PHOTOCUBE: A MULTI-DIMENSIONAL IMAGE BROWSER

Hlynur Sigurþórsson
Master of Science
Computer Science
June 2011
School of Computer Science
Reykjavík University

M.Sc. RESEARCH THESIS

ISSN 1670-8539

PhotoCube: A Multi-Dimensional Image Browser

by

Hlynur Sigurþórsson

Research thesis submitted to the School of Computer Science
at Reykjavík University in partial fulfillment of
the requirements for the degree of
Master of Science in **Computer Science**

June 2011

Research Thesis Committee:

Björn Þór Jónsson, Supervisor
Associate Professor, Reykjavík University

Laurent Amsaleg
Research Scientist, IRISA-CNRS

Hannes Högni Vilhjálmsson
Associate Professor, Reykjavík University

Marta Kristín Lárusdóttir
Assistant Professor, Reykjavík University

PhotoCube: A Multi-Dimensional Image Browser

Hlynur Sigurþórsson

June 2011

Abstract

As the scale of personal photo collections keeps growing, the simple methods commonly used today for photo browsing quickly become inadequate. We describe PhotoCube, a novel three dimensional (3-D) photo browser, based on the recently proposed ObjectCube data model. The data model allows users to seamlessly involve a large number of meta-data dimensions in each browsing session and captures a rich set of browsing actions. The architecture of PhotoCube is highly extensible, as different browsing modes are used to define the potential actions to take in many circumstances, e.g., when aggregating a large set of photos for display, or when examining a set of photos in detail. In this thesis, we describe the architecture of the prototype in detail, as well as its current state. We then present results from a preliminary user study, which indicate that while experienced computer users find the prototype interface to have a somewhat steep learning curve, they are excited about the potential of the underlying data model.

PhotoCube: Fjölvíður myndaskoðari

Hlynur Sigurþórsson

Júní 2011

Útdráttur

Þar sem stærð persónulegra myndasafna fer ört vaxandi verða þær aðferðir sem við þekkum í dag til að skoða myndasöfn ófullnægjandi. Við lýsum PhotoCube, nýjum þrívíðum myndaskoðara sem byggir á ObjectCube gagnalíkaninu sem nýlega var sett fram. Með þessu líkani geta notendur beitt víddum sem byggðar eru á lýsigögnum mynda við skoðun þeirra. Uppbygging PhotoCube býður upp á viðbætur, þar sem mismunandi skoðunarhamir eru notaðir til þess að skilgreina mögulegar aðgerðir við fjölbreyttar kringumstæður, til dæmis við samþyrpingu á stóru mengi mynda fyrir birtingu eða við nákvæmari skoðun á völdu myndamengi. Í þessari ritgerð munum við lýsa uppbyggingu frumgerðar PhotoCube og lýsa ástandi hennar. Við munum einnig birta niðurstöður úr notendaprófun, sem gefa til kynna að reyndum notendum finnst frumgerðin hafa bratt lærdómsferli en að þeir sjái mikla möguleika í undirliggjandi líkani.

Fyrir Thelmu.

Acknowledgements

I would like to thank Björn Þór Jónsson, Laurent Amsaleg and Kári Harðarson for the collaboration on this project.

Grímur Tómasson, author of ObjectCube, for short notice bugfixes, comments and help with almost everything in this project. His input into the work that this thesis stands for is invaluable.

Hannes Högni Vilhjálmsson and Marta Kristín Lárusdóttir for input and instructions on the user evaluation part of this project.

Oddur Kjartansson and Helgi Þ. Möller, for constructive criticism on the text and keeping me on track.

Publications

A demonstration was presented at the ACM International conference on Multimedia Retrieval (ICMR11) (Sigurþórsson, Tómasson, Jónsson, & Amsaleg, 2011). Part of the material in this thesis has also been submitted to an international conference. Co-authors of both are Grímur Tómasson (Reykjavík University), Björn Þór Jónsson (Reykjavík University) and Laurent Amsaleg (IRISA- CNRS). While my co-authors contributed significantly to the writing of both papers, the implementation and user evaluation is entirely my work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Photography is the art of creating images using a radiation-sensitive medium, practiced by people around the world. The first photographs were made using a light-sensitive emulsion of silver salts applied to glass plates. Subsequently, the photographic film was invented in 19th century by George Eastman, the founder of the Eastman Kodak Company (Mulligan & Wooters, 2005). Photographic films were used throughout the 19th century and are still used by many photographers today. Now, however, photographs are typically created using digital cameras with electronic image sensors based on the principle of converting light into electrons using a charge-coupled device (CCD) or a complementary-metal-oxide semiconductor (CMOS). How photography is used varies from one person to another, but the objective is typically the same, namely to transform moments into photographs.

Personal photo collections can quickly become large, as people tend to take hundreds or thousands of photos every year; for the most enthusiastic, collections can grow orders of magnitude larger. Thus, a good organization is a prerequisite for finding photographs of interest in such collections.

Before the advent of digital photography, photographs were typically stored in photo albums or (shoe)boxes, often labeled with time periods, locations or event names for photo content indication. Browsing then involved digging through a pile of albums or a stack of boxes.

Today, photographs are stored as files on computers and are viewed using image browsing applications. This digital switchover enabled us to move our photographs, from the cumbersome albums and shoeboxes, onto computers to exploit their power and flexibility to organize and browse photo collections in a better manner.

1.1 Browsing Scenarios

Despite this digital switchover, finding images in a large collection can still be a difficult and frustrating task. Let us consider two plausible browsing scenario that we might want to solve, using an image browser. In these scenarios, assume that we possess a relatively large image collection.

Browsing scenario 1. *We want to browse all images that were a) taken somewhere in Europe, b) include our friends and c) have a brightness value in a certain range.*

This scenario might be interesting if we recently traveled through Europe with our friends and we only want to view images from Europe that include our friends and are neither over- nor underlit.

Browsing scenario 2. *We want to browse all images of our family members containing at least one parent and one child, regardless of the time period, grouped by parents, children and the location where the photos were taken.*

This scenario might be interesting if we want to view family images, categorized by location and which family members are on each image.

Today's image browsers are equipped with many well accepted methods for organizing and browsing images. These methods, however, lack flexibility and power when used in solving browsing scenarios such as Scenarios 1 and 2. The reason is threefold:

1. They do not support browsing by more than one categorization at once;
2. They do not support image aggregation; and
3. They have limited, or no, support for value range filtering.

What is needed are browsers with new browsing strategies with support for the above mentioned features.

1.2 PhotoCube

In this thesis we present PhotoCube, a novel 3-D image browser which is flexible and expressive enough to handle the browsing scenarios above, as well as many other similar scenarios. PhotoCube has the following key properties:

- The graphical user interface supports three browsing dimensions, some or all of which can be used simultaneously. The interface supports basic image tagging, through automatic image analysis and user tagging.
- It is based on the recently proposed multi-dimensional data model, ObjectCube, which allows users to seamlessly involve a large number of meta-data dimensions within the 3-D browsing environment.
- The data model captures a rich set of browsing actions, such as applying various filters to dimensions to focus on particular sets of images, drilling down (or rolling up) through tag hierarchies, and pivoting dimensions on and off the screen.
- The architecture of the PhotoCube browser is highly extensible, as different browsing modes are used to define the potential actions to take in many circumstances, e.g., when aggregating a large set of images for display, or when examining a set of images in detail.

We also present results from a preliminary user study, which indicate that experienced computer users find the data model of PhotoCube both useful and engaging while they find the user interface rather raw and complicated.

1.3 Thesis overview

This thesis is structured as follows. In Chapter 2 we discuss state-of-the-art image browsing tools and the origins of multi-dimensional image browsing. In Chapter 3 we discuss the ObjectCube model and describe the components used in PhotoCube. In Chapter 4 we then discuss PhotoCube, its architecture and implementation. In Chapter 5 we discuss our user evaluation and its results. We then discuss future work in Chapter 6 and finish with a conclusion in Chapter 7.

Chapter 2

Background

In this chapter, we consider the state-of-the-art in image browsing tools. We start by describing the various meta-data types that can be associated with images, and hence potentially used in browsing scenarios (Section 2.1). We consider both commercial/freeware browsers (Section 2.2) and research prototypes (Section 2.3), before briefly introducing the origins of the multi-dimensional data model used by PhotoCube (Section 2.4).

2.1 Image Meta-Data

"A picture is worth a thousand words" is an old saying, often used to remind one of the power of images. This saying works the other way around as well, as images can be described rigorously using various image meta-data attributes. These attributes are commonly categorized by their origin as follows.

Photo Header Attributes: Camera settings and scene information are often recorded by cameras and are written into the image header. Examples of such information are shutter-speed, date and time, focal length, exposure compensation, metering pattern, and flash usage, to name a few.

Calculated Attributes: Various attributes can be calculated directly from images. These include extraction of elementary characteristics of images such as shapes, colors, and texture, as well as more advanced analysis methods such as object recognition or face recognition.

User-Generated Attributes: These are attributes that a user relates to the image content. The typical form of such attributes is a text tag, that may be linked to the image or

a part of the image. This type of attribute, along with some of the photo header attributes, is most commonly used in today's image browsers.

Turning back to the browsing scenarios in the introduction, we observe that the first two conditions of Scenario 1 involve attributes that may either be user-generated or calculated (location could be based on GPS coordinates supplied in the image header). Neither condition, however, refers directly to particular tags, but rather to a higher-level concept (Europe or friends). The last condition of Scenario 1, however, refers directly to a range of values for a photo header attribute. Scenario 2 refers to similar attributes, but focuses in grouping of images based on those attributes.

2.2 Current Image Browsers

Myriad commercial/freeware image browsers have been developed to help people with browsing and managing their personal image collections. In this discussion, we divide these browsers roughly into simple browsers and advanced browsers.

We use the term *simple image browsers* for those that merely offer browsing of file system folders and viewing of images within them. In these browsers, images can only be categorized by a single criterion; this categorization is performed outside the browser by storing images in folders. Image collections are most often browsed using the folder tree and images are presented on a two dimensional grid. Examples include early versions of several photo-specific browsers, as well as traditional file-system browsers. It is clear that Scenarios 1 and 2 cannot be served by simple browsers, as they only allow browsing by a single categorization whereas both of the browsing scenarios require browsing by two categorizations at once.

We use the term *advanced image browsers* for commercial or free-ware browsers that offer more effective strategies for organizing and browsing. Many of them allow application of tags and other meta-data to images. Some also offer construction of virtual folder hierarchies within the browsers without modifying the underlying folder structure of the images. Furthermore, some more advanced browsers offer automatic analysis of the content, such as face recognition and geographical tagging. Most of these browsers allow users to search for images using user-generated tags and meta-data information, as well as using automatically generated tags. Images are most often presented in a similar fashion as in the simple browsers, namely using a two dimensional grid, but many offer time line browsing and slide show generation as well.

Examples of advanced browsers are ACDsee[1], Adobe LightRoom[2], iPhoto[3] and Google Picasa[4]. Despite their advanced features, none of these browsers adequately support the browsing scenarios above, for the following reasons.

First, although images of friends can be tagged with their names, there is no support for describing relationships between tags. The concept of a friend is thus not implicitly understood; what is needed is some means to organize tags into concept hierarchies. An industrious user can try to get around this limitation, for example by adding specific tags to encode the hierarchy of friends and adding these to all images containing any friends, or by embedding a hierarchy within the textual tags. Such attempts, however, are labor-intensive and rarely sustainable.

Second, in Scenario 1 we are filtering on a specific numerical value range. This is not supported in these browsers as they only support searching for specific tags. Again, an industrious users may list all tags in the range to retrieve the correct set. This might be viable for small ranges, but it is impractical for large ranges and impossible for irrational values.

Finally, none of these browsers adequately support grouping of images based on contents. Some might support grouping based on a single concept, but Scenario 2 requires three-dimensional grouping, including the use of two different parts of the same attribute hierarchy.

In most recent operating systems, such as in Microsoft Windows 7, more sophisticated image browsing features have been added to the built-in file browser. These improvements include image tagging, grouping by a single tag and sorting images within folders by a single primitive image attribute or user-defined tag. Such file browsers can hardly be described as simple image browsers since these features belong to the advance image browsers category. Nevertheless, these browsers do not introduce any new methods over the advanced image browsers described above and therefore share the same limitations.

[1] `http://www.acdsee.com/`
[2] `http://www.adobe.com/products/photoshoplightroom/`
[3] `http://www.apple.com/ilife/iphoto/`
[4] `http://picasa.google.com/`

2.3 Research Prototypes

Many research projects have considered image browsing and have proposed many interesting methods related to that subject. We now review the most relevant research prototypes.

PhotoMesa (Bederson, 2001) provides a zoomable interface to multiple directories of images at once, grouping the images from the folders into clusters for maximal use of the screen. With PhotoMesa, however, both browsing scenarios are impossible, as the browser operates solely on the folder structure and only allows filtering by people tags.

Scenique (Bartolini & Ciaccia, 2009) is conceptually the most similar browser to PhotoCube, as it allows browsing images by orthogonal dimensions (called facets) in 3-D browsing rooms.

Camelis (Ferré, 2007) uses co-occurrences of tags in images, via logical relations, to deduce relationships, and uses them to facilitate browsing. In Camelis, images are browsed and searched using logical formulae.

Since Scenique and Camelis share some similarities with the browser we present in this thesis, we discuss these two browsers in more depth.

2.3.1 Scenique

Scenique is a multi-faceted image browser that allows images to be searched and browsed by facets. The system offers two types of facets:

Semantic Facets: Facets consist of tag hierarchies where node labels are tags and the root node represents the facet name. Tags can appear in different facets as well as in different levels in the same facet. This allows discrimination between the different usages and meanings of tags.

Visual Facets: Facets consist of hierarchies based on image similarity. The nodes are automatically constructed by calculating image characteristics using low-level features. Visual features that are available in Scenique are face recognition, color similarities and shape detections.

Figure 2.1 shows a screen capture of Scenique during a browsing session. In Scenique, users can select facets of interest and view them in a 3-D browsing room. Selected facets are viewed as orthogonal coordinate axes and image boxes of selected representatives are

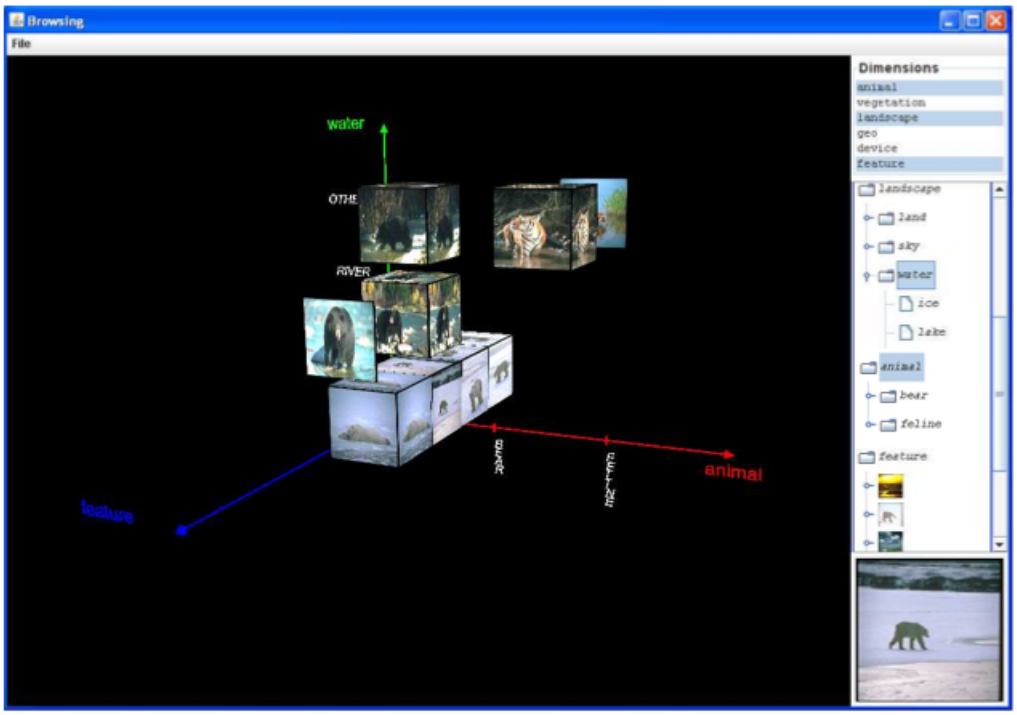


Figure 2.1: Screen capture of Scenique during a browsing session

placed at relevant coordinates with respect to the selected facets. Through the interface users can inspect facets further by selecting nodes within its hierarchical structure in the facets menu (located on the right side in the interface).

Scenique has the following key properties:

- Automatic extraction of low level features from images that can be used for searching and browsing.
- Text-based search and browsing by tag hierarchies.
- Integration of both visual facets and semantic facets by grouping the image results.

Going back to our browsing scenarios from the introduction, Scenique may not be able to handle Scenario 1 as there is no mention of range filtering. In Scenario 2 we are required to group by the same hierarcical categorization twice and view them at a different levels. In Scenique this is impossible, as a given facets can only be viewed once in a single browsing session.

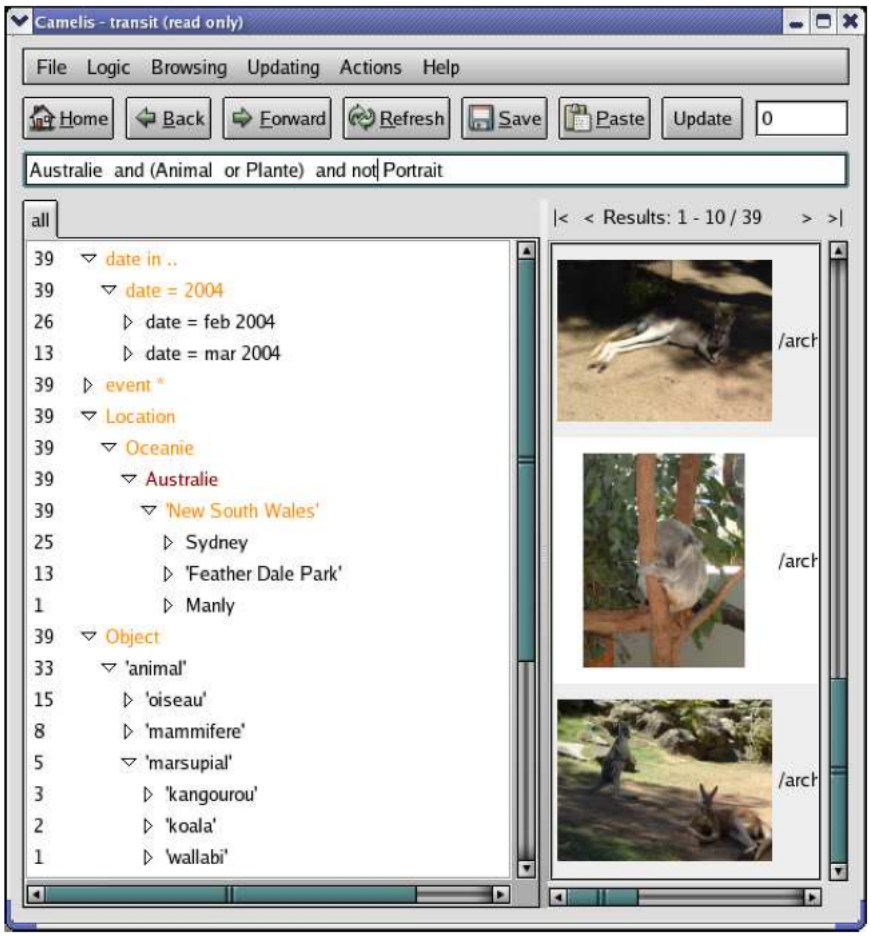


Figure 2.2: Screen capture of Camelis during a browsing session

2.3.2 Camelis

Camelis is an implementation of a logical information system (LIS) which is based on Formal Concept Analysis (FCA). LIS was introduced by the authors of Camelis to combine searching and browsing of data[5]. Figure 2.2 shows a screen capture of Camelis during a browsing session.

In Camelis, images are stored in a logical context, along with a mapping of their properties such as location, time taken and events. A logical context is defined as follows.

Definition 1 (Logical context)**.** *Let $\mathcal{L} = (L, \sqsubseteq)$ be a logic, where L is a partially ordered set of properties by a subsumption relation $\sqsubseteq$. A logical context is a triplet $K = (\mathcal{O}, \mathcal{L}, D)$ where $\mathcal{O}$ is a set of object identifiers and D is a mapping from objects to their description as a set of logical properties.*

[5] Camelis can handle any media types that can be described by their properties.

Note that Definition 1 contains two parts; first it defines a logic $\mathcal{L}$ where properties are partially ordered by a subsumption relations. Secondly it defines logical context.

Objects can be in the extent of a property without having it explicitly in their description. Property extent are defined as follows:

Definition 2 (Extent)**.** *Let p be a property, such that $p \in \mathcal{L}$. The extent of property p is defined by the set of objects whose description entails p by:*

$$extent(p) = \{o \in \mathcal{O} \mid \exists d \in D(o) \,.\, d \sqsubseteq p\} \tag{2.1}$$

Logical properties can be combined with the three well known boolean operators: $\vee$, $\wedge$ and $\neg$. Their extent are defined recursively as follows. Let q_1 and q_2 be logical formulae that can be constructed by the binary operators $\vee$, $\wedge$ and $\neg$, then

$$\begin{aligned} extent(q_1 \vee q_2) &= extent(q_1) \cap extent(q_2) \\ extent(q_1 \wedge q_2) &= extent(q_1) \cup extent(q_2) \\ extent(\neg q_1) &= \mathcal{O} \setminus extent(q_1) \end{aligned}$$

In Camelis, logical formulae can be constructed using a formulae builder or users can type them themselves into a text box directly. Properties for a given query are displayed as a tree based on its relations. The images returned appear in a linear fashion in the interface.

Applying logic on image properties and their relations for searching and browsing image collection is a neat and powerful approach but can have some drawbacks. First, this approach is only suitable for individuals who know how logical operators work. Despite the fact that simple formulae are created through the interface by clicking on available properties, more complex queries must be handwritten by users.

Second, the implementation of Formula 2.1 can be computationally expensive when the system has many images and properties relations. This is acknowledged by the authors, who state that Camelis is efficient only up to $10,000$ images (Ferré, 2007), which would be considered rather small by photographers, let alone enthusiasts.

Finally, Camelis does not support range filters. They can be emulated by creating long $\vee$ formulae, containing all the values within the range, but that would only work for a small set of values.

Camelis cannot handle the browsing scenarios described in the introduction. The logical formula People $\wedge$ Europe would return the images that we are looking for in Scenario 1, but we cannot filter out the ones that have the brightness value in a certain range.

Scenario 2 is also impossible in Camelis. In that scenario we wanted to group by three dimensions of interest, but as we stated above image presentation in Camelis is linear (see Figure 2.2) and thus no image grouping is offered in Camelis.

2.4 Multi-Dimensional Image Browsing

Together, the image meta-data attributes can be considered to define a multi-dimensional image hyperspace. Selecting dimensions, and placing them together for viewing intersecting images, is called multi-dimensional browsing. The earliest mention of multi-dimensional image browsing that we are aware of was by Harðarsson and Jónsson (2007). In that work, the authors presented a very limited 3-D image browser prototype based on the Partiview browser (Surendran & Levy, 2004), which was originally developed to browse images of galaxies. Embedding Partiview gave them a simple platform to display images in a 3-D environment and controls to navigate around the images. However, Partiview inhibited the addition of new features to their prototype, which lead them to the conclusion that it should be abandoned.

In the future work section of (Harðarsson & Jónsson, 2007), however, a wish list of browsing features was presented; most of these features correspond to operations used in on-line analytical processing (OLAP) applications, in the area of business intelligence. The ObjectCube data model, which is built on the traditional OLAP data model, was developed in (Tómasson, 2011). In that work, the aim was to create an efficient multi-dimensional data model, and a browsing engine that could be used as a back-end for multimedia browsers, such as image- or file-browsers. Since the ObjectCube model is the foundation of PhotoCube, we present the ObjectCube model in some depth in the next chapter.

Chapter 3

ObjectCube

ObjectCube is a generic multi-dimensional data model[1] and a browsing engine based on the concepts of the well known multi-dimensional analysis (MDA). In the ObjectCube data model, browsing operations are used to zoom into the media collection and construct multi-dimensional browsing sets, or cubes, similar to OLAP cubes but with objects, such as images or sound files, instead of numerical facts. These cubes are then displayed using the PhotoCube interface.

In this chapter we discuss the ObjectCube model. ObjectCube is formally defined in Tómasson (2011), but in the following discussion we review the model and the browsing engine architecture, focusing on the core aspects that are necessary for understanding the PhotoCube prototype (Section 3.1). We then discuss the architecture of a prototype implementation of the ObjectCube model (Section 3.2) and briefly discuss the results from a performance evaluation that was performed on that prototype (Section 3.3).

3.1 The Data Model

The core concepts of ObjectCube are objects, tags, tag-sets, hierarchies, and filters. Together, these concepts are used to construct multi-dimensional browsing cubes. We now describe these concepts to give a better understanding of the ObjectCube model.

[1] Although we focus on image browsing, ObjectCube is not bound to any specific media type.

3.1.1 Objects

An object is any entity that one might be interested in storing in ObjectCube for later retrieval. This can be any file type that can be described by meta-data. The actual data is not stored as a part of the object, only a reference to it. In PhotoCube, objects reference images files on disk.

3.1.2 Tags

A tag is any meta-data that can be associated with an object. For instance, we might have tags for individuals in a given photo or its brightness value. The ObjectCube prototype provides the following tag types:

Alphanumerical Tag: Alphanumerical tags consist of a string of letters, numbers and special characters. An example of alphanumerical Tag might be "Hlynur Sigurþórsson", "Canon 20D" or "Adobe Photoshop CS5".

Numerical Tag: Numerical tags consist of a single integer. This type of tag is useful for anything from ISO-speed to number of individuals in a given photograph.

Time Tag: Time tags consist of four integer values which represents a time stamp. This type of tag is useful for storing, for example, the last modified time or the length of a given sound file.

Date Tag: Date tags consist of three integer values which represent a date. This type of tag is useful for storing anything from the last modification date to the date when a given photo was shot.

3.1.3 Tag-Sets

A tag-set groups together tags that are in some way cohesive; we can think of a tag-set as a category for tags. In a tag-set named "People", for instance, the tags can be the names of people. Figure 3.1 shows an example of a "People" tag-set.

Tag-sets give context to their tags and thereby reduce ambiguity greatly by allowing us to distinguish between synonymous tags in multiple tag-sets. For instance, the tag "Mouse" might occur in the two separate tag-sets "Computer equipment" and "Animals".

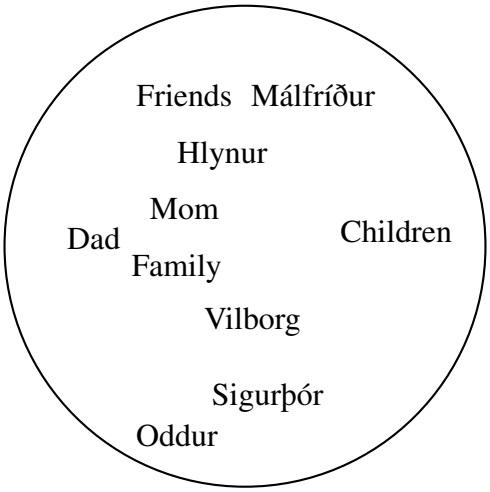


Figure 3.1: “People” tag-set

The ObjectCube prototype provides tag-set types for the four tag types mentioned above. Note that tag types cannot be mixed in tag-sets; a tag-set instance can only contain tags of the same type.

3.1.4 Hierarchies

A hierarchy is a tree structure that adds organization to a (non-strict) subset of the tags in a given tag-set. Each hierarchy belongs to one tag-set and can only use tags from that tag-set. One tag from the tag-set is selected as the root node, whereas other tags from the same tag-set can be used to extend the tree.

Figure 3.2 is an example of a hierarchy within a “People” tag-set. Tag-sets can contain infinitely many hierarchies; thus this tag-set could also contain the “Friends” hierarchy from browsing scenario 1. Both hierarchies and tag-sets are dimensions in the ObjectCube model and can be used as a dimension when constructing browsing cubes.

3.1.5 Filters

In ObjectCube, a filter is a very important concept. Filters are used to constrain the object set returned to the user. ObjectCube provides three types of filters, namely tag filters, range filters, and hierarchical filters. A filter can be applied to any dimension, tag-set or hierarchy, regardless of whether it is being shown or not.

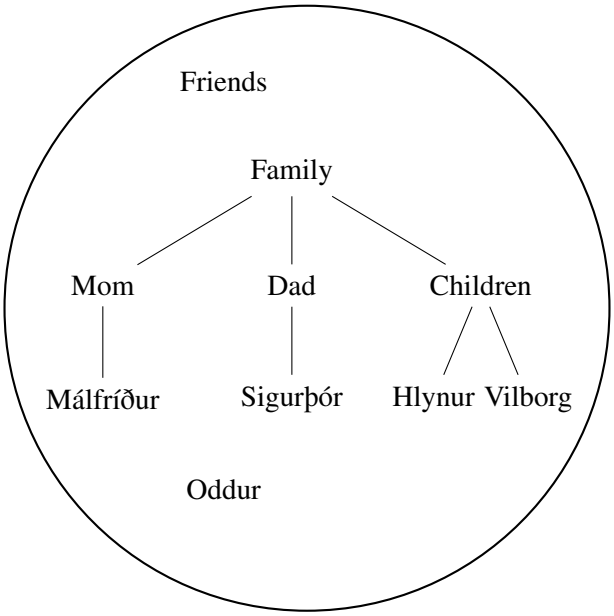


Figure 3.2: Tag-based “Family” hierarchy created within the “People” tag-set

Applying a tag filter constrains the set of objects retrieved to only those associated with the particular tag that the tag filter encapsulates. Applying a tag filter to the tag “Vilborg” in the “People” tag-set would retrieve only images tagged with “Vilborg”.

A range filter constrains the set retrieved to only those objects with one or more tags with a value within the value range of the filter. For Scenario 1, we must apply a range filter with suitable boundary values to the “Brightness” tag-set.

A hierarchical filter on a node in a hierarchy selects the entire subtree of the node, and constrains the set retrieved to only those objects associated with one or more tags in the subtree. For Scenario 1 we must apply a hierarchical filter to the “Location” hierarchy, selecting the node “Europe”.

3.1.6 Data Model Summary

We have discussed the core concepts of the ObjectCube model individually. Let us discuss how these concepts interconnect. Figure 3.3 shows an Entity-Relationship (ER) diagram of the ObjectCube model. Note that this figure has been simplified to focus on the core concepts of the model.

The Hub module in ObjectCube serves as a service layer on the model and is used to fetch objects, tag-sets, hierarchies and to apply/remove filters.

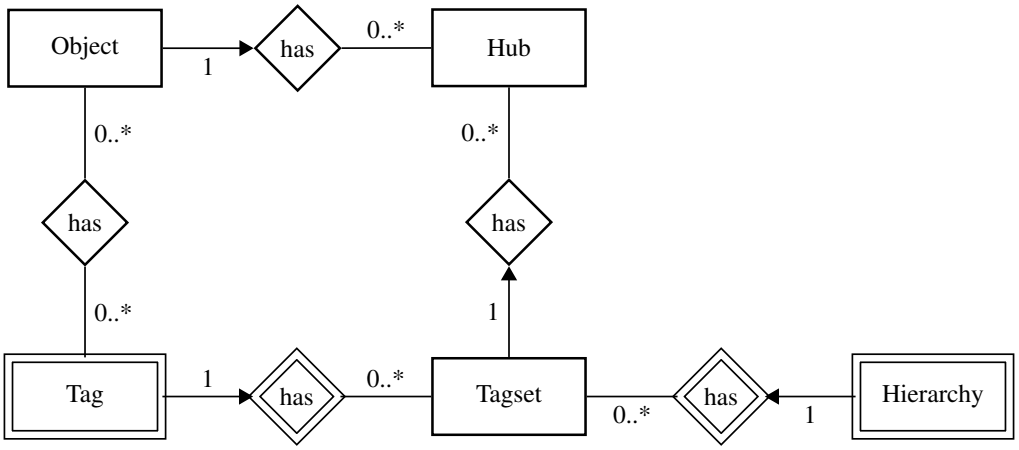


Figure 3.3: Entity-Relationship diagram for the ObjectCube model

In the figure we can see that a Hub can have zero or more objects. Objects can have zero or more tags applied to it which reside in tag-sets. A tag-set can contain zero or more tags, all of the same type. Tag-sets can then have zero or more hierarchies defined within them.

3.2 ObjectCube Architecture

A prototype of the ObjectCube model has been implemented. This implementation was written entirely in the C++ programming language with speed and efficiency in mind. Figure 3.4 depicts the overall architecture of the implementation. The architecture of the prototype is fully explained in (Tómasson, 2011), but we now discuss a few important aspects of the prototype which we utilize in PhotoCube.

3.2.1 Python Interface

The Python interface is a thin wrapper on top of ObjectCube using Boost Python. This layer allow us to easily interface with ObjectCube from Python code. This is convenient since the PhotoCube image browser is written entirely in the Python programming language.

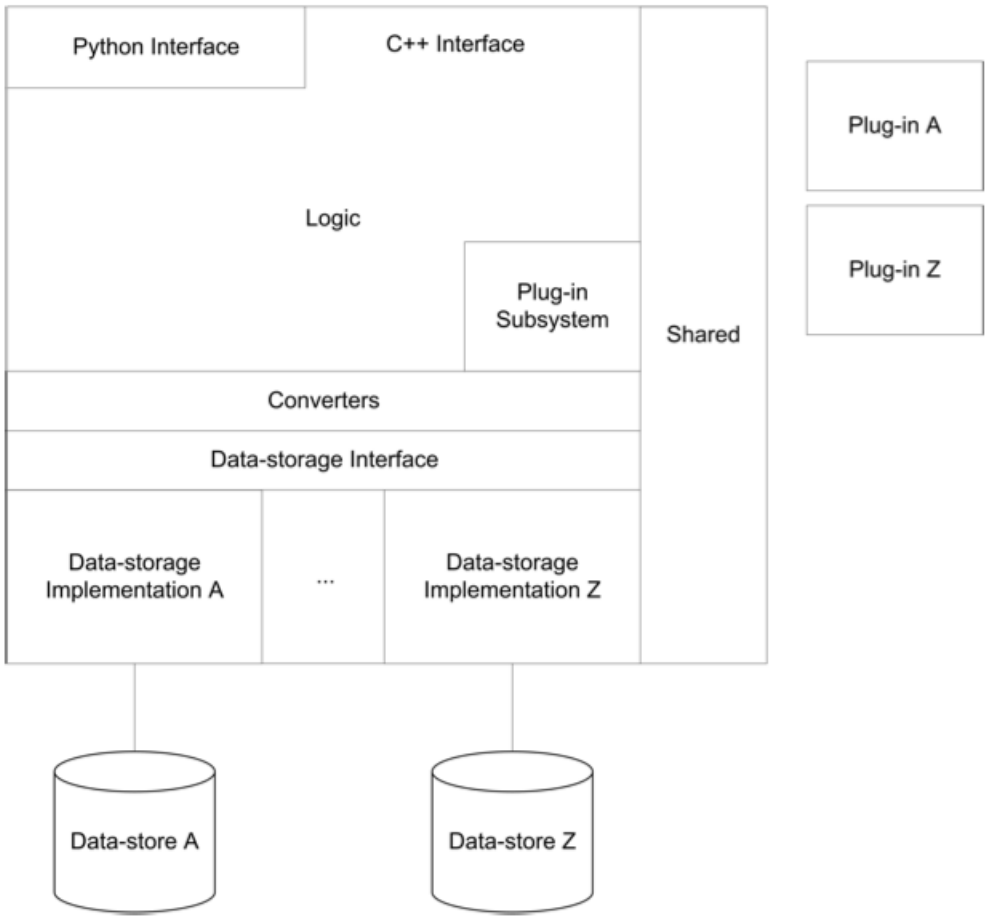


Figure 3.4: Overall architecture of ObjectCube prototype implementation Tómasson (2011)

3.2.2 Data Stores

The prototype implementation enables developers to select their own data storage. By default, the prototype supports two data storage implementations, namely for SQLite[2] and MonetDB[3]. If developers wish to use another data storage, then they must provide the data-storage implementation.

3.2.3 Plug-Ins

The prototype implementation provides a powerful plug-in architecture which allows developers to add automated analysis methods to ObjectCube. This operates as follows. When images are added to ObjectCube, they run through a pipeline of plug-ins. These plug-ins have access to the raw data of the image and can perform any analysis provided

[2] `http://www.sqlite.org/`
[3] `http://monetdb.cwi.nl/`

by the developer. ObjectCube then annotates the image with all tags returned by plug-ins.

An example is a plug-in for face recognition. When images pass through the plug-in, a face recognition algorithm would be executed and recognized faces tagged with the appropriate tags defined by the plug-in.

The plug-in architecture adds a rich flexibility the ObjectCube model where any image analysis method of interest can relatively easily be added to the model.

3.2.4 Multi-Dimensional Views

With ObjectCube, users select dimensions of interest, either hierarchies or tag-sets, and use them to construct image cubes. The model supports construction of cubes with infinite number of dimensions, but in the current version of PhotoCube we use at most three dimensions.

When a given cube has been constructed, a Multi-Dimensional View (MDV) can be requested. An MDV is a simple data structure which represents a cube as a collection of cells, indexed by their positions in the cube. A cell c in a given cube has the position:

$$c = (p_1, p_2, \ldots, p_n)$$

where n is the dimensionality of the cube and $p_1, p_2, \ldots, p_n$ are the numerical positions of the intersecting tags where the cell resides.

Objects are then placed into *appropriate* cells in the MDV. Appropriate means that all the tags which make up the cell have been associated with the object (or tags of a lower level in a hierarchy if using a hierarchical dimension in the cube).

We use the MDV to calculate positions for images in PhotoCube. Therefore we find it imperative to describe the MDV in more depth to give better understanding of how it is utilized in PhotoCube. The aim of the following discussion is to define how the appropriate cells for each image are determined. For the following discussion, we define three functions:

- $\mathcal{D}(t, i)$: Returns a set of tags with two properties. First, the tags are all part of the i'th dimension of the cube. Second, if we are working with an hierarchical dimension, then there is a path from the node for each tag in the set to the node which represents t.
- $\mathcal{A}(I, t)$: Returns $\top$ iff tag t has been applied to image I, $\bot$ otherwise; and,

- $\mathcal{T}(c, i)$: Returns the tag at position i in cell c.

We then define the a ppropriate cell for image in a given MDV as follows.

Definition 3 (Appropriate cell for image in MDV)**.** *Let $c = (p_1, p_2, \ldots, p_n)$ be a cell in an MDV. Then the cell c is an appropriate placeholder for image I iff:*

$$\forall j \exists t.[t \in \mathcal{D}(\mathcal{T}(c, j), j) \wedge \mathcal{A}(I, t)]$$

Definition 3 states that a cell c is appropriate for image I if and only if all the tags that make up c have been applied to it, or if there exist some lower layer tags within the dimension hierarchical structure that have been applied to it. Note that a given image I can have more than one appropriate cell position. When this occurs in ObjectCube, images are replicated into many cells in the cube.

The main advantages of MDV are that it:

- Facilitates cube handling;
- Positions images into correct cells in the constructed cubes; and
- Offers support for removing empty slices from the cube.

Originally, the MDV was part of the PhotoCube interface, but for efficiency reasons we decided to move it to the ObjectCube prototype.

3.3 ObjectCube Performance

A performance evaluation of the ObjectCube prototype was reported in (Tómasson, 2011). In this evaluation image retrieval was measured for the three available filter types, namely tag-filter, range-filter and hierarchical filter, to uncover any scalability or query complexity weaknesses of either the prototype or underlying data stores.

Three different data stores were considered: SQLite, MonetDB and a widely used commercial database system. Since MonetDB performed the best of the data stores, we only discuss those results here. The threshold for acceptable performance was set at one second. Note that only the meta-data was retrieved, as the retrieval time for the images is independent of the method used for meta-data retrieval.

Even when using very high selectivity of 10%, the prototype performed acceptably for 40-50,000 images. All filter types performed similarly and the overhead the prototype

imposes on the underlying data store was measured to be 140 ms or less for 1,000 retrieved images.

The key result, however, was that the performance was more dependent on the number of objects retrieved than on the size of the data-set. Using a very reasonable selectivity of 1%, the prototype performed acceptably for data-sets of over 100,000 images. Note that even 1,000 images may be an unreasonably large number of images for a user to assimilate on screen at once.

Chapter 4

PhotoCube

As discussed above, today's photo browsers do not provide an acceptable solution to image browsing, as they lack categorization and browsing by more than one criterion, as well as extensible image grouping. In this chapter we discuss PhotoCube, a novel multi-dimensional image browser built on-top of the ObjectCube browser engine, which takes these issues into account.

This chapter is organized as follows. In Section 4.1 we discuss important design goals for PhotoCube. In Sections 4.2 and 4.3 we describe the architecture and implementation of PhotoCube, focusing on currently available browsing modes. In Section 4.4 we return to the two browsing scenarios from the introduction and show how they can be solved with PhotoCube. Finally, in Section 4.5 we discuss how well PhotoCube currently meets our design goals.

4.1 Design Goals

At the beginning of this project we set forth the following design goals for PhotoCube.

- **Efficiency**
  The browser must be efficient, both in retrieving and presenting image information. This goal was one of the most important factors in our design. For efficient image presentation, e.g., we implemented asynchronous image loading and an image storage manager. As we mentioned earlier, ObjectCube is used as a back-end for PhotoCube. The prototype implementation of ObjectCube has been proven to be efficient in retrieving and calculating image informations for reasonably large number of images (see discussion on ObjectCube performance evaluation in Section 3.3).

- **Ease of Use**
  The browser interface must be easy to understand and easy to use. Though we are implementing a prototype, we wanted it to be usable and not overwhelmingly complicated in use. Therefore, we outlined the interface as a "point and click" application with as few keyboard shortcuts as possible.

- **Portability**
  The browser must be able to run on the major operating systems. PhotoCube is entirely written in Python and uses the Panda3D game engine[1] for 3-D processing. Both Python and Panda3D are available for the major operating systems, such as Linux, Mac OS X and Microsoft Windows. ObjectCube, however, is currently only available for Linux and Mac OS X. We intend to create a branch for Microsoft Windows in the future.

- **Ease of Distribution**
  It must be easy to install PhotoCube, along with ObjectCube, on end-users' computers. The installation process has not been implemented, but there are well known methods to distribute Python applications, including Panda3D, as stand-alone executable files.

- **Configurability**
  PhotoCube possesses many features that are configurable, either through a configuration file or settings dialogs. For this we implemented a singleton configuration manager that is available for all modules within the browser. Great care was taken in the implementation to make PhotoCube as configurable as possible.

4.2 Architecture

In this section we discuss the key features of the PhotoCube architecture. The aim of these features is to make the browser efficient and to provide users with a flexible and extensible browsing platform.

4.2.1 Browsing Modes

Browsing modes are the bread and butter of PhotoCube. They can be considered as visualization plug-ins on images returned from ObjectCube. The basic browsing mode is the

[1] `http://www.panda3d.org`

3-D visualization of the cube; another potential browsing mode is a slide show. Within a browsing session, the PhotoCube architecture allows for easy transitions between browsing modes, where cells in one mode can be taken into another mode with a different presentational view.

Each browsing mode supports a well-defined API and new browsing modes can be added relatively easily to PhotoCube, making the prototype very extensible.

When a given browsing mode is enabled, it gets access to the state of the current cube. The mode can then take action and present the cube, or fraction of the cube, by its definition. Modes have access to the Panda3D library and common functionality within PhotoCube.

4.2.2 Action Traces

Since browsing essentially boils down to adding and removing filters, as well as determining dimensions to visualize, it is easy to store action traces for a given browsing session. These traces can subsequently be read from disk and replayed, similar to an animated slide-show or to do an undo, or redo on actions within a given browsing session.

Actions that are performed in PhotoCube are stored in a singleton action manager during execution. This manager can then save applied actions to disk and replay them on demand.

4.2.3 Thumbnail Creation

When images are added to PhotoCube, a thumbnail of the original image is created and stored to disk. When a given image is requested, the thumbnail version is loaded first. This is done so the cube can be presented as fast as possible. If further inspection of images is requested, the thumbnails are replaced with the original images. Modes can decide which version of images is shown.

4.2.4 Image Loading

One of the main criteria in our architectural design was that images should be presented as fast as possible to users. For this purpose, we implemented the thumbnail mechanism mentioned above, as well as image buffering and asynchronous image loading.

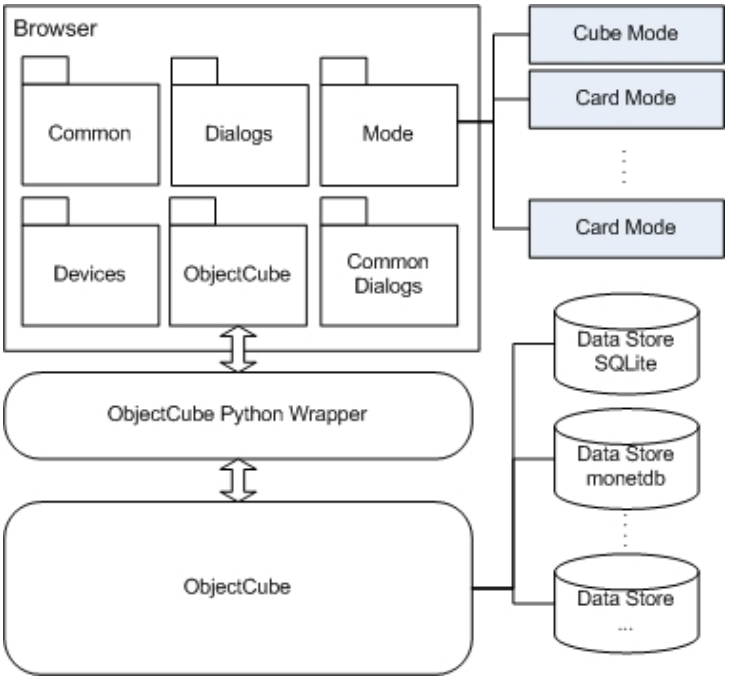


Figure 4.1: The overall architecture of PhotoCube

All images are fetched through a singleton image manager. A buffer of a predefined size is initialized and the image manager keeps the most used images in memory, based on retrieval frequency.

When images are fetched, the image manager is notified and starts loading the images asynchronously into memory. It takes into account that images from a previous browsing session might be in memory and avoids reloading them.

4.3 Implementation

We now go through the implementation of the prototype version of PhotoCube. Figure 4.1 depicts the overall architecture of the implementation. The architecture can be divided into three parts.

First we have a core, which we call “Browser”. This module is the browsing platform of PhotoCube and provides functionality for browsing modes. This module consists of a number of Python packages which have isolated purposes within the platform and provide a service layer to its functionality. The platform also implements PhotoCube’s graphical user interface, using Panda3D for graphical management.

Second, we have the browsing mode implementations (depicted as shaded boxes in Figure 4.1). Browsing modes utilize the platform for carrying out their tasks. As stated above, the purpose of browsing modes is to implement the presentation of images in browsing sessions.

Last we have ObjectCube. As Figure 4.1 shows, the platform communicates with the ObjectCube prototype through a Python layer which resides on top of the ObjectCube C++ implementation. ObjectCube handles communications with external data storage.

Together these three artifacts—the platform, the browsing modes and the ObjectCube model—form PhotoCube.

The rest of this discussion is structured as follows. In Section 4.3.1 we describe the package structure the browsing platform in more depth and discuss the role of each package individually. In Section 4.3.2 we present PhotoCube's user interface. In Sections 1 and 4.3.4 we discuss the implementation and functionality of the available browsing modes. In Section 4.3.5 we discuss the browsing mode API and how it is used to develop new browsing modes for PhotoCube. We then finish in Section 4.3.6 by discussing dependent libraries.

4.3.1 Package Structure

As stated above, the implementation of the browsing platform consists of a number of Python packages. Figure 4.1 depicts the package structure within the Browser core. We now describe these packages in terms of their purpose.

- **Mode**
  The Mode module handles all communications with external browsing modes. The functionality of this module is twofold. First, it provides interfaces for developing new modes for the browser. Second, it handles registration of new browsing modes, as well as a transition service for swapping modes during browsing.

- **ObjectCube**
  The ObjectCube module is a service layer on top of the ObjectCube Python layer. All communication with ObjectCube goes through this module.

- **Cube**
  This module handles MDVs returned from ObjectCube and converts them into cube data structures that are defined in the platform. This conversion is done for convenience and adds more information which can be used by browsing modes.

- **Dialogs**
  The Dialogs module contains dialog management in the browser. The module contains a number of pre-defined dialogs that other modules can use, as well as interfaces for browsing modes to create new dialogs.

- **Common**
  The Common module contains common code that is shared within the platform. Common contains, e.g., the browser configuration manager, system logging and action trace management. This module also contains methods which browsing modes can use to calculate positions for images on screen and to do simple animations for images.

- **Devices**
  The Devices modules handles all peripheral device communication. This module offers a rich service layer for registering mouse- and keyboards events as well as interfaces for other devices.

4.3.2 User Interface

Figure 4.2 shows a screen capture from PhotoCube's user interface. When PhotoCube is started it enables Cube mode (see Section 4.3.4) where users can construct browsing cubes. On the right side of the interface we have three menus. The top menu shows visible dimensions in the currently active cube. The middle menu is the dimensions menu. This menu contains all available dimensions in ObjectCube that can be used to construct a cube. As was mentioned in Chapter 3, tag-sets and hierarchies are both treated as dimensions; thus they can both appear in the dimension list. The bottom menu shows the currently active filters in a browsing session.

Cubes are constructed by selecting viewing orientation for dimensions in the dimensions menu. Available orientations are "front", "up" and "in". We use these names for the axes instead of x, y and z as this is a natural order for adding dimensions, but they roughly correspond to x, z and y.

Filters are added from context menus which appear when a given dimension from the dimensions list is selected.

In this project we were mainly focusing on the browsing part of PhotoCube. However, a usable browser must provide more than only browsing. To be usable and competitive against today's browsers we must implement features for tagging and searching images as well as constructing hierarchies and tag-sets. Today, there is limited support for these

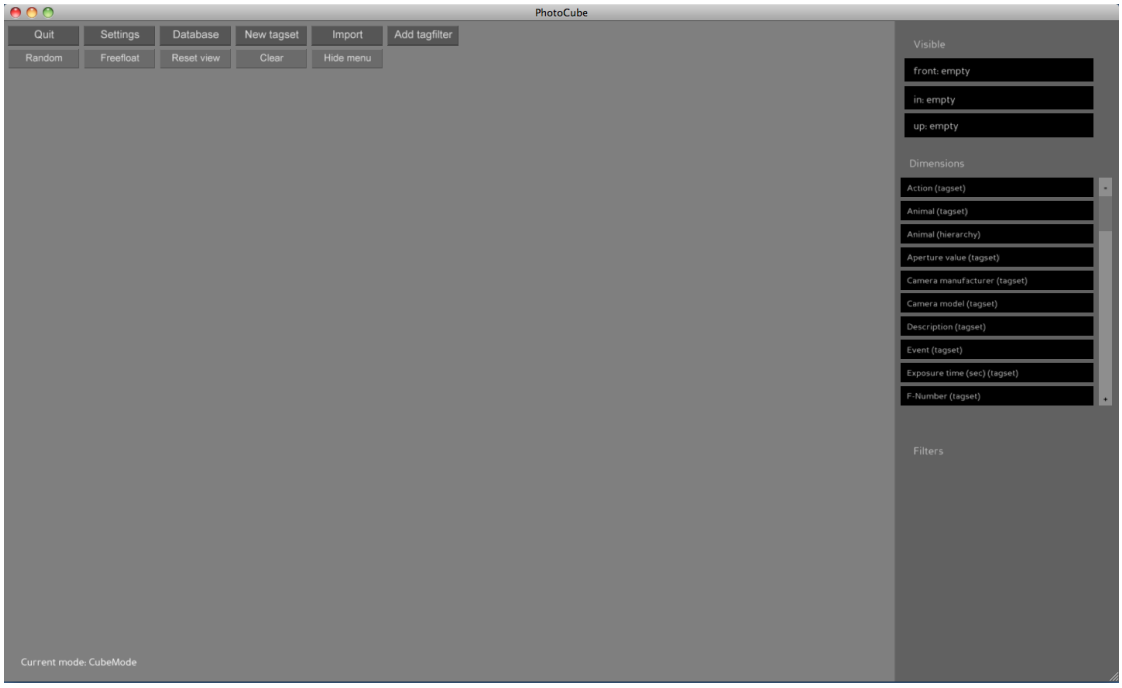


Figure 4.2: Screen capture of PhotoCube's interface

crucial operations but we outline ideas effective tagging in our discussion of future work in Chapter 6.

4.3.3 Browsing Modes

Currently, there are three browsing modes available for PhotoCube. We now discuss these modes and their behavior.

1. **Cube Mode**
   This mode is a direct 3-D representation of the ObjectCube data model and is PhotoCube's main browsing mode. Figure 4.3 shows a screen capture of the image representation by this mode. In Cube mode, dimensions are presented as orthogonal axis in a coordinate system which are built using the features supported by ObjectCube, such as filtering, pivoting, drill-down and roll-up. As Cube mode is the main browsing mode in PhotoCube, we describe its implementation in more depth in Section 4.3.4.

2. **Card Mode**
   This mode is used to view images from selected cells in more detail. In this mode, images are presented as a row of standing cards which can be easily browsed. Figure 4.4 shows a screen capture of the Card representation. Browsing is done by

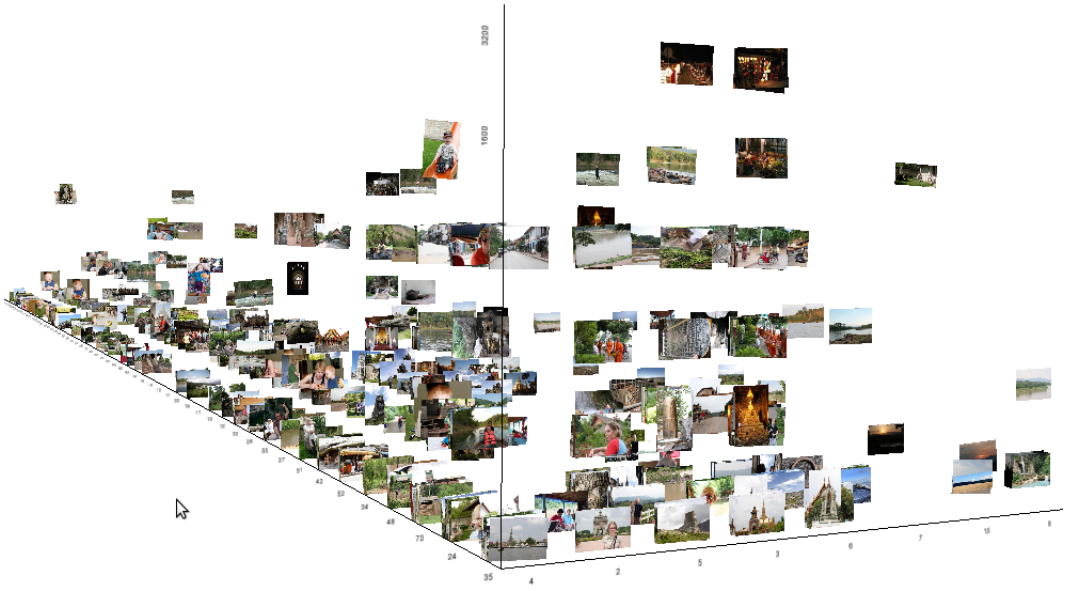


Figure 4.3: Image representation in Cube mode

flipping through the cards using either the mouse or the keyboard arrows. In the current version of the mode, image cards are ordered by the image creation date.

3. **Shooter Mode**
   This mode allows users to take selected cells from a cube into a simple shooting game. This mode was developed as a proof-of-concept mode for PhotoCube's architecture to demonstrate its extendibility.

4.3.4 Cube Mode

Cube mode is a direct 3-D representation of the ObjectCube data model. In this mode users can construct a browsing cube from available dimensions and apply all the features supported by ObjectCube, such as filtering, pivoting, drill-down and roll-up.

When a cube is constructed, the following steps are performed in PhotoCube.

1. PhotoCube notifies ObjectCube that the user wants to construct a new cube with the currently selected dimensions.
2. ObjectCube returns the MDV for the new cube.
3. The platform then converts the MDV into a cube data structure which is defined within PhotoCube. Candidate images for each cell in the cube are selected[2]. The

[2] Currently these candidates are selected randomly but there is support for adding new candidate selection algorithms.

Figure 4.4: Image representation in Card mode

platform then notifies the image manager which starts loading the selected images from disk into memory asynchronously.

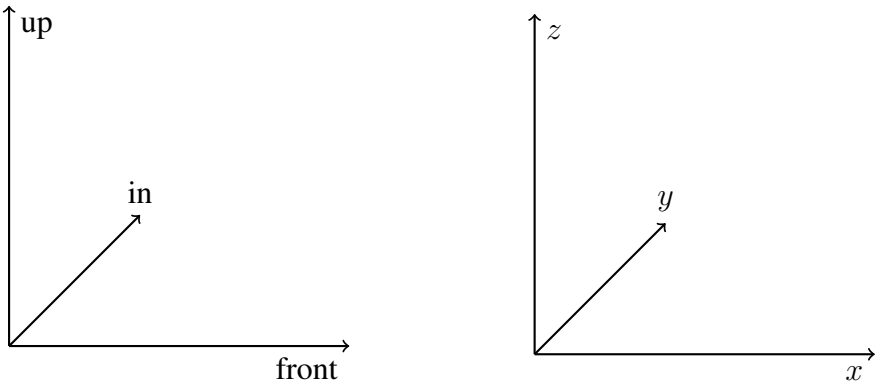
4. The platform then enables the Cube mode which presents the cube on the screen.

As stated above, PhotoCube uses Panda3D for 3-D management. Before we describe how Cube Mode presents the cube in step 4, let us briefly describe how the 3-D environment is set up and how it is used. As in other 3-D environments, Panda3D provides us with a three dimensional coordinate system and support for drawing primitive elements, such as dots and lines, or complex 3-D models into the coordinate system. Panda3D possesses many convenient artifacts which make 3-D programming easier for developers, such as:

- A camera that can easily be controlled within the environment;
- A scene graph which handles painting objects into the environment and culling of objects that are not in camera scope; and
- Support for altering object position and orientation within the environment.

The coordinate system in Panda3D is a right handed z-up coordinate system. This means that the x-axis is to the right, the y-axis is into the screen and the z-axis is up. As stated above, we abstract the coordinate names to front, up and in. Figure 4.5 depicts how the coordinate system of PhotoCube is represented in Panda3D.

We now return to our discussion of Cube mode. When the mode receives a new cube it begins by drawing axis lines for its dimensions. The length of each axis is calculated by the number of labels on the axis, the image scaling factor and the size of the space between images in the environment. The image scaling factor is a constant, which is used to scale the images that are presented. The scaling is done with respect to the image ratio. Each label on the axis represents a tag in the dimension. The length of the lines is



(a) PhotoCube coordinate system (b) Panda3D coordinate system

Figure 4.5: Interpretation of the Panda3D coordinate system in PhotoCube

calculated as follows. Let γ be the image scaling factor, σ be the image spacing length and $C = (c_x, c_y, c_x)$ be number of labels on a given axis. Then the axis length $L = (l_x, l_y, l_z)$ is defined as:

$$\begin{aligned} l_x &= (c_x\gamma) + ((c_x - 1)\sigma) \\ l_y &= (c_y\gamma) + ((c_y - 1)\sigma) \\ l_z &= (c_z\gamma) + ((c_z - 1)\sigma) \end{aligned} \tag{4.1}$$

The initial camera position is then computed based on L.

Next, the mode displays the candidate images for each cell. The screen position of each image is decided by its cell value from the MDV returned from ObjectCube, the image scaling factor and the image spacing length. The following formulae are used to map image cell position into a screen positions. Let $P = (p_x, p_y, p_z)$ be a triplet which represents the cell value for a given image, γ be the image scaling factor and σ the image spacing length. Then the screen position will be the triplet $S = (s_x, s_y, s_x)$ where each component is defined as:

$$\begin{aligned} s_x &= p_x(\gamma + \sigma) \\ s_y &= \frac{\gamma}{2} + p_y(\gamma + \sigma) \\ s_z &= p_z(\gamma + \sigma) \end{aligned} \tag{4.2}$$

The formula for s_y is different from that for s_x and s_z as the thickness of the image is taken into account.

When images are scaled, the image scale factor and the image spacing size are changed. We alter the values for σ and γ and recalculate the positions of all visible images using

Formulae 4.2 and redraw the axis. Note that in Formulae 4.1 and 4.2 we are always considering three dimensional browsing. When a given dimension is not used the values for its formula are set to 0.

Figure 4.3 showed a screenshot from PhotoCube in Cube mode viewing three dimensions. As we can see, images are placed within the coordinate system by their appropriate label using the formulae above and the positions from MDV. Note that if an image has been tagged with more than one tag from the same dimension, the image will appear in more than one place in the coordinate system as was stated in our discussion on ObjectCube's MDV in Section 3.2.4.

Cube mode support many features for working with the cube as well, such as support for scrolling through images in a given cell, rotating the cube, scaling the cube and flying around it using the Panda3D camera. Image cells can be selected and taken into other available browsing modes, such as the Card mode, for inspection.

4.3.5 The Mode API

In this section we will briefly describe the browsing mode API in PhotoCube. As we stated above, new browsing modes must implement an interface from the Mode package from the browsing platform. This interface is named AbstractMode and has the package location *browser.mode.AbstractMode* within the platform. The functions that a new browsing modes must implement are the following:

- **initialize()**
  This function is called when the mode is registered to PhotoCube. In this function the mode should load all the resources that they might need during their execution.
- **def load(cube)**
  This function is called when the mode is enabled during a browsing session. The mode service passes the active cube to the mode. From the cube, the mode can get a variety of information such as selected cells, currently visible images and all the available cells within the cube. In this function the presentation of the mode is implemented.
- **disable()**
  This function is called when the mode service disables the mode. Here the mode must release all resources which it might have been using during its execution and clean the drawing scene for the next mode. The cleanup is also done by the mode service before the next mode is enabled if this is not implemented in the mode.

```
from browser.mode import modeService
newMode = package.path.newmode.NewMode('NewMode')
modeService.register_mode( newMode )
```

Listing 4.1: Registering new mode to the mode service

Modes are registered by creating an instance and passing it to the mode service (see Listing 4.1). Modes can then be enabled through a context within the interface or by requesting it through the mode service by its name.

4.3.6 Dependent Libraries

PhotoCube uses Python Imaging Library (PIL)[3] for image processing, such as scaling, and Panda3D for 3-D management. Both PIL and Panda3D have a permissive free software license.

4.4 Browsing scenarios

We now describe how Scenarios 1 and 2 are solved in PhotoCube.

4.4.1 Browsing Scenario 1

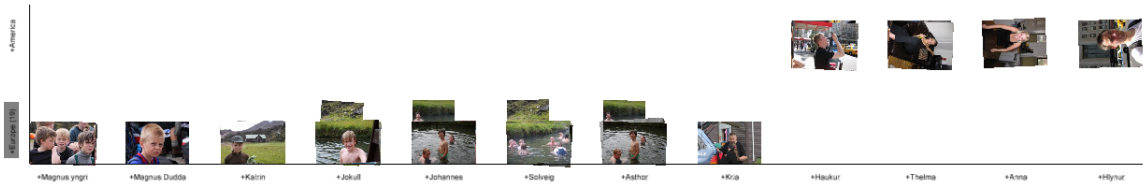
In Scenario 1, we were interested in images of our friends, taken in Europe, that had a suitable brightness value range. This is done by selecting these three dimensions in cube mode, namely the people hierarchy and the location hierarchy, and drilling down to “Friends” and “Europe” (thus adding hierarchical filters). Finally, a range filter can be added on the exposure tag-set, resulting in the correct set of images. For experienced users, this is an easy task.

Figure 4.6(a) shows the result of the first step; there we have placed the friend hierarchy on the front axis. We can see that images of our friends appear on the front axis.

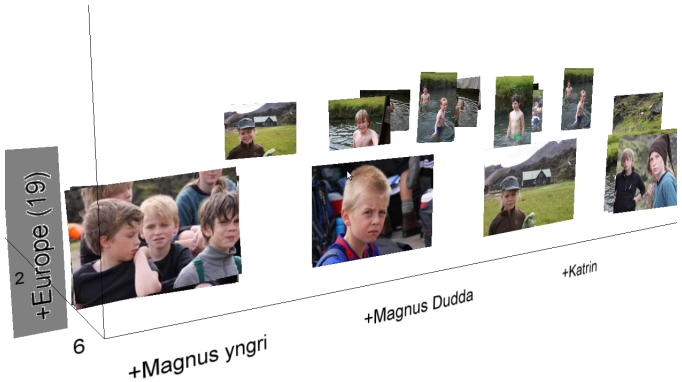
Figure 4.6(b) shows the result of the next step; there we have placed a location hierarchy on the up axis. We can see that our images have been taken in two continents, namely in “Europe” and in “America”. We can also see that our friends from America never appear on images from Europe.

[3] http://www.pythonware.com/products/pil/

(a) Friend hierarchy on front axis



(b) Location hierarchy on the up axis



(c) Exposure tag-set on in axis with range filter

Figure 4.6: Browsing Scenario 1 in PhotoCube

Figure 4.6(c) show the final step; there we have placed the Exposure tag-set on the in axis and added a range filter for the values of interest.

4.4.2 Browsing Scenario 2

In Scenario 2, we were interested in family photos, organized by parents, children, and location. In this case, we must add the “Location” hierarchy to one browsing axis and the “Family” hierarchy to the remaining two axes. Then, we must drill down into the Family hierarchy, in one case to the children, and in the other to the parents. The images will then automatically be grouped appropriately on screen. Figure 4.7 show the resulting browsing from PhotoCube.

Figure 4.7: Browsing Scenario 2 in PhotoCube

4.5 Summary

We have achieved many of the goals that we set forth at the beginning of this project, while some will need deeper consideration to be fulfilled.

- **Efficiency:** We have carefully considered efficiency in our implementation. Despite that, ObjectCube can handle vastly large image collections which our implementation cannot not handle at present. Thus finding methods to handle such cases is needed to fulfill our efficiency goal. We outline a number of ideas for this subject as future work.
- **Ease of Use:** In this project we decided to focus more on implementing features and pay less attention to the user interface. In chapter 5 we present a user evaluation where the results clearly indicate that we must pay more attention to the user interface to make it more usable and easier to use.
- **Portability:** As we stated above, PhotoCube is entirely written in Python and depends on two major libraries, namely Panda3D and PIL. Both Python interpreters and these two libraries exists for all the major operating systems. The browser is therefore highly portable between different operating system. At this time, however, ObjectCube is only available on Unix-related operating system.

- **Ease of Distribution:** The aim is to create a Microsoft Windows branch for ObjectCube. Parallel to that work, we will consider distribution of the browser for all major operating systems as well, making the project open source for other developers to take part in the project.
- **Configurability:** A careful attention was paid to the configurability in the browser, where almost all aspects of the browser can be configured through a simple configuration file or using a settings dialog within the browser interface.

Chapter 5

User Evaluation

PhotoCube is not like a traditional image browser, as it offers a completely different browsing model which might not seem natural at first to many users. Therefore we found it imperative, and at the same time interesting, to do a user evaluation on PhotoCube. The research questions of interest were:

1. *How well do users like to use the ObjectCube browsing model when browsing an image collection?*
2. *Would they like to use a similar model to browse and manage their own image collection?*

In this chapter we describe the user evaluation process and its outcome. We start by describing the experimental setup (Section 5.1) and experimental process (Section 5.2). We then discuss the results (Section 5.3 through 5.5) and end with a summary (Section 5.6).

5.1 Experimental Setup

In this experiment we considered a narrow set of users: experienced computer users. The reason for using such a skewed sample was twofold. First, the focus of this experiment was on the model, rather than on the user-interface, and more experienced computer users are more likely to understand the difference between these two concepts. Secondly, we believe that this sample of users is more likely to have used more advanced image browsing tools and we were interested in getting a comparison to the browsers they might have used.

Participant	Gender	Age	Background	Computer use
1	Male	30	M.Sc student - Software Eng.	5+ hours/day
2	Male	26	M.Sc student - Software Eng., Professional Photographer	5+ hours/day
3	Male	27	M.Sc student - Computer Sci.	5+ hours/day
4	Male	24	M.Sc student - Computer Sci.	5+ hours/day
5	Male	43	Software Developer	5+ hours/day

Table 5.1: Participant background: Demograpic Information

5.1.1 User Sample

In this experiment we got five advanced computer users to participate in our research. Four participants were graduate students of computer science or software engineering at Reykjavik University, whereas one participant has more than a decade's experience with software development. All participants are thus experienced computer users, but their experience with image browsers is varied. One participant did not possess any digital images, three participants owned between 500 and $10,000$ images, but one participant is a professional photographer and estimated his collection at about $200,000$ images. All participants were males and the average age was around 30 years. Table 5.1 contains answers from participants regarding their age, background, and computer usage and Table 5.2 contains their answers regarding their image collection and image browser usage.

5.1.2 Image Collection

The image collection used in the evaluation contained images from a five day trip along the well-known Laugavegur hiking trail in Iceland. The hiking group consisted of 9 adults and 9 kids from 5 different families. The collection consisted of $1,140$ images. Aside from 19 meta-data tag-sets extracted from the photo headers, there were 126 tags in 7 tag-sets with a total of 7 hierarchies (one user-generated tag-set had no hierarchies, while one tag-set had two). The tag-sets were:

- Events
- Days
- Locations
- People
- Objects

Participant #	Image collection	est. Size	Image browsers
1	Yes	500	Picasa, Gimp, Photoshop
2	Yes	200.000	LightRoom
3	Yes	10.000	iPhoto, Photoshop
4	No	-	-
5	Yes	5.000	Picasa, Photoshop

Table 5.2: Participant background: Image collections and browser usage

- Animals
- Impression[1]

The reason for using this set, although it was not familiar to the participants, is that it was used recently for a demonstration and was therefore well tagged (Sigurþórsson et al., 2011).

5.1.3 Browsing Tasks

The participants were asked to perform the following tasks:

1. Show images of kids by location.
2. Show images that contain a sheep.
3. Show image that contain hiking shoes and have Aperture value in the range 4 - 5.
4. Show images of people playing football.
5. Show images which contain Björn Þór, grouped by F-number, ISO-Speed and location.
6. Show me some images which you think are cool.

The purpose of the final task was to allow the users to "play around" with the prototype on their own. We believe that these task cover most of the features in ObjectCube where users must apply all the core functionality in the PhotoCube to achieve them.

The task are ordered roughly by level of difficulty. Tasks 1 and 2 can be achieved simply by viewing a number of dimensions. In Tasks 3 and 4, users must apply filters to archive a task and in Task 5 we added grouping by dimensions.

[1] The tag-set Impression contained the single tag "beautiful".

5.2 Experimental Process

The experiment was executed as follows. In the beginning, each participant was asked to fill in a consent form and a background questionnaire. The background questionnaire asked, in addition to basic demographic questions, whether users owned a digital image collection. If so, they were asked whether they used any browsing applications to manage and browse this collection, and asked to state the pros and cons of these browsing applications.

5.2.1 Task Performance

Participants were given a short presentation of the ObjectCube data model and the PhotoCube prototype, and shown how to apply PhotoCube for browsing. Then, participants were asked to solve the predefined tasks above using the prototype. The performance on the tasks was noted by the experimenter.

5.2.2 Usability Factors

Once the participants finished the tasks, they were asked to fill in a usability questionnaire, which asked whether they found the prototype to be (on a scale of 1 to 7):

- **Comfortable:** For this factor, 1 means uncomfortable and 7 means comfortable.
- **Enjoyable:** For this factor, 1 means unsatisfying and 7 means enjoyable.
- **Easy to use:** For this factor, 1 means difficult and 7 means easy.
- **Complicated:** For this factor, 1 means complicated and 7 means simple.
- **Appealing:** For this factor, 1 means boring and 7 means fascinating.
- **Pliable:** For this factor, 1 means rigid and 7 means pliable.
- **Encouraging:** For this factor, 1 means discouraging and 7 means inspiring.
- **Imaginative:** For this factor, 1 means traditional and 7 means imaginative.
- **Useful:** For this factor, 1 means unsuitable and 7 means useful.

Participant	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6
1	B	B	B	A	A	A
2	B	B	A	A	A	A
3	A	A	B	A	A	A
4	C	B	A	A	A	A
5	C	C	A	B	A	A

Table 5.3: Task performance (see Table 5.4 for explanation of performance indicators)

We believe that all good applications have these qualitative, thus we found it imperative to use these usability factors in our user studies. The questionnaire that was used in our experiments was based on Attrakdiff[2].

5.2.3 Interview

Finally, a short discussion session was held where the experimenter asked the following questions:

1. Do you find this model comfortable for browsing images?
2. What do you think are the main advantages and disadvantages of the ObjectCube model?
3. Would you consider using a similar model to browse your image collection?
4. If you were given the opportunity to alter the model, what would you change?

5.3 Task Performance

Table 5.3 shows the outcome of the task performance evaluation. Each row contain the task performance of one individual participant. Four different values were used for the performance, namely A, B, C and F; the meaning of these values is shown in Table 5.4.

Table 5.3 shows that all participants, except for participant 3, experienced some difficulties with the first two tasks. This is most likely due to the learning curve of the model and the prototype. Two users experienced significant difficulties with the first tasks and had to ask for clarifications. The instructions they received, however, consisted solely of reminding users of functionalities of the model, available operations in the browsers and

[2] `http://www.attrakdiff.de/`

Indicator	Meaning
A	Participant was able to finish the task with no problems.
B	Participant was able to finish the task but with multiple tries.
C	Participant was able to finish the task with minor help from the instructor.
F	Participant was not able to finish the task.

Table 5.4: Meaning of task performance indicators used in Table 5.3

information about the dataset. No direct help instructions were given; yet we do not see any F labels in the table. In the table we see an interesting pattern emerging. Overall, participants experienced the most difficulties with Task 1, where only one user was able to finish this task without problems. In Task 2 the performance is slightly better, but still only one user is able to finish the task without any problems. In Task 3 the performance improves further, as now two users were able to finish the task without problems and the others without any assistance. By Task 5 all the participants were able to complete the task without any problems, even though that is a relatively difficult task.

These results indicate that there is a learning curve for the browser, but that perhaps a few browsing sessions are sufficient to overcome it.

5.4 Usability Factors

Figure 5.1 shows the results from the usability questionnaire. The x-axis shows the measured usability factors, whereas the y-axis shows the average score across all participants. This figure also shows the range of the scores (the minimal and maximal scores). The usability factors are ordered from the lowest score to the highest score.

Overall, we observe that the scores are rather high across the board, ranging from 4.8 to 6.6. There are notable difference between the score for different factors, as the scores can essentially be divided into two groups. The factors on the left side, *Simple*, *Pliable*, *Easy to use*, *Comfortable* and *Encouraging* all have a comparably low average score, ranging from 4.8 to 5.6, with a wide range of scores. The widest range is for the *Simple* factor, where the average value is the lowest, as the scores range from 3 to 7. On the other hand, the factors on the right, *Enjoyable*, *Imaginative*, *Useful* and *Fascinating*, all have comparably high average scores of 6.2 through 6.6. Furthermore, the participants all seem to agree on these values, as the range of scores is quite narrow.

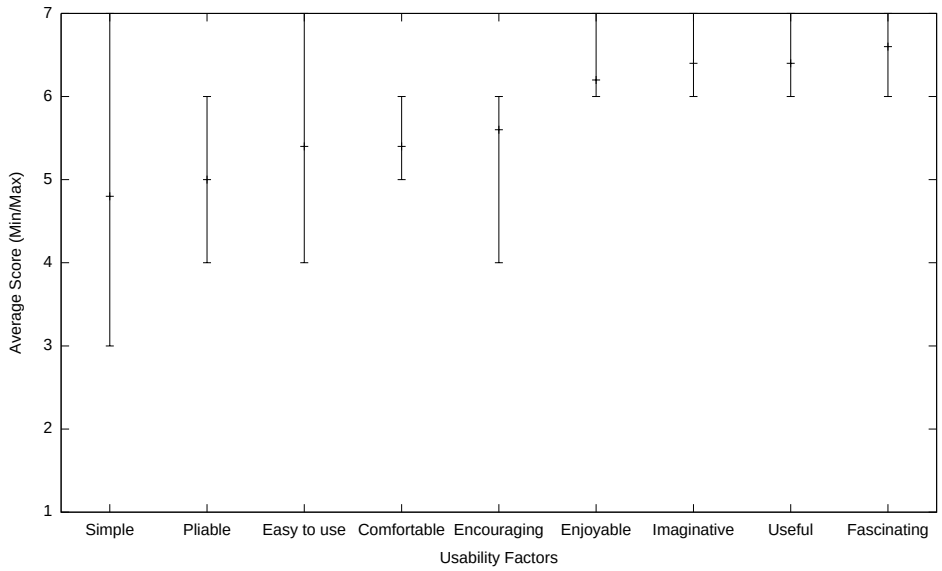


Figure 5.1: Results from the usability questionnaire

These results indicate that our participants find the prototype rather complicated and cumbersome to use, while finding it at the same time highly enjoyable, imaginative, and useful. The reason for the lower average values for the factors on the left side can be explained by the status of the prototype. The focus was to get most of the features of interest into the prototype at the cost of a poorer user interface. These features, on the other hand, might be the reason for such high values for the factors on the right side, as they can be considered exotic and innovative. The detailed data behind Figure 5.1 can be found in Appendix A.

5.5 Participant Interviews

Many useful and interesting points were brought forward in the interviews. Overall, the participants made 19 different comments on the advantages and disadvantages of the prototype. Of these, 11 comments were positive and 8 comments were considered critical, but in a constructive manner. Here we list some of the comments made by our participants. We group the comments into categories based on their focus.

5.5.1 Comments on the Interface

1. **Aggregation is convenient, particularly when multiple dimensions are viewed**
2. **The axis are convenient to understand image properties**

These two points indicate that the multi-dimensional model is suitable for image browsing.

3. **It can be hard to distinguish the difference between tag-sets and hierarchies**

This reason for this might be twofold. First, the users might have misunderstood these two concepts. Secondly the difference of these two concepts might not be adequately differentiated in the interface.

4. **The user interface is raw and cumbersome**

We are presenting a prototype and we are aware of number of issues related to our interface; Fixes for the user interface is part of our future work.

5.5.2 Comments on the Model

5. **The underlying model is closely tied to the interface, which makes it complicated to use; The model should be abstracted better in the interface.**

6. **There is a steep learning curve in the application.**

The current version of PhotoCube reflects the ObjectCube model. At the beginning of this project this was part of the design. But at later stages of the development this has been considered and will be made more comfortable in next version of the browser.

5.5.3 Comments on New Features

Many interesting comments on new features came from participants during the interview. We now state some of the ideas and discuss how they might fit into ObjectCube.

7. **Save browsing scenarios**

This is one of the features that we have implemented at lower levels in the browser and is part of the architecture. No controls have been added to the interface which allows users to use this feature yet. But this is one of the first features that we will add in the next version of the browser.

8. **Implement color coding and allow users to tag images by colors**

In fact, this is possible in the current version of the browser. Users can create a color tag-set which contains a tag for each color of interest. The users would never see the

actual colors, only tag strings. A plug-in can easily be implemented to automate the color analysis.

9. **Add drop-down list with tags and allow users to type in text to add tag-filters**

This is a great comment and would make our user interface more useful for users, as in the current version adding filters is still rather primitive. This is something that we will definitely look into doing in the next version of PhotoCube.

5.5.4 Discussion

Many of the comments confim the results of both the task performance and the usability evaluation. Some participants stated that there was a steep learning curve when solving the initial tasks. They also said that they found it difficult to distinguish the concepts of tag-sets and hierarchies, but after a number of tries these concept became clearer to them. This mirrors the results of the task performance evaluation.

Also, participants commented on the user interface. These comments were mostly on user controls, location of components and lack of action feed-backs. This confirms the low value of the factors that are related to the user experience, such as user enjoyment and usability. Many of the comments from the participants were also on how much they liked the browsing model, the representation of the images and how well the browser showed the connection between images and their attributes. This explains the high average values of the usability factors that focus on enjoyment and usefulness. At the end of the interview participants were asked whether they would like to use this model, or a similar model, to browse their own image collection; they all answered positively.

5.6 Summary

Today we have a working prototype which contains a large number of the features that we were interested in implementing, but this user evaluation has shown that the interface requires significant work. The research questions put forth, however, focused on the data model; how well users liked using the model, and whether they would like to use a similar model for their own collection. The user sample is skewed towards computer-savvy participants, and hence the research questions cannot be answered for general users. The evaluation results indicate, however, that the model has significant potential to improve the state-of-the-art in image browsing, which is very encouraging for our future work on PhotoCube.

Chapter 6

Future Work

For PhotoCube we see myriad of potential future work into many directions. In this chapter we discuss a number of ideas that we find interesting to look into in the further development of PhotoCube. We begin by discussing user interface improvements (Section 6.1). We then discuss ideas on how the ObjectCube model could be better implemented in the browser and outline additional improvements (Section 6.2). We then finish by discussing further user evaluations (Section 6.3).

6.1 User Interface Improvements

From our user evaluation, specially in the discussion session with our participants, we learned that many aspect of PhotoCubes's user interface needed refinement and alterations. First, we need to make the distinction of tag-sets and hierarchies either more clearer in the interface, or try to abstract the difference between them better. One possible solution would be to present tag-sets as hierarchies with one level.

Second, add a better functionality for adding tag filters. One suggestion which we got from one of our participant was to add an autocomplete text-box to the interface. Users could then type in tags which they would like to filter by. Synonymous tags could then be distinguished by showing their tag-set as well. This is a good idea which we would like to see in our interface and would without a doubt be a excellent addition to the browser.

At last, the most frequent comments by participants were on the GUI controls used in the interface. Currently, we are extending the GUI components from Panda3D, with a number of modifications. These controls are rather primitive and cumbersome to use, compared

to regular components, such as the ones found in today's operating systems. We would like to move our GUI components from Panda3D and use more natural components in the browser. We have been experimenting with the wxWidgets[1] GUI framework and we we will most likely integrate it into PhotoCube for handling GUI components in the future.

Many other lines of work can be mentioned for the graphical representation such as adding more flexible movements of the camera during a browsing session and adding smoother movements of images. Such features would provide a more interesting and more entertaining browsing experience for users.

6.2 Model Improvements

As we stated earlier the focus in this project was on the browsing process of the browser and creating a visual representation of the ObjectCube model where we were able to visualize dimensions, navigating hierarchies, applying filters and presenting images. This alone is only one aspect of the whole story. Users must be able to maintain the cube, add new photos, create tag-sets, construct hierarchies and apply tags to images. All this has been implemented into the browser, but the implementation is rather limited and cumbersome to use.

One line of work involves implementing a comfortable processes where these aspects are considered. For tagging we see great potential in integrating state-of-the-art image analysis methods, such as face recognition; careful handling of image meta-data dimensions containing date and time information; and effective implementation of tag-set and hierarchy management. It would also be an interesting approach to utilize the nature of multi-dimensional analysis in image tagging where cells could be dragged around and placed within cube locations for tagging.

Though searching an browsing are two different concepts in many contexts, such as on the web, they can be smoothly integrated in image browsers. For such integration in PhotoCube we see potential in adding dimensions based in image similarity, which in turn would be based on calculated image characteristics using low-level image features. Consider an example of a usage in a browsing session in cube mode: a given image is selected as the image to search by, and all similar images, based on some image characteristics, would float towards it with the final distance based on their similarity ranking.

[1] `http://www.wxwidgets.org/`

While piling images on the screen is one thing, finding an understandable presentation of large image collections is another. Another line of work involves developing methods for effectively handling large image collections. While the ObjectCube model may, in many cases, return a very large set of images, a user can probably only handle a few hundred at a time at most and we have limited resources when presenting such a large collection of images. How to aggregate the information of large sets is a very interesting open question. One way of handling large collection might be limiting the details of images and only showing a limited amount of color aggregation. Then users can analyze large sets based on most frequent colors in the image set.

6.3 Further User Evaluations

Finally, for evaluations we see many avenues of potential future work. In our user evaluation, participants were all browsing the same collection and solving the same task, whereas in personal browsing the image collection used plays a major role and to yield unbiased results the image collection must be different for each user. Executing another user evaluation experiment where image sets from the users themselves are used might give a better idea of the usability of PhotoCube. The most interesting line of work for further user studies is to distribute the browser to a number of people, get them to use PhotoCube on their image collection for some time and then invite them to participate in a larger user evaluation for the browsers. It might also be interesting to do a field experiment to see how users use PhotoCube in the comfort of their own home.

Chapter 7

Conclusions

In this thesis, we have presented PhotoCube, a novel 3D photo browser based on the recently proposed ObjectCube data model. The data model allows users to seamlessly involve a large number of meta-data dimensions in each browsing session and captures a rich set of browsing actions. The architecture of the PhotoCube browser is highly extensible, as different browsing modes can be implemented and the user can easily switch between browsing modes. In this thesis, we described the architecture of the prototype in detail, as well as its current state. We then presented results from a preliminary user study, which indicate that while experienced photo browser users find the prototype interface to have a somewhat steep learning curve, they are excited about the potential of the underlying data model.

Bibliography

Bartolini, I., & Ciaccia, P. (2009). Integrating semantic and visual facets for browsing digital photo collections. In *Proceedings of the Italian Symposium on Advanced Database Systems (SEBD).* Camogli, Italy.

Bederson, B. B. (2001). PhotoMesa: A zoomable image browser using quantum treemaps and bubblemaps. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST).* Orlando, FL, USA.

Ferré, S. (2007). CAMELIS: Organizing and browsing a personal photo collection with a logical information system. In *Proceedings of the International Conference on Concept Lattices and Their Applications (CLA).* Montpellier, France.

Harðarsson, K., & Jónsson, B. Þ. (2007). Breaking out of the shoebox: Towards having fun with digital images. In *Proceedings of the International Workshop on Computer Vision meets Databases (CVDB).* Beijing, China.

Mulligan, T., & Wooters, D. (2005). *A history of photography from 1839 to the present*. Taschen.

Sigurþórsson, H., Tómasson, G., Jónsson, B. Þ., & Amsaleg, L. (2011). PhotoCube: Effective and efficient multi-dimensional browsing of personal photo collections. In *Proceedings of the ACM International Conference on Multimedia Retrieval (ICMR).* Trento, Italy.

Surendran, D., & Levy, S. (2004). Visualizing high dimensional datasets using Partiview. In *Proceedings of the IEEE Symposium on Information Visualization (INFOVIS).* Austin, TX, USA.

Tómasson, G. (2011). *ObjectCube: Multi-dimensional model for media browsing*. M.Sc. research thesis, Reykjavik University.

Appendix A

Usability Factors data

This appendix contains the detailed answers from the user evaluation.

		Participant				
Usability Factor		**1**	**2**	**3**	**4**	**5**
Uncomfortable (1)	Comfortable (7)	5	6	6	5	5
Unsatisfying (1)	Enjoyable (7)	6	7	6	6	6
Difficult (1)	Easy to use (7)	5	4	7	6	5
Complicated (1)	Simple (7)	4	3	7	6	4
Boring (1)	Fascinating (7)	6	7	7	7	6
Rigid (1)	Pliable (7)	5	6	5	5	4
Discouraging (1)	Encouraging (7)	6	4	6	6	6
Traditional (1)	Imaginative (7)	6	7	6	6	7
Unsuitable (1)	Useful (7)	6	6	7	7	6

Table A.1: Data gathered from the usability questionnaire

School of Computer Science
Reykjavík University
Menntavegi 1
101 Reykjavík, Iceland
Tel. +354 599 6200
Fax +354 599 6201
www.reykjavikuniversity.is
ISSN 1670-8539