

# Final Report

ConceptTop – Concept Map Based Desktop

Alistair Steele – 0930435

Supervisor – Dr Frank Langbein

Moderator – Dr Xianfang Sun

## Table of Contents

Abstract .....	4
Introduction .....	4
Problem .....	4
Aims .....	5
Audience .....	5
Changes from Interim Report .....	6
Design .....	6
Relationships .....	6
User Interface .....	6
Sidebar .....	7
3D Window .....	8
Class Structure .....	9
Database Tables .....	10
Tags .....	11
Tagged .....	11
Hyperedges .....	11
Contained .....	11
Implementation .....	12
Camera Control .....	13
Connecting Renderer and Window .....	14
Retrieving and accessing query results .....	15
Moving objects in 3D space .....	16
Partial Implementation - Sectioning the map using an octree .....	17
Resulting Program .....	18
Results and Evaluation .....	18
Performance .....	18
User Interface .....	21
Evaluation .....	23
Future Work .....	24
Implementation Improvements .....	24
User Interface and Interaction .....	25
Visual Appearance .....	25
Conclusion .....	26

Reflection..... 27

References ..... 28

    Software Library References ..... 28

Appendix..... 29

## Abstract

Traditional computerised document management has focused on a 2D display involving either lists or tiles. The brief for this project was to explore how to represent the relationships between files in 3D space. In addition, a concept-map was to be used as the underlying structure for the 3D representation. This type of structure is meant to allow documents to be connected by their relationships, instead of simply by their location on machine.

To explore this problem a prototype was designed which would allow users to organise documents in 3D. This program will be written in C++ using a variety of libraries including Qt, OpenGL and SQLite3. The concept-map data structure is implemented as a graph with the edges acting as meaningful relationships.

The program developed during the project allows users to organise documents within 3D space using relationships such as tags, folders and custom labels. An objective aimed at improving performance was not achieved. Once the prototype had been created, it was tested first for performance issues and then secondly for user interaction. The program suffers from some performance issues related to the unfulfilled objective, it is believed these can be resolved with some brief future work. User testing found that most functionality was acceptable with some exceptions, where suggestions were provided. These have been incorporated into future work.

## Introduction

### Problem

Document management is a key part of using a modern computer system. A user may have many documents that relate to their work or personal life and it is important that they are able to access and organise these in a way that is clear to them. Traditionally document management software has relied on a simple hierarchical structure to organise these documents. In this format each document is contained within a folder, which in turn can be placed within another folder. It is often the case that these structures use the folders themselves to describe their contents, this is usually done by giving the containing folder a name or label. In the interim report examples of other forms of document management that had been found during research were discussed. Many of these used 'tagging' systems where each document is marked with a selection of keywords which accurately describe its purpose. A user could then look for a specific tag within the program and find all documents marked with that tag, indicating that they are relevant to the topic. In addition this allows documents to be tagged with more than one keyword if they are relevant to multiple topics.

An equally important aspect of document management is the way in which the documents are displayed. Typically this is done in a two-dimensional (2D) format where documents are entries in a list or icons in a large open area of screen space. The user then interacts with the display using their mouse, keyboard or as is becoming more popular, using a touch screen. When examples of alternative display methods were researched not many deviated from a 2D format. Most projects seemed to focus on providing superficial additions such as animations and graphical effects and did not drastically change the way in which users organised their documents.

The problem that this project aims to explore is alternative ways of organising documents with a focus on three-dimensional (3D) representations of a concept-map structure. In the context of this project, a

concept-map is an organisational structure similar to a graph, where connections exist between objects within the structure that describe the relationship between them.

## Aims

As stated above, this project aims to explore the idea of using a concept-map structure to provide users with more control over the organisation of their personal documents. The concept-map will use a variety of connection types that seek to define the relationship between two documents. The project will also look to explore and implement connections that do not directly relate two documents, but instead relate each to an abstract concept. This form of connection is much the same as the tagging example I described in the Problem section. The relationship between the two documents is not direct in this manner but still exists as each can be marked with the same tag, informing the user that they are both relevant to a single topic. This functionality was not mentioned in the Interim Report, however it was decided that it would be a worthwhile addition that would give users more control over the categorisation of their documents.

As mentioned in the interim report a more traditional folder structure is to be implemented, where documents can be grouped together under a single label. This is a format of organising that is common in modern computing as it is a simple method of grouping related documents together. From this the documents can then be seen in context with the others contained inside the folder.

Another aim is to implement an octree spatial data structure so that the prototype may more efficiently draw the concept-map in 3D. The octree will store a reference to each object within the 3D space and when the appropriate method is called, calculate exactly which objects are in view of the camera. By doing this the program will only have to draw the objects visible and not those that are out of reach, this will improve performance, particularly when viewing large concept-maps.

The final aim of the program is to store the documents added and relationships created in the session once the program has closed. This will require the data to be stored in some during the closing procedure. Along with saving the concept-map on closing, this feature will also automatically load the concept-map when it is first started. This process will occur without prompting from the user.

## Audience

This report is being written with the intention that it be viewed by a variety of groups of people. The first and most obvious group are those that wish to continue on with the work that has been achieved over the course of the project. For these readers this report will help to understand the design choices made and the process by which the concept-map is created, stored, manipulated and ultimately, rendered. In addition to helping the reader to understand the decisions made, they would also be able to see ideas for the system beyond the work I accomplished. The Future Work section details many ideas for the system that were not possible to implement or were beyond the scope of the project.

For those that are not interested in continuing the program, either through their own ideas or those detailed in Future Work, this report should still prove useful. The Design and Implementation sections discuss the various libraries and methods I used to achieve the functionality present in the program. This could prove useful to anyone who may wish to undertake a similar project or simply wishes to learn how to use the libraries such as OpenGL and Qt. A reader with this purpose however may find greater benefit in a more general report, rather than one that focuses on such specific problems as the ones this project addressed.

## Changes from Interim Report

One of the main changes from the Interim Report is the clarity of the objectives. As described above the prototype will be required to fulfil explicit areas of functionality, as well as displaying a 3D view of a concept-map structure. These four clear aims include a tagging ability, folder system, spatial data structure and seamless saving and loading. It was decided that while the project is an exploratory one looking at the effectiveness of user interfaces, it still required some clear goals. The ones mentioned have been highlighted as important aspects of document management and as such are required of the prototype.

## Design

### Relationships

As mentioned in the Introduction, the program will support a number of different relationship types, for this prototype development will be focused on three.

The first and simplest type is one which allows the user to connect two documents with a line and apply a custom label. This relationship does not have any functional impact but will help users to find related documents as the lines will go from one document to the other.

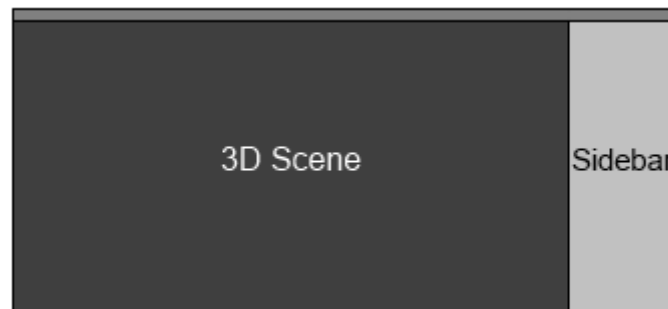
The second type of relationship that will be implemented is the tagging system. The tagging system will allow users to mark documents with one or more given keywords that describe their purpose. When the user selects an object within the 3D space, they will be able to click a button, "Tag Document". When the button is clicked, a prompt will appear asking the user to enter a term or keyword, once entered the user then presses enter or clicks "Ok" button. From then on that document will be marked with the tag given, and will only be removed when the user manually does so. Each tag found in use within the program is represented in the 3D scene by a blue sphere. This system will give the user the ability to mark multiple documents with multiple keywords. The tag objects within the 3D scene gives the user the ability to see all documents that are marked with that tag, this is described in the Sidebar section.

The third and final type available in this prototype will be the folder. As with most document management systems, the folder provides a simple way to organise several related documents into one space. In the case of this program, the documents can be added to more than one folder at a time. To view a folder's contents, the user simply sets the folder object as the focal point and then clicks once more on the object, this will then switch the view to that of the folder. When viewing the folder, each document that has been added to it is displayed in the same manner as in the normal 3D scene, a green sphere. These documents can be moved in the same way as they can in the normal view, however their positions are independent of their position outside of the folder. This lets the user move them about and arrange them in a way that is sensible given their context within the folder without affecting the outside organisation. Camera control within the folder view is locked to focusing on the origin of the scene, where a large red sphere is placed. To leave the folder view, the user simply clicks this sphere and the view will return to show the entire concept-map.

### User Interface

The program interface will be made up of two core areas, the 3D rendering and the sidebar. The 3D rendering will be a large section of the program window that will be drawn by the graphics library, as the project is largely focused on exploring the use of 3D in organising documents, it was important that the majority of screen space available to the program be devoted to this aim. As such, the 3D window will

comprise approximately 90% of the window when fully maximised. The other 10% of the window space will be used by the sidebar, this will be a collection of buttons that provide the user with quick and easy access to the program's main functionality. As well as buttons, the sidebar will also contain some textual labels that will display information about the currently selected objects within the scene, this includes both the actual documents that exist on the file system and the organising functionalities such as folders and tags.



**Figure 1: Main User Interface Mock-up**

### Sidebar

As mentioned above, the sidebar will contain buttons that control many of the options available to the user. Some of these will only be able to be clicked when certain objects are selected in the 3D scene. For example, it will only be possible for a user to connect two documents with a custom label when two documents are indeed selected. In addition some buttons will be hidden entirely when they are not applicable to the current object. An example of this is the “Delete” button, which will be hidden when no object is set as the focal point. The buttons will be displayed in a simple vertical list and ordered so that relevant or similar functions are next to each other, such as the “Add Document” and “Add Documents From Folder” buttons.

Below is a list of the buttons that will be placed on the sidebar.

- Open Document
- Add Document
- Add Documents from folder
- Create folder
- Add document to folder
- Tag document
- Connect documents/Remove connection
- Add to folder

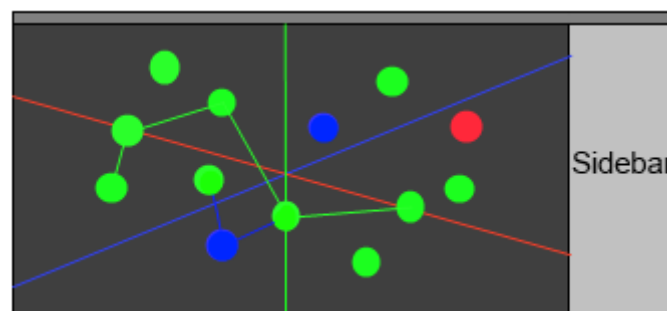
Beneath the vertical list of buttons there will be a separate area that will provide information regarding the currently selected focal point. For basic document objects this simply displays the type of object and each piece of information stored about them, including their name and path. There is also an area just below the textual information that provides another function, one that is specifically used by the tagging system. The area is populated by a scroll list of pairs of buttons, each of which performs a function when clicked. For documents, there is a button for each tag that the document has been marked with, when clicked the sidebar will then change to display the information for the tag instead. Similarly, when a tag is clicked, the

list of buttons represents each of the documents that have been marked with the tag. As expected, when one of these buttons is clicked, the sidebar changes to display the information describing that document. The second column of buttons are simply labelled “X” and give the user the ability to remove the connect that places the first button in the pair in the list. For example, when viewing a tag object and click the “X” button next to a document button, that document will no longer have that tag.

### 3D Window

The main visible component of the program is of course the 3D display of the concept-map. This window will both draw the scene and be responsible for dealing with mouse input within the 3D window. The scene will have a combination of documents, tags, folders and connecting lines within it, each type of these will have their own colour. Each of these objects is represented by a sphere, these vary in size depending on the type. Tags and folders will be drawn as larger spheres than documents as there are likely to be less of those and they act as collections of documents. Connections between documents and other objects are drawn as single lines, also coloured by type. The final element within the 3D scene will be visual markers that will aid the user in their navigation of the space, such as a flat panel that acts as a floor.

Where textual labels help to identify objects within the scene, they have been placed near the object. This means that each document has its name printed next to it in a grey font. The same is true for tags and folders. The custom connections also have a label, these are printed at the halfway mark between the documents it connects. As objects within the 3D scene may be far away from the camera position, the labels will scale in size so that the closest objects have the largest labels. When an object reaches a certain distance away from the camera position, the label is simply not drawn as the scaling would make it too small to be read and it will improve performance to ignore labels this far away.



**Figure 2: 3D Scene Mock-up**

As mentioned, the 3D scene will be responsible for handling mouse interaction with the 3D objects and the space they are drawn in. This includes selecting and moving objects and rotating the camera around the current focal point. The user can select at most two objects within the scene by performing a single click on each. These objects are selected one at a time and once marked as selected they will be drawn in the colour white instead of the default colour for their type. When one or more objects are selected, buttons in the sidebar become interactive or visible and the user can access additional functionality. When an object is double clicked, either immediately or simply by single clicking again once it has been selected, the object is set as the focal point. At this point the information relating to the object is displayed in the sidebar, including the scroll bar with buttons leading to tags or documents. The focal point is also made to be the centre of the scene, from that point onwards the camera is looking directly at the coordinates of the object.

Interacting with objects in 3D space is not a typically intuitive task, it requires the user to be highly aware of each objects position in the world. I felt that this would be made significantly easier and more user friendly



if my program gave the user the ability to drag objects around in 3D space using their mouse. This would function similarly to how dragging documents in traditional 2D managers works, the user presses and holds the left mouse button down, moves the mouse and the object follows the movement. When the mouse button is released the object remains in its new position. The movement of the object will take place on a plane that is parallel to the camera plane. This means that the object will not stay at a set distance from the camera, however the perpendicular distance between the plane of motion and the camera plane will remain constant throughout the move.

The final functionality that is tied to the mouse is direct camera control. If the user presses the left mouse button down when the mouse is placed over empty space and moves the mouse, the view will rotate around the focal point. This focal point defaults to the origin when the program is first opened or can be set by double clicking on an object. When the mouse is moved horizontally, the view rotates around the vertical axes, and when moved in a vertical direction, the camera pans and provides a steeper or shallower view. This panning ability is limited in range so that the camera is always the right way up. The last aspect of direct camera control afforded to the user is tied to the mouse wheel, using this one can zoom in and out of the scene, altering the distance the camera is placed from the focal point.

## Class Structure

The initial class diagram I included in the Interim Report detailed four classes. While these four classes still form the majority of the program code, during development it became clear that some additional classes and subclasses were needed.

I created several classes that contain information for each object in the concept-map. These classes include Node, Tag, Hyperedge and Edge. Each of the classes holds an ID and a label. In addition, the first three classes also each hold a three-dimensional position (X, Y, Z) as these are free-placed objects in the 3D space. The Node class contains a piece of data known as the path, which details which file system object is described by the object.

As well as data describing the objects themselves, these classes are designed to hold data that informs the program of the relationships between objects. For example, the Hyperedge class is used for grouping documents together and as such needs to maintain a list of all the documents, or nodes in order to properly display them. This list is stored within the Hyperedge object. A similar list is maintained within each Node object for Tags that it has been marked with.

The Edge class has a somewhat different structure, it does not contain any positional data and instead merely consists of a label, a start point and end point. The start and end points are the IDs for the two nodes that form either end of the edge. When rendering, the position for each end of the connection can be retrieved from the corresponding Node objects.

Each of the four classes mentioned above are used by the Graph object, this class deals with the logical structure of the concept-map data, tracking the connections (Edges) between the Nodes, which Tags have been used on which Nodes and which Nodes are present in which Hyperedges. The functionality provided by the Window and Renderer classes through the UI buttons and mouse interaction is implemented within the Graph class. These include methods such as adding nodes to Hyperedges and tagging Nodes with labels. The secondary task of the Graph is to deal with retrieving and storing the data structure within the database. This is done via a collection of save and load methods, one for each object type. The Graph will load the concept-map data when it is initialised and save it to the database when it is deleted.

The Window class is the top level object that will handle the UI elements. It will also be responsible for creating the Graph and Renderer objects as these will need to be directly accessible in order to provide functionality such as displaying Node information and adding connections between nodes. This class will likely use a third-party library to create the more basic elements of the interface such as buttons and text labels. Using well developed libraries will allow me to spend more time on the new functionality proposed by the brief. The UI created using this library will be composed of widgets, these can be generic types such as buttons or labels or more complex elements such as those that will be used to create a 3D rendered area.

The Renderer class will comprise the majority of development efforts, it will be the class responsible for drawing the 3D concept-map, displaying an appropriate view of the scene to user and dealing with mouse interaction within the 3D space. This object will require access to the graph in order to retrieve data necessary to draw the scene such as positions for objects and text for labelling. This class will take the form of a UI widget, which will be created by the Window class and added to the 2D program window to be displayed alongside the other widgets.

The Octree class will be responsible for maintaining a secondary list of the objects within the 3D space and organising them in a spatial data structure. This class may be broken down into an overall object and subsequent child objects, each of which will represent a node within the octree structure. This class will be used to limit the number of objects that need to be drawn by the Renderer. It will do this by calculating the exact boundaries of the camera view and then make a list of each object within that area, this list is then sent to the Renderer for drawing. This reduction in the objects drawn will improve performance, especially for concept-maps with many objects.

Lastly, the Database class is designed to provide access to an SQL database library. It includes several methods, such as those used for querying and opening transactions for faster insertions to the database. The query method in this class will perform most of the functionality, it will both insert and retrieve data. When data is retrieved using a SELECT query it is stored within a special class that will act purely as a container for such returned data. The Result class is used purely by the Database and Graph classes, it is a custom data structure that stores a number of rows that are returned by a database query. This data structure will be iterable, meaning that when a Result object is returned it can be looped through to access each row contained within it. Once a row has been accessed each value stored in it is accessible as well.

## Database Tables

In the interim report two of the tables that would be needed in the program's database were described, nodes and edges. After deciding the features of the program and more fully developing its plan it has become clear that several more database tables will be needed in order to store the data needed by the concept-map. In total six tables are needed, the schema for those not described in the interim report is listed below.

## Tags

Type	Name
Integer	id
String	label
Float	x
Float	y
Float	z

The tags table has five fields, a unique ID, a label or tag and three fields that are used to position the Tag object in 3D space. A decision was made to make the Tag objects rendered in the scene as it would provide a visual element that shows which documents have similar content.

## Tagged

Type	Name
Integer	tagID
Integer	nodeID

The tagged table simply stores a two-integer pair which declares that a particular node has been marked with a tag. This is achieved by storing both the tagID and the nodeID in a record.

## Hyperedges

Type	Name
Integer	id
String	label
Float	x
Float	y
Float	z

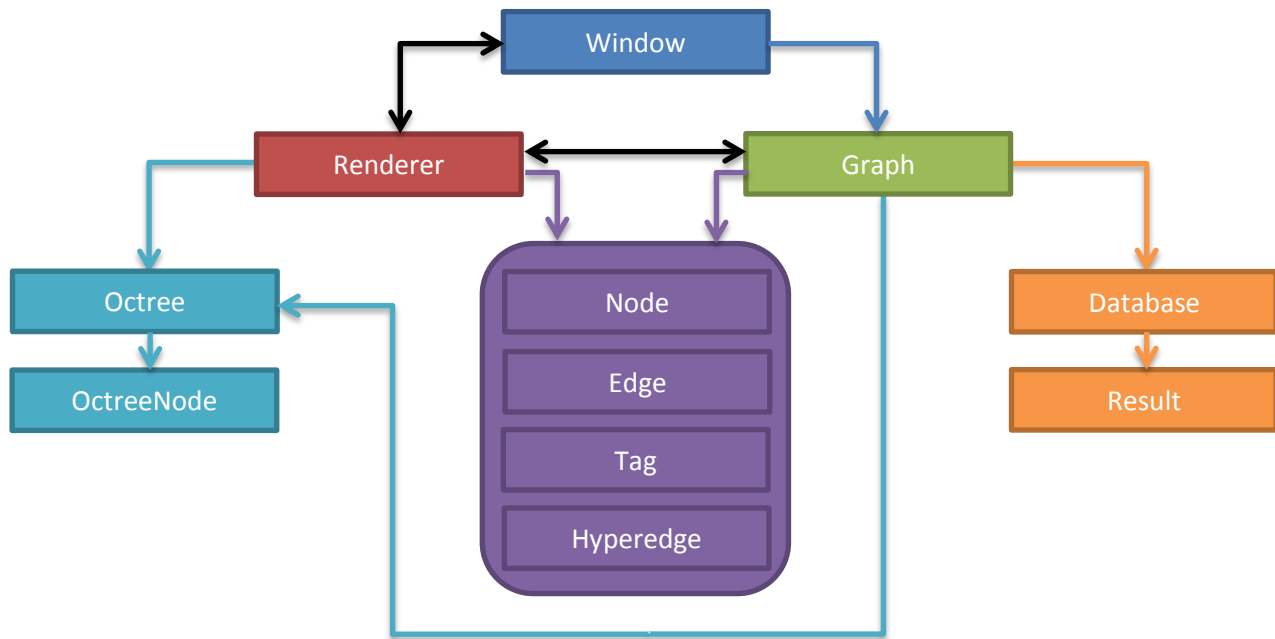
The Hyperedge object is similar to Tag, it has an ID, a label and a position within the 3D scene

## Contained

Type	Name
Integer	hyperedgeID
Integer	nodeID
Float	x
Float	y
Float	z

The Contained table is tasked with storing the data that describes which nodes are present in which Hyperedges. To do this each record holds two integer IDs, one for the Node and one for the Hyperedge. This table is very similar to Tagged, however in addition to the ID pair there are three variables that describe a position in 3D space. This position is independent of the Node's position in the normal scene, and would be used when the Hyperedge is being displayed.

The final class diagram is as follows:



**Figure 3: Class Diagram**

## Implementation

Throughout the development process the design choices were abided by as much as possible, however from time to time problems occurred that required a rethink in strategy. This section will deal with how the design was implemented, the issues encountered, and the solutions that were found to solve them.

The top level class that handles user interaction is the **Window** class, this will create and manage the interface elements such as buttons, labels and the rendered 3D frame. As discussed in the interim report, Qt was chosen as the interface library the project would use. Qt is a well-developed framework that allows a programmer to create standard UI elements as well as extend the built-in classes to suit more specific needs. This extension ability will be utilised in the **Renderer** class which will be an extension of Qt's native **QGLWidget** class.

As described in the Design section, the **Renderer** class is responsible for the retrieval of concept-map objects and the drawing of these within the 3D scene. In addition to this it intercepts any mouse actions that occur within the 3D window. **Renderer** will be an extended version of the **QGLWidget** class. **QGLWidget** is a subclass of the generic Qt widget class that allows an OpenGL instance to be placed within a Qt window. This widget can then be moved around, hidden and shown in the same manner as other **QWidgets**.

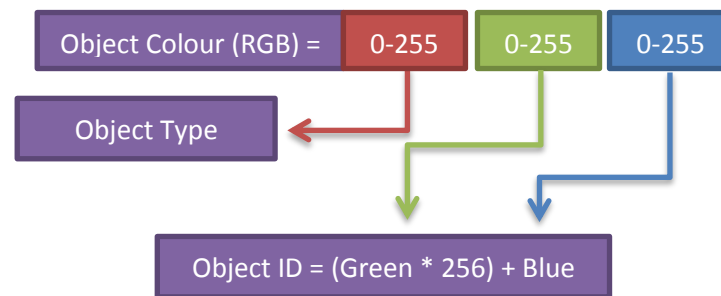
When development began it became apparent that the code to draw each of the concept-map objects onto the screen had similar rendering processes. This is because the Nodes, Tags and Hyperedges are all drawn as spheres with only the size and colour changing between them. To reduce the overall amount of code that was included in the program a separate class called Object3D was created that would hold information that was common to all three object classes. These fields comprised the numerical ID, the textual label and the three-point coordinate to describe its position in the scene. This information is all that is necessary to draw the object onto the scene. Each of the three classes was then altered to be derived from Object3D. This change allowed the program to use only one drawing method for these three classes, this method would then take a single Object3D instance as its parameter. The key benefit to removing repeated or highly similar code is to reduce the risk of errors occurring, by only having one method the effort to alter how the objects were drawn was reduced by two thirds.

## Camera Control

The centre point of the rendered scene defaults to the origin of the 3D space when the program is first opened. However the centre can be changed by making an object the focal point. To do this, the user double clicks an object with the left mouse button, the scene will then be redrawn with that object in the centre. This selected object is stored in a variable of the type Object3D\*, a pointer to the object. As such only classes that are derived from Object3D can be the focal point of the scene.

In OpenGL the camera aperture can be defined using glPerspective and glFrustum. As the camera in the program will be able to rotate, pan and zoom in and out of the focal point a convenience method called gluLookAt will be used to simplify the process, this is provided by the glu library. This method takes nine arguments, the first six of these can be considered as two sets of three-point coordinates. The first describes the position of the camera and the second describes the point that the camera is looking at. The final three arguments are used to define which vector (x, y or z) is the 'up' direction. Using these nine arguments the method calculates the perspective matrix to be used. In conjunction with this method, glFrustum to define the size of the camera plane in the 3D space. For ease of use a height of 2 and a width equal to twice the aspect ratio was used, this allows the rendered image to remain properly proportioned.

In order to interact with the concept-map within the 3D scene it is necessary to be able to select objects within the 3D space. The process used to find the object that has been clicked on in the 3D scene is called colour picking. Colour picking is a simple method of identifying objects, when the mouse button is pressed down the code to select the object under the mouse (if there is one) is activated. This code redraws the scene with several differences, it does not render the textual labels, edges, axes and the 'floor' of the scene. The rest of the objects in the scene (Nodes, Tags, Hyperedges) are drawn with one difference, the colour of the object is set to a unique value that has the object's type and ID encoded. Once the scene has been rendered in this way, the code proceeds to read the pixel buffer at the point equal to the mouse coordinates. This then returns the colour of the pixel at this location, the colour can then be decoded and the type and ID revealed. From this point, the object can then be marked as selected. The section below describes the encoding system used to store the ID and type in the colour.



**Figure 4: Colour Decoding**

Using this method of encoding, the system allows 256 different object types, it is unlikely this many types will ever be implemented and realistically this field could be reduced in size to three or four bits of the 24bit colour. This encoding also allows for 65536 possible ID values, however a reduction in the Type field would allow for more ID possibilities. Currently this means that there can be 65536 instances of each type of object.

Colour picking was not the first method of object identification that was attempted, the first method considered involved ray casting. Ray casting is a process by which a virtual ray is generated based on the camera position and the position of the mouse on the 2D user interface. Using the information known about the camera position, size of the camera plane, and the position of the cursor on the 2D user interface, the program calculates the directional vector from the camera to the location of the cursor as projected onto the camera plane in the 3D space. From this point it is possible to continue this projection (extend the ray cast) to find the first object that intersects the ray. During the development process this technique was attempted for use in revealing which object was being clicked. However the mathematical calculations proved to be quite complex and checking the objects for collision at every point along the ray was costly and inefficient.

The attempt at ray-casting resulted in the creation of a new class known as Vector. This simply acts as an easier to use container for three-point coordinates. It contains methods that allow for the manipulation of the vector such as multiplication, division and addition. Also, there are several methods that calculate the distance from another vector or an abstract point in the 3D space and the total length of the vector.

It was decided that colour picking is much simpler strategy and took very little time to implement. It was also noted that although ray casting would have likely resulted in a quicker object search, the technique would have taken a large amount of development time to finish. As the focus of this project was to explore how a 3D interface could be used to effectively organise and manage documents on a computer it was best to find an accurate approach that would not take a disproportionate amount of time to develop.

## Connecting Renderer and Window

The Renderer class is in control of the selection of objects within the 3D space, in order for this information to be displayed by the code of the Window class it is necessary that there be a method of communicating between the two objects.

A feature of the Qt library called signals and slots was utilised. Signals and Slots is a system by which two pieces of code can be connected using a meta-object code compiler that is shipped with Qt. There were three changes that needed to be made to use the signals and slots system. Firstly, when a class is defined it has one or more signals listed in its header file, these can then be called or 'emitted' within the real

methods of the class. Each signal consists of a name and a parameter list including the type of each parameter. The signalling class must also contain the Qt keyword “Q\_OBJECT” in its header file. The second place that requires special code is the header of the class that is to receive the signal, otherwise known as the 'slot'. In this case the method that is to be called when the signal is received is listed under a 'slots' keyword, again with a parameter list including types. The final step is to connect the two pieces of code, this is done using the connect() method. This method takes four arguments, the signalling object, the signal itself, the receiving object and the slot to call when the signal is received.

As with most issues encountered there were several ways to solve this problem. The first and simplest method attempted was to ensure that both objects stored a reference to the other. This would allow each class to call methods on the other and pass the selected object ID and type in the function call. Although this method is simple it does create a lack of clarity in the structure of the program. As the Window class is the object which instantiates the Renderer it is somewhat inelegant to pass a reference to Window to the Renderer object.

As the program is displaying information about the Node, Tag or Hyperedge that has been set as the focal point of the scene it is necessary for the Window class to know when the focus has been changed. For this a signal was created on the Renderer class called focusChange and a corresponding slot on the Window with the same name. This signal carries two parameters, the ID and type of the object that has been set as the focus. It is necessary for Window to know these two pieces of information so that it can display relevant information regarding the object in the sidebar. For example, a Node will have an additional field to display as it contains a path variable that links it to a file system object.

In addition to communicating focus changes made, the Renderer class is also in control of the selection made by the user. In the prototype the user can select two objects at any one time, when an object is selected it is stored in the selected array. Whenever the selected array is updated it is passed through a signal to the Window class. This class then checks the objects within it and depending on the type and number of objects within the array, enables or disables buttons within the user interface. For example, if two Node objects are selected then the button that allows them to be connected by a custom label edge is enabled, this is then altered to “Connect” or “Disconnect” depending on whether a connection exists already between the two objects.

A method-calling based system was chosen to communicate changes in the selections and focus of the rendered scene as the only other alternative visible at the time was to use a polling design. If a polling system had been created instead the Window class would have had to periodically call a method such as `Renderer::getFocus` and then update the user interface widgets if the objects retrieved were different from those returned on the last check. It was decided that this approach would cause unnecessary strain on the system and would cause a delay in updating the interface to reflect the changes made.

## Retrieving and accessing query results

The Database class is responsible for connecting to the database file and executing queries. The main method that runs queries is called `Database::query`, this method takes one parameter which consists of a string that contains an SQL statement. This query is then compiled into a prepared statement for use by `sqlite3`. If the query is successful, the program goes on to count the number of data columns returned. If the statement returns no data columns (a count of zero) then the statement was of a type other than `SELECT` and the method returns a null pointer. For `SELECT` queries the method creates a `Result` object to store the retrieved data within. The `Result` class is a wrapper that holds a `std::list` of `std::maps`, each map is

a pairing of `std::strings`. This data structure is used such that each map constitutes a row of data returned from the database. This map is then used as a key-value structure where it can be indexed by key or database column name and the map will return the value.

The Result class was created to simplify the reading of information from the database, this benefit can be seen in the only class that communicates with the database, Graph. When reading the data for the concept-map from the database, the Graph class is given Result objects by the database, it then uses the `Result::step()` method to iterate across each row and store the data within the structures contained within the relevant data structure within Graph.

One of the main reasons for choosing to create a specific class to deal with the data returned from queries was to reduce the amount of code necessary to access the information within a calling class. In addition there was a choice to give responsibility of returning individual pieces of information to the Database class itself. However this would mean that the class could only execute one query at a time without the class deleting any information retrieved from the last query. The only reasonable solution was to return the information entirely when the query is executed and to provide a way of iterating through the structure, row by row.

There is a disadvantage to the current implementation and that is that each piece of data is stored as a `std::string` within the Result class and must be converted to the expected type manually in the calling code. For example, when reading the Nodes table each row's ID must be converted from a string to an integer before being used as the index in the Graph data structure.

## Moving objects in 3D space

Moving objects around in the 3D space is an important aspect of user interface, the mouse controls this functionality. As the left mouse button is already being used to select objects in the 3D space, it was necessary to extend the code section that deals with this functionality. In addition, when not used to select an object the left mouse button acts as a method to rotate around the focal point of the scene. The methods responsible for handling mouse interaction are `mousePressEvent`, `mouseMoveEvent` and `mouseReleaseEvent`. All three of these required changing in order to add this new functionality. The `mouseMoveEvent` method is called when the user moves the mouse across the screen, this is called whether the mouse is within the bounds of the OpenGL instance or not. As the object will remain a constant perpendicular distance to the camera plane it is only necessary to calculate this distance once. This calculation is carried out in the `mouseMoveEvent` method and uses the knowledge of the object's position and the position of the mouse on the camera plane as translated into the 3D space.

For each subsequent call of the `mouseMoveEvent` method, the previously calculated perpendicular distance is used to project the new camera to camera plane vector forward into the 3D space. This is done by translating the mouse's new position into 3D space, calculating the vector between the camera and this position and then extending that vector so that the length of the vector perpendicular to the camera plane is equal to the previously calculated value.

Once the new position has been calculated the selected object is moved to the new coordinates, this gives the effect of the object 'following' the mouse. The final part of this functionality is the code within `mouseReleaseEvent`, as this method has other responsibilities such as marking objects as selected or the focal point. Therefore all that is needed is to check if the mouse was being used to move an object and if so, simply reset those variables and return.



When developing this area of the program and ensuring that this functionality would be available to the user, it was necessary to spend some time considering the mathematics of calculating an appropriate new position for the object. A decision had to be made as to which would be more useful to the user, moving the object across a plane parallel to the camera or moving it in an arc so that it was always a constant distance from the camera position itself. It was felt that the latter option would create confusion and limit the control over the object and therefore the first choice would allow the user to more accurately place the objects how they wished.

### **Partial Implementation - Sectioning the map using an octree**

When first attempting to discover which objects in the 3D scene were being selected using the mouse the ray casting technique was used. This technique required that a ray be projected from the camera, through the 3D space translation of the mouse position on the camera plane and onward through the coordinate system. As a means of increasing the efficiency of the algorithm, a system was designed and partially implemented that would reduce the number of objects which would be checked for interception with the ray. This system is based on the idea of an octree, a hierarchical structure that divides a group of objects between eight child nodes at each level. The interim report described the concept of an octree in greater detail.

The algorithm to search the octree using the vector taken from the camera interaction would subdivide each node into eight children and only check the child that the vector passed through. This results in at most four child nodes being accessed per level, the node the vector originates in and the three times the child boundaries can be crossed.

In order to implement a data structure that tracks all of the objects within the concept-map it is necessary for the Octree class to receive updates when objects are created, deleted and moved within the Graph class. It is possible for the Octree to take another approach and to poll the Graph class for changes, this could be done just before the Octree calculates which objects are in each child or whether an intersection has taken place. This method was not chosen as it would require too much development work to add a change tracking system into the Graph class, this would be necessary to efficiently communicate the changes to the Octree since the last poll.

For the Graph class to push the changes to the Octree object it needs to be aware that the object exists. To achieve this, the Renderer class which is responsible for the creation and primary use of the Octree calls the method `Graph::connectOctree()`. This method then immediately updates the Octree with the current structure of the Graph. In addition, from then on in the Graph class whenever an object is added, removed or altered the code checks to see if the Graph has been paired with an Octree. If so, it communicates the change to that object using the appropriate method.

For the most part the Octree class simply acts a wrapper for the data structure, the majority of the calculation and code is contained within another class known as OctreeNode. This class is responsible for maintaining the references to each object within its bounds.

Many obstacles were encountered in the attempted implementation of this feature, both in the theory needed to properly execute the search through the octree and in the time constraints posed by the project. When work began on this area of the program the complexity of maintaining and traversing a spatial data structure was not fully understood. Storing and dividing the objects among the child nodes was not a particularly difficult area to implement, it simply required some basic mathematics and program design to ensure that each object was within the correct child. The core issue found when storing the objects was

keeping the knowledge of the 3D space current, that is, it was difficult to ensure that the Octree class was apprised of all the changes made. It required fairly substantial additions to the Graph class to ensure that every new, moved and deleted object event was communicated to the Octree class. The layout of the initialising code also meant that the Graph had to be created before the Octree and as such the two had to be explicitly connected in a way where both objects would be aware of each other. This led to the creation of the `Graph::connectOctree()` method.

Although these changes made the code more complicated, communicating changes was not the problem that led to the failure to complete this feature. When the code to find all objects within an area (for use in frustum culling) was created it was difficult to understand the mathematics. The process required vector calculation in order to trace the four corners of the frustum through the octree. A point was reached where it was taking too much time away from other, equally or more important aspects of the program and as such it was decided to leave this feature out of the prototype. Given more time the feature would have certainly been finished and improvements made in the performance of the rendering cycle.

## Resulting Program

Appendix 4 is an image captured while the prototype was in use, the image shows several documents placed in a 3D space. Some of the documents have connections between them that describe a custom relationship. In addition, it can be seen that a Tag object is currently set as the focal point and as such each document tagged by that object is listed in the sidebar.

## Results and Evaluation

Once development had stopped on the prototype it was necessary to execute a series of tests and evaluating methods in order to assess its success.

### Performance

The first form of evaluation that was carried out was on the program performance in order to determine how efficient the software is in carrying out its functionality. One such area of functionality involves storing references to a large number of objects and keeping track of the connections between them. Due to the varying size of the concept-map that may be created by the user, it is important that the program is able to store these pieces of data in a relatively small amount of memory. During development, the program's performance and resource usage was considered and monitored using the operating system's System Monitor, a small piece of software that simply lists each running process as well as its resource usage. The monitoring program displays CPU usage as a percentage of the total and memory usage is shown in mebibytes, a unit similar to a megabyte but instead has a clear definition as  $2^{20}$  bytes. This program shows that the prototype has a period of increasing memory usage when the camera position is altered. This increase only occurs for a few seconds and then the memory usage stabilises, totalling only a few mebibytes in additional memory.

The second metric that was recorded when looking at the performance of the program is how long it takes to draw a single frame of the scene. Retrieving this information required some small additions to the Renderer class. Two global variables had to be created, the first tracks the total time spent drawing the 3D scene, and the second counts the total number of frames drawn. Using these two variables it is a simple matter of dividing the time by the count to find the average time taken to draw the 3D scene. As the average over the course of the program run was the desired metric, code was added to calculate and print

the average in the Renderer destructor. In order to accurately time how long it takes to draw each frame, the Linux sys/time.h library was used to retrieve the current time at the start and end of the Renderer::drawGraph() method. It was then a simple case of calculating the difference, adding it to the total time taken so far and incrementing the total frame count.

To see the effect of storing more documents on the memory usage and draw time of each frame the program was run several times, changing the number of documents added each time. A total of 15 different document counts were used, starting at 100 and increasing in increments of 100 each time, to a maximum of 1500. To ensure that the program was clear of any other data, the database was cleared of all records before each test, following this the program was closed and reopened it. Once the program had reopened the documents were added to the concept-map using the “Add Documents from folder” button. At this point the System Monitor was used to take the first memory count. At this point the mouse was used to rotate the view around the centre of the concept-map for approximately 40 seconds. This was done for two reasons, the first was to get a more accurate reading of the time it takes to draw the scene. By recording 40 seconds worth of frame renders it was possible to calculate a better average. The second reason was to trigger the small memory increase described above. Using this testing strategy, three pieces of data are retrieved for every number of documents tested.

The results from this testing are in section 1 of the Appendix.

After running the tests the results were analysed to see if any trends could be identified. The memory usage measurements did show a weak positive correlation between the number of documents within the concept-map and the amount of memory in use by the program. An increase in program memory usage was to be expected as the data must be stored and if the data increases, the storage requirements increase. The test aimed to assess whether or not the memory increase would be prohibitively resource intensive, this would make the program fairly impractical for large numbers of documents. I believe that the increase in memory usage is minimal and therefore the program should not cause a loss of general system performance due to allocating excessive amounts of memory. The graph below shows the data captured for both the initial program state after adding the documents and the stabilised state once the camera has been rotated for 40 second period.

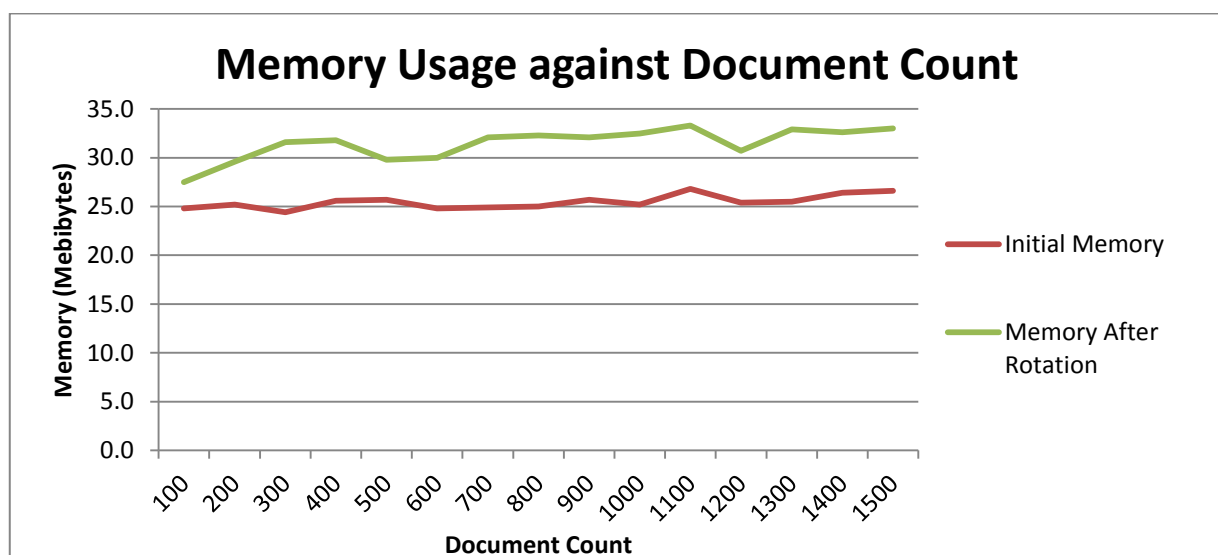
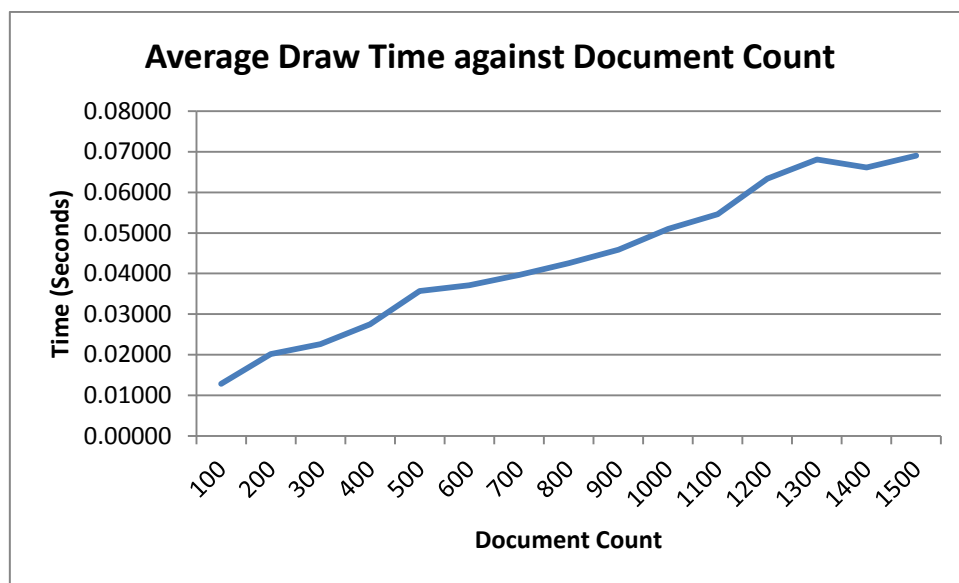


Figure 5: Memory Usage against Document Count

The second line shows the memory usage once the program has undergone user input to alter the camera position, the graph shows that this increase is never more than a few mebibytes in size. It can also be seen that the increase is fairly similar across all concept-map sizes, meaning that an increase in initial state usage does not translate to a greater increase once interaction has begun. Again, I feel that this is an acceptable level of resource usage and should not be a cause for concern when considering the real-world viability of the program.

The second part of the test looked at the average time taken to draw the 3D scene visible to the camera. The data gathered showed a much clearer correlation between the number of documents in the concept-map and the time taken. Looking at the actual numbers gathered it is clear that the program quickly becomes slow to respond once a reasonable amount of documents are added, I believe this would prove to be an issue if the program were put into real-world use. Even concept-maps containing just 500 documents took just over 0.035 seconds to draw. This means that if the program was being constantly redrawn as it was during the test, it would only reach 28.5 frames per second. This is likely the highest number of objects that can be drawn without the user noticing the reduction in frames. The graph below shows the data plotted against the number of documents, as can be seen the positive trend is quite clear and consistent.



**Figure 6: Average Draw Time against Document Count**

There are several ways that this test could have been improved, however due to time limitations it was not possible to implement an efficient way of carrying out the tests in such a way. One of the main drawbacks of the method used is that the memory measurements were not averaged over many executions of the program. Doing so would have yielded a much smoother graph. This was not done as it would have either taken far too much time, or required me to implement an automatic method of executing the program and recording the memory usage. Similarly, although the second part of the test that compared the time taken to draw the scene used an averaged value, it would have been more effective if the pattern by which the mouse was moved around the screen was identical on each iteration. An attempt to perform the same movements each time was made, however an automated test would be far more accurate than a human.

## User Interface

The second area that required testing and evaluation was the user interface. This part of the program needed to be evaluated to see how well the interface worked as a means of organising documents. The only practical way to do this was to ask real people to try using the program. A questionnaire was created that asked the user about the different features of the program and whether or not they found them suitable or intuitive to use.

The type of questions to be used within the form required careful thought, they needed to be both easy to answer and yet provide sufficient feedback to see why a particular implementation or design choice was not successful for that user. Several of the questions within the form require a feature or implementation to be scored with a value between 1 and 10. Although this form of question does not allow users to give written feedback, it does provide a scale that shows how adequate that area of the program is for them. Another form of question is one that is in two parts, the first asks whether or not the current implementation of a feature is appropriate and usable and if not the second part asks what could be done to prove it. This format of question was chosen to ensure that the feedback received would relay the problems the users were encountering when trying the program. A simple yes or no answer does serve to answer whether or not the program was sufficient in that area, however if the user answered no, it was important to record what made it an unsatisfactory design. In one of the two part questions, users were asked whether it would have been better if documents were displayed using their icon as shown within traditional document managers. This was an idea that had been considered for implementation given enough time, however the problems encountered with more important areas of the development meant this could not be achieved. The final type of question that was used were those that simply asked for additional comments, these are open questions that let the user express any other thoughts they had regarding the prototype.

The questions were broken down into three main sections, Visual Aspects, Interaction and Other. The first section dealt with how the concept-map is displayed to the user, this included questions about how easy it is to tell the documents apart and the clarity of textual labels used to display information such as the document name. In addition the user was asked to provide a general rating of the visual aspect of the program. The second section was devoted to questions regarding the interaction used to alter or explore the concept-map structure or change the camera view. As with the first section, this one began with a simple rating between 1 and 10 of the overall interaction available. Following this three questions were asked that each dealt with specific areas of the program's functionality, mouse interaction, the folder system and the tagging ability. Each of these questions were two-part, the second of which asked what could be done to improve the feature. The final section simply asked the user for any other improvement suggestions they may have and for any other comments. These were added with the understanding that some areas of the program did not have questions or sections aimed at them and as such the user did not otherwise have a place to put comments regarding those. The final questionnaire that was shown to the testers can be found in section 2 of the Appendix.

Five people were asked to test the program and fill in the questionnaire. Of these, three were Computer Science students and as such were very familiar with computers, the other two users were not as familiar however still used computers fairly regularly in their day to day life. When it was decided which audience to use for this test, I felt it was important to cover more than one type of user. As most of the people available

to test the program were Computer Science students there was a concern that the result of the questionnaire would not represent the experience of the average user.

The procedure for the user test was fairly simple, the user was presented with the Linux machine and the program already open. A printed questionnaire was placed on the desk beforehand so that they could fill it in as they tried each area of the program. Once all the tests had been done, the answers were collated into a table. This table can be found in section 3 of the Appendix.

The most notable observation from the results is the clear divide between the Very Familiar group and the Somewhat Familiar group. The first group's answers to question 1 seemed to imply they were much more comfortable with the display of the concept-map, this may be due to their experience with generally more complicated software. The second group may have been uncomfortable with the display as it is in a fairly bare state as the main focus of the project was the functionality rather than the visual look. Question 2 indicates that most testers found it difficult to distinguish between different documents, this was likely as they are all drawn in the same colour and shape. The directed part B to question 2 which asked whether an icon-based alternative would be better received a unanimous Yes. Given the trouble experienced in distinguishing between objects, this would be a likely priority for any future work done. The final question of the Visual Aspects section was aimed at the textual labels, testers in both groups agreed that the labels could be clearer and suggested improvements such as relocating them to beneath their object. Overlapping over labels was also an issue reported, as the font used is drawn with no outline it is possible that too many objects on the screen at once may make it impossible to discern any labels.

The Interaction section showed a slightly smaller division between the groups, most reported an overall score of 4 or 5. Question 5 received mixed answers with those saying that the mouse functionality did not meet their expectations including suggestions in part B. These suggestions included a context menu that could be used as an alternative means of accessing functionality such as tagging and creating custom edges as well as the ability to select objects using a dragging motion. The second suggestion would likely help greatly in the general interaction of the program, a typical user may have a large number of files and tagging each document one at a time would be a time-consuming process. The same testers that answered No to question 5 also answered No to question 6. This question pertained to the folder system, one user who replied negatively commented that they expected to be able to place folders within one another. Another explained that they expected the document object to disappear from the main view when it was added to a folder, this would prevent documents from being in multiple folders at once and would reduce the folder system to that of a traditional document manager. A greater show of support for this design would indicate that it is preferable to be able to place documents in multiple folders, and would likely result in further exploration of idea if future work were carried out. The tagging system was fairly popular, with only two comments being made, the first suggesting that tags be more prominently displayed around a document, although it was not said whether this should be when it is focused or not. The second comment ties in with a similar comment the same tester made in response to question 5, suggesting that it be possible to tag multiple documents at once.

Questions in the Other category received some replies, tester 1 suggested that adding a snap to grid option would be beneficial, it can be assumed that this is in relation to the placement of objects within the 3D space. Such a feature may help to keep the scene tidy and pleasing to the eye and could be implemented as a toggle in a future prototype. Question 10 had two replies, one each positive and negative. The positive

comment stated approval of the limited camera control feature, and the negative comment communicated an uncertainty of the use of 3D to manage documents.

The questionnaire evaluation method proved to be fairly effective at discovering common issues encountered by both more and less experienced users. Some key issues were identified such as the inability to select multiple documents at once and tag them all and a lack of distinctiveness regarding the visual look of the documents within the program. However I believe a larger scale user testing process would have yielded a more balanced set of results, particularly regarding the overall scoring questions. The questionnaire could also be expanded to include sections regarding the navigation functionality and general program usefulness. An important aspect to consider when carrying out user-based testing is the experience the user has with the type of software being tested, the small sample size used for this test shows that experience can vary largely. An expansion to include users with different experience, or perhaps a section at the start of the questionnaire asking the user about their experience with computers would help identify issues encountered by different groups.

## Evaluation

After carrying out testing and evaluating the results, it can be seen that the prototype developed achieved many of its aims regarding interaction and visual display. The interaction provided within the program was received fairly well by the testing group. They were capable of adding documents to the program and organising them using the folder and tagging systems. This functionality was accessed using the appropriate menu item in the sidebar of the program. Areas where scores were not as high as expected such as mouse interaction can be improved with a few additions in the form of drag and dropping documents into folders and selection by dragging. These ideas were suggested by the testers and would likely benefit the user greatly in their control over the concept-map. As efficient and intuitive user interaction was the core aim of the project, the low scores could be seen as a failure of the project. However, given the issues I encountered implementing certain areas of functionality including object selection and the attempted Octree system, I feel that the scores show an encouraging acceptance of the interaction provided. It would require only a few weeks more development time to implement the functionality that was suggested by the testers as improvements.

The visual appearance of the concept-map structure within the program was less well received by the testers. Specifically they felt that it was difficult to tell the difference between the documents shown and all agreed that displaying the documents using their icon as seen in a traditional document manager would be a better approach. The system used to draw the labels for the objects also received a low review, lack of clarity and correct position led users experience issues trying to read the labels. In my opinion, the project aims should have lent themselves to more time being spent developing the visual aspect of the program. Although the core aim was to explore what interaction was possible and useful to a user, document management is a very visual task and it may be the case that the underdeveloped visuals led the testers to being unsure of how the program functioned. A more user-friendly visual appearance would likely encourage less experienced users to explore the programs functionality, as it would be a welcoming display as opposed to the somewhat technical look the prototype has currently.

From a performance and system resource usage perspective, the prototype is far from efficient. While the memory usage is fairly consistent throughout operation of the program, and the increase for each added document is very low, the strain on the CPU is severe. This mostly stems from the rendering cycle, with a document count of more than a few hundred the frame rate drops to a level at which the user can see



stuttering when moving around the concept-map. I believe this issue would have been far less pronounced had I managed to complete the implementation of the Octree class, this would have allowed the Renderer to ignore objects that were not visible to the camera and reduced the subsequent strain on the processor. An alternative approach to the rendering of the scene would also have likely resulted in better performance. As mentioned earlier in the report, the material used to initially learn how to use OpenGL to create 3D scenes was fairly out of date, these techniques led the processor to take most of the rendering strain rather than the GPU of the machine. Instead of simply drawing each sphere directly, it would have been more efficient to load a sphere object into the graphical memory of the machine when the program was first opened, this would be done using Vertex Buffer Objects (VBO)[1]. With the sphere stored in graphical memory it would not be necessary for the object to be sent to the GPU every time the sphere drawing method was called. This would reduce the time taken to draw the object significantly, and therefore reduce the time to render the entire frame by a significant factor, meaning the program would remain visually smooth for higher document counts.

I feel that the approach I used for this project was not entirely correct, Too much time was spent implementing the basics of the graphical engine, such as object selection in the 3D scene. This development imbalance led to a lesser amount of work being devoted to the interaction itself, the core aim of this project. It may have been more appropriate to search for a suitable graphical library that operated on top of OpenGL and use this to draw the 3D portion of the program. By doing so, the rendering process would likely have been significantly less complicated and possibly even resulted in a more appealing visual result.

## Future Work

There are many ways in which the aims of this project can be extended to create a more complete and useful prototype. These improvements exist in a range of different areas including the visual look of the program, the interaction it provides the user with and the implementations of these aspects. There are also some additions which I feel would have been appropriate but I ran out of time to implement them.

## Implementation Improvements

One such item of functionality was mentioned in my interim report, where I discussed using an octree as an additional representation of the concept-map. One of the core benefits to using a spatial data structure such as this is that it would allow the program to be more fully aware of how each object was placed within the 3D space. From this it would be possible to calculate exactly which objects are visible to the user given the current camera position and the angle it is facing. I had intended to implement this functionality into the prototype described in this report, however I was unable to do so due to the time constraints of the project. As such, the first area that I would continue developing if given the opportunity is to finish the creation of this system. By limiting the render cycle to only drawing the objects that are actually visible the general performance of the program will improve as it will take less overall time to draw the scene. This will mean that the program will be capable of drawing a larger number of objects before the time it takes to draw the scene begins to impact on user experience.

To complete the implementation of this, I would first finish the Octree class, ensuring that it altered its internal record of the concept-map whenever changes were made to it by the user. This would likely be done through the Graph class as is described in the Implementation section. As part of the Octree class I would create a method that would take the camera position and direction as its parameters. This would use the previously described octree theory to then create a list of the objects within the visible range of the



camera, this list would then be returned. The Renderer class would use this method at the beginning of the render cycle in order to know exactly which objects need to be drawn so as not to waste any resources.

There are other ways in which performance can be improved, once such method is to use a different form of rendering within OpenGL. When I first began this project I had very little experience of 3D rendering and started by practising with tutorials and examples. However it became apparent to me towards the end of the project development period that the processes I was using to draw to the screen were somewhat out of date or deprecated. For example, each of the statements within the code that draw an object or line are called directly. A more efficient route would have been to store the objects rendered in a buffer and then simply call the buffer when the object needed to be drawn. At this late stage I could have chosen to go back and learn the newer methods but I felt that my time was better spend exploring functionality and meeting the requirements of my project. If I were to continue development on the program, I would certainly seek to amend the code to use these newer methods. Not only would the newer methods lead to a more efficient program overall but it would help to keep my program current and resistant to deprecated OpenGL methods which may be removed over time.

## User Interface and Interaction

An important area of the program's functionality that would benefit from extra development time is the user interaction. While I feel that the prototype I developed demonstrates some well designed interaction such as the mouse selection and view rotation, this could be extended further. For example, a simple change from the user's perspective could consist of allowing them to add documents to a folder by dragging the document itself onto the folder object within the 3D scene. This kind of drag and drop functionality is often seen in modern software, including existing document managers, and as such may be seen by prospective users as an essential feature.

The implementation of such functionality would not be excessively difficult, and would likely have been done by myself if not for the time constraints of the project. The new code would be entered into the `MouseMoveEvent` and `MouseEvent` methods of the `Renderer` class. Within the first method the scene would undergo a redrawing each time the method was called, during this draw phase the object currently being moved would be excluded. This would be necessary as the object is kept under the mouse while it is undergoing movement. From this the class would use the selection method described earlier to determine if any folder objects were under the mouse at the time. Once the mouse is released, the moved object would then be added to the folder if one were found under the mouse. It may be the case that once the object was added it would need to be reset to its original location from before the move. This may be confusing from the user's perspective as the object would seem to 'snap' back into place and as such there would need to be a way of informing the user that the object had indeed been added to the folder found under the mouse.

As this method would involve redrawing the scene for each movement of the mouse, it could be the case that a significant performance penalty is incurred. It would be necessary to explore whether or not this decrease in performance is worth the interaction that the improvement provides.

## Visual Appearance

While this project has primarily been aimed at exploring 3D rendering and interesting functionality to organise documents, there is a visual aspect to the program that must be considered when dealing with any user-facing software. These additions could be made by a future developer as they saw fit, it is likely

however that the functionality listed above and otherwise would take precedence over any visual improvements.

The first change I would make if development were to continue would be to change the 3D representation of a document to use the document's icon or the program that it opens with by default. This would be a feature borrowed from traditional document managers where each document or file is either given a unique icon (usually reserved for programs or other software) or uses the icon of the program that is set to open it by default. By using an icon to represent the document the user would be more able to recognise the use of the file rather than having to look directly at the name. This proposed feature may be of more use within a folder or hyperedge, where there would be a smaller subset of documents, and as such it would be easier to recognise individual documents and their purpose.

It is likely that this change would require somewhat significant changes to the source code of the program, as each icon would need to be stored within the program to avoid the expensive process of retrieving it during each render cycle. Each icon would also need to be rotated during the render cycle so that it is always facing the camera and therefore visible to the user. In addition there is an argument to be made that by using a 2D representation for the documents within the 3D space it would not take significant advantage of the 3D rendering available to the program.

One of the core issues I encountered when developing the object rendering system involved the process I was using to draw fonts. As is plainly visible on the program the textual labels for each object are anchored to the bottom left, which is positioned approximately at the same point as the object itself. The method I used for drawing these labels simply involved using a series of differently sized fonts and then picking an appropriate font object so that closer objects in the scene had larger labels. When developing this part of the code I was unable to find a simple way to justify the text to the centre, meaning that the labels look somewhat out of place in the scene and are somewhat difficult to read due to contrasting with the objects themselves.

I did consider one method of moving the text entirely to the side of the object and that was to calculate the position to the right of the object from the user's perspective. This would require taking into account the camera position and viewing angle. Although I have not attempted to implement this method, there is the possibility that calculating a nearby position for each and every object displayed for each render cycle will take a relatively significant amount of processing time. This decrease in performance may lead to visible stuttering for the user.

## Conclusion

The aim of this project was to explore the use of a three-dimensional display for the task of organising documents. The documents were to be organised using a concept-map structure, where each document may be connected to many others and the connection itself describes the relationship between the two. Certain functionality was specifically defined within the aims and objectives of the project, these included:

- the ability to tag documents with multiple keywords
- view a subset of the concept-map by means of a folder system
- save and reload the concept-map structure without user prompting
- to improve the efficiency of the program by using a spatial data structure to cull parts of the space from the render cycle.

This project was fairly successful, all but one of the core aims of the prototype were implemented. The feature that was not implemented was the efficiency benefit by using a spatial data structure and could not be added due to time constraints and a difficulty grasping the mathematics required. The time spent attempting this feature took away from the development of complex interaction, which was the core focus of the project. Therefore, although all the interactivity is present in the prototype, it is possible the implementations would have been better designed out if they had received their full development time.

Due to the failure to implement one of the objectives the prototype suffers from performance issues when dealing with concept-maps involving more than 300 or so documents. At this level, the frame rate is reduced to a point where visual stuttering is noticeable. The performance issues however do not extend beyond the poor rendering speed, memory usage is efficient and does not lead to issues. It is my belief that implementation of the incomplete aim will reduce the performance issues and this should be attempted in any possible future work.

The program developed is functional and allows a user to create a concept-map data structure in 3D that consists of documents, folders, tags and custom labels. The features present have been tested by users and on average found to be appropriate for the environment in which they are used. Inexperienced users found the program slightly confusing from a visual standpoint as the users found it difficult to differentiate between one document and another.

From this project I believe that 3D space can be used effectively in the context of a document management problem, however as this process has shown, users are easily dissuaded from using the program if it does not have an appealing visual style. This could have been achievable within the scope of this project if more time had been allocated to the visual aspect of the prototype. This time could have been sourced by using an additional library to handle the graphical engine aspect, this would have freed up the time taken to develop the object selection code for use on visual style. Overall I feel that while the project largely achieved its aims, a better prototype would have been created had the development plan been more thoroughly focused on the crucial aspects of the project, rather than creating the basic systems behind the program.

## Reflection

I've learnt much throughout the duration of this project, including both technical aspects such as using OpenGL and C++ and in project management.

Throughout the project I met regularly with my project supervisor, usually just to discuss the work that I had achieved over the last two weeks and any issues I may have had. Looking back now I see that I did not make full use of my supervisor's help, there were many times where I was struggling to understand how something needed to be done. However instead of asking my supervisor for help, I tried to solve it myself, at times to the detriment of the project. An example of this is the failed Octree implementation, where I struggled to understand the mathematics behind tracing a vector through the data structure. I believe that my supervisor would have been able to point me in the right direction had I asked.

When I began this project I had planned to focus on the interaction provided to the user, however I failed to accurately assess how much time it would take to implement the 3D engine aspect of the program. As such I declined to use a higher level library for my display and instead used direct OpenGL with a few add-

ins such as GLUT and GLU. The time it took to learn these new technologies and apply them with a reasonable amount of success drastically took away from the development time dedicated to interaction.

Not only did I fail to accurately assess how long it would take to learn OpenGL and its add-ins but I inadvertently used out of date teaching materials to understand the libraries. As such the prototype ended up using very out of date techniques for drawing objects, rather than direct draw the objects within the `paintGL()` call tree I should have been simply telling OpenGL to render an object that was previously stored in the GPU memory block. This deprecated process meant that for each object I was drawing, the CPU had to calculate the triangles that made up the shape. In the case of the prototype, where each object was either a sphere or a line, this meant that the CPU was constantly recalculating the same shape over and over. I feel that if I had simply taken some more time to do background research about how to properly use OpenGL I would have been able to improve the performance of my program significantly. Unfortunately I was fairly eager to get started on the actual programming and the research was not as thoroughly done as it should have been.

There are some areas of this project where I feel I did well and made the right choices. When it became clear that using ray-casting as a means of object selection was going to take far too much time to implement I decided to switch a much simpler method and discard the previous two weeks of work in order to move forward. I may not have made this decision had I not had previous experience with the colour picking method and instead may have spent several more weeks developing ray-casting, likely to the extreme detriment of the interaction aspect of the project.

## References

1. Vertex Specification – OpenGL.org

[http://www.opengl.org/wiki/Vertex\\_Specification](http://www.opengl.org/wiki/Vertex_Specification)

## Software Library References

These are not referenced in the main text, however may prove useful for background information

1. OpenGL – The Industry Standard for High Performance Graphics

<http://www.opengl.org/>

2. Qt Project

<http://qt-project.org/>

3. The freeglut Project :: About

<http://freeglut.sourceforge.net/>

4. SQLite Home Page

<http://www.sqlite.org/>

## Appendix

### 1. Results table for performance analysis

Document Count	Initial Memory (MiB)	Memory After Rotation (MiB)	Average Draw Time (s)
100	24.8	27.5	0.01286
200	25.2	29.6	0.02015
300	24.4	31.6	0.02260
400	25.6	31.8	0.02750
500	25.7	29.8	0.03571
600	24.8	30.0	0.03710
700	24.9	32.1	0.03961
800	25.0	32.3	0.04258
900	25.7	32.1	0.04586
1000	25.2	32.5	0.05099
1100	26.8	33.3	0.05458
1200	25.4	30.7	0.06338
1300	25.5	32.9	0.06806
1400	26.4	32.6	0.06612
1500	26.6	33.0	0.06901

2. Questionnaire for user testing

## ConceptTop – Concept-map based desktop

### User Interface Evaluation

#### **Visual Aspects**

1. How would you rate the overall visual appeal of the program? (1-10)
2. a) To what extent are the documents sufficiently distinct? (1-10)  
b) Would the document icon be a preferable representation?
3. a) Are the textual labels readable?  
b) If not, how would you improve them?

#### **Interaction**

4. How would you rate the overall interaction of the program? (1-10)
5. a) Does the mouse function as expected?  
b) Is it missing any functionality?
6. a) Does the folder system work as expected?  
b) Are there any improvements you would make?
7. a) Does the tagging system work as expected?  
b) Are there any improvements you would make?
8. Do you find the method to move objects to be intuitive?

#### **Other**

9. Are there any other ways the interface could be improved?
10. Any other comments:

### 3. Results for user testing questionnaire

#### Users more familiar with computers

Question	Part		Person 1	Person 2	Person 3
1		How would you rate the overall visual appeal of the program? (1-10)	5	4	6
2	a	To what extent are the documents sufficiently distinct? (1-10)	4	3	5
	b	Would the document icon be a preferable representation?	Yes	Yes	Yes
3	a	Are the textual labels readable?	Yes	No	No
	b	If not, how would you improve them?		Position underneath the document	Needs to be clearer
4		How would you rate the overall interaction of the program? (1-10)	5	4	5
5	a	Does the mouse function as expected?	Yes	Yes	No
	b	Is it missing any functionality?			Context menu for joining or tagging
6	a	Does the folder system work as expected?	Yes	Yes	No
	b	Are there any improvements you would make?	Drag and drop objects into folders	Less free movement for documents inside the folder maybe	Nesting folders within folders
7	a	Does the tagging system work as expected?	Yes	Yes	Yes
	b	Are there any improvements you would make?	More prominent display of the tags of a document		
8		Do you find the method to move objects to be intuitive?	Yes	Yes	Yes
9		Are there any other ways the interface could be improved?	A snap to grid option might be good		
10		Any other comments:		Restricted camera makes it easier to see what you're doing	

#### Users less familiar with computers

Question	Part		Person 4	Person 5
1		How would you rate the overall visual appeal of the program? (1-10)	3	3
2	a	To what extent are the documents sufficiently distinct? (1-10)	3	2
	b	Would the document icon be a preferable representation?	Yes	Yes
3	a	Are the textual labels readable?	Yes	No
	b	If not, how would you improve them?		Not that clear, would help if they didn't overlap
4		How would you rate the overall interaction of the program? (1-10)	4	3
5	a	Does the mouse function as expected?	No	Yes
	b	Is it missing any functionality?	Could use drag selection to tag many documents	
6	a	Does the folder system work as expected?	No	Yes
	b	Are there any improvements you would make?	I expected the document to disappear from the outside area	
7	a	Does the tagging system work as expected?	Yes	Yes
	b	Are there any improvements you would make?	Tag more than one document at a time	
8		Do you find the method to move objects to be intuitive?	Yes	Yes
9		Are there any other ways the interface could be improved?		
10		Any other comments:	Not sure how useful 3D is for this purpose	

