CHALMERS

Interactive Visualization of blog searches

Master of Science Thesis in Computer Science and Engineering

Daniel Svensson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Göteborg, Sweden, May 2009

Interactive Visualization of Blog Search

Daniel Svensson.

Examiner: Staffan Björk

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover: An image of the first draft of concept 2 [see Chapter 5.2.3].

Department of Computer Science and Engineering
Göteborg, Sweden May 2009

CHALMERS

Abstract

This report describes the project Interactive Visualization of Blog Searches from research and concept creation to the final implementation results. The purpose of this project was to develop a visual interface for browsing search results from the Twingly blog search engine which intends to simplify the users screening process and encourage browsing. The methods used to develop and implement the search interface and the two final results, the concept and the implementation, are presented in this report.

The concept is an ideal description of the proposed functionality and components of the blog search interface developed during the initial phase of this project. Three different concepts were developed and one was chosen for further development, all three concepts are presented. The final concept which was chosen for implementation is an augmented version of an existing interface originally used for presenting album covers in Apple's application iTunes.

The implementation is the realization of the concept and was implemented as a Flash application using the programming language ActionScript 3 and the 3D engine Papervision. The final application implements most of the functionality and graphical content described in the concept but unfortunately suffers from a low frame rate. The main reasons for this low frame rate and proposed improvements to the implementation are discussed at the end of the report.

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Göteborg, Sweden, May 2009

Table of Contents

1 Introduction

This chapter explains the purpose of the thesis, introduces the stakeholder, states the scope and constraints of the project and gives a short description of the document structure.

1.1 Stakeholders

The idea for this project was presented to me by Martin Källström, CEO at Twingly, when I contacted him in search of a project for my master thesis in June 2008. Twingly is a company situated in Linköping, Sweden that provides a number of services related to blogs. Their main product, Twingly Blogstream, is a widget that links back to bloggers linking to the website where the widget resides. Several online newspapers use this tool to track bloggers' comments on their articles. Twingly also provide a spam-free blog search engine, which is the focus of this thesis.

1.2 Purpose

The purpose of the master thesis was to implement a search interface for the Twingly blog search engine that allowed the user to browse the search results more efficiently while also providing a richer experience than a traditional text-based search interface provides. This problem falls under the category of data visualization which has become an increasingly popular area of interest amongst web based service providers (see background). With a background in 3D graphics I also wanted to explore the possibility of incorporating a third dimension to the solution. The following problem was the investigation of this project:

How could a visual search interface for the Twingly blog search engine be designed and implemented, using 3D graphics, to ease the user's screening process while at the same time encouraging browsing?

1.3 Constraints & Scope

In order to answer the question above, I implemented a demonstrator incorporating design choices which were based upon research of related work, GUI design literature, metaphors from the gaming industry and blog specific knowledge provided by Twingly.

The demonstrator was implemented to be launched and used as a beta version on the Twingly website for further testing and evaluation. The testing and evaluation on the Twingly website was however not planned as a part of the master thesis and is therefore not analyzed in this report. Due to the time constraints of the project no planned user tests were carried out, the tests involved were small scaled and involved only people directly connected to the project. In order to be launched on the Twingly website certain minimum requirements needed to be fulfilled. Twingly stated the following requirements:

- Must have responsive controls,
- A graphical profile matching the current Twingly profile
- Frame rate >= 24 fps
- Fault tolerant

Naturally only the last two points could be measured deterministically while the first two require testing and evaluation on a more subjective level.

As the project was targeted towards blog search engines and specifically for the Twingly search engine the result should *not* be viewed as a general solution on how to best create an interactive visual search interface. Implementation of search algorithms and data structures was not part of the project; these were provided through the existing search engine by Twingly.

Legal aspects such as copy right infringement have not been taken into account when creating the concept and demonstrator.

2 Background

This chapter describes the problem domain, "Data Visualization," and how it relates to search engines and also gives a description of Twingly's current blog search engine.

2.1 Data Visualization

Every day people are subjected to vast amounts of information. All information whether it be a television broadcast, a traffic signal or an advertisement in a newspaper has at least one thing in common, it is all screened by the potential receiver. The receiver, in this case a human, can choose to either process the information or to reject it. Decisions like these are made continuously and often subconsciously throughout a person's day. The ability to screen information is an essential skill as it serves to keep the brain from being overloaded with too much data. Fortunately the human brain is rather proficient at handling everyday information such as paying attention to a traffic light when driving a car or sifting through the ad section in a newspaper (1 pp. 215-216, 443).

When the source of information is too vast for the human brain to sift through manually, however, such as a large database or the World Wide Web (www), we must rely on search engines to screen the available information for us. Although the search engine can substantially minimize the search space, we often do not want to constrain the search to the extent that potentially interesting information is disregarded. Therefore we are many times left with a list of search results which must be browsed manually. As the list exceeds hundreds of search results, which is often the case when using a search engine on the web, the amount of information available becomes difficult to browse. One of the main reasons for this is that the brain is forced to read the information thoroughly, rather than just perceiving a light signal or quickly browsing a headline.

Several search engines have started researching and in some cases already provide visual search interfaces. A visual search interface differs from the traditional text based interface in that it provides a richer amount of visual data which is meant to aid the user in quickly browsing the search results. The incentive for providing this powerful tool is of course to attract as many users as possible in what has evolved to be a rather competitive market.

The term visualize can have different meaning depending on the context, therefore a definition and a short discussion of the term that will be assumed throughout this report is presented.

> ***Visualize*** *– to make perceptible to the mind or imagination* (2) .

Assuming this definition one might argue that a simple text based search interface such as *Google* implements a visualization of search results. To further clarify the definition assumed in the report, a

constraint that the visualization must incorporate graphics other than text such as images, graphs or icons as a main component is added.

2.2 Twingly Search Engine

One of Twingly's free products is the Twingly blog search engine which provides bloggers and blog readers with a powerful tool to find either a specific blog or blog posts associated with a topic or search phrase (3).

In order to compete with similar search engines Twingly wanted to investigate the possibility of enhancing their search engine with a more visual representation of the data, combined with a new feature which allows the user to browse a set of blogs related to a particular blog. Being able to browse related blogs can aid the user in finding results that he/she did not know they were looking for and also increase the chance of finding what they were looking for. The visual interface developed as a part of this project uses the blog search (*as opposed to the post search*) in combination with the "related search" feature as the main underlying functionality, i.e. for every blog result a list of blogs related to that blog is available. Note that this underlying functionality is provided by the Twingly search engine and not a part of the interface's functionality.

An important aspect of this new method of browsing blog results is that it introduces a new dimension to which the user can navigate. In a traditional search engine the user has the choice of browsing pages of search results by navigating backwards or forwards between these pages. When providing the user with yet another dimension it is important not to confuse the user with a poor method of navigation. Twingly wanted a method of visualizing this navigation that would aid rather than confuse the user.

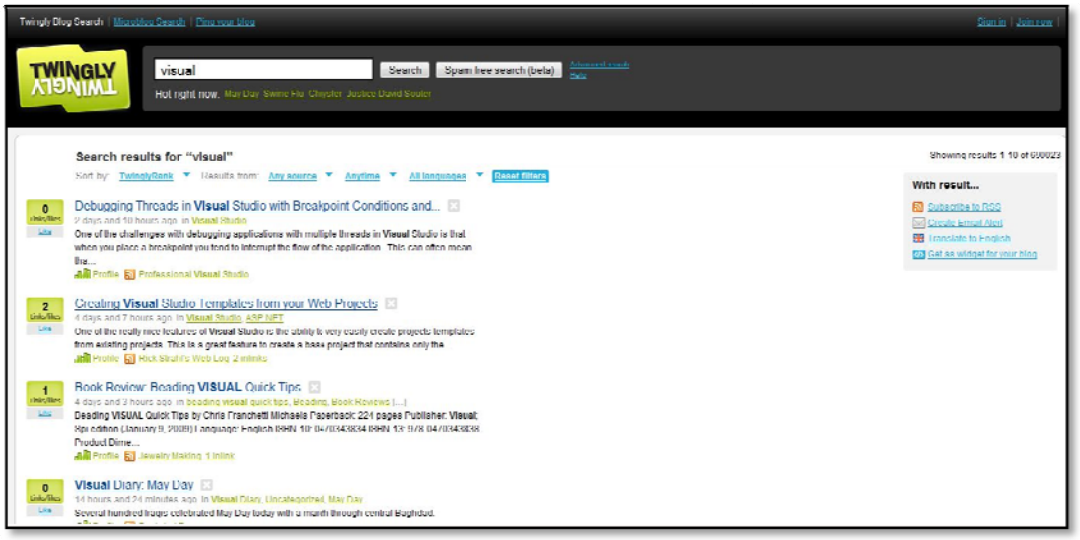

Figure 2: A screen shot of the search results using the current Twingly blog search engine (3).

3 Research & Theory

This chapter covers the theoretical background of the project. Related work summarizes the research made on projects related to this one, mainly explaining the main categories of visual search engines available. Interaction & GUI design focuses on theory, mostly from literature related to graphical user interfaces. The theory used during the software development phase has been omitted from this section, references and explanations are given in the implementation section instead. This chapter is not a regurgitation of all theory used in the project, only the most used and referenced theory is presented here.

3.1 Related work

Numerous methods and concepts exist for visualizing search results and other data on the web. The visual representation is usually highly influenced by the underlying algorithms and datastructures that organize the results and also the available information for each result. Depending on the available methods in which the data can be sorted and grouped different opportunities for presenting the data in a browser friendly manner present themselves, e.g. if the search engine has a function to group the results by language, then this might qualify as a way to group and visualize the results.

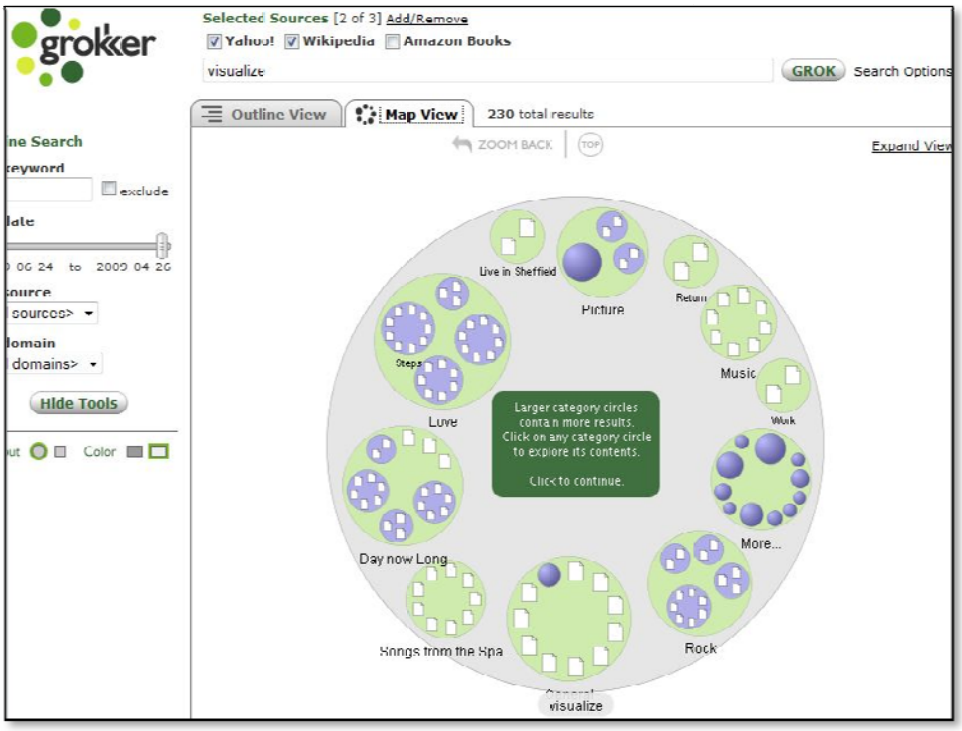

Figure 3: An example of clustering from the Grokker search engine (4).

A common method for presenting search results is to group them according to some attribute and then connect each group with a related group according to some common relation, ultimately building a graph of clusters or clouds [*see* Figure 3 *ovan*]. Another way to explain it is as a huge mind map that is generated by the search engine. The following are a few examples of search engines using this method:

- Kartoo (5)
- Quintura (6)
- Grokker (4)
- Flowser (7)

Other visual search engines only take advantage of the available graphical content that a search result might have, such as a screen shot of a web page [*see* Figure 4 *nedan*]. Some of these search engines simply append a small image of the screen shot to the text results, but the most common method by far is to visualize it by implementing some variation on the CoverFlow concept which was first seen (commercially) in Apple's iTunes application where it is applied to album covers. Several search engines have adopted this method, here are a few:

- Viewzi , provide several interfaces (8).
- SearchMe (9)
- Youtube (10)
- Coverpop , collage implementation (11).

Figure 4: A screen shot of the Viewzi search engine for web pages.

Both of the methods mentioned above seem to be the two most common ways to visualize search results from a search engine (other than simply text based), though the implementations may differ somewhat in appearance and functionality. These methods are applied to all kinds of search engines whether they search for web pages, blogs, videos or images.

3.2 Interaction & GUI design

This section is a summary of theory pertaining to the interaction- and GUI design methods used and reviewed during the course of the project. Subjects from conceptualization to prototyping and testing are covered.

Design is a concept which spans over many different disciplines such as graphical- , industrial- interior- and game design. This project spans mainly two disciplines, namely interaction design and

game design. These two disciplines share many of the same design principles as they no doubt are closely related; a game is a highly interactive product and many interactive products (aside from games) incorporate ideas from the gaming industry. This section covers theory from both disciplines.

3.2.1 Goal Directed Design

Interaction design is highly focused on aiding the user as much as possible by for example hiding complexity and adhering to people's goals and expectations. Goal-Directed design is a particular approach to interaction design which especially focuses on people's goals and that strives to overcome the shortcomings of the manner in which digital products are designed today. In About Face 3, The Essentials of Interaction Design, the authors devote a whole chapter explaining these shortcomings, but they also offer a summary in the form of three main points explaining why the industry so often fails when designing digital products:

1. *"Ignorance about users,"*
2. *"a conflict of interest between serving humans needs and construction priorities,"*
3. *"and the lack of a process for understanding human needs as an aid to developing appropriate product form and behavior."*

In essence these three points point out the fact that there seldom is a qualified person and/or process in place to play the users advocate when digital products are developed, that is, no one is focusing on the user's goals and needs (12 pp. 8-9).

Goal directed design is a model that attempts to fill the gap that resides between market research and implementation of a product. The requirement list produced by a market research does not suffice as research material upon which design choices can be made, furthermore these choices should not be made by engineers or programmers. Instead a 6-step (with iteration) model is proposed, that should be executed by Interaction designers, consisting of the following points:

1. **Research** – users and the domain.
2. **Modeling** – users and the context.
3. **Requirements** – definition of user, business and technical needs.
4. **Framework** – definition of design structure and flow.
5. **Refinement** – of behaviors, form and content.
6. **Support** – development needs.

The heart of the goal directed design process is being able to define the user's goals and sub goals as they pertain to the product, as opposed to the method of simply specifying features and functions that may or may not adhere to the user's goals.

In game design this model is mirrored by a design process which intimately involves the user/player in the design process from idea to prototyping. The interaction designer is exchanged for a game designer, but in essence they both have the same job, namely to be the users/players advocate. The main point from both disciplines is to not get distracted by details before the user's goals and way of thinking has been contemplated (13 pp. 17-26).

3.2.2 Conceptualization & Prototyping

This section describes methods designed to stimulate the mechanisms in the brain that allow ideas to be spawned and formulated and furthermore how these ideas can be formalized into concepts and working prototypes.

The activity of spawning ideas is often thought of as a talent or property that “creative” people possess from birth. Another theory describes creativity as a skill which can be achieved best by exercising different methods and routines. In the book Game Design Workshop, idea creation is proposed almost as a lifestyle where one should constantly document ideas or intriguing thoughts that pop into ones heads, with the simple justifications that they will be otherwise forgotten and that ideas can be recycled and used for future projects (13 pp. 140-142).

A perhaps more practical method of spawning ideas is also proposed, namely different techniques of brainstorming. Several methods from creating trees of interests and passions to writing down lists, making idea cards and doing randomized researches in dictionaries have been developed in order to get the idea process flowing (13 pp. 143-144). The key property relating all these methods is to not constrain or censor oneself when executing them. Alan Cooper advocates that a brainstorming session should be carried out after an initial period of research on the problem domain and user groups has been carried out, even though this introduces the dilemma of having preconceived notions of the final solution which contradicts the purpose of brainstorming (12 p. 117). No more than 1-2 hours of brainstorming is recommended.

The next step in the conceptualization process after the brainstorming phase is to edit the ideas in an organized fashion by ranking them and identifying their strengths. The most highly ranked ideas should then be picked and brainstormed upon again. This process is repeated preferably with feedback from other people between each iteration until a single idea remains that can be worked on (13 p. 149).

When an idea has been formalized and all formal elements have been decided upon a prototype of the concept is proposed so that any flaws or weaknesses can be identified before a full scale implementation is started. The idea is to make a simple prototype that encapsulates the core functionality and elements of the game, therefore very little time should be spent on details and visual appearance at this stage.

The ultimate goal of the prototype is of course to test it on the targeted user group so that feedback can be received from the people whom are actually going to use the system.

4 Project Plan

In this chapter a description of the initial time plan for the project is presented to give an idea of where the emphasis of the project is intended. The project plan is intentionally rough as there are several elements of the project which are difficult to assign a time estimate to, e.g. the implementation phase which entails research and a learning curve of the chosen API for 3D graphics. The project is planned to span over approximately 20 weeks, this time period is divided into two main phases, separating design from implementation.

4.1 Design Phase

The design phase is divided into three different stages of work.

1. *Research on data visualization, related work, Interaction & GUI design and available software tools and API: s.*
2. *Concept development and design of interactive components*
3. *Design of visual content*

These three points could have been subdivided into further and more explicit categories; this was not done in order to allow more flexibility in the creative process. Research was planned as the first part of the project so that a theoretical foundation from which conclusions could be drawn could be used in the following development process. The focus of the research phase is to get a better idea of related work and suitable tools that can be used for the project. Very little time will be devoted to research on theory as this project is highly focused on implementing a prototype. Most of the theory will be reviewed when needed during the course of the project.

Separating the concept development and design of interactive components from the graphical design intends to simplify the design process by reducing the immediate complexity of the task. Relatively little time, 5 weeks in total is allocated to the design phase as the implementation of the interface requires most of the available time.

4.2 Implementation Phase

1. *Development of interface towards Twingly search engine*
2. *Implementation of concept*
3. *Quality assurance*
4. *Documentation*

The application needs an interface allowing communication with the Twingly search engine, i.e. enabling requests to the server and reception of results from the server. This functionality is planned for the beginning of this phase as it is an essential functionality and a good starting point from which the rest of the application can be built. The implementation is the main focus of this project and will therefore be allocated most of the available time in the project plan. It is a deliberate choice not to plan the implementation phase strictly as there are too many uncertain parameters to consider. It is however planned that an Object Oriented approach with an iterative method will be used for this phase. A slot for quality assurance and documentation is reserved as this is an essential part of any serious software development process. Documentation of code and project progress is a continuous part of the process, documenting the evolution of the application, software architecture and amendments or alterations to the design concept.

5 Development

This chapter describes the evolution of the project from research and design to final implementation, focusing on design and implementation choices as well as problems encountered during the course of the project. The results of this development process are presented in chapter 6.

5.1 Research

When the project had been formally decided upon with guidelines for what it would encompass, several technical and strategic decisions remained. These included:

- Choice of programming language and 3D graphics API.
- Scope of application, i.e. how much functionality it would provide.
- Number of design concepts to be presented.
- Methods to use for design and implementation.

To base these decisions on more than intuition an initial period of research was necessary. Up to this point in the no tools or methods had been researched, although Martin Källström at Twingly had proposed some alternatives.

5.1.1 Technical research

In order to choose a suitable programming language it was necessary to first investigate existing graphics engines that supported 3D graphics in web browsers. This order was chosen as it was likely that the graphics API would support only specific programming languages. OpenGL and DirectX were ruled out as they were not supported by any browsers or browser plug-ins.

Martin Källström had early on mentioned Papervision 3D [see appendix A], a graphics engine compatible with Flash player, as a possible candidate for rendering the 3D graphics. Flash player is a plug-in compatible with the most popular web browsers and free to download. Swift3D was also considered as a possible 3D engine but disregarded because it did not offer as much functionality when it comes to creating and manipulating three dimensional objects.

Papervision 3D is written in the programming language ActionScript 3 and hence designed to be used with Flash player. For this reason ActionScript 3 was chosen as the programming language to be used for this project. Silverlight which is Microsoft's corresponding application to Flash player was also considered for the project, but disregarded as it was still quite new and lacked support for any 3D plug-ins or API's.

Having chosen a suitable programming language and necessary libraries it was natural to decide on a methodology for the implementation phase. As this project only involved one programmer, me, and a relatively small application it was not necessary to spend a lot of project time on documentation and other administrative protocols normally associated with software development methods. It was however necessary to structure the work in order to devise a plausible plan and to make a time estimate for the implementation phase. In order to simplify the programming phase a suitable Integrated Development Environment (IDE) was necessary. Unfortunately most of the IDE's associated with flash are commercial products which are not free. Adobe offers two commercial products for creating flash content, Adobe Flash CS4 and Flex Builder 3. Flash CS4 is a graphical editor for creating flash content while Flex Builder provides a development environment complete with a

debugger, syntax interpreter and testing tools. A 60 day trial version of Flex Builder 3 was chosen as this project would be highly focused on ActionScript 3 programming in which case a debugger is an essential tool.

With the programming language, necessary libraries and a method for implementing the software in place, it was time to decide on the functionality provided by the interface and also how many different concepts would be proposed in the design phase. The plan was to propose a couple of different ideas for the interface and then implement one of them as a demonstrator for testing. These choices depended a lot on how much time would be needed for the implementation and for learning the different tools and programming languages. An estimate of eight weeks was calculated for learning ActionScript 3.0 and Papervision 3D and for implementing the proposed interface. With time reserved for quality assurance and documentation only five weeks remained for the design phase [*see Chapter 4*]. It was estimated that 2-3 concepts could be proposed within this time span with complete graphical profiles for all the concepts.

5.1.2 Market research

One of the most important tasks in this part of the project was investigating related work. Related work is not only a good indicator of what not to do but was also used as a source of inspiration (12 p. 57).

No previous attempts to create a visual search interface for blogs were found at the time the research was carried out; however, similar attempts had been made for conventional search engines. Viewzi and SearchMe were the two most prominent search engines that implemented a visual search interfaces [see Chapter 3]. These two search engines were tested thoroughly and used as reference material throughout the design process. Specifically it was interesting to study the overall experience when using these search engines and especially noting the benefits and downsides of using visual data in the search results, how the experience differed from regular search engines and what needed improvement.

Figure 5: A screen shot of the searchMe search engine for web pages.

SearchMe provides a search engine which is almost identical to Apples CoverFlow concept [*see* Figure 5 & Figure 6]. One of the immediate flaws that I detected when testing this search engine was that the screen shot textures had a poor resolution and were distorted with a wave pattern. It is worth mentioning that this was a beta version of the search engine at the time of testing. Another perhaps more important observation was that there was no sense of an improvement on the text based search engine. Browsing screen shots of web pages in this manner does not provide a fast overview of the results and the actual screen shot provides very few relevant clues as to what the web page contains.

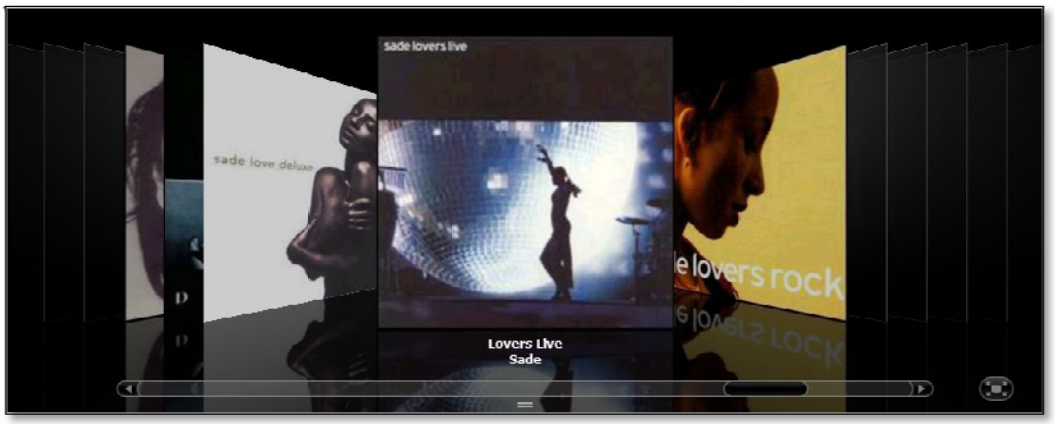

Figure 6: A screen shot from iTunes cover album browser (CoverFlow).

Although regular search engines such as Google and Yahoo essentially provide the same functionality as a blog search engine, it was important to recognize that the user's agenda and browsing pattern may differ when using one or the other. Several questions sprung to mind when researching these search engines:

1. How does searching for a web page or topic, using a search engine differ from searching for a blog or blog topic, using a blog search engine?
2. What kind of information is most useful when sifting through the search results?
3. Is the same information useful for blogs?

These and several other questions are answered in the next section which describes the design phase.

5.2 Design

The design phase started immediately after a sufficient amount of research material had been acquired and analyzed. This part of the project ties together theory and research with creativity and aesthetics. Although an initial research period preceded this phase, most of the theory was acquired and reviewed during the design process.

Brainstorming was used as a starting point in order to trigger the creative process [see Chapter 3.2.2]. This method serves to break mental blocks and to ignore stereotypes associated with the problem domain. Ironically it was essential that I left the research material aside when performing the brainstorming exercises in order to come up with innovative ideas.

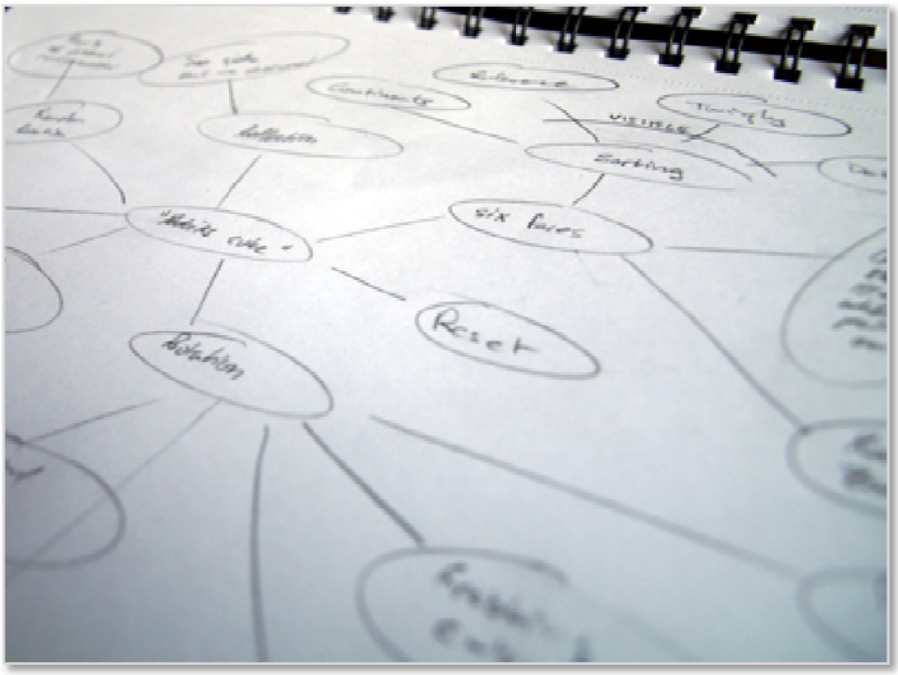

Figure 7: Example of one of the mind maps created during the brainstorming stage.

The brainstorming was split into several sessions, using different approaches and starting points each time. For each session a new keyword was used and from that keyword new keywords associated to the previous ones were spawned, finally creating a graph of words connected by relations. The result was a mind map with different keywords representing either whole ideas or in some cases just random features [*see* Figure 7 *ovan*].

A first concept was quickly derived as a first idea from one of the mind maps. This gave me a chance to figure out and test which tools could be used to create the mockups for the different concepts, a problem which had not yet been considered[*see* chapter 3.2.2 on prototyping]. As the final implementation would end up using real-time graphics and animation a 3D modeling application was used to create images and animations describing the ideas.

5.2.1 Concept one

The first idea was purely metaphorical and was derived instantly from one of the mind maps without further refinement or thought. It used the concept of a rubics cube [*see appendix A*] as a main component of interaction, which rhymed well with the method of using concepts from the gaming industry. The idea was that the blog screen shots would be pasted on the sides of the “cubelets” and rotating the cube planes would expose new sides with another set of blogs [*see* Figure 8 *nedan*].

Figure 8: An illustration of interactions using the rubics cube concept.

The rubics cube was modeled and textured in 3D Studio Max and then rendered with a real time rendering plug-in called oFusion. The reason for using a real time renderer was that the final implementation would naturally be rendered in real time and thus it gave a true representation of the type of graphics that could be expected. Using oFusion three images were created indicating in sequence how the user interacts with the cube.

As mentioned earlier this concept was still in the idea stage when a 3D sketch was created and it needed considerably more thought and refinement in order to be considered a serious proposal. Fortunately the sketch revealed prematurely that this idea was doomed to end up in the scrap bin and it was therefore not developed any further. Several factors led me to this conclusion, the main factors were:

1. The available screen real-estate for each blog was too small, rendering the screen shots useless.
2. Poor mental model (12 p. 28), rotating the cube more than once makes it hard for the user to keep track of order.
3. Insufficient overview, only 9 blogs visible at one time.

In order to make good use of the time spent on this concept as opposed to just scrapping the idea and moving on, the problems and pitfalls encountered so far were analyzed. When reviewing literature on GUI design it was clear that a classic mistake had been made when developing the first concept, namely using a metaphor as a guiding principle. This approach is discouraged, as historically it has led to poor user interfaces (12 pp. 269-285). A goal oriented and idiomatic approach is recommended instead. The goal oriented approach takes the users different goals when using an application into account. These goals can be further divided into sub-goals in order to specify the behavior and features of the application [see Chapter 3.2.1]. Compared to the metaphor-centric approach which simply tries to mimic a real entity the goal-centric approach is more analytical and requires more detailed thought.

Studying the goal oriented approach proved that there would be a tradeoff between incorporating “fun” into the interface versus making it useful and easy to use. I also realized that I had no idea what Twingly actually had in mind for the search interface regarding these fundamentally different approaches. For this reason a meeting was booked with Martin at Twingly so that we could discuss in more detail what he had in mind and so that he could be updated on the project progress.

5.2.2 Meeting with Twingly

At the time of the first official meeting with Martin Källström I was three weeks into the project and had covered an initial research phase, a brainstorming session and made a first concept. This work not only gave me a lot of insight into the problem but had also rendered quite a few unanswered questions. Some of these questions could be answered directly by the Twingly staff, others needed to be discussed and further investigated.

The following topics were discussed:

- Available screen real-estate
- Playful *versus* useful interface
- Related work
- Work methods
- Communication
- Presentation of Twingly search engine
- Search result information
- Format protocols

Available screen real-estate –deciding how many pixels of the screen that were available for the application was necessary to determine at an early stage as this would completely dictate the size and behavior of different components. Twingly proposed two different sizes; 700x520 pixels and 960x720 pixels. The smaller of the two would allow extra information in the browser window that was not directly connected to the flash application, e.g. banners or advertisement [*see* Figure 9 *nedan*].

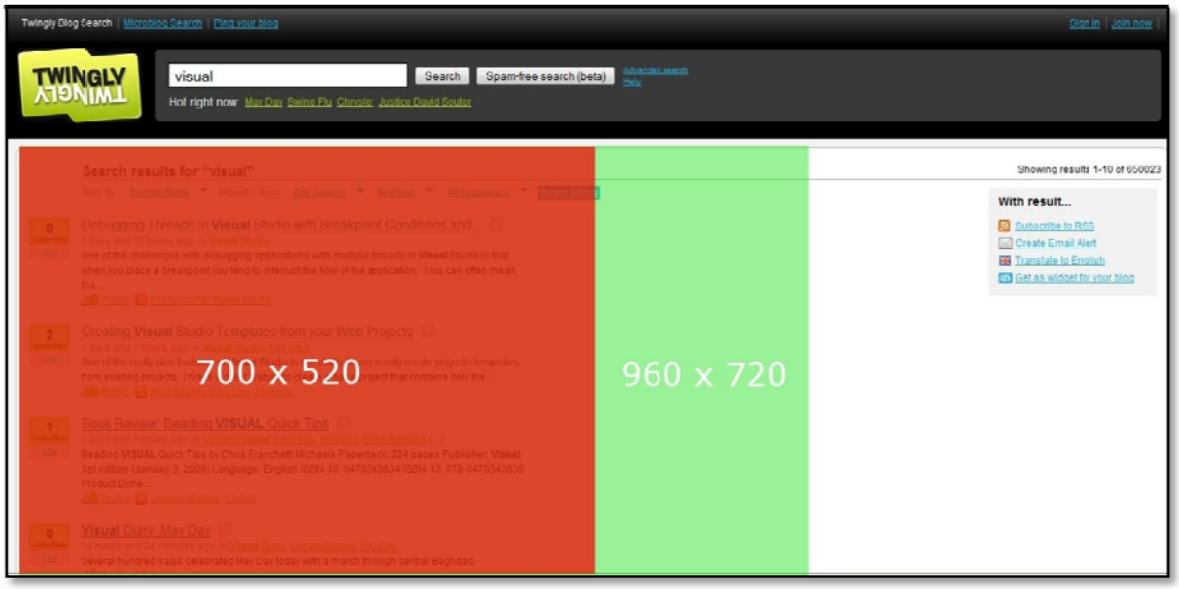

Figure 9: Illustration of the two proposed dimensions for the Flash interface.

Playful *versus* useful interface - designing for a more playful experience or designing for a user friendly interface require completely different approaches. It was clear after the meeting that Twingly wanted to create an interface that was easy to use and that improved the users experience

by aiding the user as much as possible in finding what they were looking for. Another feature that was important to Twingly was that the interface encouraged browsing the search results. In a sense Twingly wanted a user friendly search engine with an enticing GUI.

Related work - we discussed what other companies providing search engines had accomplished as far as visual search interfaces were concerned. Martin particularly mentioned searchMe and CoverFlow as concepts that he found appealing and hinted that these could be worth looking at again. It was decided that the interface would not visualize relations between blogs in a graph structure in the manner that Kazoo or Twingly's summer of code project had done [see Chapter 3.1].

Work methods- we agreed that an iterative method would be suitable for the project as this allowed functionality to be added successively to the application.

Communication – it was necessary to decide how and where the work would be carried out and consequently how communication between me and the Twingly staff was best achieved. We decided that I would work from Gothenburg and that any further meetings at Twingly in Linköping would be booked only if necessary. A technical advisor that could support me with any technical questions was also introduced to me. It was decided that Skype and telephone would be used for communication and that the project status and concepts would be documented on a blog which would be updated frequently.

Presentation of Twingly search engine – an introduction to the existing Twingly search engine was given as well as new features which had not yet been released. The function allowing searches for blogs (as opposed to blog posts) was especially interesting as this would be the version used in the project.

Search result information – we discussed the information that should be available for each search result. This topic was very useful to me because it allowed me to get an idea of how much information each result would be comprised of and consequently how much space each result would require. It was also interesting to discuss if there was any alternative information available that could improve the screening process for the user. The following information was decided upon:

- Title of blog
- Tags – words indicating type of content in the blog.
- Authority – a rank indicating popularity.
- Related blogs
- Screen shot

Format protocols – in order to retrieve the search results produced by the Twingly search engine following a search request, we needed to agree on a suitable format to communicate the information. The formats discussed were XML and JSON. No decision was made during the meeting as it was unclear whether JSON was supported in ActionScript, instead we agreed to investigate the matter and postpone the decision to a later date. An example of the JSON (Java Script Object Notation) can be found in Appendix B.

5.2.3 Concept two

The second concept made use of the experience acquired from the first attempt at creating a visual search interface [see Chapter 5.2.1] and the decisions from the meeting with Twingly. It was clear from my first attempt that a more compact representation of the search results was necessary in order to convey all the relevant information. The use of icons to represent a search result was considered at this point because they convey a lot of information using minimal amount of screen real-estate; allowing more search results to be displayed at once and they can be manipulated more easily as a group, e.g. selecting/excluding several items at one time.

The general idea with the second concept was to combine the use of icons with the third dimension. A screenshot of the "current" blog would be displayed on a tile in the background and in the foreground a 3x3 matrix of icons would be used to represent the search results. Hovering over an icon with the mouse cursor would make that blog result current and the tile display in the background would be updated with the screen shot for that particular blog.

A first sketch of the concept was developed using the same methods used when creating the first concept. Having created a three dimensional mock up of the idea allowed me to quickly identify the difficulties and possibilities with the concept.

In order to realize the concept, the icons which form the matrix needed to be carefully crafted as they were the essence of the idea. The icons had to be:

- compact
- legible
- fast (to interact with)
- informative
- intuitive

The information that each icon would convey:

- Title of blog
- Authority
- Related blogs
- Time stamp
- Tags

The Flipbook icon was developed which meets all the requirements and conveys all the information stated above, but more importantly it solved the related blogs browsing problem. Being able to browse blogs related to a particular blog was an important feature that Twingly insisted be a part of the interface. The authority of the blog was the only information which was not immediately implemented in the Flipbook icon because it needed to be discussed with Twingly first. Several ideas on how to visualize the authority existed, the most obvious was to simply show the numeral and yet another idea was to saturate the icon depending on its authority. In order to show all the tags of the blog a rolling text strip was incorporated into the icon.

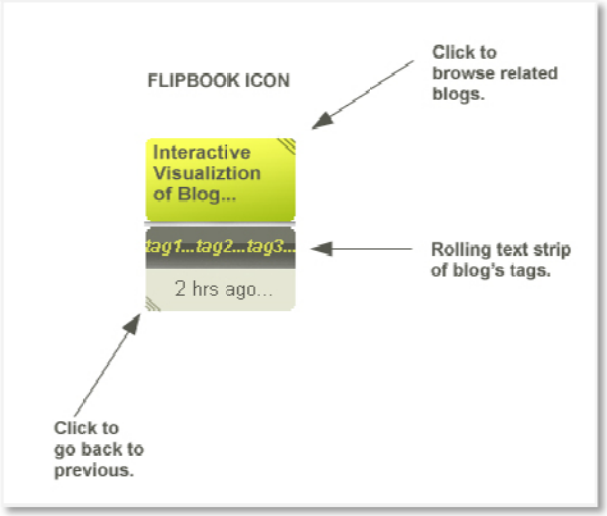

Figure 10: An illustration of the FLIPBOOK ICON.

The related blogs browsing problem was solved by creating a flipbook, hence the name of the icon [*see* Figure 10 *ovan*]. To browse the related blogs the user simply uses the icons flipping function, either backwards or forwards at the top right and bottom left corner of the icon respectively . This simple solution to the problem is quite different from presenting all the related blogs at once, which would require some type of panning and zooming functionality.

Combining this icon with the background tile showing the current blogs screenshot allowed a 3x3 matrix without obscuring too much of the tile.

The next problem that needed to be solved was how the pagination would work. I decided to use a slider which would slide the 3×3 matrix out from the screen and present a new 3×3 matrix. This closely resembles the idiom of clicking a button to switch pages, but enhances the experience by indicating a sliding animation in the direction chosen. The animation strengthens the user's mental model of where the discarded information goes, which is important because it allows the user to find his or her way back to previously disregarded information (12 p. 233).

A natural question to ask at this point was: Is this model an improvement on the text based search result? Answering this question was of course hard to do without conducting user tests. Instead I resorted to logical reasoning, not in an attempt to answer the question but rather to pinpoint strengths and weaknesses.

Using Twingly's existing search interface which is entirely text based it is possible to view 4-6 search (depending on screen resolution) results at one time without scrolling, 10 results are available without switching page. No screenshots are available; instead an excerpt or quote from the blog is available for reading.

With the proposed interface in the second concept it is possible to view 9 results without scrolling or switching page. The screenshot is available when the icon is clicked or hovered over with the mouse (instead of an excerpt from the blog). Because this model does not have as much text it is quicker to

read and quickly browse. It also gives the user an option to browse related blogs without “leaving” the current search space [*see* Figure 11 *nedan*].

Figure 11: The second concept explained with an image.

It is also important to note that the interface could be improved by extending the available screen space to 960×720. The concept was intentionally designed for the smaller resolution 700×520 as it is easier to scale up than down. Another possible extension to this concept is to be able to manipulate the icons, e.g. selecting a group and “saving” them or disregarding them.

5.2.3 Concept three

Although I was confident that the second concept had a lot of potential and was a realistic alternative to be considered for implementation, I had committed to making three proposals and therefore needed one more concept to present to Twingly before a decision could be made and the implementation phase could begin.

It was necessary for the third concept to be completely different from the second one in order to present a true alternative and not just a slight variation in theme. Therefore the icons were disregarded when developing the last concept.

The third idea for the visual search interface was strongly influenced by the search engine searchMe [see Chapter 3.2.1] and can be viewed as an extension of that concept. In addition to pure visual browsing using screenshots this interface also allowed browsing blogs related to the blog currently in scope. This is carried out by using the icon which resembles an unfolded cube, dragging the indicator rotates the cube to the selected side [see Figure 12 *nedan*]. Naturally only five related blogs can be

browsed using this method, however a second implementation was also plausible, namely continuously “unfolding” the cube allowing infinite browsing of related blogs. The information for each blog is presented in a two dimensional window as the blog is in its scope. Only one blog can be in scope at one time, hence the user must use the slider to change the blog currently in scope to view its specific information.

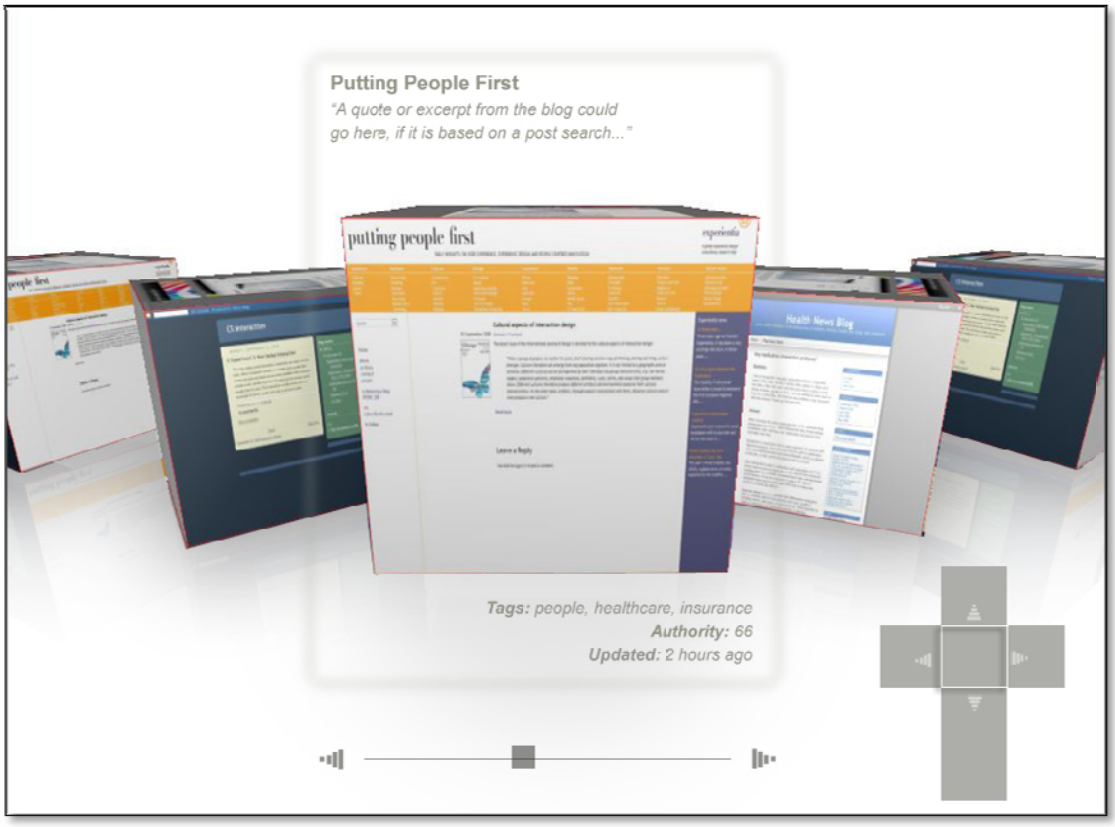

Figure 12 : The first version of concept 3 extends the CoverFlow concept with an information window.

This concept was based on a screen size of 960×720 pixels; a smaller resolution would make the interface hard to motivate as it would render the screenshot useless. Although this concept may be more visually interesting and playful than the first one it does not allow as many search results to be viewed at the same time and only shows specific information for one blog at a time.

5.2.4 Choosing a concept

Despite the fact that the third concept still needed some work and refinement it was time to present the ideas to Twingly. Because concept two and three were so conceptually different it was important to get an indication from Twingly on what they preferred so that all effort could be focused on one of the concepts. The ideas were presented on the project blog with explanations, images and animations.

After Twingly had processed the ideas we had a telephone meeting when we discussed the different ideas and the pros and cons of each concept. Only concept two and three were considered as the first concept was disregarded at an earlier stage by me. It was clear that Martin and his colleagues at

Twingly favored the third concept. The following weaknesses in concept two were pinpointed by Twingly:

- Hard to read scrolling tags.
- Screenshot hard to motivate.
- Browsing of related blogs not powerful enough.

As the third concept was not entirely specified we also discussed how this idea could be improved and augmented. Martin thought that the two dimensional frame with the blog information was a good idea and wanted to keep that feature. The cube indicator however was perceived as a difficult idiom for a first time user, furthermore the cube aspect as a whole was not considered an improvement on the original concept (CoverFlow) especially as it restricted the number of related blogs to five in total.

5.2.5 Tweaking the concept

The telephone meeting with Twingly made it clear that a CoverFlow concept such as my third proposal was what they had in mind for their blog search interface. We had already agreed to keep the 2D information window as an augmentation to the CoverFlow concept, but a good way to solve the browsing of related blogs feature was still needed.

Again it was important to consider the users mental model when creating this navigational feature. Further use of the third dimension was introduced to this end, and a stack of tiles was added to each blog result representing its related blogs. Clicking on the stack triggers the stack of related blog tiles and the camera to “fly” up above the current set of results. This animation clearly indicates where the previous search results are and where the new ones appear. Navigating backwards simply performs the reverse animation, i.e. stepping down one level and repositioning the “related tiles” in a neat stack. With this feature the user can keep browsing related blogs indefinitely. The first sets of search results are always at the bottom, which is indicated by the reflection in the “floor,” making it easy for the user to identify the original search results.

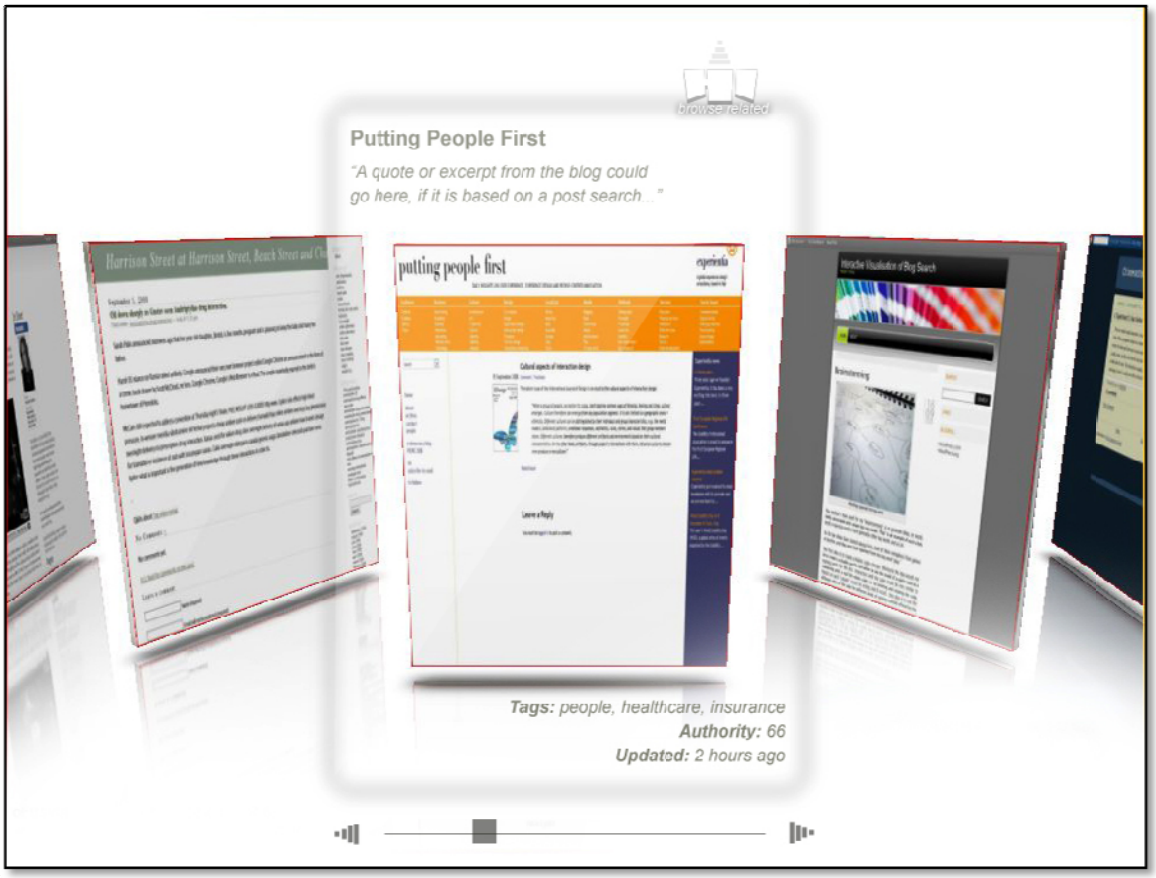

Figure 13: The second version of concept 3, were the cubes were exchanged for tiles and a "related" button was added to the concept.

With all the functional components in place only one part of the design phase remained; specifying the graphical content more strictly. In order not to be distracted by these decisions during the implementation phase it was decided that the last week of the design phase would be spent tweaking and specifying the graphical content more precisely.

Originally the stack of tiles was represented by an icon on the top right corner of the information window, but exchanged for a three dimensional stack of tiles to indicate more clearly how the navigation works for a first time user [*see* Figure 13 *ovan*].

A white background had been used instead of a black one in order to match the current Twingly search engine which presents the results on a white background. The white background had introduced a little too much contrast which can be strenuous on the eyes; therefore slight shading was introduced to the background. A loading screen was also introduced to the concept which would be used when the application is fetching data from the server. The spacing and orientation of the blogs tiles was also tweaked along with the field of view of the camera creating the final version of the concept [*see* Figure 14 *nedan*].

Figure 14: The final version of concept 3 illustrated with an image created in Photoshop and 3D Studio Max.

5.3 Implementation

Though this phase was the main part of the project it was difficult to plan it until the concept was worked out in detail. The concept description dictated which tools I would need to learn and gave an indication of how complex the task was, allowing me to plan the implementation phase.

5.3.1 Plan and priority list

Even with a detailed concept description it was difficult to create a plan as this phase also included learning many of the tools that were necessary for implementation. It would be necessary to implement and learn at the same time if the application was to be completed in seven weeks. The following programming languages, API's and IDE's constitute the package which would be used for the project:

- **ActionScript 3.0** – programming language interpreted by flash player.
- **ActionScript 3 library** – a library with utility functions for ActionScript applications.
- **Papervision 3D** – a 3d engine library written in ActionScript 3.0.
- **Adobe Flex Builder 3** – an IDE for ActionScript programming.

An iterative approach would be used adding new functionality successively. With only 7 weeks allocated for the implementation phase it was necessary to create a priority list of the functionality included in the concept description. The priority list would ensure that the most important functionality be implemented first; possibly excluding extra features should time run out. In order to create the priority list the concept description was broken down into a list of required functionality.

The following priority list was produced:

1. *Communication with Twingly server*
2. *Functional model/datastructure*
3. *Next/previous navigation, including update of the datastructure*
4. *2D GUI with functional buttons*
5. *Functional information window (glass panel)*
6. *"Related browsing" (without animation)*
7. *Loading of screenshots (including logistics/memory management)*
8. *Blog tiles with screenshots (first 3D implementation step)*
9. *"Backwards browsing" (without animation)*
10. *Animation next/previous*
11. *Reflections/Shadows*
12. *Animation related browsing*

Points 1-10 include the minimum requirements of the application. Features such as reflections, shadows and advanced animation would be implemented only if there was time left. The points are described in more detail in the iteration sections below.

Breaking the concept into smaller functional components enabled a bottom up design of the application, allowing me to create simple functions which later could be combined to create the overall application. A bottom up design was suitable as it reduced complexity and allowed me to learn the programming language while implementing simple functions which could be used for the project.

The seven weeks of the implementation phase were divided into three iterations. The iterations each include a number of the above mentioned points for implementation and testing.

- **Iteration one:** 1-4.
- **Iteration two:** 5-7.
- **Iteration three:** 8-12 (11-12 if time remained).

5.3.2 Iteration one

Iteration one also included installing the programming environment (IDE) and the libraries mentioned above (see 5.3.1). Installation of an IDE and libraries that depend on other third party libraries involved setting up dependencies in the project solution, adding environment variables and compiling source code, as an SDK was not used. The source code for Papervision was used instead of an SDK because of the limited documentation of the library. Using the source code allowed me to understand undocumented functions by examining the code.

With a working programming environment and all libraries compiled and setup the first task on the priority list was to set up a communication protocol with the Twingly server. This functionality needed to be in place first as it would allow me to get started with the heart of the application; the model that would handle all the information extracted from the server.

A simplified two step model of the communication between the search interface (my application) and the Twingly server include the following actions and information:

1. ACTION: URL request to the Twingly server.
   INFO: Search phrase URL.

2. ACTION: Response from Twingly server.
   INFO: List of blog results matching the search phrase.

When using a regular search engine the URL (INFO in point 1) is constructed after the user has entered a search phrase into the text field and pressed the search button, the corresponding URL can be seen in the address field of the browser. A search for the phrase "interface" using the Twingly blog search engine for example generates the following URL: http://www.twingly.com/search?q=**interface**. The response from the Twingly server is a new HTML page with a list of the blog results and their corresponding information matching the search phrase.

The most significant difference between a regular HTML based search engine interface and my Flash based search engine interface was that an HTML page was not wanted as a response to the request. Instead the same information structured in a fashion so that it would be easy to parse was desired. A common solution to this type of problem is to structure the information in an XML document.

The first task involved constructing the URL for a search phrase, deciding on a format for the results from the Twingly server and finally implementing a parser that could retrieve the information from the response document.

Creating the URL's and sending the request was a typical example of a simple function that could be used in a bottom up design and which was a simple enough task as a first code snippet in ActionScript. As ActionScript is interpreted by Flash player which in turn is mostly used and embedded in HTML documents a lot of functionality pertaining to web programming is available in standard libraries. Sending a URL request is an example of a function which is available in the *flash.net* library. Therefore this first function simply involved concatenating the search phrase with a URL specified by Twingly and using the library functions to send a request.

When deciding on a format for the results we wanted a simple solution that would not require a lot of work and that could be easily extended in the future. Java Script Object Notation (JSON) was a format that was recommended by my technical advisor at Twingly, Kristoffer Forsgren, as it supports datastructures such as lists and tuples. The main concern with this format was whether there were any existing parsers supported by any ActionScript library. A suitable parser was found by Kristoffer in the ActionScript 3 library (*http://code.google.com/p/as3corelib/*). After having tested the parser we decided that JSON would be a suitable format for the search results, mainly because Twingly had used the format before and also because it would not be necessary to implement a parser for an XML format.

Continuing with the bottom up design I started on the second task for this iteration; creating a model for handling the parsed data. In order to store the parsed data a representation for each blog result was necessary to hold all the information related to it. ActionScript 3 is an Object Oriented programming language supporting all the associated paradigms such as classes, inheritance and packages. A separate class, BlogResultItem, was made for the representation of a blog result. A BlogResultItem had the following attributes with accompanying *get* and *set* functions:

- title : String
- authority : int
- tags : String
- blogURL : String
- screenshot: Image

This is basically a direct translation of what a real blog result item contains. The actual class has more attributes but the ones mentioned above are the most characteristic.

The idea was that each blog result item would be stored in a list or Array. An Array sufficed as a container as the only operations performed would be sequential traversing of the datastructure, i.e. increment or decrement of an index, while also allowing dynamic memory allocation. A model class would hold this datastructure and maintain it by providing different functions such as adding or deleting items, changing current item etc.

In order to test the newly created model with the parsed and organized blog results, tools were needed to traverse the datastructure. The actual decrement and increment functions had already been implemented in the model class with the container, what was needed was a means to trigger these functions. Therefore a simple 2D GUI was implemented. This simple GUI consisted of two navigator buttons allowing forward and backward browsing (increment and decrement in model) and a simple text output showing the title of the blog item currently in scope (item at current index in model).

At this point of the implementation phase a top level design for the application as a whole had still not been made, as the intricacies of ActionScript 3 were still being learned. The problems encountered so far were mostly associated with ActionScript's method for handling events and IO operations. Events are an important part of ActionScript because it is the only mechanism available to disrupt the sequential flow of the program. Many of the functions in the flash libraries use the event model to notify surrounding objects of important events instead of blocking. An example of this is reading from a file; instead of blocking until the whole file has been read, the regular flow of the application continues and the "file reader object" casts an event when it is finished reading or if the read failed. Quite a lot of time was spent learning the event handling system of Flash player.

At the end of this iteration a functional model with a simple datastructure holding the blog results, a means to navigate the results and a working interface towards the Twingly server had been implemented. The model did not support pagination and related browsing (browsing blogs related in some way to the current blog) at this point.

5.3.3 Iteration two

Equipped with a class for storing and maintaining the BlogResultItems and with improved knowledge of ActionScript it was now necessary to make an overall design of the application. A model for the application that would be easy to understand and to rebuild if necessary, mainly because the end result would be delivered to Twingly and possibly be augmented or altered by them. To get a better understanding of the application a flow diagram indicating the sequence that the program would need to follow was constructed. This painted a clear picture of the different components that the program could be split into.

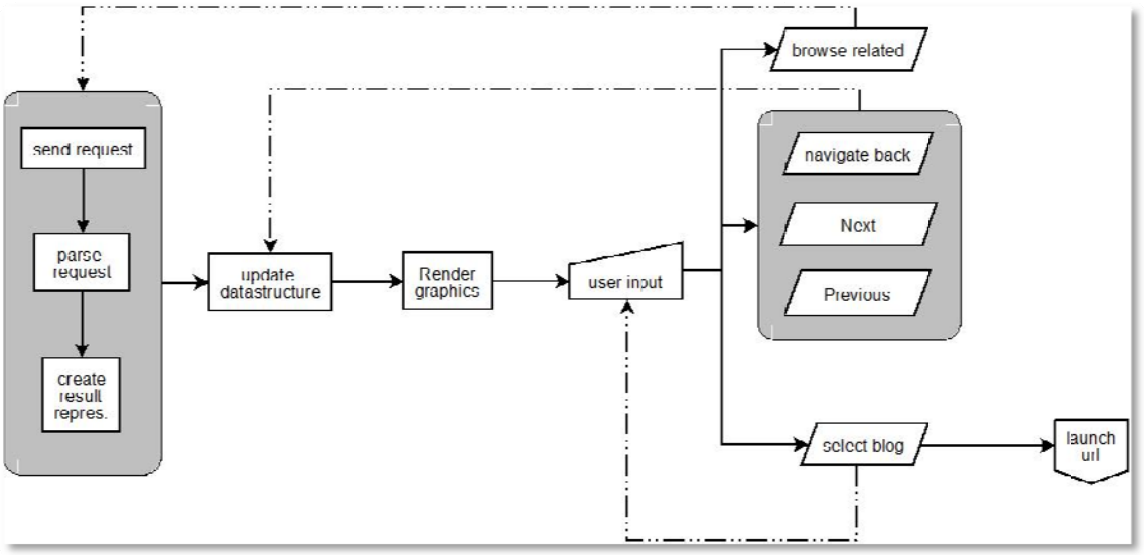

Figure 15: This image describes the flow of the application.

The *package* containing all the classes and utility functions would be called *com.twingly*, naming the package after the domain that the application resides in, in accordance with standard ActionScript 3 coding standards (14 p. 41). A natural design pattern for this application was the Model View Controller (MVC) pattern, which is a simple and easy to understand pattern for applications with graphical user interfaces (15 p. 370). The main idea with MVC is to separate the graphical and controller components from the datastructures and models holding and maintaining the program state. Below is a table with all the classes indicating how they fit into the pattern.

Model	**View**	**Controller**
ResultContainer	Scene	ModelController
BlogResultItem	Navigator	SearchCenter
ResultParser	InfoWindow	
	ArrowIcon	
	BlogTile	

The View components are all derived from the *flash.display.sprite* class which is the base class for most visual components in Flash. With the MVC structure it is easy to re-implement the visual components without having to alter the model and vice versa. As the 3D components of the concept had not yet been implemented this method (MVC) allowed me to first start with a simple 2D graphical representation of the blog results without having to worry about conflicts with the model later on.

As a bottom up method was used without a top level design it was necessary to refactor the code written in iteration 1, so that it would conform to the new design before continuing with the next task. The functions were simply put into appropriate classes and refactored to accommodate the new structure.

With the new design in place task 5 on the priority list was implemented. This task simply entailed creating the InfoWindow class which is the 2D overlay displaying the blog information. The InfoWindow class has access to the current blog and its information. The visual features of this class (the frame) were created in Photoshop and imported as an image which can be displayed due to the fact that InfoWindow inherits the Sprite class in *flash.display.sprite*.

Task 6 on the priority list, allowing browsing of related blogs, mainly affected the ResultContainer class (model in MVC). Browsing related blogs means that the application must send a URL request to the Twingly server, requesting all blogs related to the blog currently in scope. The Twingly server then responds by sending a new list of blogs and their corresponding information. This task did not entail updating the graphical component (view in MVC) with the animations listed in the concept description.

In order to accommodate the new functionality of browsing related blogs the datastructure was changed from an Array to a matrix (or Array of Arrays). Doing so was necessary as the original search results could not be deleted because the user must be able to browse backwards after having navigated to the "related" results. The original search results are only deleted if a completely new search is executed. The "related" search results however are deleted if the user chooses to browse backwards. The controller functions to navigate the datastructure were simply augmented with an extra index pointer, allowing both rows and columns to be accessed in the model.

The last task in this iteration was to make use of the screen shot for each blog result. Up to this point only the URL of the screen shot had been stored in the BlogResultItem object without actually fetching the image from the database. The screen shot needed to be fetched from the Twingly server by sending a request with the URL for the blog's screenshot. In this first version of the application the BlogResultItem's visual representation was a BlogTile represented by a *Sprite*. All the BlogResultItems with their corresponding BlogTiles are created immediately after the parser has finished parsing the result list. The screen shots are fetched and loaded during this creation process.

Figure 16: A screen shot of the application at the end of iteration two.

At the end of this iteration the application was able to show one blog result at a time in a two dimensional GUI (Information window + screenshot) supporting next/previous browsing and browsing of blogs related to the blog currently in scope (in the information window) [*see* Figure 16 *ovan*].

5.3.4 Iteration three

The tasks in iteration three mainly involved altering the View by incorporating 3D graphics and animation in the application. Some adjustments to the Model were also necessary in order to improve performance.

As is usually the case with Open Source projects the documentation is rather poor, Papervision is no exception. Therefore the initial part of this iteration was spent on learning Papervision 3D by looking at tutorials, browsing forums and blogs and, as a last resort, by studying the source code. The first step when using Papervision or any 3D engine is to setup the scene with elements such as cameras, lights, render engine and a viewport in order to get some polygons on the screen. After having browsed a number of forums it was clear that many people were using the same default template called PaperBase to setup the scene. This is not advanced code; the class simply sets up the elements listed above. Instead of reproducing the template was used as a base class for my Scene class. Because PaperBase has circulated on the Papervision community it is difficult to assert whom the proprietor is, hence no such reference is given, instead the community as a whole is acknowledged for the existence of this setup code which is a great tutorial for new time users of Papervision.

The setup code allowed me to quickly start creating the blog tiles. In the previous iteration the blog tiles were represented by a *Sprite* object holding the screen shot of the corresponding blog. A *Sprite* is the most commonly used object used to present visual elements in ActionScript. The new blog tiles needed a different representation in order to match the description in the concept, namely a box shape with the screen shot pasted on the front facing side. Papervision provides classes for the most common primitives such as spheres, cones, planes and of course cubes which was the primitive used to represent the blog tiles. Instead of inheriting from the Sprite class the blog tiles would now inherit from Papervision's Cube class.

The first version of the three dimensional blog tile was a red box without the screen shot present. This version only served as a first trial to get polygons on the screen. With this visual representation it was now possible to position the tiles relative to each other and to match the dimensions and field of view (of the camera) with the concept image [*see* Figure 17 *nedan*].

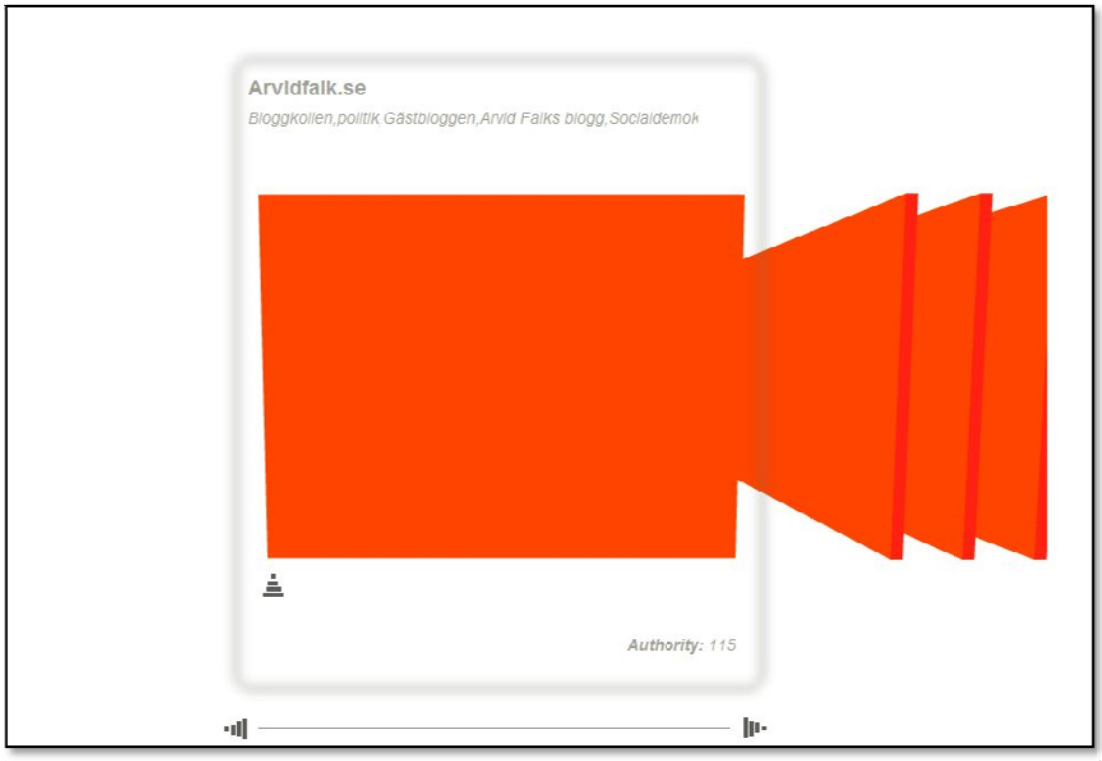

Figure 17: First use of PaperVision in the application.

Positioning the tiles was a not as straight forward as anticipated; the following model was used to accomplish the tiles positions relative to each other.

The position of a tile depends on the blog's:

- *order in the result list (column index),*
- *it's "focus" relative the information window*
- *and the level of the blog (row index).*

The focus indicates if the blog is in scope (encompassed by information window), immediately to the right or left of the scope or simply just in line. For blog tiles that are in line a constant offset was used, for the three other states specific positions were used.

The level indicates how many times the user has clicked the "related" search button and is used to calculate the z- value (elevation) of the blog tile. Recall that if a related search is triggered the camera is elevated above the original search with the list of related blogs listed above the previous results; this action is repeated every time the related button is pushed. The level information is also used to determine if the blog tile has a reflection or not, only the tiles associated with the original search results (level 0 tiles) have a reflection.

This model for positioning the blog tile needed to be dynamic, as the position of a tile changes when the user triggers one of the navigation buttons.

The next problem that needed to be solved was pasting the screen shot of the blog to the blog tiles front facing side. Papervision offers a material called *BitmapFileMaterial* which fetches an image from a server and applies it to a 3D object. Naturally this was the method attempted first as it solves both the fetching of the screen shot and the texturing of the blog tile. Unfortunately the *BitmapFileMaterial* turned out to be a problem which I spent several days trying to solve. The *BitmapFileMaterial* simply did not work and as it turns out I was not the first to encounter problems with this Papervision feature. As a Beta version (at the time) of Papervision was used the assumption that this was a bug was adopted. The problem was identified as an event handling problem, the event indicating that the file had been fetched and loaded was never received and therefore the screen shot was never textured to the blog tile. A work around was used instead of spending more time on solving the issue. The solution was to fetch the screen shot using the regular *Flash.net* library and loading the image into another material in the Papervision library; the *CompositeMaterial*. With this class the screen shot was successfully loaded and applied as a texture on the cube.

When the screen shot was first applied as a texture it did not look quite right when it was rendered to the screen, the texture looked "wobbly." This phenomenon was recognized from the visual search interface searchMe where I had first seen this effect. This "wobbling" interference occurs when too few polygons are used. This correlation between number of polygons and texture quality was unfortunate as it was necessary to keep the polygon count down as much as possible. Simply increasing the polygon count would have a negative effect on the frame rate when animating the tiles.

According to the priority list the next task was to add the functionality of browsing backwards from a related search, i.e. removing the current row in the matrix and resetting the navigator to the previous row (model representation) and stepping down one level (view representation). The order of the priority list was altered at this stage by solving the next/previous *animation* first. This was done because there was no means of debugging the backwards navigation without a visual representation of the action and also because at this stage a better idea of what kind of performance could be achieved when animating the blog tiles was necessary.

Animating the action of fetching the next or previous blog in line serves as a means to strengthen the users mental model of where the disregarded blogs go and where the next set of blogs in line come from.

A smooth animation of an object's movement from point A to point B during a time interval T is accomplished by interpolation. A number of different functions are plausible that carry out this interpolation. The interpolation function determines how each position is calculated for every time value t in the interval T, and hence determines the visual appearance of the movement. As the frame rate was a concern it was necessary to keep the calculations as simple as possible for each frame, which is why linear interpolation was used for animating the blog tiles movement. Linear interpolation is the least calculation intensive form of interpolation. The drawback of using linear interpolation is of course that the animation is not as smooth when compared to using for example an ease-in ease-out function [appendix].

The interpolation was written as a separate function so that it could easily be exchanged by me or by Twingly for a more elegant function, rendering a more pleasing animation, if the frame rate would not suffer too much.

It was not until this point, when a functional animation of the next/ previous navigation was implemented, that the performance issues could be seen and actually be evaluated if the concept would be a realistic alternative as a visual search interface. Several issues presented themselves when the animation functionality was completed.

Performance issues:

1. ***Poor texture quality with too few polygons.***
2. ***Faulty depth sorting with too few polygons.***
3. ***Poor frame rate with too many polygons.***

If only two polygons were used for the front face of the cube, issues one and two presented themselves, but the frame rate was still good. Issue 2 was a consequence of the sorting algorithm used by Papervision, the Painter's depth sorting algorithm, which requires a certain amount of polygons in order to not get a faulty depth sorting.

Simply increasing the polygons solved issues 1 and 2 but lead to an unacceptable frame rate. It was important to keep in mind at this point that the polygon count would increase even more when the reflections were added. Therefore a method to drastically reduce the amount of polygons in the scene needed to be figured out without invoking issues 1 and 2. The only options were to minimize the amount of polygons for each cube to the limit where issue 1 and 2 were still not invoked and decreasing the number of visible blog tiles at one time in the scene. A flag setting in Papervision which affected the precision of the rendering of textures was also altered. The default value was *imprecise*; changing it to *precise* affected the frame rate but also allowed me to reduce the number of polygons, ultimately producing a gain in both visual appearance and frame rate.

Before the next task could be started an issue which had been postponed needed clarification with Twingly. They had not yet solved the issue of pagination, i.e. how many results would be received in each response package. The JSON feed we had been working with contained only the first 10 matches and there was no way of requesting or accessing the next set. At Twingly I was told to simply replicate the first search request until they had solved the pagination issue, a similar solution would be used for the "related" search as this functionality was not yet available either.

With this temporary solution I started working on how the application would handle the pagination issue both internally (in the datastructure) and visually. The first thing I needed to do was to limit the animation to only the visible tiles, animating all the tiles was both unnecessary and a frame rate killer. Secondly I needed to decide when the next set of results (next page) would be fetched; I decided that this would occur when the blog tile in the middle of the current set was in focus. Results are only removed from the datastructure when browsing backward. This was a deliberate design choice and one that I knew would have to be altered if the interface was going to be deployed on the web, as it could potentially overflow the memory of the host computer. Because my main focus was to create a functional prototype this detail was ignored for the time being. The same design choice was made for related browsing, allowing "infinite" related searches without freeing memory. The solution to this problem is not difficult, but was not something that I wanted to spend time on at this stage; it was added to the end of the priority list as point 13.

When the animation of the next/previous navigation was in place with a satisfactory frame rate I decided to tackle task 11, reflections and shadows, before solving the backwards browsing (task 9). It was clear at this point that shadows with a lighting model would not be implemented as this would lower the frame rate considerably and also because it is not directly supported in Papervision, which meant that I would have to implement the shadow casting. An alternative method for faking shadows was considered, but dropped because it would require too many extra polygons. In the end I opted to not use shadows but to rely only on the reflections to indicate the "ground" level.

Several methods for implementing the reflection were possible:

- Raytracing.
- Rendering reflection to a bitmap viewport.
- Mirroring the geometry in the xy-plane.

But only the third method was a realistic option. Raytracing the reflection would require too much computation for each frame and hence be much to slow and the viewport option was too complicated for the effect that I was trying to achieve. The reflections were added as an attribute of the BlogTile class. The reflection was a plane with a rotated and flipped version of the screen shot as a texture. In order to achieve the reflection fall off, a gradient texture from white to transparent was composited on top of the screen shot, using Papervision's CompositeMaterial.

Two tasks remained before the implementation phase was completed, the ability to browse backwards (from a related search) both the animation and the model implementation and the animation when triggering a related search. In the concept the idea was to have the stack of tiles representing the button for the related search fly up above the current blog tiles and spread out to form a new list of blog tiles with the camera following that movement. Unfortunately I realized that I would not have time to implement this animation and had to settle for a simpler version of this original idea. The animation I ended up implementing was only the camera movement which used the same animation function as the next/previous animation uses, with the new blog tiles simply spawning in their final position. Implementing the backwards browsing was only a matter of removing the blog results from the datastructure and reversing the camera movement when triggering a related search.

5.3.5 Quality assurance

This phase of the project was scheduled to ensure that any remaining bugs and undesired behavior was eliminated from the application before it was deployed on the web for testing. At this point I had a discussion with Martin Källström regarding the testing of the application and if Twingly had any intention of deploying it for testing. Because the application did not fully meet the requirements, mainly due to a lower frame rate, Twingly decided that the application was not ready for deployment. I suggested improvements that could be made (see extensions and improvements) in order to increase the frame rate but also explained that I would not have time to implement these improvements and that it was not within the scope of the thesis to do so.

Even though the application would not be deployed for testing, bugs encountered during the test phase at the end of the last iteration were resolved. One major issue that was encountered was a memory leak which was the result of a reference that was not removed when the backwards browsing was executed. Because flash uses automatic garbage collection it was difficult to find this memory leak.

6 Results

This chapter gives a detailed presentation of the final results of the project. The design concept and the final implementation are treated as two separate results in this text for convenience.

6.1 Final design concept

The final concept is best described as an augmented version of the well known Cover Flow concept that can be found in different flavors on various media today. A short description of Cover Flow is given first.

Cover Flow presents the desired objects visually in the form of planes textured with an image that describes the object in question, e.g. a cover album. The objects can be browsed to the left or to the right, by clicking on the next object or by using the provided slider; either action will trigger an animation that moves the objects in the selected direction. The object currently in focus faces the user and is slightly larger than the other objects.

The final concept for the visual search interface mimics the behavior and to a degree the visual appearance of Cover Flow, but differs in several ways as the objects in this case are not music albums but blogs. The concept is described in terms of its main components, provided interaction and the graphical profile.

6.1.1 Components and interaction

Blog tile – the blog tile is the visual representation of each blog in the result list. It is represented with a tile shape with an aspect ratio of 4:3. The blog's screen shot is presented on the tile's front facing side. Seven blog tiles are displayed at one time in the application viewport. The blog tile that is in the center of the application viewport is clickable; clicking it spawns a new browser window navigating to the blog's URL. The cursor changes its visual appearance to a hand cursor when hovered over the blog tile in the center, indicating its "clickability."

Information window – displays the information extracted from the blog tile that is currently in focus. Only one blog at a time is represented in the information window. The information displayed can be configured but will typically be the blog's title, tags, authority and update history. This information is updated when the new blog is centered in the window.

Navigation slider – allows navigation to the next or previous blogs in line. This is accomplished either by clicking one of the two arrows or by using the box slider. Using the arrows will browse to the next blog. If the box slider is used the number of blogs browsed is determined by the distance that the box slider has been dragged. All navigation using this slider triggers an animation which moves the blogs in a smooth motion either to the left or to the right.

Related stack – is the stack of miniature blog tiles that can be found at the bottom left corner of the blog tile that is centered in the information window. This stack represents blogs that are in some way related to the blog currently in focus. The related stack is clickable, which is indicated by an arrow pointing upwards, a highlight of the stacks outline when the cursor is hovered over it and the fact that the cursor changes to a hand cursor. Clicking on the stack triggers an animation and a URL request which fetches the information of each related blog. The animation is an upward movement of the camera and a morphing + translation of the miniature tiles to their final size and position. An example of the animation can be found on the project blog (15).

Reflection – the reflection of the blog tiles serve the purpose of indicating when the user is viewing the original search results. If a user has clicked the related search button, the camera is translated upward and a new set of blog results are displayed, without a reflection.

6.1.3 Graphical profile

The graphical profile is best understood by examining the mock-up image of the concept. A light background was used instead of a dark one mainly to match the existing search engine which presents the results on a white background.

A light grey tone was used for all the components in order to get as little conflict as possible with the colors present in the screen shot or surrounding ads. Using the same color for all the components also serves the purpose of uniting them as navigational elements.

6.2 Final implementation

The final implementation for this project is a first trial version of the visual search interface described in the concept above. This implementation strives to meet the requirements of the concept and those set forth by Twingly [see 1.3] at the beginning of the project as far as possible. In this section the implementation will be described in terms of the functionality provided, but also where and why it differs from the concept and requirements. Furthermore an overview of the architecture will be presented.

6.2.1 Version 0.9

The basic functionality which is needed to browse blogs is supported in this version of the application. The priority list that was produced at the beginning of the implementation phase lists all the functionality that is needed to meet the requirements of the concept description. The list is reproduced here for convenience:

1. *Communication with Twingly server*
2. *Functional model/datastructure*
3. *Next/previous navigation, including update of the datastructure*
4. *2D GUI with functional buttons*
5. *Functional information window (glass panel)*
6. *"Related browsing" (without animation)*
7. *Loading of screenshots (including logistics/memory management)*
8. *Blog tiles with screenshots (first 3D implementation step)*
9. *"Backwards browsing" (without animation)*
10. *Animation next/previous*
11. *Reflections/Shadows*

Version 0.9 as I have chosen to call the present version of the search interface implements all the points listed above with the exception of shadows and a complete animation of the related browsing action (a simplified animation was implemented).

The functionality that is not implemented in this version is the connection to the current search interface. In order to pass arbitrary search phrases to the flash application it was planned that the text field present on the current Twingly Blog search would be used. As the application did not meet the requirements set by Twingly it was not deployed on the web and hence the functionality to pass search phrases to the application was not implemented, although a plan for that implementation did

exist. Another functionality which is not present in the current version is the ability to browse related blogs “for real,” mainly because the functionality was not provided by Twingly at the time, instead a faked version of this functionality was implemented [*see Chapter 5.3.4*].

The performance of the application was tested by me and by the staff at Twingly. The requirement was that the frame rate should never fall below 24 fps on a regular computer. I tested the application on a regular computer and on a computer that would be considered “fast.” On the regular computer an IBM T43 Laptop (single processor) the performance was slow and clearly below 24 fps. The second computer I tested it on was a HP Pavilion dv7 Laptop (dual core) which achieved an acceptable frame rate. Unfortunately I do not have any actual figure for the achieved frame rate but my guess is that the frame rate on the Pavilion was in the range 20-25 fps.

The visual appearance of the application is quite close in resemblance to the concept image and graphical profile presented in the concept description [*see* Figure 19 *nedan*]. The biggest difference regarding the visual appearance is the lighting model used for the blog tiles. In this version the shading is static and “faked,” i.e. the sides of the blog tile have different predefined shades of grey. A true lighting model with a material was tested but did very little to improve the visual performance and had a remarkably negative effect on the frame rate. Note that the concept image is a raytraced image and has been manipulated in image processing software, the application image is a real-time rendered image from flash that has not been postprocessed.

Figure 18: Screen shot of the final application.

Figure 19: Illustration of the concept, replicated here for comparison.

6.2.2 System architecture

The system is built on the design pattern Model View Controller (MVC) which separates graphical components from the underlying functionality. The main classes reside in the *com.twingly* package and derive functionality from the packages *flash.display* and *org.papervision*. All visual components inherit either the *flash.display.Sprite* class or the *org.papervision.cube* class. The ModelController and the ResultConatiner classes are both referenced by several of the other classes in the application and never have more than one instance, therefore these were implemented as singletons (see appendix). Below is a short description of each class and the functionality it provides.

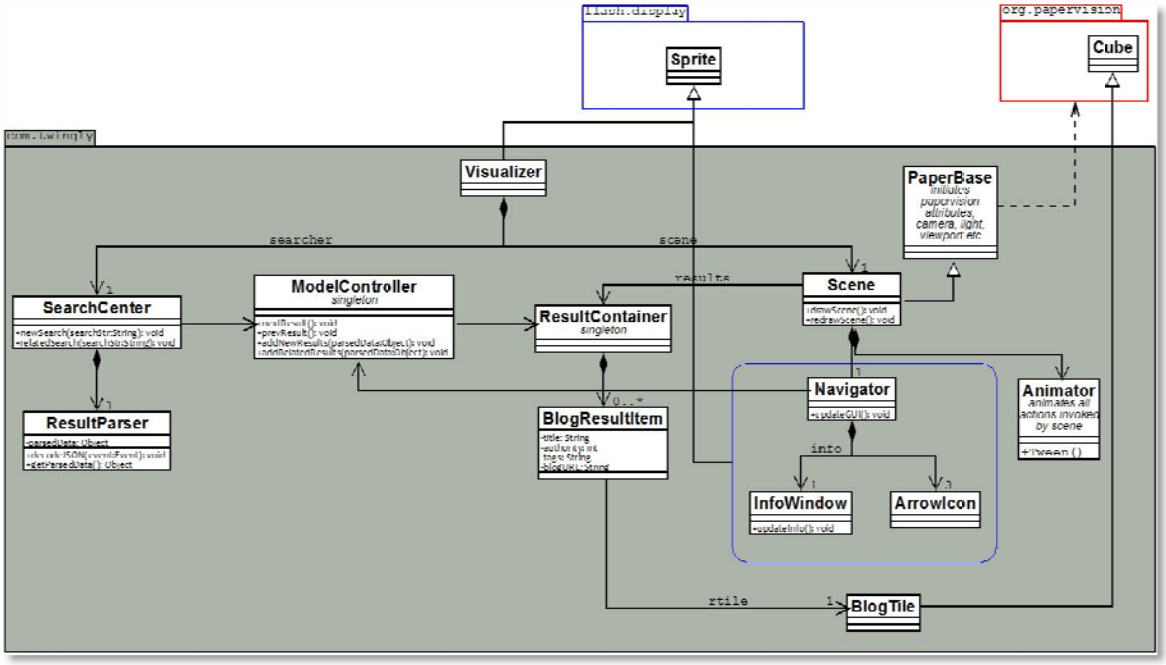

Figure 20: Overview of the architecture of the application.

Visualizer – Is the class that contains the initialization code setting up the necessary objects and listeners. It also provides the top level functions to perform searches of different kinds as it holds the only reference to the SearchCenter class. Visualizer inherits the Sprite class only because it needs to be a part of the Event loop.

SearchCenter – Initiates and prepares the components that perform the searches. It holds references to the ModelController and ResultParser classes. This class is also responsible for updating the model with the results from a search when it receives an event indicating that the results have been fetched.

ResultParser – Contains the actual functionality for sending a request to the Twingly server and for parsing the results with the JSON decoder.

ModelController – Contains the top level functions that update the model (ResultContainer) according to different actions communicated with events. These events include actions initiated by the user through the navigational components and actions initiated by the search parser in response to a new set of results available from the result parser.

ResultContainer – Holds the datastructure that organizes the BlogResultItems. Also provides a number of functions to maintain the structure. Only the ModelController uses these functions.

BlogResultItem – Is the representation of a blog result. Contains information about a blog and also holds a visual representation in the form of a blog tile.

BlogTile – Is the visual representation of a blog in the form of a tile with a screen shot of the blog on its front facing side. Extends the functionality of the *org.papervision.cube* class. Also implements the reflection.

PaperBase – Initializes all components that are necessary in order to render polygons on the screen, e.g. cameras, viewports, lights, renderengine etc. This code is not written by me and can be found as a tutorial on the PaperVision community blogs.

Scene – Extends the PaperBase class and handles all updates to the scene and is the root node of all visual components allowing it to listen to all events in the scene. Is responsible for adding and removing blog tiles form the scene and has an instance of the Animator class in order to animate actions provided by the Navigator class.

Navigator – Implements the 2D GUI and is responsible for alerting involved components when an action has been triggered. It holds a reference to the ModelController singleton in order to update the model. It is also responsible for alerting the information window when a new blog is in focus.

InfoWindow – Holds the information window graphical representation and updates the window with the information from the blog that is currently in focus.

ArrowIcon – Implements the arrow buttons used as navigational components.

Animator – Contains functionality to animate objects with an interpolating function.

7 Discussion

This section discusses the development process by analyzing the weaknesses and strengths of the different methods used during the course of the project and how they were used. The end results of the project in the form of the concept, implementation and lessons learned are also analyzed and discussed.

7.1 Design phase

The only formal methods that I used for the design phase were brainstorming and prototyping. Only a limited amount of research on users was done because of the tight time constraints. This project involved both creating a concept and implementing it on a relatively short time span; therefore many of the formal methods advocated by interaction designers were not a realistic option for this project. Because of this the Goal Directed Design described in chapter 3.2.2 was only followed to an extent, but not all the way, as it requires a lot more research and testing. The same is true for the methods described by the conceptualization and prototyping chapter. Although these methods were not used to the extent described in chapter 3, many of the ideas and the core ideology behind the methods were used.

Goal Directed Design for example, is all about keeping the users interests and goals in mind when making critical design choices. This was still possible for me to do, although a larger databank of information on the users probably would have given more insight into some choices. It is my belief that following this concept to the point would require the entire project time span or an extra human resource dedicated solely to the interaction design.

The prototyping was a central part of the design phase and the process that I spent most time on. The use of 3D modeling and rendering software made it possible for me to test spatial issues very early on in the design phase. Many times this is not possible until the implementation phase has started; in this case it allowed me to discard the first concept very early on. Another great benefit of using 3D rendering software was that it was much less time consuming to create content and variations of concepts compared to using only image processing applications or pen and paper. Hence it allowed me to create detailed presentations of the concepts with graphics closely resembling that of the final implementation. This method also proved advantageous when communicating with the staff at Twingly as the images and animations clearly described my ideas. Quite a few times I was asked to alter something or to tweak some details, a task which proved simple when using 3D software.

Ironically the strongest point with this method also turned out to be its weakest point. The amount of detail and advanced graphics presented, I believe, may have given the impression of being a final concept and not a mock-up. In my opinion concept #2 was disregarded by Twingly much too quickly. The arguments against using this concept (see 5.2.4) could have been averted simply by altering the concept a little or by creating an animation to test the supposed flaws. In principle I believe that the distance between me and Twingly might have affected the final choice of concept, had I worked at Twingly I would have pushed harder for the second concept and had explained the strengths of the concept more thoroughly.

The communication with Twingly was almost entirely carried out with Skype chat or by telephone. This method worked but more face to face meetings would have allowed me to present the different

ideas in more detail, which ultimately may have lead to a completely different concept. It was sometimes difficult to balance the wishes of Twingly with the research I had done as I was sometimes forced to make design decisions contradicting the research material. The CoverFlow concept for example does not give a very good overview of the search results and can only display a very limited set of the results at one time. I do however believe that the final concept was a good one when it comes to browsing related blogs and giving the user a clear mental model of how to navigate the results.

During the course of the project it seems that many search engine providers where also implementing different kinds of visual search interfaces. At the end of this project there were many more visual search engines than when I started research at the beginning of September 2008. I have no exact figure, but very many of these visual search engines are using some kind of CoverFlow concept. In my opinion it would perhaps be more advantageous to develop a concept that does not resemble CoverFlow at all but brings something new to the table. Interestingly I did find a concept (at the end of this project) which strikes an uncanny resemblance to my Rubics cube idea (concept #1) [*see* Figure 21 & Figure 22 *nedan*]. The Visual Search Cube has augmented the cube with an image of the result that is pointed at with the mouse (16).

Figure 21: Screen shot of the Visual Search Cube (16).

Figure 22: The Rubics Cube concept (concept #1).

Having tested this search engine (www.search-cube.com) only confirms that the idea was not a good one. I think it is also worth noting that none of the visual search engines so far have succeeded in revolutionizing the way people search for topics, blogs or images on the internet. If they in fact were an aid, why hasn't Google, Yahoo or one of the biggest search engine providers developed or bought such an interface?

SearchMe which is referenced quite often in this thesis, mainly because the same underlying concept CoverFlow is used has also been altered and augmented during the course of the project. When I first tested searchMe very little information about the web page in scope was available. SearchMe now provides a similar function to my information window that also implements a magnifying glass that can be used to study the web page closer without navigating to it.

As far as blog search engines go, I have not yet found a blog search engine (dedicated solely to blogs) that implements a CoverFlow concept or one that provides a related search. Perhaps bloggers and blog readers should be treated separately as their search habits differ from a user using a regular search engine. In any case it would certainly be interesting if Twingly enhanced the final concept by implementing the extensions and improvements recommended in chapter 8 and tested it on their users, only then can the concept be properly evaluated.

7.2 Implementation phase

The implementation phase went very much according to the plan and I was quite pleased with the methods used and the planned iterations. Starting with a bottom up design was a very good choice and I can recommend it to anyone whom is learning a new programming language because it allows you to start simple. Taking time to split the application into smaller pieces of functionality and creating a priority list takes some time but is definitely worth it as it makes it very easy to keep track of the progress and overall functionality, not only for yourself but also for the "customer" or stakeholder.

The MVC (Model View Controller) structure used for the architecture was a given choice as it is a proven pattern that is very suitable for applications that are very “GUI oriented.” This also allows for example Twingly to use the underlying model while completely altering the visual interface or vice versa.

The biggest problems that I encountered during the implementation phase were mostly ActionScript 3 related. As I have mentioned earlier ActionScript 3 relies very much on an asynchronous event handling system which sometimes makes it difficult to debug the application for a novice. Because of this it would have been a good idea for me to early on implement an error handling system which would catch errors and handle them appropriately. Creating such a system properly does require time and a well thought through plan for how different errors should be handled. Unfortunately I did not plan for this and that made it more difficult for me in the long run as I encountered strange behavior and unnecessary programming mistakes.

As always testing and quality assurance was in a way given too little time in the project plan. On the other hand, if more time had been spent on testing, perhaps some of the features on the priority list would not have been implemented.

It was unfortunate that I did not have the time to implement the extensions and improvements proposed in the next chapter. Having done so might have enabled a deployment on the Twingly server for actual user tests. Martin Källström at Twingly did propose that such a test phase could be realized if the application met the requirements. It is also a fact however that a lot of the functionality that was supposed to be supplied by Twingly (related searches, pagination issues in JSON) was not finished at the time, therefore I decided to do manual testing myself as there were no guarantees of this functionality being finished in time.

The resulting implementation provides the core functionality that is described in the concept with emulated pagination and related searches and I feel that with the improvements proposed in the next chapter that the interface could be a powerful tool for bloggers and blog readers.

8 Extensions & improvements

In this section improvements and possible extensions are proposed for both the concept and the implementation. The main focus however is on the implementation as the concept requires more thorough testing in order to be further analyzed.

8.1 Improving the concept

The main improvements that can be proposed to the concept at this stage without further testing are mostly connected with the graphical representation of the components and there placement. Many people have commented on the arrow icons and that they can be confused for “volume” buttons, this is naturally not a successful icon design and therefore I believe they should be redesigned.

During the course of the project I tried to get access to more information that could be used and displayed in the information window in order to aid the user in his/her browsing. An example of information that could be displayed is when the blog was last updated. The main point is that there is screen real-estate available that could be used for more information.

The navigation is today constrained to the navigational buttons; another option would be to allow clicking on the blog tiles in line, navigating directly to the clicked blog tile. This interaction is available on most variations of Cover Flow, e.g. SearchMe implements this. The same interaction could be used to navigate back from a related search by clicking the blogs on the level below.

Another appearance issue that can be experimented with is the spacing and rotation of the blogs that are in line. Rotating them slightly with the front face towards the screen and increasing the distance between them will enhance the visibility of the blog tiles and decrease the number of blogs that are displayed at one time. This would not only increase the visibility of the blogs immediately to the left and right but also increase the frame rate as less polygons are visible at one time in the view.

Figure 23: An example of how the blog tiles can be spaced and rotated differently.

These settings can be adjusted in the BVConst class which holds constants such as spacing and other parameters in one place for convenience. It is important to note however that some of these constants are dependent on each other; this is commented in the code.

8.2 Improving the implementation

As the implementation is a first prototype there are naturally quite a few improvements that can be proposed, the most important ones however are suggestions that will improve the performance. The frame rate is too slow for the application to be deployed on the web; therefore performance enhancing measures are necessary. The following list of enhancements is proposed:

1. Exchange cubes for planes.
2. Refactor memory management.
3. Solve pagination issue.
4. Find unnecessary calculations (profiling).
5. Replace tween function.
6. Implement related search animation.

Points 1-3 are all performance enhancing measures that should have a positive effect on the frame rate. By exchanging the cubes for planes the visual appearance is affected, but in my opinion the gain in frame rate overshadows this minimal change in appearance. Making the blog tiles planes instead of cubes saves a few polygons for each tile and also makes it possible to reduce the amount of triangles used for each tile. Recall that one of the problems with using too few polygons was that the depth sorting was negatively affected, when the cubes are exchanged for planes the sides of the cubes are no longer a part of the depth sorting, hence the planes do not need as many polygons as the front facing side of the cube requires. The overall effect is a large reduction in polygon count and an increase in frame rate.

Point 2 addresses the issue of allocating and deallocating memory which is carried out when a new search, related search or next page search is triggered. Allocating memory is usually a time consuming task which requires many cycles for the CPU. Due to the time constraints I have not spent a lot of time optimizing this feature. Point 3 is related to this issue in the sense that there probably is an optimal amount of results per page, before the next page needs to be fetched. In the existing version of the application the whole next page is fetched at one time about halfway through the current page which causes a glitch in the frame rate. A better solution would probably be to fetch not the whole page at once but perhaps one or two continuously and putting the results into preallocated memory. This solution however requires that the pagination is incorporated into the JSON format used today; this had not been solved by Twingly during the course of the project. The same solution can be used for the related blogs, pre-fetching them before the related search would reduce the loading time a great deal. This loading time can be hidden to an extent by a longer animation but reducing the waiting time for the user is a better solution.

Another point which is worth looking into is reviewing the calculations that are carried out for each frame. These operations will typically be found in the animation class and the scene class. Some calculations perhaps are unnecessary to perform every frame, especially calculations which involve floating point operands and divisions are very costly for a CPU to perform. One might ask why I have not considered this directly instead of saving it for a refactoring session. The reason for this is that I have worked with the motto “first make it work, then make it fast.” There are several reasons for

using this order, the most compelling reason is that it is sometimes difficult to manually optimize code better than the compiler/virtual machine does, rendering these alterations useless. It can also be harder to read optimized code, which is not a good thing when the application is still under development. This optimization should be carried out using a profiler, which makes it a lot easier to find the bottlenecks of the code. Sometimes the bottlenecks are not where we think they are, especially in this case because it is difficult to know how much work the GPU is performing and how much the CPU is performing. Optimizing for the GPU is different than optimizing for a CPU.

Points 5 and 6 are not performance enhancing but appearance enhancing tasks. The Tween function used in the Animation class is a simple linear interpolation. This function should be exchanged for a smoother interpolation which uses an ease-in ease out function. Altering this code is rather trivial and can be experimented with in order to achieve the desired results. The last point is simply to implement the animation which is performed when the related search is triggered. The existing animation only animates the camera movement once the tiles have been loaded and put into place. The original idea was to animate the small blog tiles movement from the “button” placement to their final position above the current blog tiles. This animation can be seen as a gif animation on the project blog. This animation was not implemented because the related search had not been implemented or prepared for the JSON format by Twingly, which is necessary in order to realize it. Implementing the animation does not really require a lot of code, in essence the existing tween function could be used, but if more control of the scaling, rotation and translation is desired a new function is necessary.

9 Conclusion

In this project I have created a concept and a prototype of a visual search interface for Twingly's blog search engine that incorporates 3D graphics using Adobes Flash player and a software 3D engine for the web called PaperVision. The concept mimics the CoverFlow concept most commonly known from iTunes album cover browser and augments it with functionality related to browsing blogs and features inherent in the Twingly blog search engine.

The purpose of creating this interface was to provide bloggers and blog readers with a more fun and alternative manner of browsing and searching for blogs in the hope of aiding him/her to find results quicker and in finding results they perhaps did not know they were looking for.

The concept developed during the course of this project is closely related to and inspired by the searchMe search engine. The main feature that separates these two interfaces is that searchMe is a search engine for web pages while Twingly's search engine is directed towards blogs. Having tested both of these search engines I do not believe that the CoverFlow concept is an enhancement when it comes to searching for web pages. Because searching for a blog differs somewhat from searching the web for a topic I still believe that the CoverFlow concept might have a future in blog search engines.

The final implementation of the concept developed during the course of this project still requires some extensions and improvements in order to meet the requirements set forth by Twingly in order to be deployed on the web for testing (see chapter 1.3). Although the application has not been deployed for testing I believe that the concept has a possibility of succeeding as an alternative to the text based blog search engine and that it could be worthwhile for Twingly to implement the extensions and improvements proposed in chapter 8. It is mainly the feature of browsing related blogs and the implementation of this visualization that sets it apart from other search engines mimicking the CoverFlow concept, and I firmly believe that it is one of its strengths.

10 Bibliography

1. *The Handbook of New Media.* London : SAGE publications Inc., 2006.

2. Dictionary.com. [Online] [Cited: 01 20, 2009.] http://dictionary.reference.com/browse/visualize.

3. Twingly Blog Search. [Online] [Cited: May 3, 2009.] http://www.twingly.com.

4. Grokker Enterprise Search Management. [Online] [Cited: 04 22, 2009.] http://www.grokker.com/.

5. Kartoo.com. [Online] [Cited: 03 15, 2009.] http://www.kartoo.com/.

6. Quintura. [Online] [Cited: 04 22, 2009.] http://www.quintura.com/.

7. Flowser. [Online] http://www.flowser.com/.

8. Viewzi. [Online] [Cited: 04 12, 2009.] http://www.viewzi.com/.

9. Searchme.com. [Online] [Cited: 05 01, 2009.] http://www.searchme.com/.

10. Youtube. [Online] [Cited: 05 10, 2009.] http://www.youtube.com/.

11. Coverpop. [Online] [Cited: 04 01, 2009.] http://www.coverpop.com/.

12. **Alan Cooper, Robert Reimann, David Cronin.** *About Face 3, The Essentials of Interaction Design.* Indianapolis : Wiley Publishing, Inc, 2007.

13. **Tracy Fullerton, Christopher Swain, Steven Hoffman.** *Game Design Workshop: Designing, Prototyping and Playtesting Games.* San Francisco : CMP Books, 2004.

14. **Roger Braunstein, Mims H. Wright, Joshua J. Noble.** *ActionScript 3.0 Bible.* Indianaplois : Wiley Publishing, Inc, 2008.

15. **Svensson, Daniel.** Interactive Visualiztion of Blog Search. *Interactive Visualiztion of Blog Search.* [Online] [Cited: 05 04, 2009.] http://datx0208.wordpress.com/.

16. search - cube - the Visual Search Engine. [Online] [Cited: 05 04, 2009.] http://www.search-cube.com/.

17. Flowser on Amazon. [Online] [Cited: 04 25, 2009.] http://www.flowser.com/.

18. **Sommerville, Ian.** *Software Engineering 8.* Harlow : Pearson Education Llimited, 2007.

Appendix A: Definitions

3D Studio Max – A 3D modeling and rendering software.

ActionScript – An Object Oriented (from version 3.0) programming language interpreted by Adobe's Flash Player.

API – Acronym for Application Programming Interface

Flash player – Flash Player is a cross-platform browser plug-in…

JSON – Stands for Java Script Object Notation and is a carrier format for arbitrary data primarily intended for JavaScript but is also supported by several other programming and scripting languages. Provides support for datastructures such as tuples and lists that are commonly used in programming languages. JSON is a powerful alternative to XML in many situations.

Papervision 3D – a 3D engine for Flash player which supports the most common functionality necessary for creating, reading , rendering and manipulating 3D scenes.

Rubik's Cube – The **Rubik's Cube** is a 3-D mechanical puzzle invented in 1974[1] by Hungarian sculptor and professor of architecture Ernő Rubik.

Singleton – A design pattern commonly used when an object only should have one instance and needs to be referenced by several other objects. The Singleton pattern ensures that only one instance of the object can exist at one time.

Appendix B

Example of JSON formatting:


```
{
  "totalFound":1,"items":
            [
                        {
                                    "id":"2667480418487502938",
                                    "summary":"This summer the Swedish
political blogosphere has been in flames over a law proposal allowing a
civil organization to monitor all Internet traffic crossing Swedish
borders, with the purpose to fight t&hellip;",

            "url":"http://blog.twingly.com/2008/09/18/twingly-report-about-
the-fra-debate",
                                    "pubDateMs":1221764829000,"title":"Twingly
Report about the FRA Debate",
                                    "websiteName":"Twingly Blog",
                                    "websiteUrl":"http://blog.twingly.com/",

            "websiteRssUrl":"http://blog.twingly.com/feed/",
                                    "inlinks":"39",
                                    "likes":"0",
                                    "blogoscopeX":null,
                                    "blogoscopeY":null,

            "screenshot":"http://images.twingly.com/screenshots/153407522123
97415993.jpg",
                                    "twinglyAuthority":"105",
                                    "tags":[
                                                "Blogosphere",
                                                "Europe",
                                                "FRA",
                                                "report",
                                                "Twingly"
                                    ]
                        }
            ]
}
```