the environment is too dark, if the animal has no eyes, or is blind. Echolocation works by creating a sound and receiving the echoes that return from the sound waves reflecting off objects. By comparing the sound waves created, to the ones received, the brain can produce images to interpret the animal's surroundings (Jones, 2005).

Echolocation, as a term, was first coined by Donald R. Griffin in 1944 in a scientific journal. He came up with it because he wanted a term for how bats and blind persons navigate their surroundings using echoes (Griffin, 1944).

3.2.1 In bats

Bats use echolocation to perceive their surroundings and hunt for prey. The range of frequencies for bat echolocation calls are between approximately 11kHz and 212kHz, with most insect-eating (insectivorous) bats using frequencies between 20kHz and 60kHz for their calls (Jones & Holderied, 2007). The limit of a normal human's hearing is bounded between approximately 20 to 20,000Hz (0.02kHz to 20kHz) with the upper bound of this range decreasing somewhat with age (Purves et al., 2018, p. 282). This puts the majority of bat echolocation calls in the ultrasonic frequency range (Berg, 2017).

Sound travels at a constant speed of $\approx 340ms^{-1}$, and because of this bats can determine how far away objects are based on the difference in time between their emitting of the call and the returning of the echo from the surface of an object (Russ, 2013). The reason for bats' use of ultrasonic frequency comes from the size of their insect prey. The intensity of a returning echo diminishes a lot when the wavelength of the emitted call is greater than the size of the insect, so in order for a bat to effectively detect an insect's presence, the wavelength of the call must be equal to or less than the size of the insect (Russ, 2013). Bats commonly hunt mosquitoes for prey (Wilson, 1997), and with mosquitoes ranging between 2-19mm in length (Service, 2012, p. 2), the frequency required to detect a mosquito ranges from 170kHz to 17.895kHz (Eq. 3.3).

When a bat uses a higher frequency call, it will get more detail about the scene in front of it because, the higher the frequency of the call, the more directional it is. This means that the returning echoes are more concentrated to one area and as such they can provide a more accurate picture of their prey (Russ, 2013).

3.2.2 In sonar

Sonar (sound navigation and ranging) is commonplace in boats and submarines (Allwood, 2021), and has been since World War I. The fist Sonar type device was invented in 1906 by Lewis Nixon as a method of detecting icebergs. After this, because of the threats posed by submarines in WWI, the interest in Sonar technology was increased, with the first passive submarine detecting Sonar being created in 1916 and the first active Sonar system being created in 1918 by British and U.S. scientists (Bellis, 2020; The Editors of Encyclopedia Britannica, 2019).

Despite these being the earliest known developments of Sonar systems, the first recorded use of the technique dates back almost 450 years prior with Leonardo da Vinci who, in 1490, wrote "If you cause your ship to stop, and place the head of a long tube in the water and place the outer extremity in your ear, you will hear ships at a great distance from you." (Fahy & Walker, 1998, p. 375).

Sonar is classified into two categories: passive Sonar, which is where sounds are heard coming from external sources, and active Sonar, which is where a sound is generated, and the reflected echoes are detected. What da Vinci wrote about was passive sonar, as he was listening for sounds from other ships, whilst active sonar is seen in Side-scan Sonar systems that are often used to map the sea floor (Geoscience Australia, n.d.).

Chapter 4

Project Management and Methodology

Multiple well-established methodologies were considered for use, however none of these considered methodologies were chosen due to the belief that they would not be beneficial to this project.

Due to the exploratory nature of the project, requirements gathering happens at multiple stages throughout it's duration, as this sort of project has not been attempted before in any form that I can find, and this meant that a list of complete requirements could not be formulated at the beginning. In light of this, the Waterfall method was discounted due to it's rigid structure not allowing for this discovery of new requirements once that stage has passed (Royce, 1970, p. 329).

Whilst considering the Agile methodologies, the recurring theme found within them was the fact that their basic principles are tailored towards projects with customers/clients/sponsors with statements such as "... harness change for the customer's competitive advantage" and "Our highest priority is to satisfy the customer through..." ("Principles behind the Agile Manifesto", 2001). As this project has no customers/clients/sponsors, the workflow doesn't need to be focused around customer input and satisfaction.

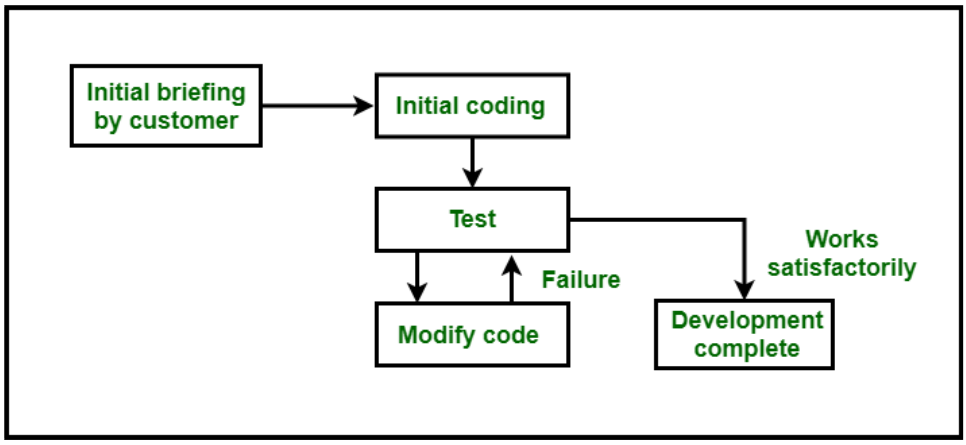


Figure 4.1: Exploratory style of software development.
Source: itskawal2000 and surbhikumaridav (2020)

4.1 The Explorative method

As a result of finding no established methodologies which seem to suit the needs of this project, a methodology was devised specifically for this project, based on insight gained from previous projects that have been undertaken.

The overall process of the methodology is not overly dissimilar to the Exploratory style of software development detailed by itskawal2000 and surbhikumaridav (2020) (Fig. 4.1). With the main differences being that this project has no customer and so the "Initial briefing by customer" is removed, and instead requirements are added when discovered throughout development.

4.1.1 Requirements elicitation

Before the development process could begin, some requirements elicitation was done through: observing the existing system at work, changing small parts of it to see how it operated, and figuring out how the different features of the system interact with one another. This allowed some cursory requirements to be made which could be used to begin development.

As the initial requirements primarily serve to give an idea of where devel-

opment should begin, more requirements are bound to be discovered during the other phases of development. In contrast with the Waterfall method, requirements that are discovered later in development are not just documented for reference when developing a subsequent system, but are instead documented alongside the existing project requirements, at which point it is left to the sole developer to re-prioritise the requirements in order to determine when this new addition should be focused on.

This method of adding more requirements and adjusting the order in which they will be tackled, whilst it may become overwhelming to do in a large project, especially one with a team that have to co-ordinate their work effectively and may uncover many additional requirements in quick succession, it is a method which has been deemed suitable enough for a project of this scale, with only myself as the developer.

4.1.2 Development process

The implementation of this method appreciates the level of volatility that can arise when making decisions throughout development as a solo developer. Without being a part of a team setting, all decisions regarding this project are decided by myself alone. I can gather some wisdom and guidance from my supervisor and other academic staff, however with the lack of a team structure with frequent communication, some design choices and implementation methods may have unforeseen drawbacks due to a lack of experience with specific technologies.

As a result of this, all phases of the development process are unrestricted in where they can go next. If a flaw in the design is found after deploying the system, then the design phase can be revisited without issue.

Overall, this method allows complete freedom in development by removing the restrictions that can be found in established methodologies. This method would likely be very unstable and not suitable for larger scale projects requiring more complex architecture such as databases and APIs, or projects involving a team of developers. However for the scale and complexity of this project, I have considered it to be an acceptable method to follow.

4.2 Project Management

Development projects in business settings usually require developers to provide regular progress updates to their superior/s, who is often a lead developer, or the whole development team will meet regularly to discuss progress made, as it is under Scrum methodology with Scrum meetings. In lieu of other team members or a superior to report progress to, my frequent supervisor meetings were used as progress reports. These supervisor meetings allowed this project to receive regular critique and useful insight, as well as encouraging me to make constant progress without straying too far off track.

As mentioned prior, in section 4.1.1, requirements were added throughout the entire development of this project. As these requirements were created, the development plan changed to accommodate them. The final development plan used for this project is shown in figure 4.2.

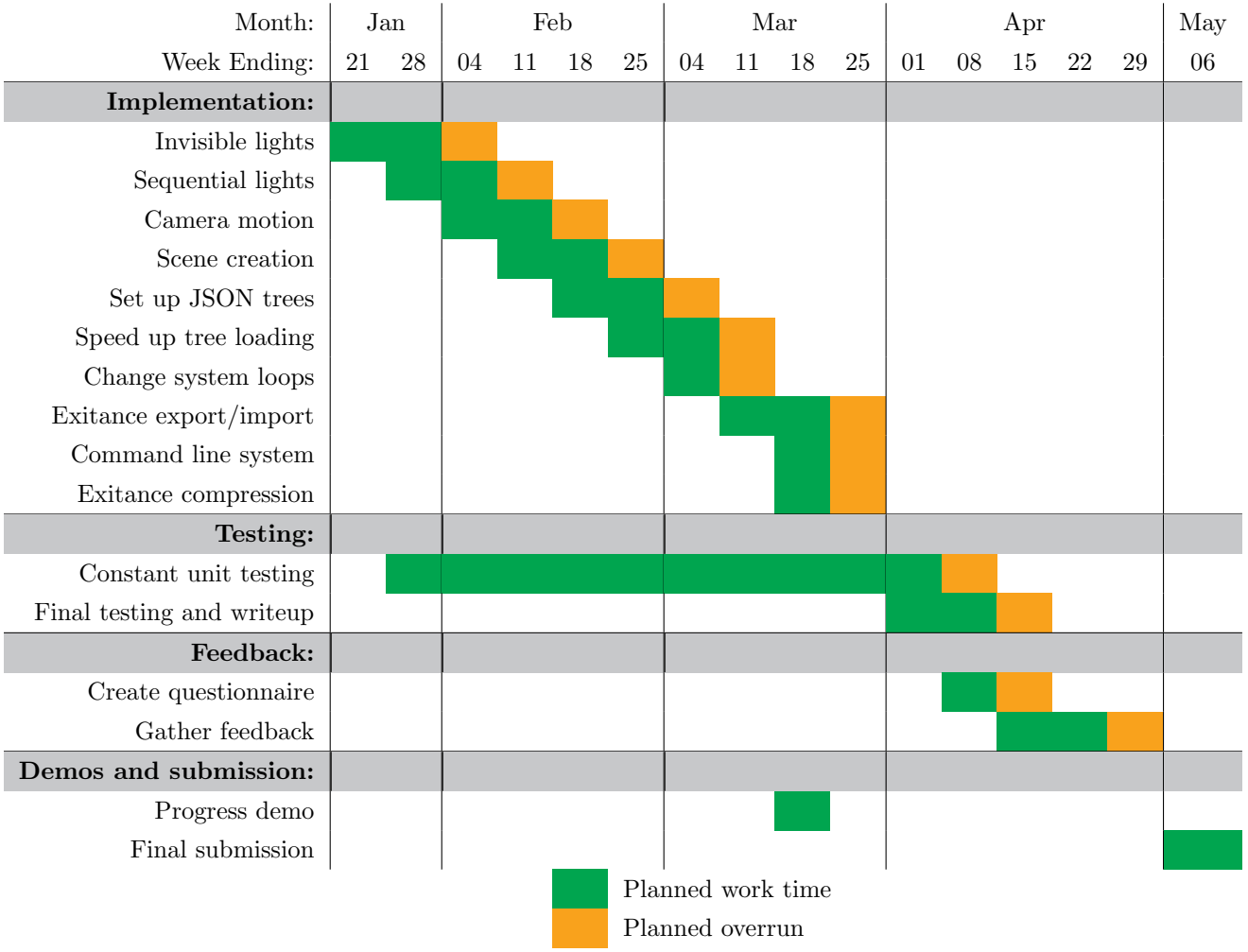


Figure 4.2: Software artefact development plan.

Chapter 5

Requirements

Due to this project's lack of a client and exploratory nature, a decision was made to gather initial requirements through observation of the existing system, and doing background research into the field. Later requirements were then included based on the results of constant testing during development.

5.1 Must Have

MH1 - A moving camera on a pre-planned path. (*Functional*)
In order to simulate the flight of a bat, the camera must be mobile in the scene. Because exitance values are computed before the animation plays, each scene's light sources must be in fixed positions, and the camera must follow a path that intersects with every light source.

MH2 - Application served on GitHub Pages. (*Functional*)
The system is served on a static page, which means that it does not need dedicated server architecture to function. When deploying the system to users for both feedback gathering and general use post-deployment, GitHub Pages is a perfect candidate as it is free and it requires no set-up for users in order to function.

MH3 - Pre-loaded exitance data. (*Functional*)
It became apparent in appendix E.6.3 that scenes with many vertices would take a large amount of time to load in the browser. Because of this, in order to allow participants to test the system and give feedback, the exitance values for all vertices should be computed before deployment and fetched from a file when the system is in use.

MH4 - Light sources that activate sequentially. (*Functional*)
In order to have more than one pulse of light per scene, light sources must have some way of activating when required, and not just at the start of the scene as it is in the existing system[1].

MH5 - New method of storing and loading trees. (*Functional*)
While testing the ability to load trees from STL form in appendix E.4, it was identified that Chrome would exceed its memory limit and crash when loading just one tree in STL form into an empty scene. For this reason, a different method of storing the models must be devised, so that the scene can have multiple tree objects contained within it.

5.2 Should Have

SH1 - Command line calculation ability. (*Functional*)
Because of the memory limitations imposed by browsers, it was discovered in testing that some scenes were able to load into the browser, however the page would crash whilst attempting to calculate the exitance values for the scene (Sect. 8.3.3). Creating a method of calculating and saving exitance values from the command line will allow more complex scenes to be created for simulation.

SH2 - Light sources that aren't visible. (*Functional*)
Because the light sources are replicating the sound pulses from a bat, they should not be visible as physical objects in the scene, as this would mean that the future light sources could be seen by the camera, and would also be casting shadows in the scene.

[1] Original system: radiosity/slowrad.js L168-L177

SH3 - Compressed exitance data files. (*Functional*) (Extends **MH2**)
When exporting a complex scene's exitance data, the JSON file had a size of 374MB. Pushing this file to a Git repository required the use of Git LFS, however files using LFS are restricted from being served on GitHub Pages. Because of this, a method of compressing these files should be used in order to allow deployment.

5.3 Could Have

CH1 - Minor background noise. (*Functional*)
Bats in nature are bound to be near other bats. All of which will be using echolocation for navigation. Because of these other sources of sound, bats likely experience varying levels of interference to their echolocation.

CH2 - Exportable images. (*Functional*)
In order to compare the differences in simulation outputs between different stages of development, a way of exporting the current simulation view to an image file could prove useful.

CH3 - Exportable animation. (*Functional*)
Being able to export the simulation output to an animation file would allow users to view the output, even if they find themselves unable to run the simulation on their current device, thus enabling the outputs of the simulation to be helpful to a wider audience.

5.4 Won't Have

WH1 - Creation of custom scenes and flights. (*Functional*)
The decision was made not to include the creation of custom scenes and flights, as creating a 3D scene editor was felt to be too large a task to fall within the scope of this project, if done well.

WH2 - Different colours to illustrate the doppler effect. (*Functional*)
Allowing for different frequencies of light in the scene would require the rewriting of a large amount of the initial system, and after having to compress exitance files to allow them to be deployed to GitHub Pages in section 7.5.2, the decision was made to only store a single colour channel, thus making this requirement unachievable in this project.

Chapter 6

Design

Because of the exploratory nature of this project, the requirements were ever-changing, and so the design had to be consistently updated to keep up with the changes. This chapter will cover the main design choices made during development.

6.1 User interface

The existing system has a well-formed UI, so the general layout has remained the same. The only change made was the addition of three new buttons. One to toggle between the orbital and flight cameras, another to save an image of the current frame, and a final one to display the help page. These buttons were added to the existing menu bar in the top left corner so that the UI wouldn't become cluttered from additional containers. In keeping with the existing theme, the buttons had a light coloured backdrop and used Font Awesome 5 icons ("Font Awesome", 2020).

6.2 Scene design

Bats have many natural habitats, however the one chosen to model in this project is the forest. Forest scenes were chosen because the layered detail that can be found in a forest made it feel like the final animation would be

more visually appealing and interesting than if some other scene, such as an urban setting, had been modelled.

6.2.1 Trees

It was discovered during testing that STL trees were unable to load into the scene without crashing the browser due to lack of memory (Sect. 8.3.2). JSON has been chosen as the replacement for STL models as it is well-suited towards storing complex data structures, and JavaScript has built-in methods for handling JSON data.

6.2.2 Model placement

To create a realistic forest scene, the tree models needed to be placed with a natural distribution. After some research to try and figure out how to do this, the most common result found was *Poisson disc sampling*. As it is a well-established algorithm, and more than one layer of it can be used in one scene with ease, this was the method chosen to position the tree and bush models. Two different sample sets were used. One was generated for the tree placements, and another, with a smaller minimum radius, was used for the bush placements in each scene.

6.3 Exitance files

As per requirement **MH3**[1], the exitance data for scenes should be calculated once and then imported into the system when required. The exitance data for every vertex in the scene, for every time step, is stored in `vertex.futureExitances`. Once calculations have been completed, a large 2D array can be constructed from these `futureExitances` values, and the array can be exported to a JSON file. As above with the trees (Sect. 6.2.1), JSON is chosen here because it is suited towards structured data and JavaScript has functions to handle it.

[1] MH3 - Pre-loaded exitance data

6.4 Command line system

Testing the browser's ability to export exitance files in appendix E.7, revealed that a new method of calculating the exitance files had to be devised, as the browser could not be trusted to reliably load a scene and perform the necessary calculations. Given that the entire system for these calculations is already implemented in JavaScript, attempting to use some other tool to make the calculations would have taken far too long to implement, which left two viable options to consider: Increasing the testing browser's memory limits to allow the exitances to be calculated in the browser, knowing that they will only need to be calculated once; or creating a minimal version of the system that can run outside of the browser to perform the calculations. The latter option was chosen, as increasing the memory limits of the browser, whilst quick and easy, would only be a short-term solution for this device; however a minimal system running from the command line would be beneficial in the long run, as not all systems support browser hardware acceleration by default.

6.5 Summary

In this chapter we've covered the design choices made, and the reasoning behind each of them. From the visuals of the user interface, the methods used to design scenes, and the handling of exitance files to be used by the system. The next chapter will go over the implementation of the design choices.

Chapter 7

Implementation

This chapter covers the implementation choices that were made throughout the project. As this project serves to create an artefact as a proof of concept, rather than a distributable product, speed of implementation was the primary factor considered, with efficiency and maintainability of the resulting code being secondary factors.

The implementation of the image saving feature, the poisson disc sampling for object placement, as well as the changing of loops in the system to speed up loading can be found in appendix D.

7.1 Development setup

During the implementation stage, linting was done with ESLint (ESLint, 2021) to improve the readability and maintainability of the resulting code. ESLint was configured using the `eslint-config-portsoc` package (Kopecký et al., 2020), and the Atom text editor (GitHub, 2020) was used, with the `linter-eslint-node` package (Dupont et al., 2022) to enable real-time linting.

7.2 Light sources

In order to have the light sources act as origins for echolocation calls, two new properties were created: `isLight`, and `activeTime`. These properties will come into play when calculating `Patch` form factors, and when initialising when each light emitting `Patch` should emit their light.

The `Surface3` class acts as a parent for `Patch` instances, which contains the lighting information regarding their child patches. As `isLight` and `activeTime` are directly related to lighting, it was decided that the `Surface3` class will be where these properties are held.

7.2.1 Invisible lights

The `isLight` property is a boolean which describes whether or not a given surface is part of a light source. It is used both to tell the `ThreeJS` renderer (Cabello, 2020) not to render the surface, and also to ensure that the surface receives no light from the scene, as well as preventing it from occluding any light in the scene and casting a shadow.

Another method of making light sources invisible would have been to just replace any check for the `isLight` property with a check to determine whether a given surface had a non-zero emittance value, and if so then to make the surface invisible. This method was disregarded as it would have stopped requirement **CH1**[1] from being able to be implemented.

Preventing rendering

There are two functions inside the `renderer.js` module that deal directly with the rendering of surfaces: `showEnvironment()`, and `updateColors()`. The `showEnvironment()` function creates new geometry for every `Instance` instance[2] in the environment, and the `updateColors()` will update the colour properties of every surface in the environment when a colour setting, such as gamma or exposure, is changed. Both of these functions were

[1] CH1 - Minor background noise implementation.

[2] "`Instance` instance" refers to a given instance of the `Instance` class, which contains the surfaces and vertices that make up a scene object.

modified to ignore any surface which has `isLight = true`, and so light sources will never be shown on screen.

Preventing shadows and reflections

The projection algorithm, which calculates form factors, was modified to not project any patch with `Patch.parentSurface.isLight = true`. This means that no light source will ever receive light from another patch in the scene, and that surfaces that may be behind the light source will receive lighting as if the light source isn't there.

Radiosity reciprocity

Form factors in the radiosity algorithm are usually calculated from the perspective of the receiving patch (Ashdown, 1994, p. 48). This means that for every patch, the light being received from every other patch is calculated. One of the key principles of the radiosity algorithm is the *reciprocity relation*, which states that if the form factor F_{ij} can be calculated from patch E_i to patch E_j, then we can easily calculate the reciprocal form factor F_{ji} from patch E_j to patch E_i (Ashdown, 1994, p. 51).

$$A_i F_{ij} = A_j F_{ji} \tag{7.1}$$

Where A_i and A_j are the areas of patches i and j respectively.

Because of this reciprocal relationship, the form factors can be calculated in reverse, from the perspective of the emitting patch. The existing system does in fact make this reversed calculation, and it's for exactly this reason that it is possible to make the light sources receive no light whilst still being able to emit light into the scene.

In making this change, the fundamental *reciprocity relation* is being broken, as the form factors are no longer calculatable in reverse. However no negative outcomes were foreseen that would interfere with this project's success, and so development continued with this broken relation in mind.

```
// Emit between steps 0 to 9
activeTime = [0, 9]

// Emit between steps 0 to 4, and between steps 10 to 14
activeTime = [[0, 4], [10, 14]]
```

Algorithm 7.1: Example activeTime values.

7.2.2 Lights on timers

The `activeTime` property holds the information about which frames of the simulation a light source should emit light. It is stored as an array either containing two integer values, or a set of arrays which all contain two integer values. These integer values represent the first step that the light is emitting, followed by the last step that the light is emitting (Alg. 7.1).

The `initExitance()` method of the `HemiCube` class in the existing system was set to make all light sources emit for the first 10 time steps of the simulation[3]. This method was modified to emit based on the times given in the `activeTime` property.

7.3 Flight path

To simulate a bat flying around a static scene, the light sources must be positioned on a pre-defined path and the camera needs to be able to move along the same path. Because of this, details relating to the flight path are stored in the `path.js` module, which is imported by any other module that requires access.

7.3.1 Parametric equations

Parametric equations were chosen for the basis of the flight path due to their ability to generate a different value for each of the three axis based on a single independent variable, which is time. *Sin* and *Cos* were used for the

[3]Original system: /radiosity/slowrad.js L168-L177

parametric equations because of their ability to create a perfect loop.

Trigonometry in JavaScript deals with angles in radians rather than degrees which meant that the values passed into the equations needed to be scaled to between 0 and 2π. All animations are assumed to be 1,000 steps long, which meant that the conversion from time step to radians was a simple calculation. After scaling the values between 0 and 2π, a speed multiplier is also added, which for this project is a multiplier of two.

The specific equations chosen for the simulation are:

$$\begin{aligned} x &= 31 \times Cos(t) \\ y &= 31 \times Sin(t) \\ z &= 2 \times Cos(t) + Sin(3t) + 16 \end{aligned} \tag{7.2}$$

7.3.2 Placing objects along the path

Placing objects along the flight path is a simple process. When creating a scene, an object that should be positioned on the path should be created and translated based on the returned value from `flightPath(t)` where `t` is the time step in which the camera should intersect the object.

7.3.3 Moving the camera along the path

Within the `renderer.js` module, a new ThreeJS perspective camera was made to act as the flight camera. The `animate()` function was modified to pass the current time step, every step, into the `path.js` module, and then take the returned values and pass them into the flight camera's `position` value.

After testing whether this worked (Sect. 8.3.1), it was found that the camera's path wasn't lining up with the light sources that had been placed. Based on information gathered after testing, it was determined that ThreeJS uses a different co-ordinate system to the rest of the system, and as such the co-ordinates had to be rotated by $+90°$ around the x-axis before they were provided to the camera (Fig. 7.1). This equates to the new y value being the original z value, and the new z value being the negative of the original y value (Alg. 7.2).

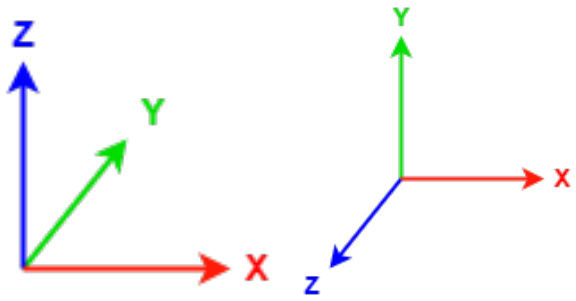


Figure 7.1: Co-ordinate system before and after $+90°$ rotation around x-axis.

```
const [x, y, z] = flightPath(currentStep);
flightCam.position.x = x;
flightCam.position.y = z;
flightCam.position.z = -y;
```

Algorithm 7.2: Fixed camera position assignment.

7.4 JSON trees

To enable trees to be loaded into a scene, a new module `js-output.js` was created which converts the tree structure generated by `generate-tree.js` to JSON format. It is a direct replacement for `scad-output.js` however it outputs the trees in JSON format as opposed to SCAD format.

7.4.1 Fixing branches

The branches weren't loading correctly when opening the trees from JSON format (Appx. E.5.3). This was caused by some incorrect matrix rotations, and they were repaired by removing the need for rotations at all. Rather than storing the branches' start positions and rotations, their start and end positions are stored and they're constructed between the specified points, thus fixing the issue (Fig. 7.2). Without calculating the rotations, the branches get thinner as they get more horizontal, this is because the branches are made of two connected triangles that have no height and so the surfaces that connect them form a narrower tube as the triangles' z values approach each other (Fig. 7.3).

(a) Before **(b)** After

Figure 7.2: Default tree branch structure before and after fixing branch construction.

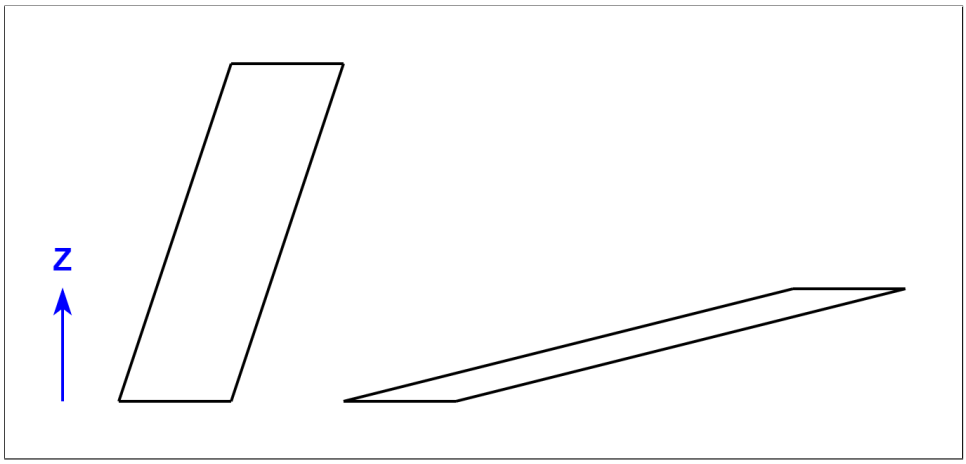


Figure 7.3: Branch structure narrowing as start and end approach same z value.

Cutoff	0.1	0.2	0.3	0.4	0.5
Surfaces	4,695	2,295	1,914	1,455	1,155
Cutoff	0.6	0.7	0.8	0.9	1.0
Surfaces	885	573	432	324	261

Table 7.1: Number of surfaces needed for the branches at different width cutoff values.

(a) Without leaves. **(b)** With leaves.

Figure 7.4: Tree model after reducing the number of branches.

7.4.2 Loading improvements

The tree models can take a large amount of time to load in, and this is due to their complexity. The generated trees have many branches and leaves, and so to improve their loading time, the number of branches loaded in was reduced based on the width of the branch. Different cutoff values were experimented with before deciding which one to use (Tab. 7.1), and in the end a cutoff value of 0.7 was used as it significantly reduced the number of branches whilst having a minimal impact on the tree's appearance (Fig. 7.4)

7.5 Exitance files

When loading complex scenes, the system would take a long time calculating form factors and exitances before the simulation could play. To avoid this, the ability to export exitance files was added so that these files could be loaded into the system to reduce the loading time for each scene. In some situations, the browser would crash due to a lack of memory whilst performing the exitance calculations, and so the decision was made to create a command line system to generate exitance files, as a command line process can be allocated more memory than a browser-based one.

7.5.1 Command line system

A new module `calculate.js` was created which imports all the environments from the `environments-list.js` module. Using these environments, the system can load all the data that defines an environment and run the exitance calculations on them, before exporting them as JSON files for the browser to load.

7.5.2 File compression

Due to the size limit of 100MB for files hosted on GitHub pages, the exitance files needed to be compressed, as one of them reached $\approx$ 374MB in size (Tab. E.2). The possibility of using another hosting platform for the exitance files was considered, however that would have made the system slightly harder to maintain, and another free hosting platform may not have been available.

To reduce the file sizes being saved, it was decided that only one colour channel's value needed to be stored and that value can be loaded back into all three channels when running. The implication of this means that the system can only ever be in grayscale, which isn't ideal, however it doesn't stop the system from meeting any of the requirements set out in chapter 5[4].

[4] ***WH2*** - *Different colours to illustrate the doppler effect* will be impossible to meet, however it was planned as a requirement not to be met, so this is acceptable.

For the actual compression, Run-Length Encoding (RLE) was the first method used for compression as it is lossless and so the simulation could remain at peak accuracy, however RLE barely reduced the file size due to many values being close to one another but not equal.

Two other methods of compression were used in tandem with RLE. The first of these was converting all numbers to their exponential form, and removing all numbers after the decimal point in the mantissa (`1.04e-5`→`1e-5`) which lowered the number of characters being used to store the number. The second technique used was rounding all numbers between `1e-9` and 0 to whichever of those two values they fall closest to. This means that all numbers are stored using a maximum of four characters.

The implications of these lossy compression methods were monitored, however looking at the resulting differences between the uncompressed data and the fully compressed data (Appx. I.3), there is minimal difference and so the use of lossy compression feels justified, however in some circumstances the differences may be more noticeable and so it should be monitored in the future.

7.6 Summary

In this chapter, we have looked at how light sources in the scene were made invisible and given timers to determine when they activate. Then the implementation of a flight path for the camera, and the difference in co-ordinate systems that had to be resolved to get the motion working. Finally we covered the inclusion of JSON trees and the improvements made to speed up their loading times.

Chapter 8

Testing

All projects are prone to issues and bugs, both expected and unexpected. Stopping them from making it to release is an important stage of the development process which helps ensure a quality output. As outlined in chapter 4, this project was subject to testing throughout the entire course of development, and this allowed the project to grow at a constant rate without enabling issues to get out of hand.

Because of the frequent testing that was being done throughout the development process, the artefact was tested in-house, as this meant that the implementation wasn't slowed down whilst waiting for third-parties to test the system.

8.1 System setup

All in-house tests were done from the following system setup.

8.1.1 Testing machine

CPU	Intel i7-9700K @ 3.60GHz
RAM	2x8GB DDR4 @ 2133MHz
GPU	NVIDIA GeForce GTX 1050 Ti 4GB @ 1.29GHz
OS	Windows 10 Home 21H2 (19044.1586)

8.1.2 Testing browser

Version	Google Chrome 100.0.4896.127 (64-bit)
jsHeapSizeLimit	4,294,705,152 bytes (4.295GB)

8.2 Tests done

Some unit tests were done on every feature of the artefact after they had been implemented. Table 8.1 shows the results of these unit tests. More detailed breakdowns of each test can be found in appendix E, whilst the analysis of the tests that didn't pass will be discussed next.

Test	Pass / Fail	Appendix
Invisible light sources	Pass	E.1
Lights activating on timers	Pass	E.2
Placing objects along path	Pass	E.3.1
Moving camera along a path #1	Fail	E.3.2
Moving camera along a path #2	Pass	E.3.2
Loading STL tree	Fail	E.4
Loading JSON tree	Pass	E.5
Faster loading of JSON tree	Pass	E.6
Browser exitance exporting and importing	Inconclusive	E.7
Command line exitance generation	Pass	E.8
Exitance file compression	Pass	E.9
Deployment for testing	Pass	E.10

Table 8.1: Unit testing table.

8.3 Issues that arose during testing

8.3.1 Moving camera along a path #1

When attempting to make the camera move along the path (Appx. E.3.2), the scene loaded up and the animation was played, however the camera's motion was not lining up with the objects that had been placed along its path. Because the positioning of these objects had been verified in the

previous test (Appx. E.3.1), it was clearly an error in how the co-ordinates were being provided to the camera.

After giving the camera some preset hard coded co-ordinates, it became clear that ThreeJS was using a different co-ordinate system to the rest of the system; one where the horizontal plane was the x, z plane as opposed to the x, y plane. Based on this information, and the assumption that ThreeJS most likely uses a right-handed co-ordinate system, it was determined that the y-axis was likely pointing vertically upwards. Knowing this, the co-ordinates provided to the camera were changed accordingly (Sect. 7.3.3), and in the second test everything worked as expected (Appx. E.3.2).

8.3.2 Loading STL tree

The ability to load STL trees into the scene was tested in appendix E.4 and in doing the test, the browser crashed due to a lack of memory. Investigating why this happened led to the realisation that STL files are used commonly by 3D printers and other technology that requires shapes to be possible in the real world. This means that in STL, a shape must have a minimum of four faces, and cannot be hollow[1] as this is the minimum required for a shape to be created in the real world.

Given that computer graphics have no such requirement for shapes to be solid, it was clear that STL models were not the way forward, and that to allow trees to be loaded, a new format had to be used. After some consideration, JSON was chosen as the replacement format (Sect. 6.2.1) and STL models were no longer considered in development.

8.3.3 Browser exitance exporting and importing

When attempting to export the exitance files from the browser (Appx. E.7), the testing was unable to be completed due to the complex scene crashing before the exitance data had been calculated. This crash made it clear that the browser did not have the memory capacity required to reliably load a scene and calculate exitance data. As stated in section 8.1.2, the testing

[1] "hollow shape" refers to something like a triangular prism that's had its ends removed.

setup has ≈ 4.3GB of memory that JavaScript can use. Understanding that this is likely more than some everyday systems have, a way of taking the calculations away from the browser had to be devised in order to allow more users to be able to access the system. It was decided that these calculations could be made from a new command line interface, and the resulting files could be imported into the system (Sect. 6.4).

Chapter 9

Evaluation

This chapter will cover the evaluation against the requirements set out in chapter 5, as well as the feedback gathered from prospective users of the system. Additionally, the project's success against its aim laid out at the start of the report (Sect. 1.1), and the overall success and suitability of the methods used throughout this project will also be discussed later on.

9.1 Evaluation against requirements

The evaluation against requirements was conducted using Alpha-Beta Testing, where the Alpha testing consists of the tests carried out in chapter 8, as well as the general observation of the system in action; and the Beta testing was carried out by users, which will be covered later in section 9.2. Alpha-Beta testing was chosen as this project is a proof of concept for a new system, and so the Alpha testing allowed the features to be tested internally as they were developed, whilst the Beta testing provided the opportunity for users to provide feedback regarding the performance of the system, and potentially spot issues that were missed in the Alpha tests.

Table 9.1 lays out all the requirements specified in chapter 5 and whether or not they were met, excluding the **Won't have** requirements as these were designed not to be met. The explanation of those that weren't met is next, whilst a breakdown of all requirements can be found in appendix F.

ID	Requirement	Met or not met
MH1	A moving camera on a pre-planned path.	Met
MH2	Application served on GitHub Pages.	Met
MH3	Pre-loaded exitance data.	Met
MH4	Light sources that activate sequentially.	Met
MH5	New method of storing and loading trees.	Met
SH1	Command line calculation ability.	Met
SH2	Light sources that aren't visible.	Met
SH3	Compressed exitance data files.	Met
CH1	Minor background noise.	Not met
CH2	Exportable images.	Met
CH3	Exportable animation.	Not met

Table 9.1: All project requirements and whether or not they were met.

9.1.1 Requirements that weren't met

CH1 - Minor background noise implementation

Whilst the background noise would not have been hard to implement, and indeed, anyone with access to the final project's source code could create a scene with background noise implemented by just creating more surfaces that emit light. The decision not to implement background noise was made because it would have generated more surfaces, and with the system already using enough memory that it crashes on some computer setups, it felt like a very unimportant thing to spend time implementing without pushing the system to crashing on even more setups.

CH3 - Exportable animation

The idea of an exportable animation is something that could still prove to be a useful feature of the system, however the research needed to implement it would have exceeded the time constraints of the project, and so it was left out in order to make time for more important features. The decision to scrap this particular requirement, was made because screen-recording software already exists, and this makes the feature less useful when weighed

up against other features, especially when accounting for the time it would take to implement.

9.2 User feedback

Whilst the in-house testing in chapter 8 revealed which features of the system worked as expected and which ones were faulty thus enabling faults to be promptly rectified. Gathering feedback from users gives insight into whether the system works as other people might expect. This is especially important for this project as no previous input has been received from potential users, and so this is the only time that the output of the project is able to be evaluated against what a user might consider to be successful.

To gather feedback, potential users were emailed both a link to the deployed system, and to a feedback questionnaire (Appx. G). The questionnaire goes through various questions relating to the technical performance, potential applications, and the general use of the system.

Unfortunately, due to illness, the collection of user feedback was unexpectedly delayed. Given that this project's primary aim was to create a system that could be used to fill a gap in educational materials regarding echolocation, the planned target audience for this system is teaching staff, specifically those with experience in biological sciences. Because of the unexpected delay, most schools were out of term time and so it was difficult to reach an audience of teaching staff.

Because feedback could not be gathered from specifically the target audience, the decision was made to open up the feedback to any users so that the performance and ease of use of the system could still be evaluated.

9.2.1 Results analysis

The results gathered from the feedback questionnaire highlighted some issues regarding the performance of the system and helped to provide insight into what the users would want to be incorporated into the system. All the responses can be found in appendix H.

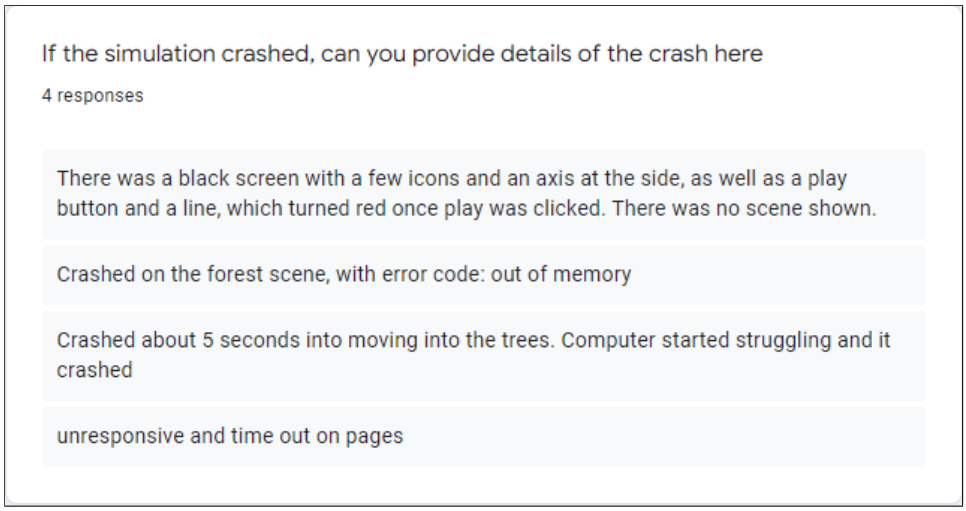


Figure 9.1: User feedback with crash information.
Source: Appendix H

Technical performance

Over 50% of the users reported that the system crashed whilst it was in use, with exactly 50% specifying that it crashed before they could properly test the system. Based on these crash statistics, information provided by the users (Fig. 9.1), and the discovery of the browser crashing during testing (Sect. 8.3.3), it is clear that the system requires too much memory in its current state to be deployed effectively for people to use as even reducing the browser's load by running the calculations externally doesn't seem to save enough memory for most user setups to handle.

Applications of the system

Everybody that was able to test the system reported that they felt the system had artistic value, and two thirds of these users said that they feel it may be useful for demonstrating echolocation in an educational capacity. However because of the large amount of crashes experienced, combined with the fact that the feedback could not be elicited specifically from the project's target audience; whilst this feedback is promising, and shows that the system may have some general appeal to it, I would consider these results to be inconclusive as the sample size isn't large enough, and is lacking in users

from the target backgrounds.

Future additions

In the feedback questionnaire, the users were asked if there are any features that could be added to make the simulation more aesthetically appealing, or that could make the simulation more useful for teaching purposes.

Regarding the aesthetics, it was identified that the environments present didn't have enough variety within them and that they could feature different objects. This is definitely something that would be considered in future developments, however the ability to have more interesting and diverse scenes is currently rather limited by the memory required for the system to run. It was also noted that tool tips present when hovering over buttons should be capitalised.

For teaching purposes, a suggestion was to have the camera be user-controlled and to add a means of 'activating' echolocation pulses and using them to navigate an environment. This would definitely be a good addition to add, however the radiosity algorithm would not be the way to go about doing this as it would require the form factors and exitances to be calculated every time a pulse is used, which would massively increase the performance costs. Another suggestion was to add some kind of "birds eye view with arrows / lines to show the sound bouncing", however it's unclear exactly what is meant by this.

9.3 Success against the project's aim

In chapter 1, the aim of this project was discussed which explained how the desired output of this project is to have a web application with accompanying YouTube videos to act as an educational tool to teach people about echolocation.

This project has successfully created a web application and deployed it via GitHub Pages, and YouTube videos have been created and uploaded for people to view[1]. However whilst these two outputs have been successfully created and made available to the public, the overall success of this project as an educational tool is still undetermined, this is because, as stated in section 9.2, the project was unable to be tested by people with experience in the desired fields.

Just because the project hasn't proven to be successful as an educational tool, does not mean that it has failed. The Alpha-Beta testing that was carried out highlighted some major performance issues that need addressing before the software artefact can be considered complete, and once these issues have been fixed further testing will need to be done to ensure it meets the required specifications for an educational tool. However the project still shows promise that it may prove to be useful, and as a proof of concept, I believe it has demonstrated enough success to benefit future work that may be done in and around this field of interest.

9.4 Critique

9.4.1 Approach

Overall, I believe the methodological approach outlined in chapter 4 was the right way to go about the implementation of this project, as having the ability to discover new requirements throughout the development meant that there were opportunities for me to backtrack and correct myself, which allowed me to create a higher quality artefact. However if this project was to be continued for the purposes of creating a completed deployable product, a different methodology should be used, as forethought and planning will be paramount to ensuring the maintainability of the system in the long run.

[1] https://www.youtube.com/playlist?list=PLVg2-G6HwbO6lhhChUvL4ntMTlgaTRLAv

9.4.2 Research

The initial research I did into the topics surrounding the project was sufficient enough to gain enough understanding of the field. However I neglected to do any primary research into what features potential users might expect/want to be implemented in a system like this. This lack of primary research for requirements elicitation made it harder to judge the success of this project as an educational tool.

Additionally, a mistake was made by assuming that literary research had to be done in full before beginning development. However after seeing how the system worked and being able to practically apply knowledge during its development, my understanding of the topics being used was increased dramatically. In future exploratory projects, this fact should be exploited such that development should have a designated break within which more research can be done regarding the mechanics surrounding the project so they can be better integrated in the final result.

9.4.3 Model suitability

Light vs sound

The largest problem with the radiosity model for this system was the fact that radiosity is designed to simulate a basic lighting system, and whilst light and sound are both waves, and they share many properties, they do have their differences.

For example, if a room is made out of brick on all sides, with no doors or windows, if there was a light source inside the room such as a torch, then you might expect all the light to be blocked by the brick walls, regardless of the strength of the light. However if there was a loud sound system in the room, then one could expect the sound waves to travel through the walls.

Conversely, if the walls were made out of thick glass, then light would almost definitely be seen from the outside, whilst the sound waves may be more obscured.

These differences in what materials light and sound can pass through are largely due to their wavelengths. Visible light ranges in wavelength from 400nm to 700nm, and with the speed of light being 299,792,458ms^{-1}, this gives light waves a frequency ranging from $\approx$ 749THz to $\approx$ 428THz (Eq. 3.3). This puts the lowest frequency light wave (violet light) at about 21,313,747,000 times greater than the 20kHz upper bound on audible sound.

As discussed in section 3.1.1, the decay of a wave as it travels through a medium is likely greater for waves of higher frequency. The immense difference in frequencies between visible light and the upper bound of a bat's echolocation call at approximately 212kHz means that the radiosity algorithm, and any other algorithms design to simulate light may be inherently unable to model sound without serious modification.

Resolution

The radiosity algorithm will calculate reflections of light on all surfaces, with near perfect clarity. This is fine when simulating light, however given the wavelength differences between sound and light outlined above, it's highly unlikely that bats have perfect clarity regarding the environment around them because, as discussed in section 3.2.1, most insectivorous bats use a frequency up to 60kHz, which equates to a wavelength of $\approx$ 5.667mm, this means they can't detect things less than 5.667mm in size. With this minimum detection size, would a bat be able to detect a leaf from a side on angle? Having a method of excluding objects from being detected based on different properties of the light in the scene would allow this frequency to resolution correlation to be implemented correctly, however I'm not sure if a feature of this sort would be possible using the radiosity algorithm.

Chapter 10

Conclusions

This project as a whole has been mostly successful. At the beginning of the implementation chapter it was mentioned that this project served to create a proof of concept to show that there was some life to this idea (Sect. 7).

Almost all requirements set out in chapter 5 have been met, with only two **Could have** requirements remaining unfulfilled[1]. However despite these requirements not being met, the system still operates at an acceptable level to be able to say that it is working as intended. The testing carried out in chapter 8 revealed no bugs or features not working as expected, however it was discovered that the system requires a significant amount of memory to operate and this did impact the users that tested the system, with the majority of them suffering crashes during their usage (Sect. 9.2.1), and it's for primarily this reason that the system is not deemed to be worthy of distribution.

[1] **CH1** - Minor background noise, and **CH3** - Exportable animation

10.1 Future considerations

10.1.1 Different shape light sources

Currently the light sources used along the bat's flight path are cubes that have undergone no orientation changes, which means that light is emitted in all directions equally. Changing these shapes to tetrahedrons would lower the vertices required per light from eight to four, which could have a positive impact on the time taken to calculate all the exitance values in a scene.

Taking this a step further, as bats direct their echolocation calls forwards (Jakobsen et al., 2013), a tetrahedron with different emittance values on each face may provide more accurate visualisations as it could have the face with a stronger emittance directed forwards, or perhaps even a single surface angled forwards might suffice. The ability to orient these direction-dependent implementations already has some ground work laid out in `path.js` where the derivatives of the parametric equations used for movement are already stored[2]. These derivatives provide the direction of motion of the bat at any given time in the simulation.

10.1.2 Improved tree loading

Currently every section of a trees trunk and branches are independent of the others, only bound together in the fact that they make a single `Instance`[3]. Could a tree be made using less vertices if each branch was created as a single shape rather than multiple smaller sections (Fig. 10.1)? This would lower the amount of vertices in the scene, and so would likely make the exitance calculations faster, and potentially even save memory to allow the system to be used in more browsers without risk of crashing.

[2]Source Code: /modeling/path.js L20-L22

[3]Source Code: /modeling/json-tree-loader.js L47

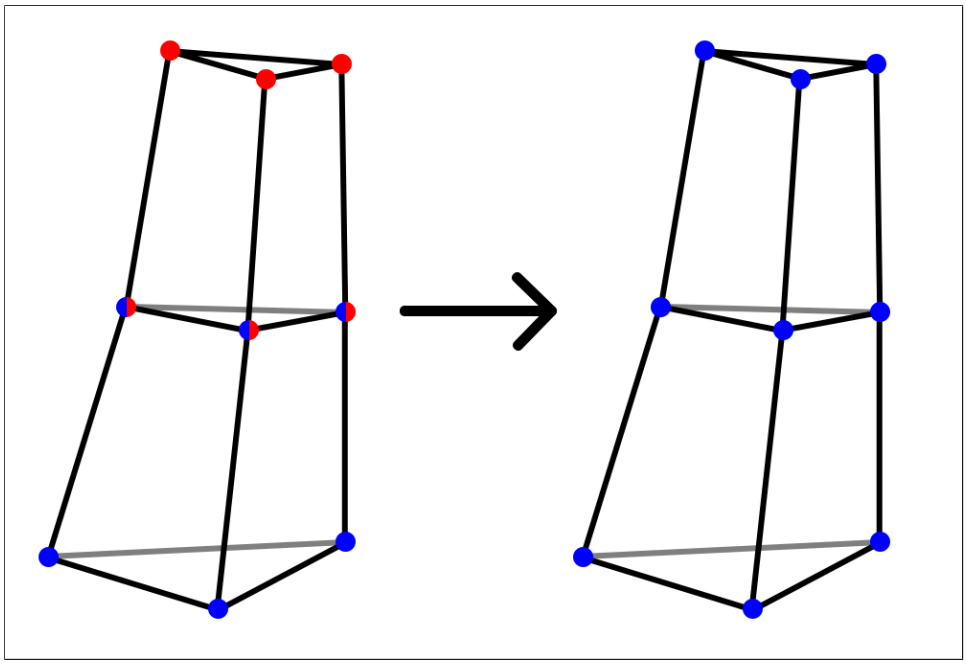


Figure 10.1: Conversion of two trunk sections into one whole surface. Different coloured vertices equate to different surfaces that they form.

10.1.3 Remove the system from the browser

Applications running in browsers can have strict memory limits imposed on them. Given that the system now has command line exitance calculation, could steps be taken to make the system entirely independent of the browser? And would removing it from the browser improve the performance of the system?

10.1.4 Adjusting the radiosity model

As mentioned in section 9.4.3, the radiosity model isn't necessarily an accurate model for simulating sound, as sound waves and light waves behave differently. Would it be possible to allow some objects to be slightly translucent? This could be done using a property similar to spectral reflectance, and it could be used to allow objects like windows and tinted plastic sheets to be created in a scene. Not only would this allow for more intricate light effects, but it could also be used to allow light to pass through objects in the same way that sound could.

If one could figure out how much of a wave should come out the other side of an object, then it would simply be a matter of setting the face that the wave emerges from with some emittance value at that moment. Because light and sound both diffract, the way that that surface emits light wouldn't need to be changed to accommodate anything beyond what it would do with a normal Lambertian reflection.

10.1.5 Adding sound

Considering the entire idea of the system is to simulate echolocation, which is an audible method of navigation, incorporating sound into the system would likely help users become more immersed, and might help make the system feel more genuine. If the sound of the echoes were also simulated, with 3D positional audio, then maybe this system could be used to actually help people actually learn how to echolocate.

10.2 Personal reflection

When beginning this project, the anticipated workload was fairly overwhelming as I had never attempted a project of this scale before, I'd never heard of the radiosity algorithm before, and I'd never used ThreeJS in any capacity. However, being able to examine how the Slow-light Radiosity system worked and being able to make small changes to see how all the different pieces interacted with one another, and being able to ask Jacek questions when I got confused, allowed me to figure out what I needed to do to make my system operational.

Throughout the project, a number of issues arose, many of which were solved by looking up help forums and documentation, however I think my previous experience in programming, especially JavaScript, was the biggest aid to the problem solving required. Taking the *3D Computer Graphics and Animation* module in my second year definitely helped me to understand the basics required to embark on this project, as local and global illumination were both covered as part of its content.

If I were to do this project again, I would spend less time attempting to do all the research required prior to development, as with a project of this nature, development and research should be in conjunction with one another so that the practical application of knowledge can help deepen the understanding of the research being done. I think it would also be beneficial to write a brief diary of the things done so that all the notes taken can be easily accessed, and they can be used to develop a more detailed plan, as the plan set out in chapter 4, whilst it was functional and did help me to manage my time, it could have been more in-depth and potentially covered the techniques I was wanting to use.

Overall, I'm happy with the way that this project has turned out, however I'm aware that there are parts of the project that could be improved upon. I feel I would be much more confident carrying out a project of this scale in the future as I can use the strengths and weaknesses found in this project to fuel my future decisions.

References

3d common properties. (n.d.). Retrieved April 17, 2022, from https://www.neurobs.com/pres_docs/html/03_presentation/04_stimuli/03_visual_stimuli/02_picture_stimuli/04_3d_graphics/02_3d_common_properties/index.html

Adobe. (2022). *Photoshop* (Version 23.2.2) [Computer software]. https://www.adobe.com/uk/products/photoshop.html

Allwood, G. (2021, August 20). *Know your Navy - all the ships and subs in the Royal Navy*. Forces Network. Retrieved April 23, 2022, from https://www.forces.net/services/navy/know-your-navy-all-ships-and-subs-rn

Ashdown, I. (1994). *Radiosity: A Programmer's Perspective*. John Wiley & Sons.

BBeck1. (2016, September 26). *Ambient and diffuse light are not even remotely the same thing*. Retrieved April 17, 2022, from https://community.khronos.org/t/difference-between-ambient-diffuse-material/75109/4

Bellis, M. (2020, March 27). *The History of Sonar*. ThoughtCo. https://www.thoughtco.com/the-history-of-sonar-1992436

Berg, R. E. (2017, October 6). *Ultrasonics*. Encyclopedia Britannica. Retrieved April 23, 2022, from https://www.britannica.com/science/ultrasonics

Cabello, R. (2020, March 25). *Threejs* (Version r115). https://github.com/mrdoob/three.js/tree/r115

Caulfield, B. (2022, March 23). *What is path tracing?* Retrieved April 17, 2022, from https://blogs.nvidia.com/blog/2022/03/23/what-is-path-tracing/

Chege, J. (2020, November 10). *Javascript iterations - which one is faster?* Retrieved March 13, 2022, from https://www.section.io/engineering-education/javascript-iterations-which-one-is-faster/

Dupont, A., Brix, T., & Zamir, B. (2022, March 20). https://github.com/AtomLinter/linter-eslint-node/tree/v1.0.4

Elias, H. (2000). *Radiosity*. Retrieved April 22, 2022, from http://freespace.virgin.net/hugo.elias/radiosity/radiosity.htm Archived at https://web.archive.org/web/20010607021839/http://freespace.virgin.net:80/hugo.elias/radiosity/radiosity.htm

ESLint. (2021, March 26). https://github.com/eslint/eslint/tree/v7.23.0

Fahy, F. J., & Walker, J. G. (Eds.). (1998). *Fundamentals of Noise and Vibration*. Spon Press.

Font Awesome. (2020, March 23) (5.13.0). Retrieved April 4, 2022, from https://github.com/FortAwesome/Font-Awesome/releases/tag/5.13.0

GeoGebra GmbH. (2020). *GeoGebra 3D Calculator* (Version 5.0.580.0). Retrieved March 4, 2022, from https://www.geogebra.org/3d

Geoscience Australia. (n.d.). *Sidescan sonar*. GOV.AU. https://www.ga.gov.au/scientific-topics/marine/survey-techniques/sonar/sidescan-sonar

GitHub. (2020, May 18). https://github.com/atom/atom/tree/v1.47.0

Goral, C. M., Torrance, K. E., Greenberg, D. P., & Battaile, B. (1984). Modeling the Interaction of Light Between Diffuse Surfaces, 213–222. https://doi.org/10.1145/800031.808601

Griffin, D. R. (1944). Echolocation by blind men, bats and radar. *Science*, *100*(2609). https://doi.org/10.1126/science.100.2609.589

Honsberg, C. B., & Bowden, S. G. (2019). *Absorption Coefficient*. Photovoltaics Education. https://www.pveducation.org/pvcdrom/pn-junctions/absorption-coefficient

itskawal2000, & surbhikumaridav. (2020, August 22). *Introduction to Exploratory Style of Software Development*. https://www.geeksforgeeks.org/introduction-to-exploratory-style-of-software-development/

Jakobsen, L., Brinkløv, S., & Surlykke, A. (2013). Intensity and directionality of bat echolocation signals. *Frontiers in Physiology*, *4*. https://doi.org/10.3389/fphys.2013.00089

JGuerra. (2019, September 23). *The History of Ultrasounds: From Bats to Babies*. Conquest Imaging. https://conquestimaging.com/ultrasound-blog/history-ultrasounds-bats-babies/

Jones, G. (2005). Echolocation. *Current Biology*, *15*(13). https://doi.org/10.1016/j.cub.2005.06.051

Jones, G., & Holderied, M. W. (2007). Bat echolocation calls: Adaptation and convergent evolution. *Proceedings of the Royal Society B*, *274*(1612). https://doi.org/10.1098/rspb.2006.0200

jun.yoshino. (2021, September 28). *Georadiosity*. Retrieved April 16, 2022, from https://enlighten.atlassian.net/wiki/spaces/SDK400/pages/2352592123/GeoRadiosity

Kintel, M. (2021, February 7). *OpenSCAD* (Version 2021.01). https://openscad.org/

Kopecký, J., Boakes, R., & TLDRQwerty. (2020, December 10). https://github.com/portsoc/eslint-config-portsoc/tree/09f213

Kopecký, J., & Mattone, T. (2020). *Slow-light Radiosity* (Version ea2f61f) [GitHub Repository]. University of Portsmouth: School of Computing. Retrieved February 21, 2022, from https://github.com/portsoc/Slow-light-Radiosity/tree/ea2f61f

Kumar, S., & Lee, H. P. (2019). The Present and Future Role of Acoustic Metamaterials for Architectural and Urban Noise Mitigations. *Acoustics*, *1*(3), 590–607. https://doi.org/10.3390/acoustics1030035

Kurtus, R. (n.d.). *How Obstacles Affect Sound Waves*. School for Champions. https://www.school-for-champions.com/science/sound_obstacles.htm

Lague, S. (2018, November 23). *[Unity] Procedural Object Placement (E01: poisson disc sampling)* [Video]. YouTube. https://www.youtube.com/watch?v=7WcmyxyFO7o

Langley, L. (2021, February 3). *Echolocation is nature's built-in sonar. Here's how it works.* National Geographic. https://www.nationalgeographic.com/animals/article/echolocation-is-nature-built-in-sonar-here-is-how-it-works

Mazda, F. (Ed.). (1993). *Telecommunications Engineer's Reference Book.* Butterworth Heinemann.

Mirrors. (n.d.). Retrieved April 16, 2022, from https://courses.lumenlearning.com/boundless-physics/chapter/mirrors/

Mischi, M., Rognin, N. G., & Averkiou, M. A. (2014). *Ultrasound imaging modalities* (Vol. 2). Elsevier.

Modeling Global Illumination with Radiosity. (2018, March 18). Retrieved April 16, 2022, from https://help.autodesk.com/view/3DSMAX/2022/ENU/?guid=GUID-C5A3C77B-794B-4444-9783-7F2EA11C16BD#GUID-C5A3C77B-794B-4444-9783-7F2EA11C16BD__GUID-C5A3C77B-794B-4444-9783-7F2EA11C16BD

NVIDIA. (n.d.). *Nvidia rtx ray tracing.* Retrieved April 17, 2022, from https://developer.nvidia.com/rtx/ray-tracing

OpenSCAD. (n.d.). Retrieved April 3, 2022, from https://openscad.org/

Principles behind the Agile Manifesto. (2001). https://agilemanifesto.org/principles.html

Purves, D., Augustine, G. J., Fitzpatrick, D., Hall, W. C., LaMantia, A.-S., Mooney, R. D., Platt, M. L., & White, L. E. (Eds.). (2018). *Neuroscience.* Oxford University Press.

Reflection, Refraction, and Diffraction. (n.d.). The Physics Classroom. https://www.physicsclassroom.com/class/sound/Lesson-3/Reflection,-Refraction,-and-Diffraction

Royce, W. W. (1970). Managing the Development of Large Software Systems, 328–338.

Russ, J. (2013). *British Bat Calls: A Guide to Species Identification.* Pelagic Publishing.

Seddeq, H. S. (2009). Factors Influencing Acoustic Performance of Sound Absorptive Materials. *Australian Journal of Basic and Applied Sciences*, *3*(4), 4610–4617.

Service, M. (2012). *Medical Entymology for Students* (5th ed.). Cambridge University Press.

Tabellion, E. (2010). Global Illumination Across Industries: Ray Tracing vs. Point-based GI for Animated Films. *ACM SIGGRAPH 2010 Courses*. Retrieved April 16, 2022, from https://cgg.mff.cuni.cz/~jaroslav/gicourse2010/

The Editors of Encyclopedia Britannica. (2019, May 30). *Sonar*. Encyclopedia Britannica. Retrieved April 23, 2022, from https://www.britannica.com/technology/sonar

Wilson, D. E. (1997). *Bats in question: the Smithsonian answer book*.

Young, H. D., Freedman, R. A., & Ford, A. L. (2012). *Sears and Zemansky's University Physics with Modern Physics* (International 13th ed.). Jim Smith.

Appendix A

Ethics Certificate

Certificate of Ethics Review

Project title: Simulating Echolocation Through The Use Of Slow-Light Radiosity

Name:	Daniel Ellis	**User ID**:	UP940148	**Application date**:	31/03/2022 19:57:47	**ER Number**:	TETHIC-2022-103005

You must download your referral certificate, print a copy and keep it as a record of this review.

The FEC representative(s) for the **School of Computing** is/are **Haythem Nakkas, David Williams**

It is your responsibility to follow the University Code of Practice on Ethical Standards and any Department/School or professional guidelines in the conduct of your study including relevant guidelines regarding health and safety of researchers including the following:

- University Policy
- Safety on Geological Fieldwork

It is also your responsibility to follow University guidance on Data Protection Policy:

- General guidance for all data protection issues
- University Data Protection Policy

Which school/department do you belong to?: **School of Computing**
What is your primary role at the University?: **Undergraduate Student**
What is the name of the member of staff who is responsible for supervising your project?: **Dr. Jacek Kopecký**
Is the study likely to involve human subjects (observation) or participants?: Yes
Will you gather data about people (e.g. socio-economic, clinical, psychological, biological)?: No
Will you gather data from people about some artefact or research question (e.g. opinions, feedback)?: Yes
Will the study involve National Health Service patients or staff?: No
Do human participants/subjects take part in studies without their knowledge/consent at the time, or will deception of any sort be involved? (e.g. covert observation of people, especially if in a non-public place): No
Will you collect or analyse personally identifiable information about anyone or monitor their communications or on-line activities without their explicit consent?: No
Does the study involve participants who are unable to give informed consent or are in a dependent position (e.g. children, people with learning disabilities, unconscious patients, Portsmouth University students)?: No
Are drugs, placebos or other substances (e.g. food substances, vitamins) to be administered to the study participants?: No
Will blood or tissue samples be obtained from participants?: No
Is pain or more than mild discomfort likely to result from the study?: No
Could the study induce psychological stress or anxiety in participants or third parties?: No
Will the study involve prolonged or repetitive testing?: No
Will financial inducements (other than reasonable expenses and compensation for time) be offered to participants?: No
Are there risks of significant damage to physical and/or ecological environmental features?: No
Are there risks of significant damage to features of historical or cultural heritage (e.g. impacts of study techniques, taking of samples)?: No
Does the project involve animals in any way?: No
Could the research outputs potentially be harmful to third parties?: No
Could your research/artefact be adapted and be misused?: No
Will your project or project deliverables be relevant to defence, the military, police or other security organisations and/or in addition, could it be used by others to threaten UK security?: No

I confirm that I have considered the implications for data collection and use, taking into consideration legal requirements (UK GDPR, Data Protection Act 2018 etc)

I confirm that I have considered the impact of this work and and taken any reasonable action to mitigate potential misuse of the project outputs
I confirm that I will act ethically and honestly throughout this project

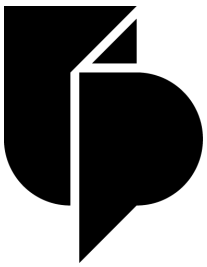
Supervisor Review

As supervisor, I will ensure that this work will be conducted in an ethical manner in line with the University Ethics Policy.

Supervisor's signature**:** Date**:**

Appendix B

Project Initiation Document



UNIVERSITY OF PORTSMOUTH

School of Computing
Project Initiation Document

Daniel Ellis

Simulating Echolocation Through The Use Of Slow-Light Radiosity

Engineering Project

1. Basic details

Student name:	Daniel Ellis
Draft project title:	Simulating Echolocation Through The Use Of Slow-Light Radiosity
Course:	Software Engineering
Project supervisor:	Dr Jacek Kopecký

2. Degree suitability

Please describe how your project satisfies the criteria for your current course. For example, if you are a Software Engineering student, please explain why your project is suitable for a Software Engineering degree.

In each section please write your text below ours in regular (non-italic) font.

As I'm studying Software Engineering, my project is suitable for my degree because at the end of it I will have created a web application.

3. Outline of the project environment and problem to be solved

*For engineering **projects without a client**:*
What is the problem that you will investigate? *Why is it worth working on?*

- Understanding/visualising how echolocation works.
- Educational resource? Final animations could be used to show younger students how echolocation works in biology.

4. Project aim and objectives

What is the overall aim of the project?
What are the objectives that will lead to you meeting that aim?

The aim is to improve the existing system so that it becomes possible to generate simulations or visualizations of echolocation for educational purposes. For example, at the end, users could use the system to enter a potential flight path of a bat, and the system will output an animation showing what the bat 'sees' as it flies along the path.

The existing system can already do these things:

- load a pre-defined 3d environment and shows it
- allows movement through it
- computes the lighting with slow-light radiosity

Objectives:

- Users should be able to input a desired flight path
- A animation should be rendered and output back to the user
- create an environment for the bat flight simulation
- prepare a sample final animation

5. Project deliverables

For an engineering project, what information system artefacts will be developed? What documents will be produced? This always includes your project report, but could also include supporting documentation for your client such as requirement and design specifications, test strategies, user guides, that are useful outside of the project report.

For a study project, are there anticipated outcomes besides the report, for example datasets or recommendations to external bodies?

- A final animation of the flight, uploaded to YouTube
- Live system deployed on GitHub Pages
- GitHub repository with documentation for future contributors
- The project report

6. Project constraints

What constraints are there on your solution to the problem? For example, you could not test a medical system on real patients.

At this point in time, I don't believe there are any constraints that would affect my problem or solution.

7. Project approach

How will you go about doing your project? What background research do you need to do? For an engineering project, how will you establish your requirements? For a study project -

can you refine your larger research area into research questions that you can meaningfully answer? What skills do you require and how are you going to acquire those that you do not already have? What methodologies are you going to use?

- For my background research, I will need to investigate how echolocation works.
- I'll need to be able to program in JavaScript, have a moderate understanding of 3D graphics, and some understanding of how radiosity works. These are skills which I already have.

8. Literature review plan

What are the starting points for your research? (e.g. specific books or papers in journals, existing reports or documents, online resources, existing systems)

Research into Echolocation

- https://www.researchgate.net/figure/Echolocation-simulation-in-real-world-scanned-environments-During-training-the-agent_fig1_346371740

Research into MIT GameLab's *'A Slower Speed of Light'*

9. Facilities and resources

What computing/IT facilities will you use/require?
What other facilities/resources will you use/require?
Are there constraints on their availability? If funds are required to acquire them, have these been allocated? Will they be available in time?

For example, you might need a specialist lab or equipment at the university, which might be in use in teaching and by other project students. Your own computer and free software, or software you already have, do not normally need to be mentioned.

I don't believe I need any facilities or resources beyond my own personal computer and the software already installed on it

10. Log of risks

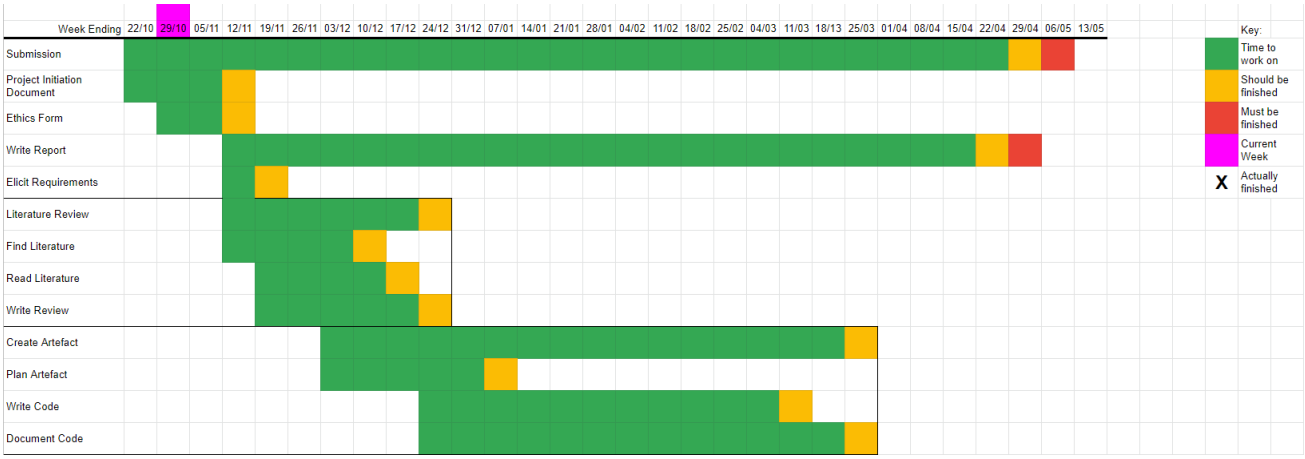
What risks will you encounter when doing your project? What backup plans do you have if identified things go wrong?
What is your plan for reviewing risks? Remember that risk probabilities, and hence priorities, will change over the course of the project, so this section should be maintained.

Use a table like below.

Description	Impact	Likelihood	Mitigation	First indicator
COVID-19 outbreak means I cannot get into a lab for usability testing	*Severe*	*Likely*	*Get in while I can, prioritise lab tasks in time.* *Make an alternate test plan that does not need the lab.*	*University informs that lab closure is likely*
My computer might stop working, which could stop me working on my project for a period of time	Severe	Unlikely	Utilise cloud storage systems such as GitHub and Google Drive	Screen goes black?
I could become sick, and be unable to do work on my project	Severe	Unlikely	Do as much work as I can early on, and aim to complete my project early so that I have extra time if I need it	Feeling ill
My supervisor could become sick and we might not be able to have meetings	Moderate	Unlikely (?)	Plan regular meetings with clear aims, so that most questions I may have can be answered early on	Email from supervisor / Auto-response email

11. Project plan

What do you need to do to create the artefact / do the primary research and write the report? Walk through your proposed approach and break it down into tasks.
When are you planning to perform these tasks? When do you need access to other people or resources? Usually a Gantt chart is a good way of presenting the plan.
Note that plans can change over the course of the project, so this plan should be maintained.



12. Legal, ethical, professional, social issues (mandatory)

What are the legal/ethical/professional/social issues that may impose constraints on the project? How will you ensure that they will be addressed, or what steps will you take to avoid/mitigate their effects?
Because my project doesn't require any hardware such as cameras/sensors, and it doesn't require the collection or storage of any personal information, or in fact any interaction with anybody outside of the project 'team' (Myself and my supervisor), I believe that there are no legal, ethical, professional, or social issues that need addressing.

Whatever project work you are doing, you must consider its security implications, for the data you generate or use, or for the software artefact itself. Please describe how you are taking these into account. There is also a question about security on the ethics review form (see below)

*All students must complete the ethics review form at https://sums.soc.port.ac.uk/ethics at this time. Has your supervisor (and the FEC representative, if required) seen and approved your ethics form? **Remember** – this is obligatory and must be completed **now**.*
The school's FEC representatives are Dr Matt Dennis and Dr Philip Scott.

Appendix C

Source Code

All of the code used in this project can be found in a public repository hosted on GitHub at github.com/UP940148/up940148-fyp

The live version of this project can be found at up940148.github.io/up940148-fyp

Appendix D

Small implementations

D.1 Speeding up loops

Whilst attempting to speed up the time taken for trees to load (Sect. 7.4.2), it was noticed that almost every loop in all Radiosity classes were implemented as `for...of` loops. These loops generally improve readability, but can lead to larger loading times when compared with other loop types (Chege, 2020).

After testing the speeds of different loops, the `while` loop came out with the best performance speed (Tab. D.1a), and so it was used in place of all the `for...of` loops where, except for those that loop over generator functions. The changes made in order to alter those loops are detailed next.

Loop type	Time taken (s)
`For...of`	561.09
`For`	389.15
`While`	379.85

(a) Before changing generator functions.

Loop type	Time taken (s)
`For...of`	300.87
`For`	274.19
`While`	244.50

(b) After changing generator functions.

Table D.1: Time taken to load default tree when using different loops.

JavaScript's generator functions return an iterator when called, so that they can be re-entered when required. A `for...of` loop can take this iterator and loop through it, however a `while` loop won't iterate through it in the same way. For this reason, all generator functions, except for `SlowRad.prepGenerator()`, have been replaced with functions that return lists (Alg. J.2). The `for...of` loops that iterate over generators were then able to be replaced with different loops that iterate over a list of values.

After removing the use of generators, the different loops were tested again, and the `while` loop still gave the fastest result (Tab. D.1b), for this reason the `while` loops were used in place of `for...of` loops all throughout the Radiosity classes.

D.2 Image saving

To be able to download the image, the canvas element that ThreeJS renders to must be initialised with `preserveDrawingBuffer` set to true. This stops the drawing buffer and the display buffer from being swapped when the scene is drawn to the canvas. By default, `preserveDrawingBuffer` is set to false, and changing it to true can cause performance issues due to the cost of copying all of the image data from one buffer to the other. No performance issues have been noticed by implementing this change, but alternative solutions may need to be investigated if performance issues arise on different systems with potentially higher resolutions.

Because ThreeJS renders the scene to a canvas element, the image can be converted to a blob using `canvas.toBlob()`. This then allows the image to be downloaded using `URL.createObjectURL(blob)`, and by setting the href of an anchor element to the object URL, the image can then be downloaded by sending a click event to the anchor element.

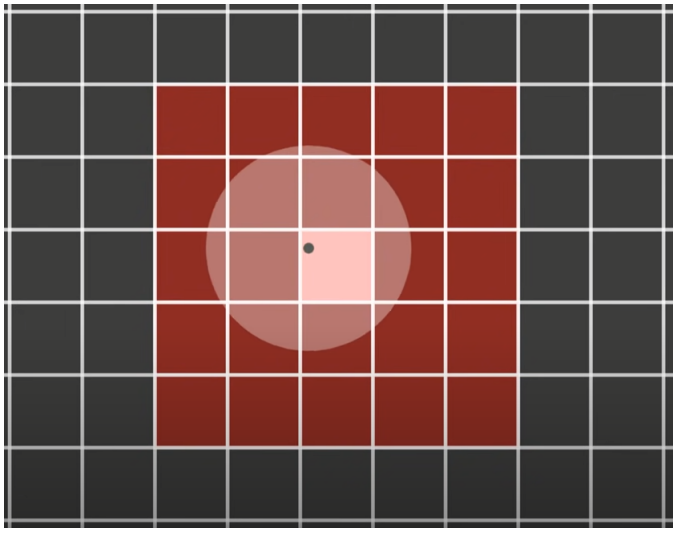


Figure D.1: Poisson Disk Sampling grid. Pink cell is the container for the point. Red grid shows the only cells that need to be considered when positioning other points.
Source: Lague (2018)

D.3 Poisson disc sampling

As mentioned in the design, Poisson disc sampling is being used for the placement of trees and bushes in the scene (Sect. 6.2.2). Poisson disc sampling works by dividing a sample space into a grid of cells, such that the diagonal width of each cell is equal to the minimum distance allowed between two points. This grid layout means that when a point is positioned in the sample space, the minimum distance won't extend beyond the 5x5 grid that is centred on the cell which the point is placed inside (Fig. D.1). This means that when placing a new point in the plane, only points that fall within this 5x5 grid need to be checked.

This algorithm was implemented in a folder called `poisson-sampling/`, and it can be called from the command line where the minimum radius, number of samples, and size of the square sample space can all be specified as arguments. It will output the list of sampled points as a list into a JSON file, which can then be read into the scene when positioning objects.

Appendix E

All tests carried out

E.1 Invisible light sources

E.1.1 Scene setup

Walls

The scene features two walls, both of size 10×10.
Wall one is rotated by $90°$, to face the positive y direction, and is then translated to centre at $(-5, 0, 0)$.
Wall one is rotated by $-90°$, to face the negative y direction, and is then translated to centre at $(5, 0, 0)$.
Both walls are sub-divided 64 times, giving a total of 8,192 patches per wall.
Both walls have an emittance value of $(0, 0, 0)$ and a reflectance value of $(0.5, 0.5, 0.5)$.

Light source

The light source is a single plane of size 1×1 centered at $(0, 0, 0)$, and rotated by $-90°$ to face wall one.
The light source has an emittance value of $(100, 100, 100)$ and a reflectance value of $(1, 1, 1)$.
Light is set to emit between time steps 0 and 10 (inclusive).

Figure E.1: Scene setup, for *Invisible light sources* test.

E.1.2 Expectations

Before making light invisible

It is expected that before setting the light source's `isLight` attribute to `true`, the light source will emit a pulse of white light, which will be reflected by wall one (left). The reflected light will travel towards wall two (right) and will be visibly reflected all along it's surface with the exception of a black square in the centre of the wall where the light source casts it's shadow.
Whilst the light is visible on wall two, red light should be visible on the surface of wall one. This is due to the light source having a natural number value in only the red channel of it's reflectance value, thus causing only red light to be reflected off it's surface.

After making light invisible

When the scene loads, the light source with `makeLight` set to true should not be visible in any capacity.
The light source should still emit a pulse of light, wall one should still reflect it towards wall two. When the light reaches wall two, there should be no shadow cast on the surface, and no light should reflected from the light source back towards wall one.

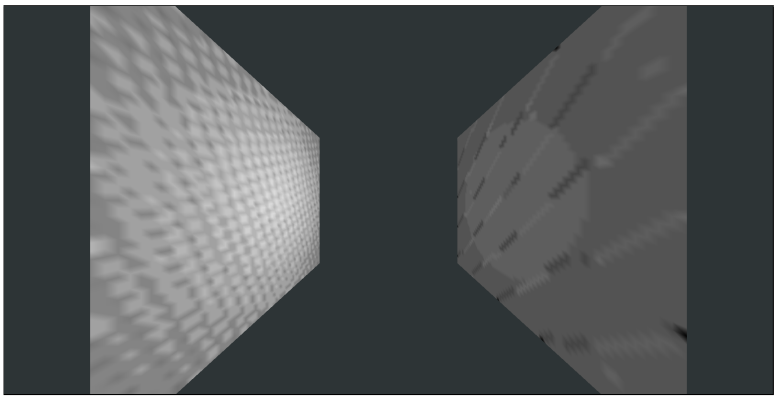


Figure E.2: Result of test when *isLight* = *true* at time step 8.

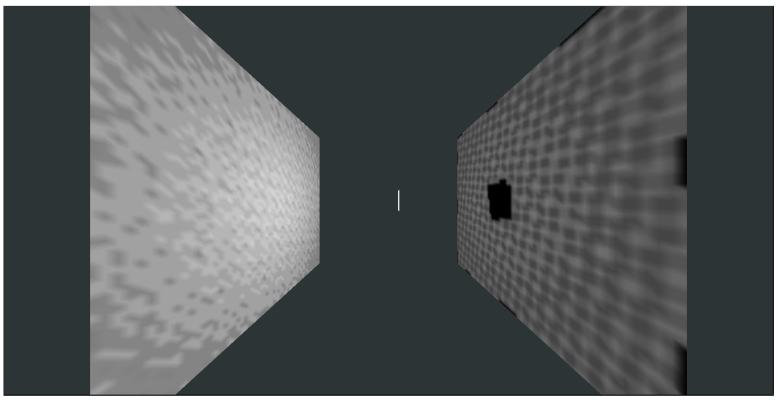


Figure E.3: Result of test when *isLight* = *false* at time step 8.

E.1.3 Results

As is shown in figures E.3 & E.2, both tests were successful and neither test revealed any unexpected behaviour.

E.2 Activating lights on timers

E.2.1 Scene setup

Floor

The floor is a single plane of size 50×50 centered at $(0, 0, 0)$, with no rotations applied. It has an emittance value of $(0, 0, 0)$ and a reflectance value of $(1, 1, 1)$.

Light source

The light source is a single cube of size 1×1 centered at $(0, 0, 0.5)$, with no rotations applied. It has and emittance value of $(1, 1, 1)$ and a reflectance value of $(0, 0, 0)$.

The light source has been set to activate between steps 0 to 10 and 30 to 40 (inclusive).

E.2.2 Expectations

- During steps 0 to 10, the light source should be emitting light, which will be reflected by the floor plane.
- Between steps 11 to 29, the light source should be emitting no light. For the first few frames of this period, the reflections of the previously emitted light will be seen fading away.
- Steps 30 to 40 should see the light source emitting light once again, with the same level of reflection in the floor plane.
- From step 41 onwards, there should be no light in the scene, except for the reflections of the previously emitted light slowly fading out.

E.2.3 Results

As expected, the light source activated and deactivated at the given times with no issues.

E.3 Flight path

E.3.1 Placing objects along path

To test that the `flightPath()` function is working as expected, a basic scene was set up with 500 cubes positioned along the path at evenly spaced intervals. A white background was used and all cubes were set with a reflectance of $(0, 0, 0)$ to make it easier to view.

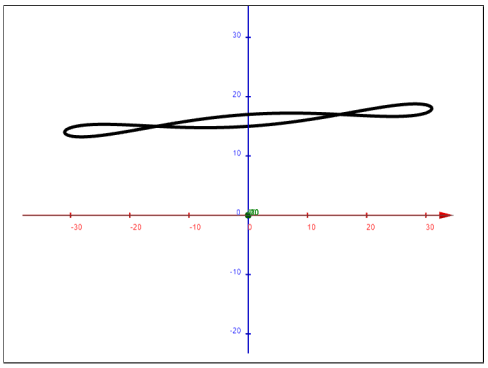
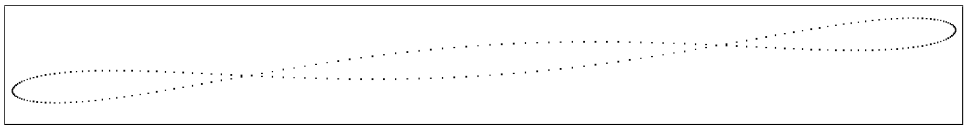
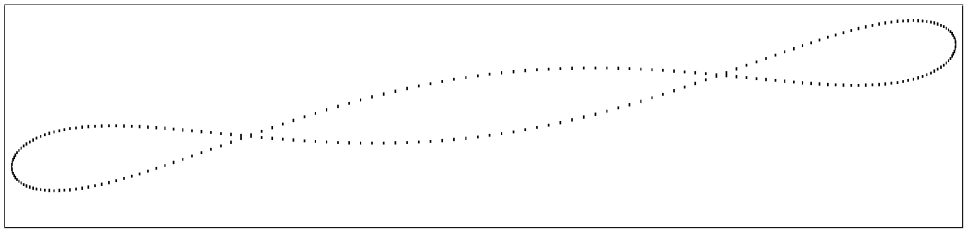


Figure E.4: Expected curve, modelled in GeoGebra's 3D Calculator.



(a) Curve with standard orthographic view.

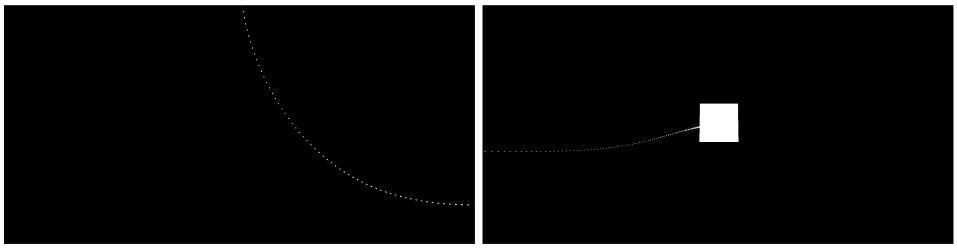


(b) Curve when orthographic view stretched vertically (For clearer view).

Figure E.5: Flight path curve output by simulation.

The curve produced from the parametric equations in use (Eq. 7.2), was plotted using Geogebra's 3D Calculator (GeoGebra GmbH, 2020)(E.4). After loading the simulation scene, a visual comparison was made between the expected curve and the curve projected in the simulation. In order to verify the similarity of the curves, the scene's orbital camera was changed from a perspective camera to an orthographic camera, as GeoGebra displays plotted curves from an orthographic view.

After inspecting the curve projected by the simulation (E.5), no visual differences were noticed when compared with the expected curve. Because of this, the functionality to position objects along the camera's path was considered to be successful.



(a) First test **(b)** Second test

Figure E.6: Simulation output at step 0 during camera motion tests.

E.3.2 Moving camera along path

For both tests in this section, the scene from the previous test (Sect. E.3.1) was used.

First test

Using the `flightPath()` function, the x, y, and z co-ordinates for the camera at every step of the animation were retrieved and passed straight into `flightCam.position`. The view from the flight camera was then tested in the simulation to see if it passed through all the cubes set out along its path.

Before beginning the simulation, it was clear that the camera was not lining up as expected because the simulation at step 0 showed a vertical view of the scene (Fig. E.6a).

Second test

After rotating the co-ordinates by $+90°$ around the x-axis before passing the co-ordinates into `flightCam.position` (Alg. 7.2), the flight camera view was displaying as expected as it passed through every cube along its path (Fig. E.6b).

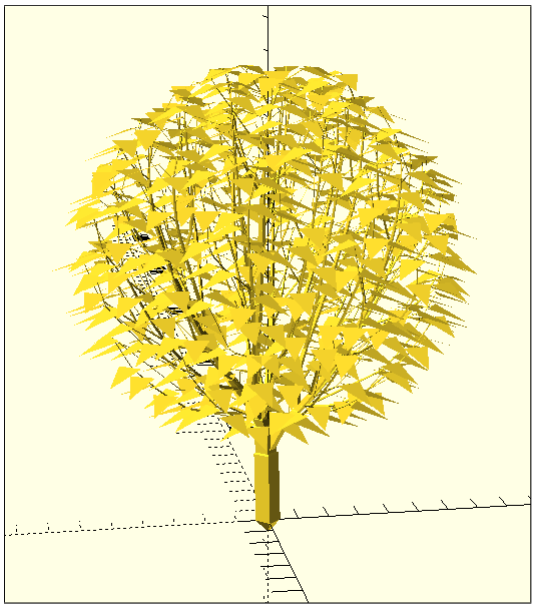


Figure E.7: Tree with default settings rendered in OpenSCAD.

E.4 Loading STL tree

E.4.1 Scene setup

The testing scene is completely empty, spare for a single tree that has been created using the `generate-tree.js` module, opened with OpenSCAD (Kintel, 2021), and then exported as a .stl file.

The tree (Fig. E.7) was generated using the default values found in the existing system [1].

E.4.2 Expectation

It was expected that the tree would render in, with some latency due to the complexity of the model.

[1] Source Code: /modeling/trees/generate-tree.js

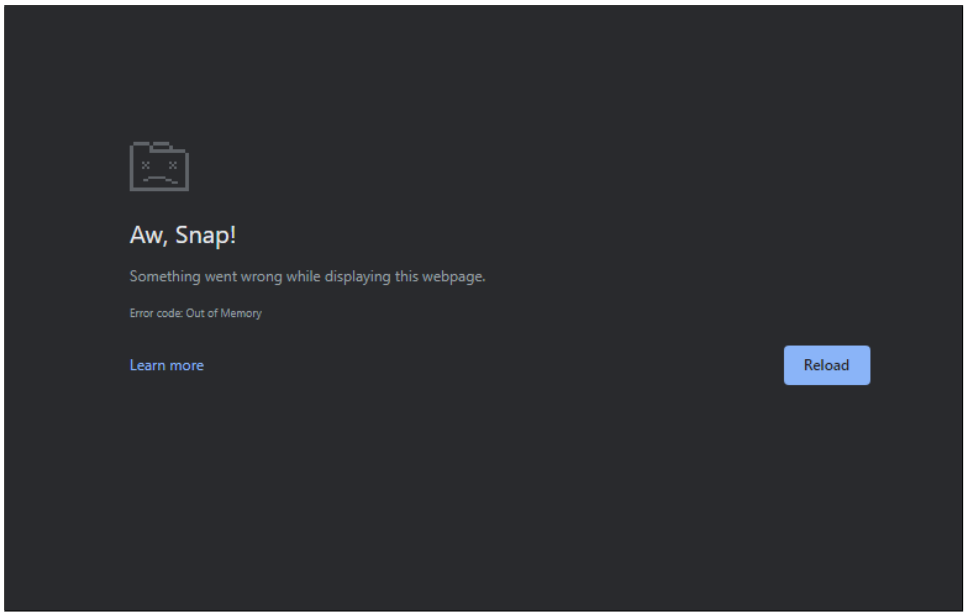


Figure E.8: Chrome's 'Out of Memory' error screen.

E.4.3 Observation

When attempting to open the scene in the browser, there was a long delay which ended when the browser crashed due to lack of memory, shown in figure E.8.

The reason for this crash is believed to be because OpenSCAD is designed for creating solid objects which can be used in CAD software ("OpenSCAD", n.d.). Because of this requirement for objects to be solid, all shapes made in OpenSCAD must have at least four faces, as this is the minimum number of faces required to create a polyhedron.

This limitation creates a requirement for every leaf in the model to have four faces. However when 3D modelling for rendering purposes, a leaf could be comprised of only two faces, a top and a bottom face. This increase in faces, and the increase in vertices by proxy, is likely the cause of this memory error.

E.5 Loading JSON tree

E.5.1 Scene setup

As it was with section E.4, the scene consists of a single tree and no other objects. In this scene however, the tree is imported from a JSON file.

E.5.2 Expectations

It is expected that when the scene loads, there will be some delay and then a tree that is structurally identical to figure E.7 will render in.

E.5.3 Observation

As expected, there was a delay when the scene loaded, however the tree rendered in exactly as anticipated (Fig. E.9) with the exception of some branches which appear to be incorrectly positioned.

The most pressing issue with this scene is the time taken for the exitances to be calculated. By using the `performance.now()` function to log the time taken, it was found that the scene with just this single tree took an average of 49 minutes and 57 seconds (2,997,299ms) to load and calculate all the exitance values, when tested three times.

E.6 Faster loading of JSON tree

E.6.1 Scene setup

The scene is completely empty, except for the default tree, which has had all branches with a width value less than 0.7 removed from it's structure.

E.6.2 Expectations

It is expected that there will be a noticeable improvement in loading time when compared to the 49 minutes 57 seconds observed in the previous test (Sec. E.5.3).

Figure E.9: First attempt JSON tree.

Test number	1	2	3	Average
Time taken (ms)	367,145	367,838	276,869	337,284
Time taken (m:ss)	6:07	6:08	4:37	5:37

Table E.1: Time taken to load the default tree with branches trimmed

The time taken to load will be determined by taking the output of the `performance.now()` function, after the tree has loaded and all exitance values have been computed.

E.6.3 Results

As can be seen from table E.1, reducing the number of branches has caused an immense performance increase for the system.

E.7 Browser exitance exporting and importing

E.7.1 Basic scene

When attempting to export the exitance data of the '`TEST: Invisible light source`' scene before any compression (Sec.), the exported file came

out to 128,049KB (125MB). Importing this exitance data into the scene worked correctly with no visual faults.

E.7.2 Complex scene

When testing on the '`Forest`' scene, the browser crashed due to a lack of memory (Fig. E.8). The crash only occurred whilst calculating the exitances, after both the RFF and distance arrays had been calculated.

E.8 Command line exitance generation

E.8.1 Basic scene

Saving the exitance data of the '`TEST: Invisible light source`' scene resulted in a file that was completely identical to the one generated in the previous test (Sec. E.7.1).

E.8.2 Complex scene

When attempting to export the '`Forest`' scene's exitance data from the command line, the system didn't crash as it did when attempting to export from the browser (Sec. E.7.2). It saved a file that was 365MB in size, and when it was loaded into the system in the browser, everything looked as expected.

E.9 Exitance file compression

E.9.1 Lossless RLE compression

The run length encoding was applied to the '`TEST: Invisible light source`' and '`Forest`' scenes' exitance data, and the files were compressed from 125MB to 119.7MB and 365.5MB to 364.8MB respectively (Tab. E.2).

Loading the exitance data back into the scene gave no errors, and everything looked as expected.

Scene	Uncompressed	RLE only	+ Precision	+ Rounding
Forest	374,236	373,511	89,316	55,045
Invisible light	128,049	122,543	23,338	676

Table E.2: File size (KB) of exitance data under different compression settings.

E.9.2 Lossy compression

Precision

Changing the precision to a zero decimal place exponential compressed the file sizes from 119.7MB to 22.7MB and 364.8MB to 87.2MB for the '`TEST: Invisible light source`' and '`Forest`' scenes respectively (Tab. E.2).

No errors occurred, and there were minimal visual differences in the simulation when compared to the uncompressed data (Fig. I.1b & I.2b).

Rounding

Implementing the rounding feature to the exitance compression system caused a further decrease in file size from 22.7MB to 675KB and 87.2MB to 53.7MB for the '`TEST: Invisible light source`' and '`Forest`' scenes respectively (Tab. E.2).

When comparing the simulation output produced from the uncompressed exitance files (Figs. I.1a & I.2a), with the output produced from the exitance files that had their exponential precision set to zero and their near-zero values rounded (Figs. I.1c & I.2c), there's very minimal noticeable difference. The differences can be seen more clearly in figure I.3.

E.10 Deployment for testing

In order to receive user feedback, the system was deployed to GitHub Pages and checked to make sure everything worked as expected.

(a) Before **(b)** After

Figure E.10: Forest scene before and after opening a different coloured scene.

E.10.1 Discoveries

With the existence of some scenes that have coloured trees, occasionally a scene which should be greyscale will load in with coloured trees. This problem only occurred when loading a greyscale scene after loading a colour scene (Fig. E.10).

This issue was caused by a previously unknown feature of JS imports, whereby an imported module is only evaluated once, regardless of how many times it is imported across different files. The use of global variables determining the reflectance of the surfaces making up the tree structure, meant that when one scene was loaded in colour, it would update the reflectance variables for every scene that is viewed after it.

This was fixed by removing the global variables in `json-tree-loader.js` and replacing them with local variables inside the `load()` function, which can be set when the function is called, and then passed as parameters to the other functions that need the values.

With the exception of this colouring issue, no faults were discovered when deploying the system.

Appendix F

Full evaluation of successful requirements

F.1 Must Have

MH1 - A moving camera on a pre-planned path.
The system features two cameras, the default orbital camera provided by the existing system, and the flight camera which was created in section 7.3.3. The ability to toggle between the cameras during runtime is enabled through the use of a button in the side menu.

MH2 - Application served on GitHub Pages.
The system is successfully deployed to my GitHub pages[1]. Since deploying the system, no errors of any kind have arisen.

MH3 - Pre-loaded exitance data.
The system is able to load in pre-generated exitance data with no issues, and if no exitance data is available for the system to use, it will generate it at run-time. With more complex scenes, the browser is unable to calculate the exitance data, however with the creation of the command line system (Sect. 7.5.1), the exitance data can be generated externally, allowing for more complex scenes to still be viewed in the browser.

[1] https://up940148.github.io/up940148-fyp/

MH4 - Light sources that activate sequentially.
Sequentially emitting light sources were successfully implemented in section 7.2.2. Light sources can be set to activate at whatever time steps they are needed.

MH5 - New method of storing and loading trees.
Tree models are now successfully stored in JSON format, and the new tree loading module allows trees to be loaded into the browser without crashing.

F.2 Should Have

SH1 - Command line calculation ability.
The command line system is fully operational and can successfully generate the exitance data for any scene made available in `environments-list.js`. For some more complex scenes, the NodeJS command line option`--max-old-space-size` may need to be used due to the JavaScript's heap limit.

SH2 - Light sources that aren't visible.
Light emitting surfaces in the scene can be made invisible by setting their `isLight` property to true. Whilst this ability to have objects that can't receive light but can emit light does break the reciprocity rule of the radiosity algorithm, no errors have arisen as the result of this feature.

SH3 - Compressed exitance data files.
The exitance files generated by the system are compressed correctly, and can be uncompressed for use with no issues. Unfortunately lossy compression was needed in order to allow the files to be deployed to GitHub Pages, however this lossy compression has minimal effect on the simulation output, as can be seen in figures I.1, I.2 and I.3.

F.3 Could Have

CH2 - Exportable images.

The button added to the side menu allows the current simulation output to be downloaded as a PNG, with none of the interface obscuring the scene (Sect. D.2).

Appendix G

Participant Feedback Form

Echolocation with Slow-light Radiosity Feedback Form

User feedback form

*Required

Informed Consent

Hello,

I am a final year Software Engineering Student at the University of Portsmouth. I am doing an engineering project where I have created a web application that uses a slowed down form of the Radiosity algorithm in a 3D environment, to attempt to simulate how echolocation detection is perceived by a bat in a forest.

The deployed system can be found at https://project.d-ellis.net/

The purpose of this feedback form is to gather information regarding the performance and potential applications of the simulation. Collecting this data will help me write an informed evaluation of the artefact for my final report.

If you have any issues or concerns regarding the artefact or this feedback form, please contact me at up940148@myport.ac.uk.

Many thanks,

Daniel Ellis

Please read through the Participant Information Sheet for this study.

The Participant Information Sheet can be found at this link:
https://docs.google.com/document/d/12D1YfwJdyPCqjjPwBsnuuvsSPd8Z-uHQcbrIKm14x48/edit?usp=sharing

1. Please confirm the following *

Tick all that apply.

- [ ] I confirm that I have read and understood the Participant Information Sheet for this study.
- [ ] I understand that data collected during this study will be processed in accordance with data protection laws, as explained in the Participant Information Sheet.
- [ ] I understand that my participation is voluntary, and I am free to withdraw at any time up until 23:59 on April 29th 2022 without giving any reason.
- [ ] I understand that by completing and submitting this feedback form, I am providing implied consent for my responses to be used for the purposes outlined in the Participant Information Sheet.

About you

2. Please enter here the unique ID that I provided to you in the email that directed you to the artefact. *

This ID will be used to destroy your data if you wish to withdraw from the study at any time.

3. Do you have experience in teaching? *

Mark only one oval.

- ◯ Yes
- ◯ No
- ◯ Prefer not to say

4. Which of these best describes the area in which you have most experience? *

Mark only one oval.

- ◯ Biological Sciences
- ◯ Visual Arts
- ◯ Technology
- ◯ Other

Technical Feedback

This section of the form will go over the questions relating to the technical performance of the simulation.

5. What browser did you run the simulation on? *

Mark only one oval.

- ◯ Chrome
- ◯ Firefox
- ◯ Internet Explorer
- ◯ Microsoft Edge
- ◯ Opera
- ◯ Prefer not to say
- ◯ Other: _____

6. What version of the browser did you use?

Leave blank if unsure

_____

7. Did the simulation crash at any point during use? *

Mark only one oval.

- ◯ Yes
- ◯ No

8. If the simulation crashed, can you provide details of the crash here

What type of crash, what scene it crashed on, etc. Leave blank if unsure.

9. Did you encounter any of the following performance issues?

Select all that apply

Tick all that apply.

- ☐ Low framerate in animation playback
- ☐ Excessively long scene loading time (longer than 10 seconds)
- ☐ Scene not loading at all
- ☐ Animation freezing/stuttering
- ☐ Other: _____

10. If you encountered any of the above performance issues, please elaborate with any additional details here

11. Are there any other notes about the technical performance of the system that you wish to mention?

Leave blank if no

12. Did the system crash before it could be tested? *

For example: The system crashed when it was loading up

Mark only one oval.

- ◯ The system crashed before it could be tested *Skip to question 26*
- ◯ The system did not crash before it could be tested

Simulation applications

This section will go over questions relating to potential applications for the simulation.

13. Do you think that this system has any artistic value to it? *

Mark only one oval.

- ◯ Yes
- ◯ No
- ◯ Unsure

14. Are there any features that could be added to the simulation itself (not the user interface), that would make this more aesthetically appealing?

Difference speeds, more complex models, colour to represent the doppler effect, etc.

15. Do you think that this system may be useful for demonstrating the concept of echolocation to students studying biology? (At any level of study) *

Mark only one oval.

- Yes
- No
- Unsure

16. Following on from the previous question. Are there any features that could be added to make the simulation more useful for teaching purposes?

Leave blank if unsure

17. Are there any other notes about the potential applications of the simulation that you wish to mention?

Leave blank if no

User Interface

This section will go over questions relating to the design of the simulation's user interface

18. Did you have to refer to the user interface help page at any point whilst using the system? *

Mark only one oval.

- Yes
- No

19. If so, then what made you visit the help page?

20. How easy did you find it to change the scene? *

Mark only one oval.

	1	2	3	4	5	
Very easy	◯	◯	◯	◯	◯	Very difficult

21. How easy did you find it to move through the animation? *

Using the play, pause, step forward, step backward, go to start, go to end, switch direction, and repeat controls

Mark only one oval.

	1	2	3	4	5	
Very easy	◯	◯	◯	◯	◯	Very difficult

22. How easy did you find it to change simulation settings? *

Changing playback speed, gamma, exposure, etc

Mark only one oval.

	1	2	3	4	5	
Very easy	◯	◯	◯	◯	◯	Very difficult

23. Overall. How easy to use did you find the interface? *

Mark only one oval.

	1	2	3	4	5	
Very easy	◯	◯	◯	◯	◯	Very difficult

24. Were there any features of the interface that you were expecting, which weren't present?

25. Are there any features you would want to see added that you think would make the system more useful?

Final remarks

Thank you for taking the time to go through this form!

26. Do you have any additional feedback points that weren't covered in the rest of the form?

If so, please enter them here

Appendix H

Participant Feedback Responses

This shows the responses submitted to the feedback form, with the participants' unique IDs omitted.

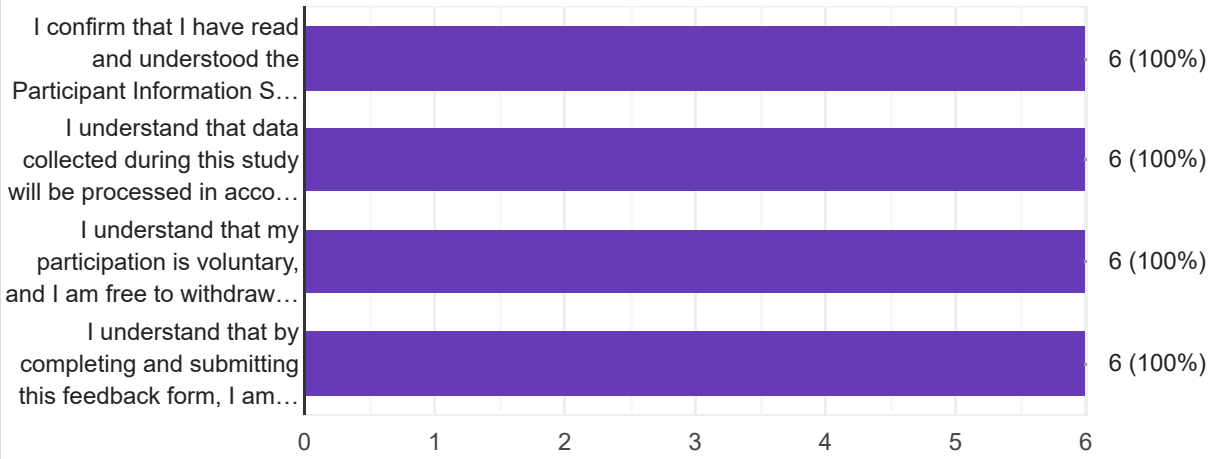
Echolocation with Slow-light Radiosity Feedback Form

6 responses

Please confirm the following

Copy

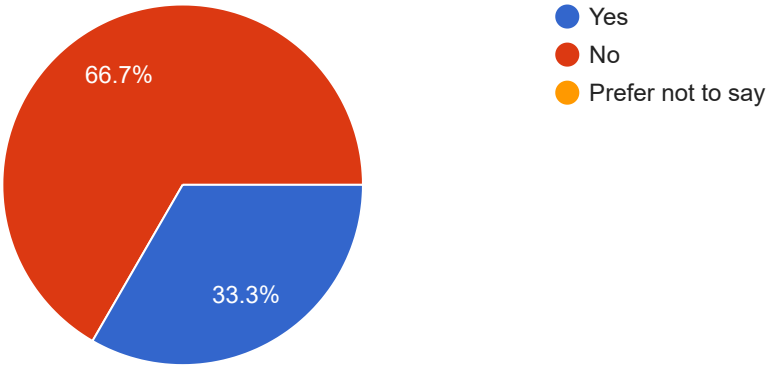
6 responses



Do you have experience in teaching?

Copy

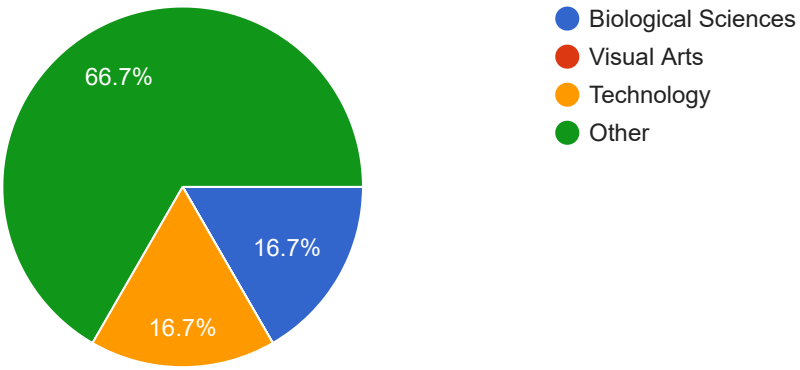
6 responses



Which of these best describes the area in which you have most experience?

Copy

6 responses

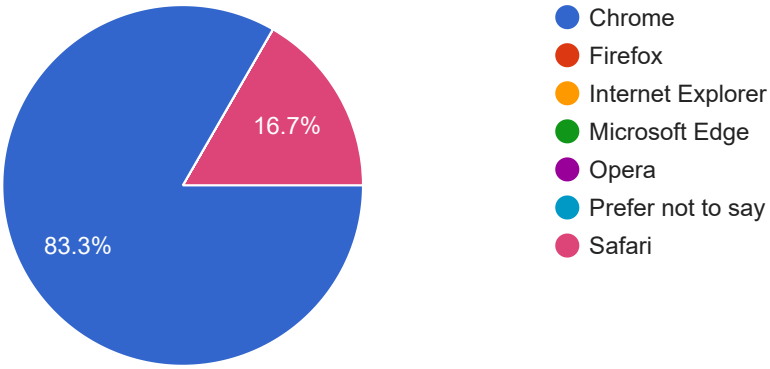


Technical Feedback

What browser did you run the simulation on?

Copy

6 responses



What version of the browser did you use?

2 responses

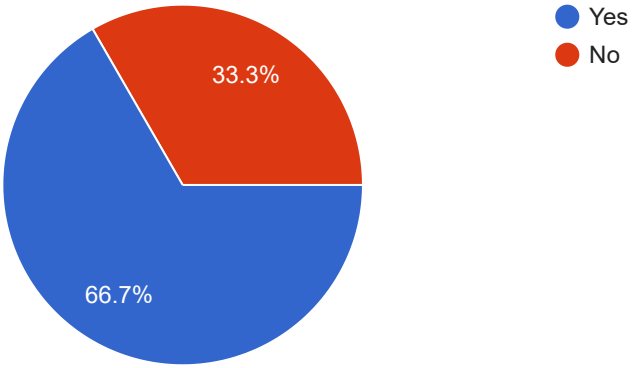
Version 100.0.4896.127

100.0.4896.127

Did the simulation crash at any point during use?

Copy

6 responses



If the simulation crashed, can you provide details of the crash here

4 responses

There was a black screen with a few icons and an axis at the side, as well as a play button and a line, which turned red once play was clicked. There was no scene shown.

Crashed on the forest scene, with error code: out of memory

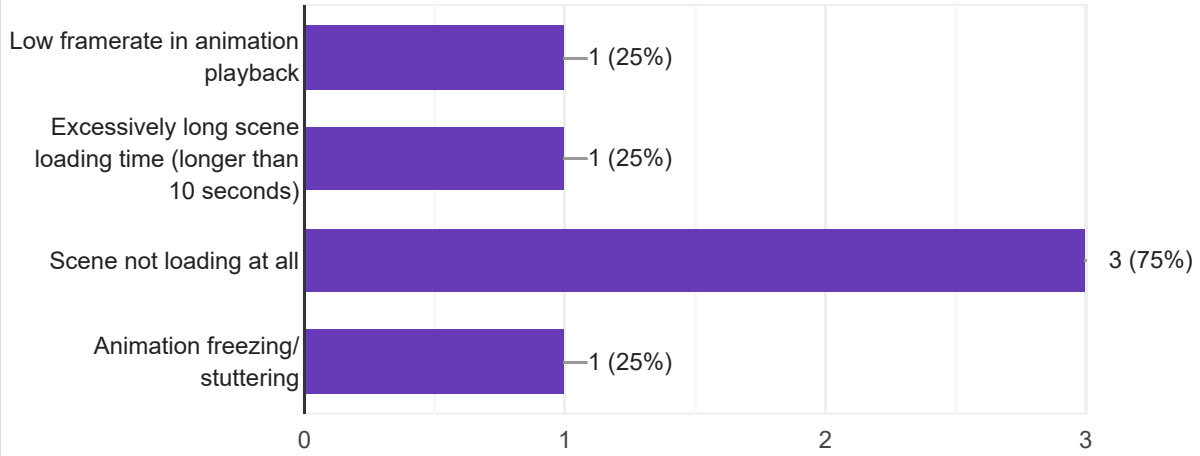
Crashed about 5 seconds into moving into the trees. Computer started struggling and it crashed

unresponsive and time out on pages

Did you encounter any of the following performance issues?

Copy

4 responses



If you encountered any of the above performance issues, please elaborate with any additional details here

4 responses

The initial program opened, but nothing was really usable and then an aw snap page opened and said Error code: Out of Memory.

The forest scene loaded quickly and immediately after crashed

Took a while to load, then animation had low framerate and freezes while playing

Some pages loaded very slowly but animation playbacks wouldn't load and then crashed

Are there any other notes about the technical performance of the system that you wish to mention?

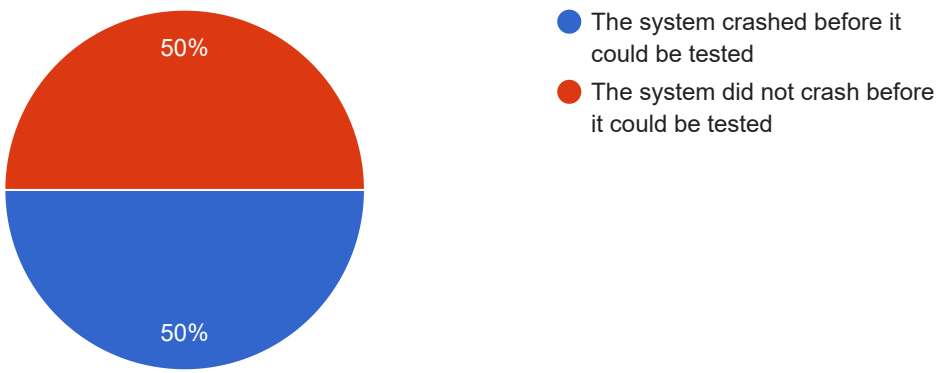
1 response

It didnt let me access any of the other options in the side bar besides the eye one. Think my laptop is too slow

Did the system crash before it could be tested?

Copy

6 responses

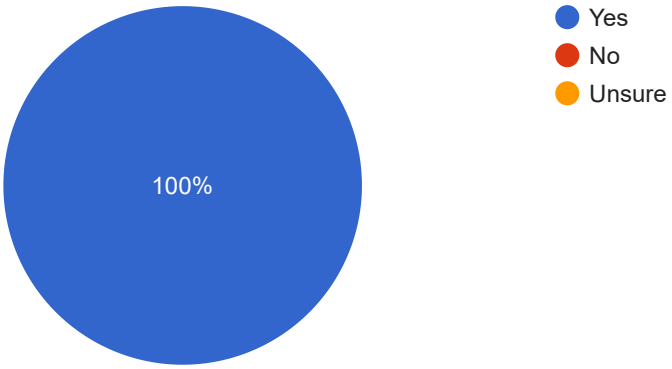


Simulation applications

Do you think that this system has any artistic value to it?

Copy

3 responses



Are there any features that could be added to the simulation itself (not the user interface), that would make this more aesthetically appealing?

2 responses

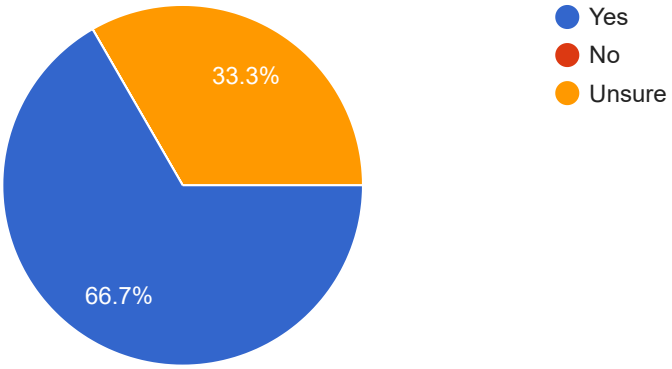
More variety to the environment, different objects maybe.

Silly one but capitalization of words on sidebar panel, when hoovering over the pictures its all lowercase

Do you think that this system may be useful for demonstrating the concept of echolocation to students studying biology? (At any level of study)

Copy

3 responses



Following on from the previous question. Are there any features that could be added to make the simulation more useful for teaching purposes?

2 responses

Opportunity to control the motion and direction of the camera. Attempt to avoid objects using echolocation.

maybe birds eye view with arrows / lines to show the sound bouncing

Are there any other notes about the potential applications of the simulation that you wish to mention?

0 responses

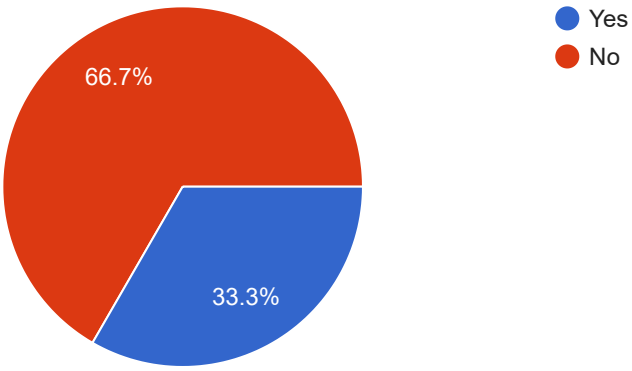
No responses yet for this question.

User Interface

Did you have to refer to the user interface help page at any point whilst using the system?

Copy

3 responses



If so, then what made you visit the help page?

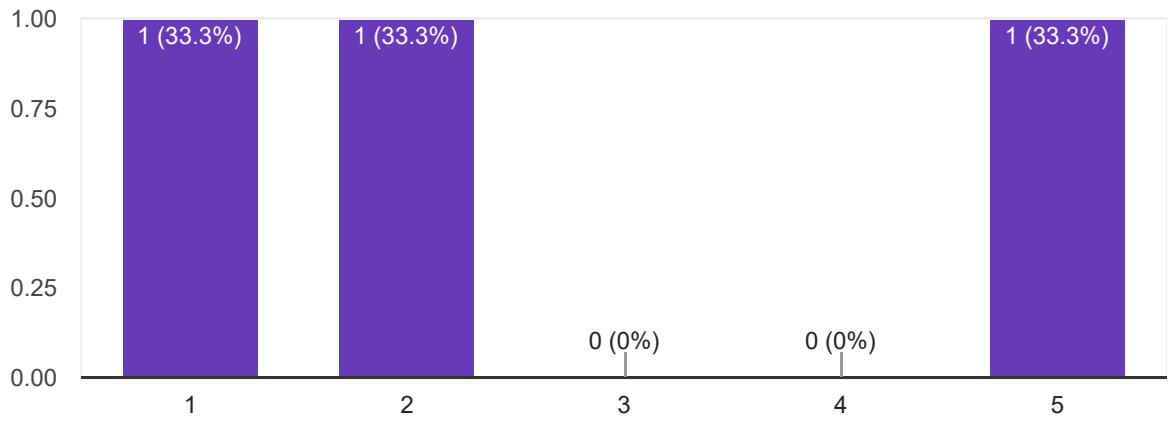
1 response

I wanted to see what else could be done.

How easy did you find it to change the scene?

Copy

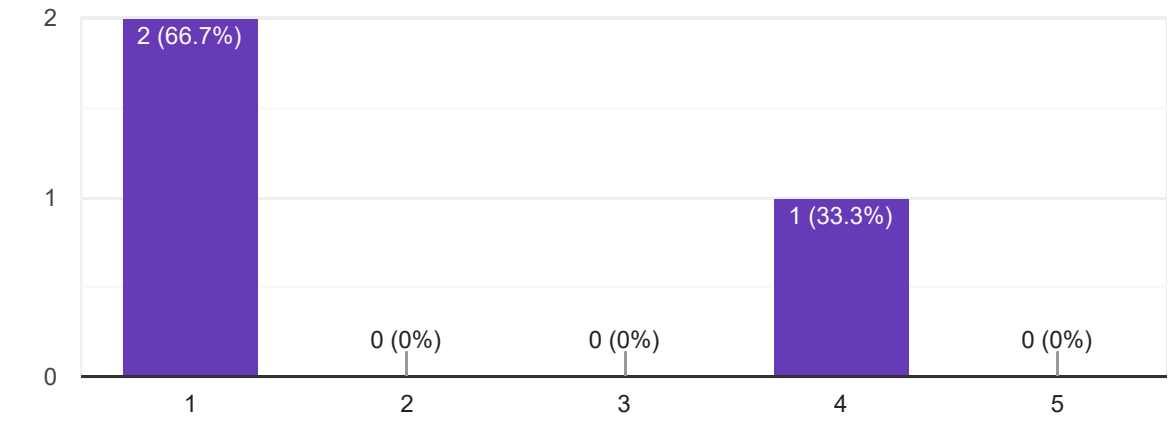
3 responses



How easy did you find it to move through the animation?

Copy

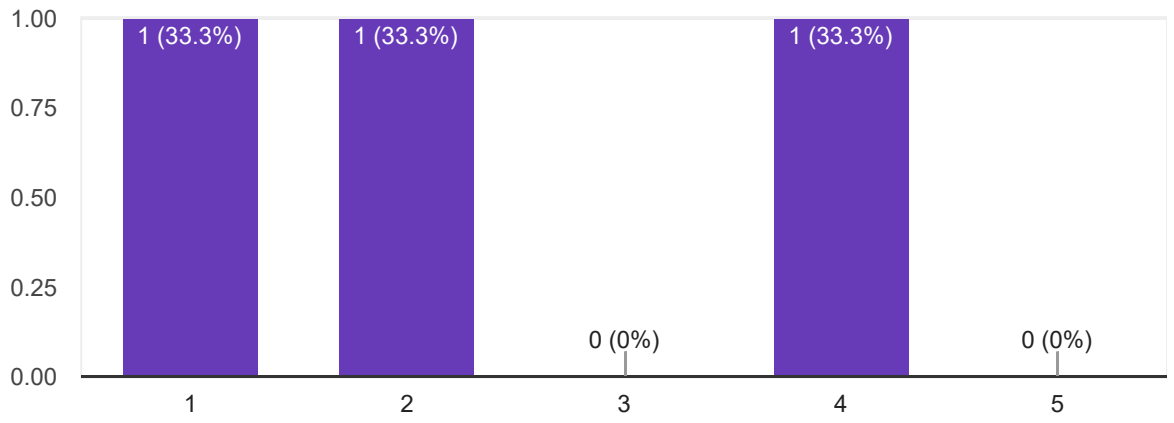
3 responses



How easy did you find it to change simulation settings?

Copy

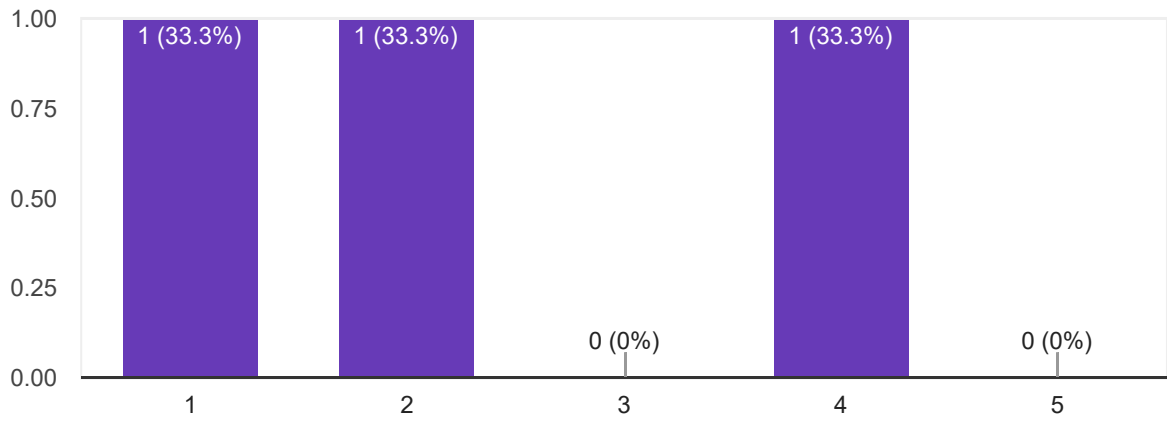
3 responses



Overall. How easy to use did you find the interface?

Copy

3 responses



Were there any features of the interface that you were expecting, which weren't present?

1 response

Only as previously mentioned

Are there any features you would want to see added that you think would make the system more useful?

2 responses

Only as previously mentioned

More accessible to laptops and slower processing computers

Final remarks

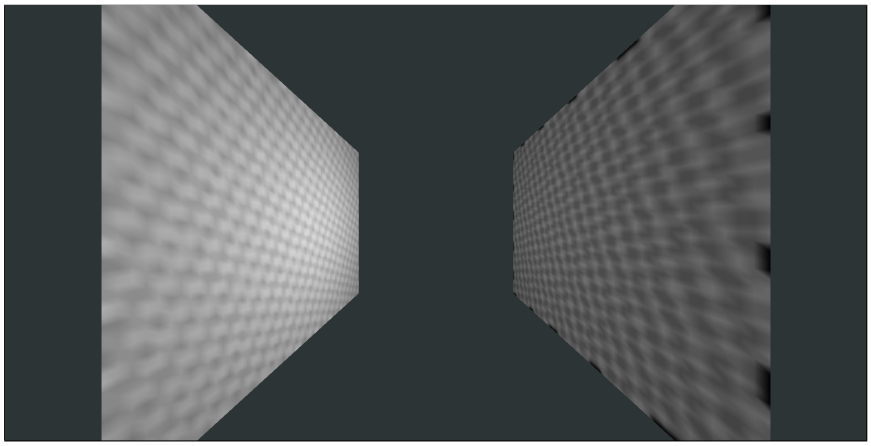
Do you have any additional feedback points that weren't covered in the rest of the form?

0 responses

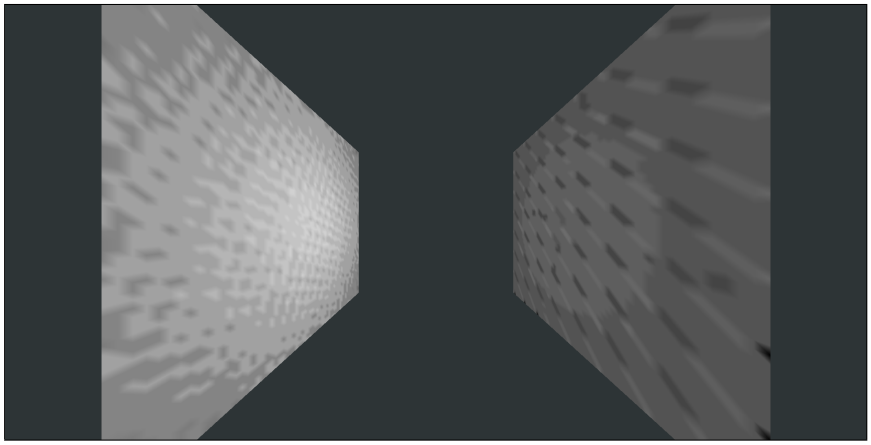
No responses yet for this question.

Appendix I

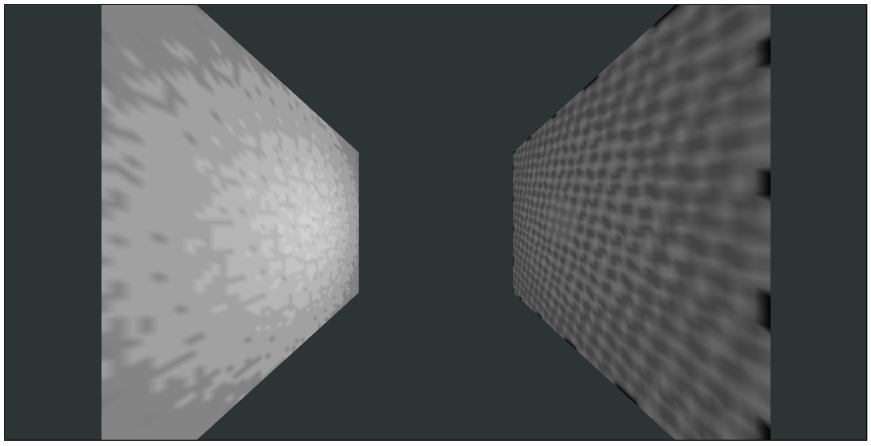
Simulation Differences From Compression



(a) Uncompressed.



(b) When exponential precision set to 0 decimal places.



(c) When rounding values that are close to 0.

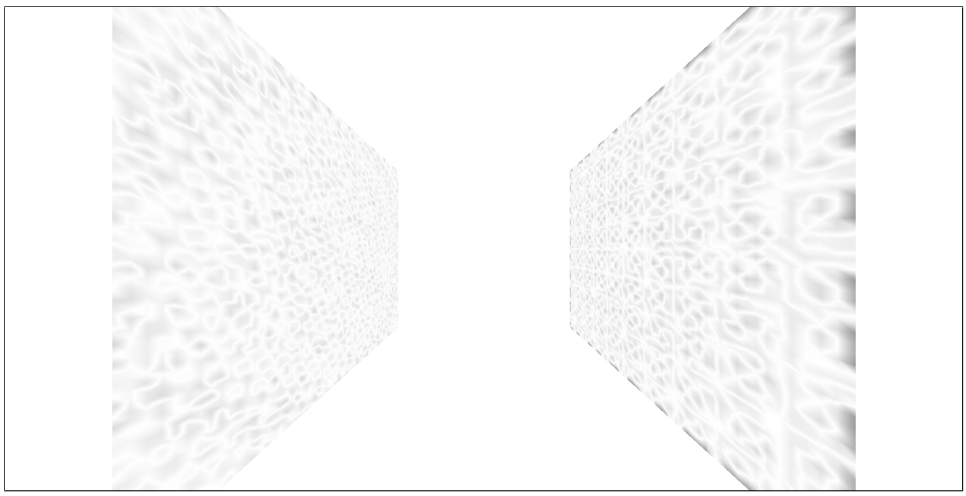
Figure I.1: The '`TEST: Invisible light source`' scene at time step 7, with different levels of compression.

(a) Uncompressed.

(b) When exponential precision set to 0 decimal places.

(c) When rounding values that are close to 0.

Figure I.2: The '`Forest`' scene at time step 118, with different levels of compression.



(a) Difference in '`TEST: Invisible light source`' scene at time step 7.

(b) Difference in '`Forest`' scene at time step 118.

Figure I.3: Absolute differences between uncompressed exitance data and the fully compressed exitance data. Created by overlaying images in Adobe's Photoshop and using the 'Difference' blend mode (Adobe, 2022).

Appendix J

Code Comparison

```
// Old way of getting vertices
* _vertexIterator() {
    for (const i of this.instances) {
        for (const v of i.vertices) {
            yield v;
        }
    }
}

get vertices() {
    return this._vertexIterator();
}
```

(a) Original generator function (Kopecký & Mattone, 2020, environment.js L62-L72).

```
// New method of getting vertices
get vertices() {
    const verts = [];
    let i = 0;
    while (i < this.instances.length) {
        let v = 0;
        while (v < this.instances[i].vertices.length) {
            verts.push(this.instances[i].vertices[v]);
            v++;
        }
        i++;
    }
    return verts;
}
```

(b) New method which returns a list (Source code, environment.js L68-L80).

Algorithm J.2: Comparison of generator function versus returning a list when getting environment vertices.