Rendering Invisibility

Final Report for CS39440 Major Project

*Author*: Katherine Rose Farmer ([krf@aber.ac.uk](mailto:krf@aber.ac.uk))

*Supervisor*: Dr. Bernie Tiddeman (bpt@aber.ac.uk)

12th April 2015

Version 1.0 (Draft)

This report is submitted as partial fulfilment of a BSc degree in  
Computer Graphics, Vision and Games inc. Industrial Placement (G451)

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Wales, UK

**Declaration of originality**

In signing below, I confirm that:

* This submission is my own work, except where clearly indicated.
* I understand that there are severe penalties for plagiarism and other unfair practice, which can lead to loss of marks or even the withholding of a degree.
* I have read the sections on unfair practice in the Students’ Examinations Handbook and the relevant sections of the current Student Handbook of the Department of Computer Science.
* I understand and agree to abide by the University’s regulations governing these issues.

Signature ……………………………………………. (Katherine Rose Farmer)

Date …………………………………………………

**Consent to share this work**

In signing below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Signature ……………………………………………. (Katherine Rose Farmer)

Date …………………………………………………

**Ethics Form Application Number**

The Ethics Form Application Number for this project is: 983.

**Student Number**

# Acknowledgements

I am grateful to Bernie Tiddeman for being my supervisor and helping me progress through this project. The amount of mathematical and graphical problems that I had seemed insurmountable at first but Bernie was always there to give me a hand.

I’d like to thank Oliver Roe for being a steadfast rock at my side when the stress of this project was getting too much. I am almost certain I would not have made as much progress if I didn’t have such brilliant emotional support and rubber-ducking coming from him! I’d also like to thank many of the other fourth years for being supportive when I was stressed and for being there to bounce ideas off when I was stuck. Also to my parents for putting up with all the technical jargon I was throwing their way when they asked how this project was going!

I’d also like to thank Aberystwyth University for the last 4 years and the chance to complete this course and all the work that has been done to support my continuing progress through this course. This project is the culmination of all that I’ve learned here and I can only hope it reflects the high quality of teaching I have received here.

# Abstract

Invisibility being a realistic possibility, without the use of optical illusions, is a recent discovery and is in large part due to the progress made in the field of transformation optics and metamaterials. With this recent progress, the idea of virtually simulating the theory presented is an interesting one. Hence why this project was undertaken. A virtual simulation can model any potential pitfalls that could occur while science cannot currently create a structure that would render an object invisible to the naked eye using the theories presented.

The main aim of this project was to produce a virtual model that was based on the theory presented and then to adjust any of the parameters involved to see what effects could be achieved. A side aim of the project was also to test the various geometrical shapes the cloak could take and how this would affect model. The project aimed to prove that the theory could be presented in a virtual simulation and that by adjusting certain parameters, certain effects could be achieved.

The model was produced using WebGL in combination with Three.js. The model has been created in a spherical, cylindrical and conical shape, all with a solid and shell structure included. The effects that were achieved were perfect invisibility, a chromatic effect and showing the cloak from a single viewpoint, looking into how the cloak might appear if you were able to view the cloak from one point all the way through. The project has proven that it is possible to render an invisibility cloak using the theory presented and to produce various effects based on the theory. My results are somewhat compromised by the language used and Three.js as the results are inconclusive based on the rendering being controlled by the language in a way that has produced some results that appear to be correct according to the theory but when tested separately produces incorrect results. In conclusion, further work would be needed to create a completely accurate model, mostly into determining how far the language is affecting the model and how that can be counteracted.

Contents

[Acknowledgements 2](#_Toc418512693)

[Abstract 4](#_Toc418512694)

[Background, Analysis & Process 8](#_Toc418512695)

[Introduction 8](#_Toc418512696)

[Reason for Project Choice 8](#_Toc418512697)

[Background 9](#_Toc418512698)

[Refraction 9](#_Toc418512699)

[Transformation Optics 10](#_Toc418512700)

[Metamaterials and Invisibility Cloaks 11](#_Toc418512701)

[Similar Projects 11](#_Toc418512702)

[Analysis 12](#_Toc418512703)

[Research Method 15](#_Toc418512704)

[Design 17](#_Toc418512705)

[FDD Design 17](#_Toc418512706)

[System Design 18](#_Toc418512707)

[Model Designs 20](#_Toc418512708)

[Spherical Model 20](#_Toc418512709)

[Cylindrical Model 22](#_Toc418512710)

[Conical Model 23](#_Toc418512711)

[Effects 24](#_Toc418512713)

[Implementation 25](#_Toc418512714)

[Tools 25](#_Toc418512715)

[Initial Implementation 25](#_Toc418512716)

[Three.js 25](#_Toc418512717)

[GitHub Pages 26](#_Toc418512718)

[Version One 27](#_Toc418512719)

[Skybox 27](#_Toc418512720)

[Glass Spheres 27](#_Toc418512721)

[Version Two 28](#_Toc418512722)

[Shell Structure 29](#_Toc418512723)

[Hemispheres 29](#_Toc418512724)

[Sandboxing 30](#_Toc418512725)

[Initial Sandboxing 30](#_Toc418512726)

[Shaders and Models 31](#_Toc418512727)

[Updated Refraction and Refraction.java 32](#_Toc418512728)

[Chromatic Shaders 32](#_Toc418512729)

[Ray Path-tracing 32](#_Toc418512730)

[Single and Multiple Viewpoints 32](#_Toc418512731)

[Final Versions 32](#_Toc418512732)

[Spherical Model 32](#_Toc418512733)

[Cylindrical Model 32](#_Toc418512734)

[Conical Model 32](#_Toc418512735)

[Testing and Results 32](#_Toc418512736)

[Unit Testing 32](#_Toc418512737)

[Tutorial 32](#_Toc418512738)

[Version One 32](#_Toc418512739)

[Version Two 32](#_Toc418512740)

[Solid Spherical Model 33](#_Toc418512741)

[Shell Spherical Model 33](#_Toc418512742)

[Solid Cylindrical Model 33](#_Toc418512743)

[Shell Cylindrical Model 33](#_Toc418512744)

[Solid Conical Model 33](#_Toc418512745)

[Shell Conical Model 33](#_Toc418512746)

[System Testing 33](#_Toc418512747)

[Experiments 33](#_Toc418512748)

[Results 34](#_Toc418512749)

[Conclusions 34](#_Toc418512750)

[Overall Conclusion 34](#_Toc418512751)

[Issues Encountered 34](#_Toc418512752)

[Critical Evaluation 35](#_Toc418512753)

[Research 35](#_Toc418512754)

[Design 35](#_Toc418512755)

[Implementation 35](#_Toc418512756)

[Testing 35](#_Toc418512757)

[Overall Evaluation 35](#_Toc418512758)

[Appendices 36](#_Toc418512759)

[A. Third-Party Code and Libraries 36](#_Toc418512760)

[Three.js 36](#_Toc418512761)

[OrbitControls.js 36](#_Toc418512762)

[KeyboardState.js 36](#_Toc418512763)

[Github Pages 36](#_Toc418512764)

[Fresnel Shader 36](#_Toc418512765)

[B. Mathematical Formulas 39](#_Toc418512766)

[Snell’s Law 39](#_Toc418512767)

[Snell’s Law – Vector Form 39](#_Toc418512768)

[Fermat’s Principle – Optical Path Length 39](#_Toc418512769)

[Maxwell’s Equations 39](#_Toc418512770)

[C. Code Samples 40](#_Toc418512771)

[Refract Method [4] – Cg 40](#_Toc418512772)

[Refract Method[26] – GLSL 40](#_Toc418512773)

[D. Outline Project Specification 41](#_Toc418512774)

[E. Design Specification 42](#_Toc418512775)

[F. Test Specfication 43](#_Toc418512776)

[Annotated Bibliography 44](#_Toc418512777)

# Background, Analysis & Process

## Introduction

Invisibility cloaks are starting to enter more and more into the realm of possibility. There are already many projects that both theorise and display practical solutions to the problem of how to create true invisibility such as the ‘Rochester Cloak’ [1] that using a series of lens of differing focal length to bend light around an object. Currently this is limited by the angles that it can viewed from and some edge problems that ruin the invisibility by hinting at the object cloaked or distorting the background behind the object.

What this project aimed to tackle was the idea of a 3-D rigid structure that would cloak an object within it. The difference was this project aimed to render such a structure in the virtual space and thereby obtain results about how well current theories into metamaterials, transformation optics and their relation to invisibility can be displayed in a virtual space and what this can inform us about those theories. One of the reasons for working in a virtual space when it comes to these theories is that currently they cannot create structures big enough to cloak in all directions and still work on the visible light spectrum in a reasonable amount of time. This is due to the structure needing to be less than a few hundred nanometres for each metamaterial ‘atom’ in the structure [2]

The direction the project ended up taking was to create multiple geometrical models of the invisibility cloak using the WebGL refraction model and manipulating it to use transformation optics, the mathematical equations behind metamaterials being used to create invisibility cloaks, and then to manipulate those equations to create effects that would be comparable to an invisibility cloak if it were created in reality.

## Reason for Project Choice

The author’s reasons for this project choice are threefold. One is that the subject of invisibility is universally fascinating and being able to delve more deeply into this project and the subject matter behind was a brilliant opportunity to expand and gather more knowledge in a field they had little knowledge of.

The second reason was that the author’s course and skills were more directed towards graphical projects, with previous experience in Javascript, WebGL and HTML5 being most useful for this project. This would provide a good example of their graphical skills as well as an example of their web and research skills. The ability to delve more deeply into extra libraries such as Three.js and to use WebGL for a more complicated challenge was also a draw to accepting and completing this project.

The third reason was that their industrial placement had provided little chance to use their graphical skills in a professional environment or a project of a large size. This meant that will their skills were more directed towards this area, these skills were unused and their other software engineering skills were enhanced. By using this opportunity to enhance their graphical skills while maintaining their other software engineering skills, they will show a wider range of abilities to future employers.

## Background

As the author was not familiar with the topics involved with this subject, a lot of background research was required. The research can be broken down easily into the four main topic areas.

### Refraction

While it may appear to be a basic area to start with, it was a good place to gain a level of understanding that could be used in the future research topics.

Refraction is the occurrence of light passing between two different mediums and the difference causing the velocity of the light to change. An example of this is the change between air and water. If we look into a pool of water and there is an object at the bottom of the pool, the object will appear bigger than it actually is due to refraction.

Snell’s Law is used to describe the relationship between the angles of incidence, the incoming ray of light towards the refractive medium, and the angle of refraction that occurs after the ray has passed into the medium. [3] This law can be used in vector form to obtain the refraction vector using the incident vector and the refractive index of the two mediums the light passes through. The refractive index is a number that describes how light passes through the medium. The formulas for Snell’s Law and its vector form can be found in Appendix B.

Another reason for looking into refraction was to understand how the graphical languages being contemplated for this project, OpenGL and WebGL, recreated refraction. This was made simpler because WebGL is based upon OpenGL ES and therefore uses the same implementation of refraction as OpenGL.

The refraction model is based upon the code presented in GPU Gems 2 [4] that describes how to simulate generic refraction. This code is included in Appendix C but will be commented on here. As can be seen from the code, it uses texture mapping to recreate the how refraction should appear in real life. The texture map co-ordinates are transformed by a small amount that is based on the refractive index of the material you are attempting to imitate in your simulation.

This idea is used in OpenGL, but it transforms the texture map using vectors instead. The refract method for OpenGL is also included in Appendix C and be discusses in more detail here. This refract methods uses the vector form of Snell’s law found in Appendix B to find the refraction vector. This occurs in the vertex shader, the shader that is used in OpenGL to calculate the vertices of the program and what kind of data is found at each vertex. The refracted vector is then passed to the fragment shader. The fragment shader works out the colour of each vertex so that it can be rendered ass that colour. The easiest way to use the vector in the fragment shader is to have a cubemap that you have passed to the shader available. The vector is then used, much like the NVIDIA refract method, to transformation the map so that it appears refracted according to the refractive index you chose in the vertex shader.

Understanding how this refraction model worked was crucial to implementing this project and therefore implementing the theories that were researched. However it was less crucial to understanding the theory being dealt with in this project. The next few sections will be dealing with the different topics that helped to shape the author’s understanding of the invisibility cloaks being theorized and produced in the last couple of decades, due to the relative infancy of the invisibility cloaks created using metamaterials.

### Transformation Optics

Transformation optics is a specialized subsection within optics, the study of light. Transformation optics focuses on how to manipulate light through geometric transforms. The way it creates these geometric transforms is via metamaterials.

Transformation optics begins with Fermat’s principle. Fermat’s principle, also known as the principle of the least time, is the principle that light travelling between two points will take the path of the least time. In Cartesian co-ordinates, this is always a straight line between the two points. Fermat’s principle was also a basis for Snell’s Law, talked about in the previous section. The reason Fermat’s Principle is an important starting point is that Fermat’s Principle also works in transformed co-ordinates, the kind that transformation optics deals with. By proving that it continues to work in transformed co-ordinates, and that the shortest path in these co-ordinates tends to be a curved path, proves that by using metamaterials to trick the light into acting as if the space has been transformed, the light can be guided and therefore an object made invisible

However, to work with metamaterials, the light must be treated as a wave, not a ray. This is where Maxwell’s equations come in. Maxwell’s equations are used to describe the laws that govern electromagnetic fields work. These were the equations that first proved that visible light was on the electromagnetic spectrum by showing the speed of light was the same as the speed of the propagation of electromagnetic waves. These equations are needed as light must be described as a wave to understand how it would interact with the metamaterials due to the materials they are made of and the way they work. Maxwell’s equations are included in Appendix B for reference.

The co-ordinate transformations that are required for invisibility cloaks are such that the co-ordinate leave a hole in space in the middle of the so-ordinates. As such, all objects in that space would be invisible to any waves that are propagated around the space. An example of what this co-ordinate space would look like can be found in [5] in Figure 9.

There is a lot more to transformation optics than what has been discussed here, but these sections should suffice to give an understanding of the kind of research that was undertaken and the topics that are necessary to explain the next two sections.

### Metamaterials and Invisibility Cloaks

Metamaterials have been around a lot longer than people may think when they hear the word mentioned. The first occurrence of metamaterials can be found in Roman times. [6] Ruby glass contains nano-scale gold droplets that make the glass appear ruby based on the concentration of the droplets. The difference between this metamaterial and modern metamaterials is the amount of control we have over the structure of the metamaterial.

Two uses of the metamaterials are superlens and cloaking devices, the latter being what this project was about.

Superlens are also an interesting use of metamaterials being used to manipulate electromagnetic waves. A superlens is designed to go beyond the diffraction limit, so there is no loss of resolution at smaller levels. Conventional lens have a limit to what microscopic level they can go too without losing information. A superlens is designed to go beyond this using negative refraction. Negative refraction sounds like something that cannot be, but Sir John Pendry described this effect in a lecture he gave at the Institute of Physics, ICL. [7] [8] By using metamaterials that are constructed to have a refractive index of -1, the lens can go beyond the diffraction limit and super scale imaging can occur. Currently no superlens have been built as the metamaterials required would be non-trivial, but a hyperlens, which puts objects in the near-field into the far field, has been built for the UV spectrum. [9]

For invisibility cloaking using metamaterials there has been many more proof of concepts constructed. Invisibility cloaking uses the theories presented in transformation optics to propagate the electromagnetic waves around a shielded object. A common structure in these metamaterials is a lattice structure made of split rings, described in a paper on the cloak created to work on microwave frequencies [10] The split rings are placed in a lattice formation and work like crystals do upon the waves. These structures have to be smaller than the frequency that the waves you are planning to work with are which means that these structures are likely to be at the nano-level. This is where the problems occur. Creating specific nano structures, without any imperfections, is still very non-trivial. Hence why most invisibility cloaks are proof of concepts or classed as imperfect invisibility cloaks. The invisibility cloaks that this project aimed to simulate were ones found in theory. For a spherical cloak, the lecture given by Sir John Pendry [7] gives a good example of the kind of structure we’re looking at. For cylindrical cloaks, a cloak was presented by K. Elassy et al [11] that describes the fundamentals of designed a cylindrical cloak and includes many examples of the structures involved. These designs will be discussed more in the design section as they are relevant to how the author designed the structures for the virtual representations.

### Similar Projects

Another research path was looking into any similar projects to the one proposed. However it appeared after numerous researches that there were no project sufficiently similar to this project that would give be some idea of what systems were available or viable.

The closest project I discovered what a virtual representation of the path of the rays through different geometrical cloaks. This did not deal with a complete virtual representation, but it did include a good system for testing the cloak visually using the ray-traced paths. [12]

There are many examples of invisibility cloaks and cloaking device within video games, such as the tactical cloak from the Mass Effect series.



Example of tactical cloak being used in Mass Effect 3. [Image](http://oyster.ignimgs.com/mediawiki/apis.ign.com/mass-effect-3/thumb/d/dd/Tact_cloak.jpg/228px-Tact_cloak.jpg) belongs to IGN, Bioware and EA.

While the implementation of these cloaks is nothing close to the implementation of the theories presented, the effects that rendered a cloaked enemy or hero slightly visible were an inspiration for the kind of effects that might be expected from an imperfect cloak on the visible light spectrum. An example of an effect that seems likely to occur is a prismatic or chromatic effect. This is more of an informed assumption that a proven point. Reasons for this are that there is currently not a 3D cloak that works on the visible light spectrum, or research into imperfect invisibility cloaks on the visual spectrum, due to reasons discussed above. However as metamaterials refract light, not dissimilar to crystals, the idea of having an imperfect structure that acts as a prism is not ridiculous.

## Analysis

Analysis was not a beginning concern for this project, more of an iterative concern throughout the project. Much of this was to do with continuing research that took place throughout the project and therefore changed what conclusions where previously created through problem analysis. Much of this section will talk about the beginning analyses but will also dip into the analyses that took place further into the project as these shaped the final product by a large degree. One of the first major decisions was the programming language that was chosen for the project: WebGL.

WebGL is an API for Javascript that is widely supported by modern web browsers such as Google Chrome, Mozilla Firefox and Apple Safari. It is used for rendering 2D and 3D graphics in browsers without the use of a plugin. It is based on OpenGL, which is a multi-language 3D and 2D graphics API that is available to use on multiple platforms including embedded systems. There are multiple reasons as to why WebGL was chosen as the language of choice for this project.

One of the driving factors behind the choice of WebGL was the Three.js library. An early decision in the project was that the creation of the geometrical shape of the cloak, such as a sphere or a cylinder, should be as simple as possible so that the work on the implementation of the invisibility and various effects would not be hindered. Three.js is especially good at dealing in abstractions. Creating a renderer, a scene and objects is extraordinarily easy, compared to base WebGL where it can require a comparatively significant amount of time to create an entire scene and then work on it.

Another benefit to Three.js was that there were numerous examples, thorough documentation and tutorials available which would aid in learning the library as quickly as possible and the possibility that any problems encountered further along in the project would likely have an answer within the widespread Three.js community.

The choice of WebGL was also facilitated by the author’s choice of version control in GitHub. The author had used github in numerous projects previously and was aware of its many benefits when it came to using it for version control. Github has extensive documentation on how to use its shell interface and also a desktop version that is simple enough for day-to-day commits. The reason GitHub worked so well with WebGL was that GitHub has support for webpages attached to projects called GitHub Pages. These can be projects by themselves, such as the GitHub Page the author used for the project blog, or webpages attached to already created projects, such as this project. This meant that the GitHub Page could be used as both version control and a display for this work.

The author’s experience in using WebGL for a previous university assignment was also a factor in choosing WebGL for this project. As the subject matter was relatively unknown to the author as well as being complex physics that would require some time to research and understand, it made sense to choose a language that was both suited to the project and known to the author so as to reduce any learning time needed for the language and use that time instead for the background reading.

After deciding upon the language, the next decision was what the model would focus upon depicting and the questions that the research and implementation would answer. The main questions that the research was aiming to answer were questions along the lines of:

* **Is it possible to render an invisibility cloak, which functions using the theories presented, in a virtual simulation?**
* Are there any problems with representing an invisibility cloak in a virtual simulation? What causes these problem? Can they be fixed in the virtual simulation?
* Any there are strengths in representing this invisibility cloak in a virtual simulation? Why are these strengths occurring? Can these strength be used when creating the cloak in reality?
* What does the virtual simulation show about the theories presented? Does it confirm them? Does it reveal any new information?
* What effects are created in the virtual simulation, through imperfections or otherwise, that could occur in a cloak produced in reality?

The bold question is the main hypothesis of this project, with the other questions being derived from the main question and also used to obtain more detailed information about the main question. The effects are where a lot of the compromise happened, as well as the type of virtual simulation that ended up being modelled.

An example of these constraints affecting the problem analysis is the difference in what the main concern was at the beginning of the project. It was decided that the structure of the cloak would be a main issue to contend with, as metamaterials are the enabling force behind these types of invisibility cloaks. However being able to create an entirely new structure that interacts with its environment in such a specific way would have required a project all of its own to accomplish. In light of this decision, it was decided to focus on the visuals, with the refraction model that we ended up using being tweaked to work with the theories provided by the research for this project. This may have ended up making it more difficult to manipulate the theory to fit the model, but it meant that what was produced is a good starting virtual simulation for this theory, rather than an example of a metamaterial in a virtual simulation.

As to be discussed in the next section, the lifecycle model decided upon was Feature Driven Development. This meant that a feature list needed to be created as part of the lifecycle. As part of the iterative analysis that was included in the lifecycle, this feature list changed over time. The feature list that was originally decided upon can be seen in the Design Specification (Appendix Something). The later feature list will be discussed in the Implementation section as this is where many of the changes to the feature list occurred. Reasons for these changes were mainly time constraints, such as the change from ray-tracing or photon mapping as a rendering method.

This was a beginning feature as the refraction model in WebGL is more of an approximation than a true-to-life model of it. This meant that it would probably have some effect on the implementation and would probably not produce a realistic result. The ray-tracing and photon mapping would have been a more realistic model and definitely was something the project was aiming towards. The problems with these methods is that they are not particularly viable for real-time rendering due to the slowness of the algorithms. While there are examples of real-time ray tracing in examples such as a demo of refraction in water using ray tracing [13], the algorithms used were extremely complicated to implement and would have required more time than was available to implement. This meant that the decision to achieve real-time rendering with the WebGL refraction model was more viable, though not as precise.

As can be gathered so far, a lot of this project was unknown at the beginning and unveiled as development and further research became necessary. This is another reason why iterative analysis was required. As some unknowns, such as the complexity of the ray-tracing algorithms and their slow performance, came to light, the project had to take different steps to what was decided upon previously. This was both a boon and detriment to the project. A boon as the project developed in a somewhat natural progression in accordance with the author’s skill and knowledge. It was a detriment in that many ideas had to be discarded that could have produced interesting results, such as what the outside world would look like from inside the cloak. While this has been estimated in the finished model, it is not a true representation of the idea of how it could be achieved in a true-to-life model.

## Research Method

The lifecycle model that was chosen for this project for Feature Driven Development (FDD). FDD is an agile methodology that was introduced in 1997 by Jeff de Luca and Peter Coad. It is highly adaptive and designed to produce tangible working progress and to track that progress at all stages of the development. It is typically designed for medium to large projects as the code reviews and quality reviews can be hard to achieve as a single developer. However the iterative development and the emphasis on working on features is what drew the author to this particular methodology.

The project contained a great deal of unknowns at the beginning of the project, due entirely to the lack of similar projects and systems available to compare and contrast with and also the complexity of the subject matter. The idea of using FDD was to help deal with these unknowns by structuring the produce into small steps that could be easily adapted depending on how the unknowns affecting the project. By also maintaining quality, as suggest by FDD, the code should be easily refactored for minor changes.

FDD also gave the author the idea of using a project blog to track progress. By updating the project blog and therefore explaining the current features completed and steps taken each week, the project would be traceable and therefore give a good measure of how well the project was progressing. Something that the author also added to track progress and to help with the structuring of the project was to include a design specification and a test specification as deliverable. These documents are normally more associated with the Waterfall model which is not an agile methodology, but plan-driven development. However the author felt that with the amount of unknowns, having deliverables that would enable a structuring of the research and ideas brought forth by that research would help them produce a better quality of project.

FDD have a 5 step model for project development with the last two steps being iterative. These are:

1. Develop an Overall Model
2. Build a features list
3. Plan by feature
4. Design by feature
5. Build by feature

Developing an overall model was included in the analysis stage of this project. By analyzing the project based on the background research undertaken, the author was able to decide on an overall direction the project could take. This was documented in the Outline Project Specification (Appendix Something). Another important section of the project that was developed at this point was what kind of research methods would be appropriate according to the research that was done.

Research methods fall into two categories: qualitative and quantitative. A mixture of these methods is normally required to produce robust results as well as solid proof for the hypotheses presented. Examples of qualitative methods are interviews and observations as these are bettering at finding out the ‘whys’ and ‘hows’ of hypotheses and tends to not provide results that must be explained in writing rather than with graphs and statistics. For this project, observational research seems appropriate as it is a visual project, and observations can answer a number of the questions proposed.

Qualitative research methods such as interviews and questionnaires would not be suitable for this type of project as they are more about finding people’s opinions and views on a piece of subject matter. If this project was about the ethics of invisibility then these two methods would be suitable to obtain a general sense of how people feel about invisibility in an ethical sense. This is why these two methods have not been considered for this project.

Quantitative research methods were difficult to decide upon as the background research had provided me with a great amount of detail on the theories and mathematical equations needed, but not how those equations might be represented in regards to visible light in terms other than the invisibility of the object contained in the cloak. A paper on invisibility cloaking with regards to ray-tracing [12] did produce a visual representation of how a ray might act within an invisibility cloak. Using this method to produce a similar path that curves around the object would provide a good quantitative measure on proving whether the theory does work in virtual space.

A method that would have been more preferable would have been a table of vectors or indicators of what the differences in vectors might be after they have been transformed by the cloak. This would have been easy to compare and contrast with in regards to the model. However, the background research did not provide such a source and this method was disregarded due to lack of comparison material available.

Another method that was used is using the plotted points on the traced ray and then determining whether the vector direction is correct in regards to what direction the beginning ray was travelling in. This fits in with invisibility requiring the light rays to travel out of the cloak in the same direction as they entered the cloak so they appear un-refracted. However as there are no indications in the papers the author researched that this should be the case, this could be misleading. This will be taken into account when using the method and then it’s weighting in the results.

# Design

## FDD Design

FDD design is designed to be done by feature. The process is normally such that you will design a feature, then implement it, and the cycle continues as such until all features are complete. This is the point when a project is considered feature complete.

The author adapted this process to include a complete design specification before implementation began. This was used to tie up all the author’s research on the project into a cohesive document that could be used as a basis for the rest of the project. This Design Specification can be found in Appendix E. This was especially useful for tracking progress as the initial feature list as well as the initial gathering of information into one document helped to gather the author’s thoughts and therefore continue on the project with less unknowns available to them.

The feature list included in the Design Specification was changed as the project progressed. This was mainly due to time constraints. The final feature list, after changes made due to the path of the project took, is included here:

1. Preliminary Research

a. Research into current theories surrounding invisibility cloaking, metamaterials, transformation optics and refraction.

2. Preliminary Learning of New Technologies.

a. This included Three.js, WebGL, and Ray-tracing (to determine constraints).

3. Outline Project Specification

a. This outlines the projects and any deliverables that are required from the project.

4. Prototype of Sphere Shell Structure

a. Implementation of a spherical transparent shell structure with differing refraction ratios.

6. Design Specification.

a. Discussion of feature list, model design and system design.

7. First Model – Spherical Invisibility Cloak (Solid and Shell Models)

a. First fully implemented model created based on design specification and early prototyping.

b. Includes the effects that were implemented. (Examples include chromatic effect and the multiple viewpoints).

8. Test Specification

a. Test specification detailing evaluations and tests to be used on current and future models.

9. Second Model – Cylindrical Invisibility Cloak (Solid and Shell Models)

a. Second fully implemented model created based on design specification and first model.

b. Includes the effects that were implemented.

9. Third Model – Conical Invisibility Cloak (Solid and Shell Models)

a. Third fully implemented model created based on design specification and first model.

b. Includes the effects that were implemented.

10. Viewpoint from within cloaks.

a. The ability to look from inside the cloak and what kind of images would be shown.

13. Project Diary

a. A blog that has tracked the progress of the project.

14. Final Report

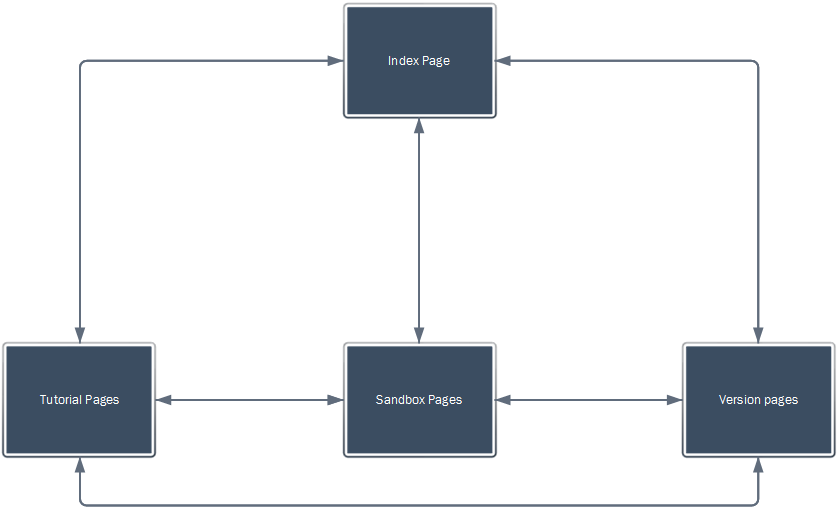
a. Report detailing project progression, completion and results.

As you can see from the differences between this list and the list in the design specification, there is little to no mention of ray-tracing and photon mapping. After initial attempts at prototyping a ray-tracing renderer, the decision was made that the progress being made on it was too slow for the time remaining. Instead, the models were designed using the Three.js refraction model as a base.

The next two sections describe the designs from both the specification view and the FDD designs that were made as the project progressed.

## System Design

The system designed was intended to be a simple hierarchy of web pages. It was designed this way as the web page design was not as important as the working models. The author had considered using a design pattern for the website and the project but after looking in the Design Patterns book by the Gang of Four[14], there were no design patterns that fitted the simplicity of the system. The webpages are designed to follow the MVC pattern and reflect that in their creation. The model is the JavaScript files that run the program, the view is the webpage displayed on the monitor and the controls are defined by a third-party library called OrbitControls.js, mentioned in Appendix A. The hierarchy of the webpages was designed as such:



This system is meant to enable easy access to any point on the site using links/navigation menu provided. The different sections are as follows:

**Index Page**

The main page of the site that is the first page the user will see. The user will also be able to access documentation from the project from the index page.

**Tutorial Pages**

This is where the tutorial work from the preliminary research and preparation will go. This will not be tested and is used to show what the initial research consisted of.

**Sandbox Pages**

This is where any sandbox work for the different versions will be stored. This will include non-final versions of the models chosen.

**Version Pages**

This is where each version that is a final prototype of a feature will be stored. This will include a first version that shows a spherical object made of glass as a geometrical model prototype, a second version showing the shell structure of the cloak and a third version which will be the final demo version of each of the models.

## Model Designs

The model designs discussed here will not include any details on the implementation involved and will discuss more about the research into what structures have been theorized for each of the geometric models. For the effects, it will be a more general discussion about the kind of effects that were hoped to be implemented. It will be mentioned in the next section if they were or were not able to be implemented.

For each model, there is two types of model to discuss. The two types are a model with a shell structure and a model with a solid structure. A shell structure expects more than two layers to the cloak and that the light will be guided using the layered design. The solid structure is to have a solid geometrical shape with a hollow centre. It works on the layered basis as well but expects that it will not be created in shells but as a solid single shell. They are not expected to work any differently, as the equations involved will be the same, but the structures are expected to appear differently, with the solid structure only showing certain points of the ray passing through, whereas the shell structure is expected to depict each stage.

### Spherical Model

The lecture that Sir John Pendry contained a diagram of a spherical shell that directed the rays around an inner shell. This is what gave the author the idea of creating a more solid structure that would, in the visualization, contain an outer and inner sphere surrounding an object. This was also the inspiration for having a solid structure model for each geometrical model. The reason for this is that a solid sphere with a hollow core would not allow for traversal through the sphere that would allow you to observe the different shells from inside.

The design for the solid sphere was this:



The design for the shell sphere was this:



### 

### Cylindrical Model

The cylindrical model is the most widely described. The cloak that was designed to work on radio waves [10] was a short cylinder only designed to work on the horizontal plane for the object. The paper which inspired the model the most was one describing the structure of a 3D cylindrical cloak made using silver-silica based metamaterials[11]. It contains a diagram showing how the layered structure would need to be arranged, which contained a shell structure with different refractive indexes. The author decided this would be a useful structure to work with and might work for other models as well.

The design for the solid cylinder was this:



The design for the shell cylinder was this:



### Conical Model

The idea for the conical cloak came from the paper that was the closest to this project. It discussed invisibility cloaking using ray tracing and had virtual models of many different structure. While the author had already found enough material on the spherical and cylindrical models, it did contain examples of ellipsoidal and conical models. The conical model appeared the most interesting out of the two, the top point being an interesting factor when considering manipulating light.

The design for the solid cone was:

The design for the shell cone was:

### 

### Effects

The effects that were discussed with the author’s supervisor were these:

* Chromatic/Prismatic
  + This was theorized to occur when the cloak had some imperfection that caused it to split the light as if it was a prism.
* Caustics
  + This is the name applied to the light scattering that happens on the surface of water when it ripples. If the cloak had had a flexible surface this may have been a viable effect, however the author decided it was not appropriate to the cloaks being designed.
* Multiple/Single Viewpoint(s)
  + The author decided it would be interesting to see what happened if you treated the cloak as if the light was only coming from the front half of the cloak but you could still see the back, how would the visualization handle that? Multiple viewpoints is what happens normally when the light comes from all angles, at this point you would expect invisibility no matter what angle you looked at it from.
* Centre Rays/Inner View
  + The central rays cannot physically travel all the way round in the same amount of time it would take to travel straight through the cloak. This is due to the constant speed of light. The author and their supervisor theorized that if you could use the central ray in some way to add light to the inner part of the cloak, the person inside a cloak might be able to see out of the cloak.

The only effects from these that were decided to be used was Chromatic, Single Viewpoint and Inner View. These were not designed before most of the implementation had occurred, as these relied on the implementation for the cloak being correct. These effects will be discussed more in the implementation section.

## 

# Implementation

## Tools

The tools used for creating the website and the project were relatively minimal. The version control was GitHub [15] which was mainly accessed via the desktop client, though there were a few cases where the Git shell was necessary as the checkout function was not accessible through the desktop client.

The editor used for the JavaScript files, which contained all the WebGL code for the models, as well as any editing needed for the webpages, to update links between the pages, was done in Atom.io editor[16]. Web development doesn’t tend to need an IDE so the author chose to use an editor that had a customizable colour scheme, to highlight the different parts of the files, and was able to work with all the different types of files involved with the project blog and the project webpages. It also was able to connect to the project’s GitHub repository and keep track of the changes being made.

There was a small program implemented in Java to test the different mathematical equations before placing them into the shader files. This was because the shader files do not have good debugging tools available, so being able to test any logic failures outside of the main program was useful throughout the project. It was a lot faster than testing the maths by hand and meant that any logical fallacies could be worked out relatively quickly. This was worked on in NetBeans IDE [17], which includes good support for GitHub repositories so any commits could be done in the IDE. It also kept track of any changes and has excellent debugging and code checking tools available.

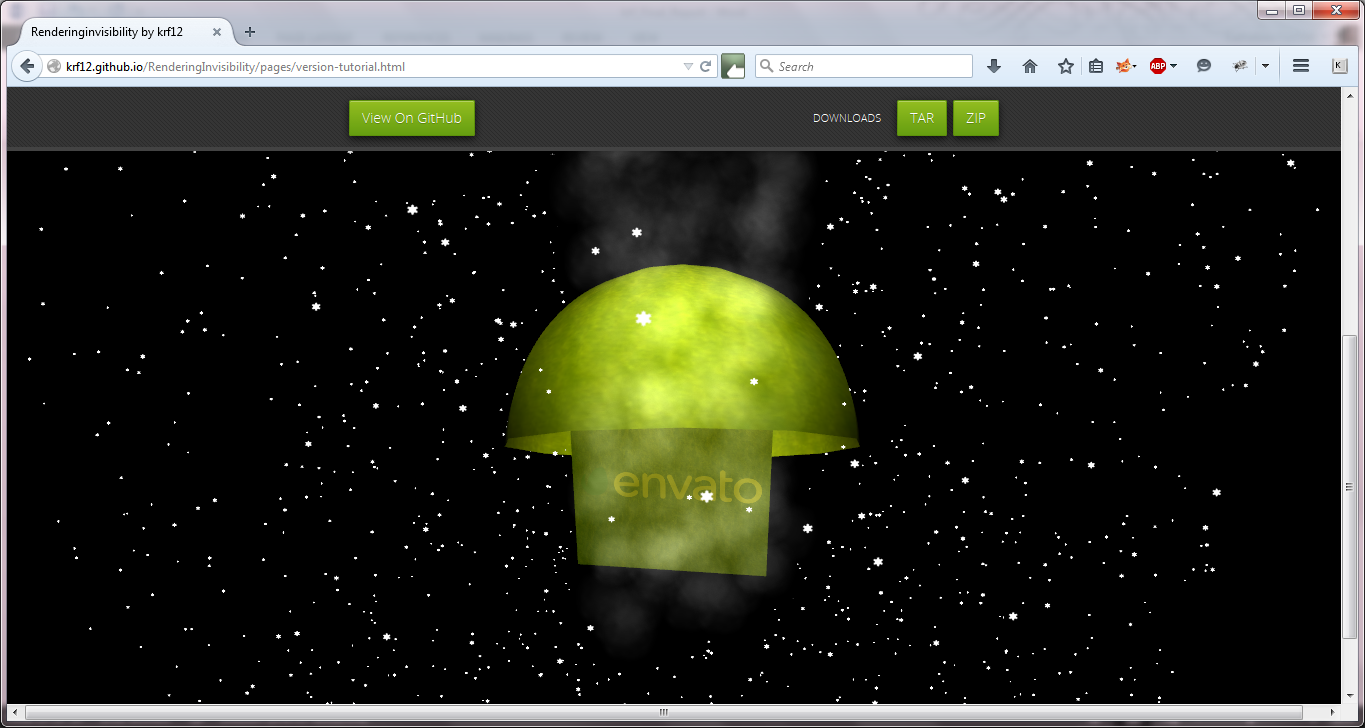
## Initial Implementation

The initial coding was based around improving the author’s proficiency with WebGL while also learning how to use the three.js library as well as setting up the two websites to be used during the project.

### Three.js

The author accomplished learning how to use three.js using a tutorial on Tuts+ [18] that explained how to set up an initial scene and how to use the various geometries associated with three.js.

The tutorial introduced the concepts of Geometries, Materials, Meshes and many other types of three.js objects. The author discovered that three.js was good at abstracting away a lot of the complexities involved in starting a WebGL program. This is definitely something that the author was looking for in using three.js because the program surrounding the model was not as important as the model itself and therefore being able to set up a working scene for the model quickly was essential for getting an early start on implementing the model.



This is the finished result of the tutorial. The different object displayed here are CubeGeometry, SphereGeometry (used to create the hemisphere) and ParticleGeometry (to create the snowstorm).

Three.js also contains many examples on its website detailing how to use the various geometries and a well maintained support form. The author of three.js, mrdoob, also spends a lot of time answering questions on Stack Overflow[19]. Having a good support system in place was definitely something the author needed for this project as there were many unknowns involved in creating the model from scratch and the author anticipated problems occurring.

### GitHub Pages

The author also set up the two GitHub Pages that would be used throughout the project. The project blog was set up at [krf12.github.io](http://krf12.github.io) while the project site was set up at [krf12.github.io/RenderingInvisibility](http://krf12.github.io/RenderingInvisibility). The project blog was made using the default theme and was not changed from the original style. The project blog posts were made using markdown, which was easy to use and produced professionally styled blog posts.

The project website was set up using a theme in the GitHub Pages that fit the type of look the author wanted for the project website. The html was then edited to include links to the outline project specification and to the three.js tutorial work that the author was working on. This was relatively simple as the author had previous experience using HTML5. The tutorial page was also edited to include a section that the JavaScript file would be loaded from, creating the canvas area that the model could be viewed in. The same structure was used throughout the project to ensure that the site conformed to the same style. The only pages that would be subject to change were the sandbox pages, which were treated more as test areas and did not include the themes associated with the rest of the site.

#### Problems with GitHub Pages

The author did discover relatively quickly into implementation, even during the implementation of the tutorial that the GitHub servers seemed to not be able to load WebGL quickly. This is still a continuing problem. This might be fixed with some performance management or using better servers to hold the project, but the creation of the model was deemed more important than performance and thus the GitHub servers are currently sufficient.

## Version One

The first version to implement was a structure that would be used for each subsequent model. The easiest way to do this was to find a skybox texture that could be used for the future models and create a glass sphere. These two parts would create a model to build upon.

### Skybox

The tutorial had not provided knowledge of how to create a skybox in three.js, so the first step was to look for a tutorial on how to implement a skybox and what kind of texture image was need to create ‘realistic’ surrounding. This was something that was important to the author as it was a virtual simulation of something that could feasibly work in real life. Using a texture that depicted real-life scenery would enable a more true-to-life test in the future.

The tutorial that the author found was on a blog by romano1la [20]. This blog post explained both how to create a skybox and how to map that onto an object. These two skills were both things that would be required for this first version and could be built on top of for future versions.

To create a skybox, a cubemap texture was needed. Luckily, there is a creative commons site that contains scenery from around the world in cubemap form, designed to be used for skyboxes. The only requirement to use the textures was to include a copy of the license [21] on pages using the photo. The specific skybox used was from Måskonåive [22] and depicted a mountainous scene, which seemed like an interesting background to use for an invisibility cloak. The relatively uniform division between sky and land would make it easier to see any refractions that were occurring wrongly and rectify them.

This texture was then applied to a large cube that served as the boundaries of a scene. The second skill applied more to the next step in the first version.

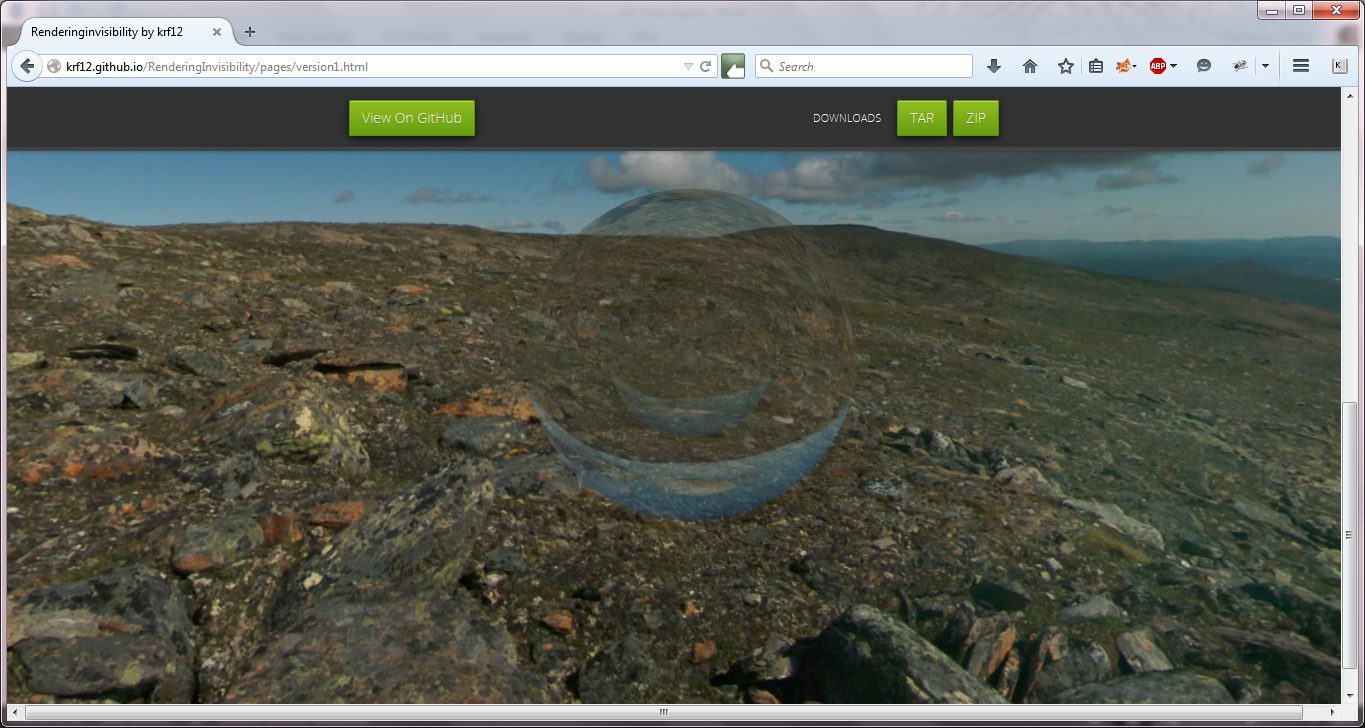
### Glass Spheres

As the spherical cloak was to be the first model on the feature list, this version was used to create a sphere that used refraction to appear like glass. This appeared to be a good test of the refraction model used by WebGL and three.js and is a good test of the knowledge gained from the tutorial.

A glass sphere is created in three.js by created a Mesh that has a SphereGeometry object, which describe the radius and number of segments that make up the surface of the sphere, and a Material object, which will be explained shortly, to describe the sphere. The segments in the geometry decide how well rendered the sphere will appear. Too few segments and the sphere will begin to look more like a multi-faceted object, Too many and you might be using too much power just to render a sphere. The normal amount to make a sphere appear completely spherical with no fragments on the surface is 64.

The Material is what describes how the object looks, whether it looks metallic, reflective etc. The way to create a refractive material that looks like glass is to use the cubemap you created for the skybox and create a refractive map out of it. This can easily be done in three.js using the CubeMapRefractionMapping method when loading the cubemap as a texture. This mean that when you use the cubemap in a Material object, by adding it as an envmap variable, and you specify that the object is refractive, it will apply that refraction to the map and then map the object with the cubemap that is respective to how refractive the object is described as. This is described using a refractive ratio between 0 and 1.

By combining all this properties, you can easily create a glass sphere, like the ones that were implemented for version one. This version was implemented to look like the solid structure for one half of the future models.



This is the finished result of version one. As you can see, there are two glass spheres, one inside the other. The refraction ratio of the spheres is the same as the refractive index for glass, as you can tell from the upside down image.

Having implemented the refraction in this manner, there were already problems emerging based on what the author’s research was suggesting. The refraction index could only between 0 and 1, which did correspond with the refractive indexes shown in the cylindrical cloak structure in [11], but as at this point it was not clear whether or not these were refractive indexes, and therefore yet to have the ratio between them worked out, as this is the number that WebGL uses to work out the refraction, or that these were those values were the ratios between the shells.

If the latter was correct, the ratios were not a problem. If the former was correct, then the refractive method being used here would not be adequate for the model being used. This would not be determined until the sandbox stage, as the next implementation was the shell structure, according to the feature list, not the sandboxing or the theoretical physics.

## Version Two

The second version to implement was also the next feature on the feature list to complete; the prototype of the shell structure. The shell/layered structure was proposed in the cylindrical cloak design in [11]. In this version, the problem of how to affect one side of the model over the other was attempted to be tackled. As the refraction needs to change from one half to the other half, if you’re looking at it from a single viewpoint and following a single ray or set of rays from a similar direction, there needs to be a way to texture each side differently.

For this version, there was an attempt at creating hemispheres in three.js and then placing them so they would appear spherical. These would then be textured differently. An explanation on why this seemed to be the correct course of action will follow.

### Shell Structure

Following on from the last version, the shell structure was created using multiple spheres. In the early version of this, it was made simply using a for statement that increased the shells radius each iteration as well as decreasing the refraction ratio each time to replicate what was expected in the invisibility cloak.

This shell structure was still made of glass spheres and not implementing the theory presented in the papers read during research. However it showed the difference in refraction ratio easily and meant it was easy to see that the iterative method was working. There was however a realization at this point. If a ray was to travel all the way through, the ratio would change halfway. If the project was aiming to show what it might look like if we were able to follow a ray through, then the half way point on each shell should have a different material with a different refraction ratio.

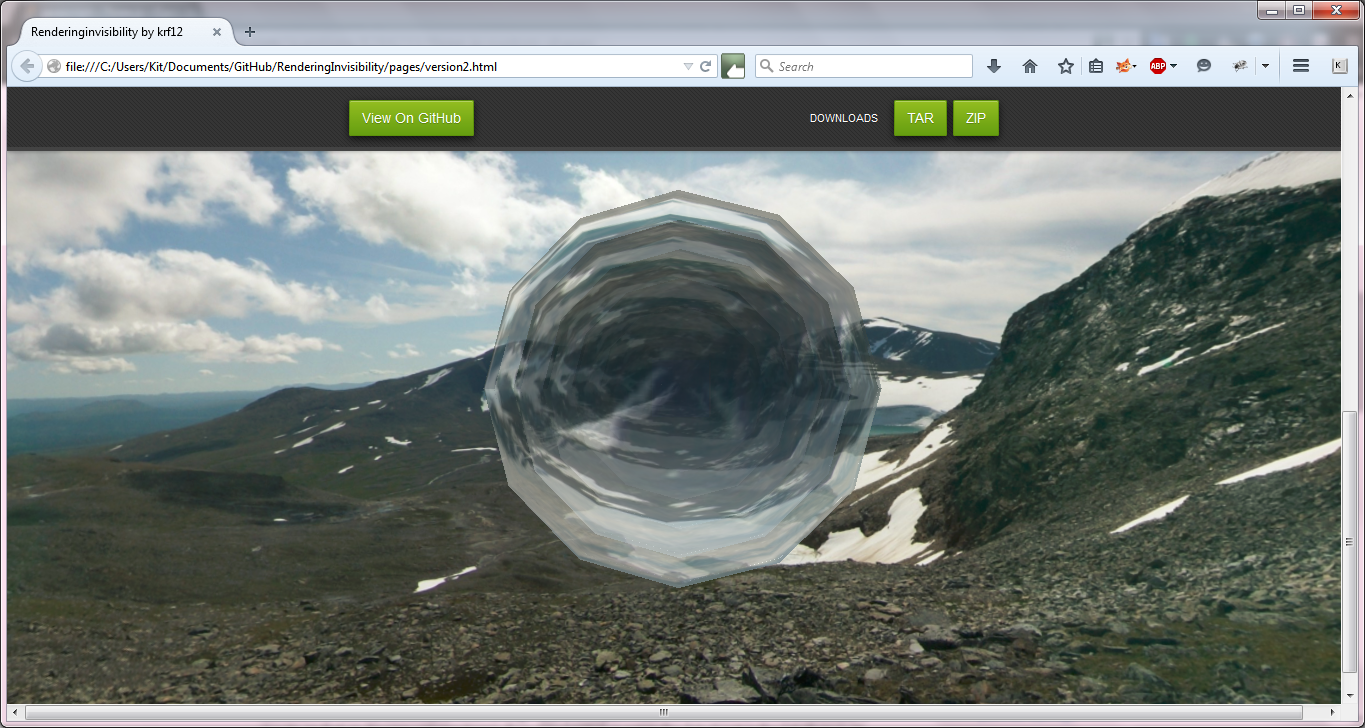
After researching how to fix the problem, the author discovered three ways around the problem. The first way was to apply two materials as suggested by a Stack Overflow answer [23]. However as the answer states, there was some distortion. This would disrupt the look of the invisibility cloak. The second was to use two hemispheres and to place them in the correct position to look like a complete sphere. This will be talked about in the next section. The third way was to use a ShaderMaterial instead and use the surface normals to restrict the textures to the correct halves of the sphere. This ended up being the method used, but was not included in this version. It was used in the sandboxing section after problems arose with the hemispheres.

### Hemispheres

As suggested above, the version two ended up using hemispheres to display different refraction ratios on either half of the each shell layer. This seemed like a simple task but ended up being extremely difficult to accomplish in a satisfactory manner.

The first way suggested by another Stack overflow answer [24] and only works well when there is one hemisphere. It presents a hollow hemisphere, and can have another hemisphere of smaller size contained in size and still look good. However, placing another hemisphere on the other side has an odd effect. If you at if from the side you expect to be convex, it is instead concave. Having tried multiple inputs to try to fix this, the author accepted that this method was not going to solve the problem.

The next method was also proposed by a Stack Overflow answer [25] and included a method to create a hemisphere using a LatheGeometry. This involved specify the radius and using the equation proposed by the answerer to create an array of point for the LatheGeometry. This seemed like it would solve the problem at hand and was implemented into the second project. It is still the solution in the current version two but as can be seen from the screenshot below, it does not produce a smooth hemisphere. No matter how small you make the angle detail value, meant to increase the smoothness of the hemisphere, it still appears more like a multi-faceted geometrical shape.



As can be seen from the screenshot, the sphere is not smooth at all. This produces very poor shells and ruins performance as well.

There is also the other problem with creating the hemispheres this way. The performance drops dramatically, to the point where it cannot even load in a reasonable amount of time on the GitHub page. The author has kept this as a record of the attempts to make a structure this way to show the performance issues and because the third way of creating the hemispheres, with ShaderMaterial, would require a large amount of refactoring, which would be easier to accomplish in the next stage of the project.

## Sandboxing

This stage was when the project began to incorporate the mathematical formulas and cloak designs discussed in the research. It is also where the ShaderMaterials began to be used which meant the beginning part of this stage was to refactor the code and create shader files. The shader files were all based on a Fresnel Shader by alteredq, which is included in Appendix A as a code sample, but has been heavily edited in every instance of the models as it was mostly used to ascertain how to write a three.js shader and also to see a way of creating the chromatic effect used to create the bubbles in the Fresnel Shader as this could be useful in recreating such an effect for the invisibility cloak.

### Initial Sandboxing

The first thing that needed doing in the sandbox stage was to attempt to create the hemispheres using ShaderMaterials. The author decided to work on the solid structure and used the code for the first version as a basis for the refactoring. The ShaderMaterials require a shader to be written describing how the object is to be rendered. There are a number of standard shaders contained in ShaderUtils.js in the three.js library but the shader for this needed to be something that could be edited by the author as it would also contain the mathematical equations.

This was accomplished by looking at the three.js examples and using the Fresnel shader example to base the shaders for this project on. This particular shader was used as it showed how to use texture mapping from a cubemap in the shaders, what the previous materials had already been doing to recreate refraction, and also how to use the refract method in GLSL to do this texture mapping so that the material would make the object appear refracted. This was also a good shader to look at as the Fresenel shader was used to create a chromatic effect that made glass spheres appear as bubbles. This effect was similar to the effect that the author wished to recreate for the later effects, though it was particularly muted here and the author wished to change it to be a more extreme effect so as to show a more imperfect invisibility.

### Shaders and Models

The solid structure was easy enough to adapt to use the ShaderMaterials, though it was at this point that the author realized that each layer would require a different shader as passing variables to the shader to distinguish between the layers would not work and each layer only appeared as the first layer, therefore the first shader loaded. This was not a problem for the solid structure as it only required two shaders. The shell structure however would require ten shaders which would create a lot of work for refactoring and maintenance. However no solution to this problem seemed available and as such was the same problem at the end of the implementation as well as at this point.

While the shaders had not been completely finished at this point and the feature list said that the next feature was the prototype of the spherical invisibility cloak, the author decided that considering the amount of shader required for each model, it might be easier at this point, while refactoring was a minimum to create other geometrical models. The cylindrical model was the first to be finished and was created using the CylinderGeometry which was very similar to the SphereGeometry but included a height value as well. The conical cloak was not added at this point but created a little later into the project when the author had decided that the conical cloak was going to be the other model. It was both based on the ray-tracing paper’s conical cloak[12], as stated in the earlier design section, and that the cone was built using the CylinderGeometry as well, which made refactoring the code to create the cone model was a lot easier as it did not require working out how to use another geometry in three.js, which was particularly time-consuming if the geometry was not well explained in its use, which was a problem that occurred when using the LatheGeometry earlier.

The next step after creating the base models and the shaders needed for the shell and solid structures was to update the refraction to be more realistic and closer to what an invisibility cloak required to be good virtual representation.

### Updated Refraction and Refraction.java

It was at this point that the maths in the papers needed to be examined to decide what parts were related to the structure, and therefore beyond the means of this project, and what was required for the refraction. Most of what was required for the refraction was found in the last papers the author looked at, the paper on ray-tracing[12] and the paper on the cylindrical cloak design[11]. The author suspected that the reason for this was that the metamaterials were the more interesting points in the other papers and therefore were more heavily featured and discussed. In these two papers, while focusing on the cloak designs and therefore structures, it also focused on the fundamentals behind the cloak. This included the path that light would need to travel through the structure.

Before using the maths in the program, the author wanted to ensure that the maths being used would produce results that seemed reasonable. The assumption that the author made was that the vectors would decrease as they moved towards the centre and then increase as they moved away and returned to something similar to the starting vector. This assumption was made because of the structure described in the paper on the cylindrical cloak design[11]. After re-visiting the papers and re-reading them, the author realized that the design of the shells contained details on their refractive indexes, not the ratios between each shell. This was a basis for the above assumptions because, as shown by the paper, the path of the ray would travel through each shell on its path at most twice. The ratios would be reversed half way through and as such the reverse refraction would be applied to the path of the ray. Hence the assumption that the vectors would go from a beginning point, and then back to it, or at least something similar in the same direction.

This was an assumption and the reason for the need for assumptions was because the papers were heavy on the steps taken to reach a final formula, but not the vectors/numbers being inputted and outputted from the formulas. This meant that by using the formulas, the assumption was that whatever came out should be correct and produce the correct effect: invisibility. But as stated, this needed testing to ensure that the maths produced something that seemed usable, so no NaNs or zero vectors. The way the author went about this was by writing a small Java program that would run through the iterations of one path that would go through all shells. This meant ten iterations inwards and ten iterations out. The reason for writing this in Java was the author’s experience in using Java. By writing the program in a language that they were highly familiar with, they could write the program in a minimal amount of time and then edit it as new information came up easily. The program contained a third party file, included in Appendix A, that describes a Vector object, not the java.utils.Vector object used by Java, that contains method for dot product, normalizing etc. These methods are extremely useful in the implementation of the maths required.

The formulas used in the implementation are contained in Appendix B as well as Appendix C. The formula in Appendix B is used for the first and final iterations to work out the refraction between the air and the cloak. The code for the refract method used by OpenGL provided that formula for refraction as it was the same as the vector form of Snell’s Law, which seemed the most likely formula to use as the ones suggested by the papers required the use of tensors. Tensors required knowledge of the materials being used in the structure which wasn’t what the author was aiming for.

The next step after confirming that the maths was working was to add it to the shaders. This was easier for the solid structures due to the smaller amount of shaders required for it. This was another stage where it showed that the amount of shaders in the shell structure was a problem but with how GLSL works, at a very low-level with no easy way to move variables between different instances of different shaders, this was a problem that was going to be encountered constantly and just meant more time was set aside for any work on the shell structures.

The maths was implemented the same in each shader, using the normal the three.js shaders additions provided. This did present a problem that may have caused inaccuracies in the maths. As the normal were not carried from shader to shader, as there was no way of doing this, the normal vectors were almost definitely calculated as different vectors at each layer. However, as the spheres were centred on the same origin point and only have different radiuses, the author theorized that the normals would have the same direction as the bigger sphere, producing extremely similar results that any inaccuracies might be too small to cause problems.

The implementation first only contained the vector form of Snell’s Law, applied in code form by OpenGL as mentioned before. This was changed from GLSL to Java quite easily and then placed in an iterative loop. At first the vectors appeared to not be following the expected pattern. This was rectified by using the air to cloak refraction as well, which appeared to fix the problem by making the vectors correct to start with. The vectors used to test this implementation where the position of the light in the scene implementation (1000, 1000, 1000) and a surface normal that had been worked out using the formula found in Appendix B.

After assuring that the vectors seemed to working correctly, the code was placed in the shaders. The author found that it was easier to not use a for-loop as debugging the shaders was not an easy task. The only good way to debug a shader is using the fragment shader to output colours for if statements. If the for-loop had gone wrong, it would be harder to pinpoint the step it had gone wrong at. By spacing the statements out, they could debug any of the vectors to see if any outputted a zero vector. The vectors were then split between the beginning and end vectors for the outer shell and the middle vectors were split between the beginning and end of the middle vectors for the inner shell for the solid structure. For the shell, the vectors were chosen for whatever shell layer they were at currently. This all occurred in the vertex shader.

In the fragment shader, the beginning vectors were applied to the front of the sphere and the end vectors were applied to the back of the sphere. This was accomplished by only applying the end vectors as the fragment colour when the normal z co-ordinates were less than 0. These were then used in conjunction with the cubemap and the texturecube method to work out the color to be used at the current vertex.

At first this seemed to not be working, the front half of the outer sphere was appearing as expected, an invisible sphere. The inner sphere appeared to be a further refraction on the outer sphere, appearing more distorted but maintaining the invisible effect by showing what lay through the sphere, and therefore, what the author expected at that stage. The back half however was appearing extremely solid. It had the convex look to it that showed that the cloak was there. This was extremely troubling and did not appear to have a solution. The effect was improved after removing the normalization of vectors after the first incident vector, as Snell’s Law states that the vectors that are produced are normalized refraction vectors when using the vector form of Snell’s Law. However it remained convex.

After looking back at the maths and the research that had been done previously, the author came to the conclusion that this was occurring because instead of taking into account all the rays entering the sphere, it was instead following rays coming from the front of the sphere and exiting through the back of the sphere. This was what was causing the strange effect because this would not occur in the actual cloak. The author decided this was a very interesting effect and while it would not be one that would occur in reality, as light would always be travelling from all angles, it was something to consider keeping as an interesting visualization.

This meant that the cloak needed to take into account multiple viewpoints and rays of light. The author decided the best way to visualize this was to only use the beginning refraction vector calculations on the outer sphere. As the rays would be coming in from all directions, this is the maths that would be applied on entrance. This produced the effect of a perfect invisibility cloak that only appeared slightly off. The author suspected this was something that was due to the cloak being rendered on top of the skybox and therefore the renderer had attempted to distinguish it somehow from the rest of the scene. This was something the author had no control over and therefore left it in.

The next step was to add the effect shaders into each model. The single viewpoint shaders were already part of every sandbox model. To add to this, the author implemented the multiple viewpoint shaders, using the method described in the previous paragraph. The chromatics versions of all these shaders were added as well. The chromatic shaders were created by combining the refraction methods in each shader with the Fresnel effect achieved in the shader these were based on. This was achieved by instead of using one of the refraction vectors, three refractions vectors were used and in the fragment shaders these were applied the red, green and blue values of the fragment colour. This was a good visualization of what might occur if the cloak was imperfect and the rays were split through the layer, though as mentioned before in the research and design sections, this was an assumption based on how light can be affected by structures it passes through.

### Ray Path-tracing

## Final Versions

### Spherical Model

### Cylindrical Model

### Conical Model

# Testing and Results

## Unit Testing

-Testing of models using manual testing

### Tutorial

### Version One

### Version Two

### Solid Spherical Model

### Shell Spherical Model

### Solid Cylindrical Model

### Shell Cylindrical Model

### Solid Conical Model

### Shell Conical Model

## System Testing

-Testing of project website system

## Experiments

* **Is it possible to render an invisibility cloak, which functions using the theories presented, in a virtual simulation?**
* Are there any problems with representing an invisibility cloak in a virtual simulation? What causes these problem? Can they be fixed in the virtual simulation?
* Any there are strengths in representing this invisibility cloak in a virtual simulation? Why are these strengths occurring? Can these strength be used when creating the cloak in reality?
* What does the virtual simulation show about the theories presented? Does it confirm them? Does it reveal any new information?
* What effects are created in the virtual simulation, through imperfections or otherwise, that could occur in a cloak produced in reality?
* -Testing possibility of simulation
* -Strength of simulation
* -Weaknesses of simulation
* -Effects created, their implementation and implications in real world

## Results

* What results I obtained from testing and experiments. Is the system robust, is it functional, what answers I achieved to the questions.

# Conclusions

## Overall Conclusion

-Conclusion drawn from testing.

## Issues Encountered

-Issues in design, implementation, testing and overall project.

# Critical Evaluation

## Research

-Was my research thorough enough, did I take into account all possibilities, what could I have also researched and how might that have affected my concluions and results.

## Design

Was FDD the right method for design, did I not do enough preliminary design and could I have done more. What might this have done for the project?

## Implementation

Was the implementation solid, robust? Do I feel I did enough? What more could I have done? Did any constriants add to my feelings on my implementation?

## Testing

Was my testing thorough enough? If it was, why, if not what could have made it more through? What other testing was there available and how might that have affected my project?

## Overall Evaluation

How well did I do? What would I do differently given the chance? What problems did I encounter that were detrimental, what was good, what was bad?

# Appendices

* 1. Third-Party Code and Libraries

### Three.js

### OrbitControls.js

### KeyboardState.js

### Github Pages

### Fresnel Shader

At the start of this project, this shader was not included in the three.js library as standard and was instead included in a three.js example. As of now it is a part of ShaderUtils.js in the three.js library. This code is being included as it was the basis for all the shaders used and while the shaders do not resemble to shader greatly, it is acknowledged as the basis for them and as such should be included.

|  |
| --- |
| /\* ------------------------------------------------------------------------- |
| //      Fresnel shader |
| //      - based on Nvidia Cg tutorial |
| ------------------------------------------------------------------------- \*/ |
|  |
| 'fresnel': { |
|  |
| uniforms: { |
|  |
| "mRefractionRatio": { type: "f", value: 1.02 }, |
| "mFresnelBias": { type: "f", value: 0.1 }, |
| "mFresnelPower": { type: "f", value: 2.0 }, |
| "mFresnelScale": { type: "f", value: 1.0 }, |
| "tCube": { type: "t", value: null } |
|  |
| }, |
|  |
| fragmentShader: [ |
|  |
| "uniform samplerCube tCube;", |
|  |
| "varying vec3 vReflect;", |
| "varying vec3 vRefract[3];", |
| "varying float vReflectionFactor;", |
|  |
| "void main() {", |
|  |
| "vec4 reflectedColor = textureCube( tCube, vec3( -vReflect.x, vReflect.yz ) );", |
| "vec4 refractedColor = vec4( 1.0, 1.0, 1.0, 1.0 );", |
|  |
| "refractedColor.r = textureCube( tCube, vec3( -vRefract[0].x, vRefract[0].yz ) ).r;", |
| "refractedColor.g = textureCube( tCube, vec3( -vRefract[1].x, vRefract[1].yz ) ).g;", |
| "refractedColor.b = textureCube( tCube, vec3( -vRefract[2].x, vRefract[2].yz ) ).b;", |
| "refractedColor.a = 1.0;", |
|  |
| "gl\_FragColor = mix( refractedColor, reflectedColor, clamp( vReflectionFactor, 0.0, 1.0 ) );", |
|  |
| "}" |
|  |
| ].join("\n"), |
|  |
| vertexShader: [ |
|  |
| "uniform float mRefractionRatio;", |
| "uniform float mFresnelBias;", |
| "uniform float mFresnelScale;", |
| "uniform float mFresnelPower;", |
|  |
| "varying vec3 vReflect;", |
| "varying vec3 vRefract[3];", |
| "varying float vReflectionFactor;", |
|  |
| "void main() {", |
|  |
| "vec4 mvPosition = modelViewMatrix \* vec4( position, 1.0 );", |
| "vec4 mPosition = modelMatrix \* vec4( position, 1.0 );", |
|  |
| "vec3 nWorld = normalize( mat3( modelMatrix[0].xyz, modelMatrix[1].xyz, modelMatrix[2].xyz ) \* normal );", |
|  |
| "vec3 I = mPosition.xyz - cameraPosition;", |
|  |
| "vReflect = reflect( I, nWorld );", |
| "vRefract[0] = refract( normalize( I ), nWorld, mRefractionRatio );", |
| "vRefract[1] = refract( normalize( I ), nWorld, mRefractionRatio \* 0.99 );", |
| "vRefract[2] = refract( normalize( I ), nWorld, mRefractionRatio \* 0.98 );", |
| "vReflectionFactor = mFresnelBias + mFresnelScale \* pow( 1.0 + dot( normalize( I ), nWorld ), mFresnelPower );", |
|  |
| "gl\_Position = projectionMatrix \* mvPosition;", |
|  |
| "}" |
|  |
| ].join("\n") |
|  |
| }, Vector.java Used in the Refraction Java program to implement the vector methods needed to test the maths. A minor change was made in the program to allow for getting out the vector data after it had been added to the Vector, but this is the original code used.  */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *\* Compilation: javac Vector.java*  *\* Execution: java Vector*  *\**  *\* Implementation of a vector of real numbers.*  *\**  *\* This class is implemented to be immutable: once the client program*  *\* initialize a Vector, it cannot change any of its fields*  *\* (N or data[i]) either directly or indirectly. Immutability is a*  *\* very desirable feature of a data type.*  *\**  *\**  *\* % java Vector*  *\* x = (1.0, 2.0, 3.0, 4.0)*  *\* y = (5.0, 2.0, 4.0, 1.0)*  *\* x + y = (6.0, 4.0, 7.0, 5.0)*  *\* 10x = (10.0, 20.0, 30.0, 40.0)*  *\* |x| = 5.477225575051661*  *\** <x, y> *= 25.0*  *\* |x - y| = 5.0990195135927845*  *\**  *\* Note that java.util.Vector is an unrelated Java library class.*  *\**  *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*  public class Vector {  private final int N; *// length of the vector*  private double[] data; *// array of vector's components*  *// create the zero vector of length N*  public **Vector**(int N) {  this.N = N;  this.data = new double[N];  }  *// create a vector from an array*  public **Vector**(double[] data) {  N = data.length;  *// defensive copy so that client can't alter our copy of data[]*  this.data = new double[N];  for (int i = 0; i < N; i++)  this.data[i] = data[i];  }  *// create a vector from either an array or a vararg list*  *// this constructor uses Java's vararg syntax to support*  *// a constructor that takes a variable number of arguments, such as*  *// Vector x = new Vector(1.0, 2.0, 3.0, 4.0);*  *// Vector y = new Vector(5.0, 2.0, 4.0, 1.0);*  */\**  *public Vector(double... data) {*  *N = data.length;*  *// defensive copy so that client can't alter our copy of data[]*  *this.data = new double[N];*  *for (int i = 0; i < N; i++)*  *this.data[i] = data[i];*  *}*  *\*/*  *// return the length of the vector*  public int **length**() {  return N;  }  *// return the inner product of this Vector a and b*  public double **dot**(Vector that) {  if (this.N != that.N) throw new **RuntimeException**("Dimensions don't agree");  double sum = 0.0;  for (int i = 0; i < N; i++)  sum = sum + (this.data[i] \* that.data[i]);  return sum;  }  *// return the Euclidean norm of this Vector*  public double **magnitude**() {  return Math.**sqrt**(this.**dot**(this));  }  *// return the Euclidean distance between this and that*  public double **distanceTo**(Vector that) {  if (this.N != that.N) throw new **RuntimeException**("Dimensions don't agree");  return this.**minus**(that).**magnitude**();  }  *// return this + that*  public Vector **plus**(Vector that) {  if (this.N != that.N) throw new **RuntimeException**("Dimensions don't agree");  Vector c = new **Vector**(N);  for (int i = 0; i < N; i++)  c.data[i] = this.data[i] + that.data[i];  return c;  }  *// return this - that*  public Vector **minus**(Vector that) {  if (this.N != that.N) throw new **RuntimeException**("Dimensions don't agree");  Vector c = new **Vector**(N);  for (int i = 0; i < N; i++)  c.data[i] = this.data[i] - that.data[i];  return c;  }  *// return the corresponding coordinate*  public double **cartesian**(int i) {  return data[i];  }  *// create and return a new object whose value is (this \* factor)*  public Vector **times**(double factor) {  Vector c = new **Vector**(N);  for (int i = 0; i < N; i++)  c.data[i] = factor \* data[i];  return c;  }  *// return the corresponding unit vector*  public Vector **direction**() {  if (this.**magnitude**() == 0.0) throw new **RuntimeException**("Zero-vector has no direction");  return this.**times**(1.0 / this.**magnitude**());  }  *// return a string representation of the vector*  public String **toString**() {  String s = "(";  for (int i = 0; i < N; i++) {  s += data[i];  if (i < N-1) s+= ", ";  }  return s + ")";  }  *// test client*  public static void **main**(String[] args) {  double[] xdata = { 1.0, 2.0, 3.0, 4.0 };  double[] ydata = { 5.0, 2.0, 4.0, 1.0 };  Vector x = new **Vector**(xdata);  Vector y = new **Vector**(ydata);  System.out.**println**("x = " + x);  System.out.**println**("y = " + y);  System.out.**println**("x + y = " + x.**plus**(y));  System.out.**println**("10x = " + x.**times**(10.0));  System.out.**println**("|x| = " + x.**magnitude**());  System.out.**println**("<x, y> = " + x.**dot**(y));  System.out.**println**("|x - y| = " + x.**minus**(y).**magnitude**());  }  }  *Copyright © 2000–2010, Robert Sedgewick and Kevin Wayne.* |

* 1. Mathematical Formulas

### Snell’s Law

\frac{\sin\theta_1}{\sin\theta_2} = \frac{v_1}{v_2} = \frac{n_2}{n_1}

Source: Wikipedia

### Snell’s Law – Vector Form

 r = n_1 / n_2 

 c = -\mathbf{n}\cdot \mathbf{l}

\mathbf{v}_{\mathrm{refract}} = r \mathbf{l} + \left( r c - \sqrt{1 - r^2 \left( 1 - c^2 \right)} \right) \mathbf{n}

Source: Wikipedia

### Fermat’s Principle – Optical Path Length

S=\int_{\mathbf{A}}^{\mathbf{B}} n\, ds\ 

Source: Wikipedia

### Maxwell’s Equations

#### Gauss’ Law

#### \nabla \cdot \mathbf{E} = \frac {\rho} {\varepsilon_0}

Source: Wikipedia

#### Gauss’ Law for Magnetism

#### \nabla \cdot \mathbf{B} = 0

Source: Wikipedia

#### Faraday’s Law

\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}} {\partial t}

Source: Wikipedia

#### Ampere’s Law (with Maxwell’s addition)

#### \nabla \times \mathbf{B} = \mu_0\left(\mathbf{J} + \varepsilon_0 \frac{\partial \mathbf{E}} {\partial t} \right)

Source: Wikipedia

### Formula for Air-to-Cloak refraction [12]

k2 = k1 + *λ*n

*λ*=−k1 · n± (k1 · n*)*2 +1−*k*21

k2 – Refraction Vector

k1 – Incident Vector

n – Surface Normal

### Formula for the Surface Normal of a Sphere

x=r \, \sin\theta \, \cos\varphi

y=r \, \sin\theta \, \sin\varphi

z=r \, \cos\theta

Source : Wikipedia

* 1. Code Samples

### Refract Method [4] – Cg

half4 main(float2 bumpUV : TEXCOORD0,

float4 screenPos : TEXCOORD1

uniform sampler2D tex0,

uniform sampler2D tex1,

uniform float4 vScale) : COLOR

{

// fetch bump texture

half4 bumpTex=2.0 \* tex2D(tex0, bumpUV.xy) - 1.0;

// compute projected texture coordinates

half2 vProj = (screenPos.xy/screenPos.w);

// fetch refraction map

half4 vRefrA = tex2D(tex1, vProj.xy + bumpTex.xy \* vScale.xy);

half4 vRefrB = tex2D(tex1, vProj.xy);

return vRefrB \* vRefrA.w + vRefrA \* (1 - vRefrA.w);

}

### Refract Method[26] – GLSL

genType refract( genType I, genType N, float eta);

k = 1.0 - eta \* eta \* (1.0 - dot(N, I) \* dot(N, I));

if (k < 0.0)

R = genType(0.0); // or genDType(0.0)

else

R = eta \* I - (eta \* dot(N, I) + sqrt(k)) \* N;

* 1. Outline Project Specification
  2. Design Specification
  3. Test Specfication

# Annotated Bibliography

[1], [2], [5], [7], [10]–[12], [18], [27]–[29]

[1] J. S. Choi and J. C. Howell, “Paraxial ray optics cloaking.,” *Opt. Express*, vol. 22, no. 24, pp. 29465–78, Dec. 2014.

[2] T. Ebbesen, S. Hell, J. B. Pendry, and A. Brown, “2014 Kavli Prize in Nanoscience: A Discussion with the Laureates,” 2014. [Online]. Available: http://www.kavliprize.org/events-and-features/2014-kavli-prize-nanoscience-discussion-laureates. [Accessed: 26-Apr-2015].

[3] Wikipedia, “Snell’s\_law,” 2015.

[4] T. Sousa and NVIDIA, “Generic Refraction Simulation,” in *GPU Gems 2*, Second., M. Pharr and R. Fernando, Eds. Pearson Education Inc., 2005.

[5] U. Leonhardt and T. G. Philbin, “Chapter 2 Transformation Optics and the Geometry of Light,” *Prog. Opt.*, vol. 53, pp. 69–152, May 2009.

[6] F. E. Wagner, S. Haslbeck, L. Stievano, S. Calogero, Q. A. Pankhurst, and K. P. Martinek, “Before striking gold in gold-ruby glass.,” *Nature*, vol. 407, no. 6805, pp. 691–692, 2000.

[7] J. B. Pendry, *Metamaterials and the Science of Invisibility*. London, UK: Youtube, 2013.

[8] J. B. Pendry, “Negative Refraction Makes a Perfect Lens,” *Phys. Rev. Lett.*, vol. 85, no. 18, pp. 3966–3969, Oct. 2000.

[9] J. Wang, Y. X. H. Chen, and B. Zhang, “Ultraviolet dielectric hyperlens with layered graphene and boron nitride,” p. 20, May 2012.

[10] D. Schurig, J. J. Mock, B. J. Justice, S. a Cummer, J. B. Pendry, a F. Starr, and D. R. Smith, “Metamaterial electromagnetic cloak at microwave frequencies.,” *Science*, vol. 314, no. 5801, pp. 977–980, 2006.

[11] K. S. Elassy, N. H. Rafat, M. E. Khedr, and M. H. Aly, “Fundamentals of designing cylindrical high-order transformation optics invisibility cloaks using silver–silica metamaterials,” *Appl. Phys. A*, vol. 115, no. 2, pp. 531–539, Nov. 2013.

[12] M. M. Crosskey, A. T. Nixon, L. M. Schick, and G. Kovačič, “Invisibility cloaking via non-smooth transformation optics and ray tracing,” *Phys. Lett. A*, vol. 375, no. 18, pp. 1903–1911, May 2011.

[13] E. Wallace, “WebGL Water,” 2011. [Online]. Available: http://madebyevan.com/webgl-water/. [Accessed: 28-Apr-2015].

[14] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*, vol. 206. 1995.

[15] C. W. Tom Preston-Werner PJ Hyett, “GitHub,” *2008*, 2013. [Online]. Available: https://github.com.

[16] Atom.io, “Atom.io.” 2015.

[17] Oracle, “Netbeans.” 2015.

[18] M. Sopylo, “WebGL With Three.js: Basics - Tuts+ Code Tutorial,” 2013. [Online]. Available: http://code.tutsplus.com/tutorials/webgl-with-threejs-basics--net-35688. [Accessed: 03-Feb-2015].

[19] Stack Exchange, “Stack Overflow,” 2013. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Stack\_Overflow&oldid=571742484. [Accessed: 02-May-2015].

[20] romano1la, “JavaScript — Skybox and environment map in Three.js,” 2014. [Online]. Available: http://blog.romanliutikov.com/post/58705840698/skybox-and-environment-map-in-three-js. [Accessed: 02-May-2015].

[21] Creative Commons, “Creative Commons — Attribution 3.0 Unported — CC BY 3.0,” 2012. [Online]. Available: http://creativecommons.org/licenses/by/3.0/. [Accessed: 02-May-2015].

[22] Humus, “Måskonåive,” 2010. [Online]. Available: http://www.humus.name/index.php?page=Textures&ID=77. [Accessed: 02-May-2015].

[23] L. Stemkoski, “javascript - three.js - What’s the best way to put multiple textures/images on a single Sphere? - Stack Overflow,” 2013. [Online]. Available: http://stackoverflow.com/questions/18305318/three-js-whats-the-best-way-to-put-multiple-textures-images-on-a-single-spher. [Accessed: 03-May-2015].

[24] WestLangley, “Create a concave half sphere with three.js - Stack Overflow,” 2012. [Online]. Available: http://stackoverflow.com/questions/12425014/create-a-concave-half-sphere-with-three-js. [Accessed: 03-May-2015].

[25] G. Profenza, “javascript - Three.js - Custom Shapes? - Stack Overflow,” 2011. [Online]. Available: http://stackoverflow.com/questions/8284233/three-js-custom-shapes. [Accessed: 03-May-2015].

[26] Khronos Group, “refract - OpenGL 4 Reference Pages,” 2011. [Online]. Available: https://www.opengl.org/sdk/docs/man/html/refract.xhtml. [Accessed: 29-Apr-2015].

[27] J. B. Pendry, “The Science of Invisibility,” *Procedia Computer Science*, vol. 7. pp. 20–21, 2011.

[28] W. Cai, U. K. Chettiar, A. V Kildishev, and V. M. Shalaev, “Designs for optical cloaking with high-order transformations.,” *Opt. Express*, vol. 16, no. 8, pp. 5444–5452, 2008.

[29] M. Gharghi, C. Gladden, T. Zentgraf, Y. Liu, X. Yin, J. Valentine, and X. Zhang, “A carpet cloak for visible light,” *Nano Lett.*, vol. 11, no. 7, pp. 2825–2828, 2011.