LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
Department "Institut für Informatik"
Lehr- und Forschungseinheit Medieninformatik
Prof. Dr. Heinrich Hußmann

Diplomarbeit

Informed Browsing of Digital Image Collections

Dominikus Baur
dominikus.baur@stud.ifi.lmu.de

Bearbeitungszeitraum:	4. 4. 2007 bis 28. 9. 2007
Betreuer:	Dipl. Inf. Otmar Hilliges
Verantw. Hochschullehrer:	Prof. Dr. Andreas Butz

Abstract

The proliferation of digital cameras and their low operating costs lead to an abundance of digital photos that are not only laborious to organize but also lack the tangibility of their physical counterparts. Informed Browsing tries to ease the organization of a large media collection by applying automatic analysis methods. Tabletop displays allow for a more direct handling of virtual objects and facilitate multi-user interaction and communication.
In the course of my diploma thesis I developed and implemented an application, *flux*, that takes concepts from Informed Browsing and puts them in a tabletop interface to combine a convenient, physical treatment of photos with the advantages of digital information: manipulability and easier organization.
This work presents the basic motivation, the idea of Informed Browsing and its roots in the photowork process, related work in the same field, the design process with multiple iterations and user feedback in form of a focus group and the implementation with selected aspects and the hardware and technology side.

Zusammenfassung

Durch die weite Verbreitung von Digitalkameras und ihre niedrigen Betriebskosten entstehen eine Unmenge von digitalen Fotos, die nicht nur aufwändig zu organisieren sind, sondern auch die Greifbarkeit ihrer physikalischen Gegenstücke missen lassen. Informed Browsing versucht, die Organisation großer Mediensammlungen zu vereinfachen, indem es automatische Analyseverfahren anwendet. Tabletop Bildschirme erlauben eine direktere Handhabung von virtuellen Objekten und erleichtern Kommunikation und Zusammenspiel zwischen mehreren Benutzern.
Im Zuge meiner Diplomarbeit habe ich eine Anwendung, *flux*, entworfen und implementiert, die Konzepte aus Informed Browsing nimmt und sie auf Tabletop Computer portiert, um einen angenehmen, physikalischen Umgang mit Fotos mit den Vorteilen von digitalen Daten, Anpassbarkeit und einfachere Organisation, zu verbinden.
Diese Arbeit stellt die grundlegende Motivation, die Idee hinter Informed Browsing und seine Wurzeln im Arbeiten mit Fotos, verwandte Arbeiten im gleichen Bereich, den Designprozess, der in mehreren Stufen stattfand, und Nutzerresonanz dazu in Form einer Fokusgruppe und schließlich die Implementierung mit ausgewählten Aspekten und die benutzte Hardware und andere verwandte Technologien dar.

Aufgabenstellung

The Fluidum project investigates interaction with different types of information units in instrumented environments. In order to represent and manipulate potentially large collections of data, interactive visual representations are needed. These visualizations should support searching, grouping, categorizing and sharing of data and should work in different contexts of usage (single user, co-located collaboration on a shared display, remote collaboration, etc.).

Recently more and more technology is on the leap from our working environments into our living rooms. Large screen flat TVs, internet enabled set top boxes and media-center PCs are only the beginning. With more screen real estate and more content available at our finger tips we need to think about new ways to organize, access and interact with digital media.

To browse large photo collections efficiently a new technology to gain an overview of the collections' content has to be developed. One possible approach would be to use state of the art information retrieval technologies to pre-cluster image collections (e.g., by object, person, event) in order to reduce visual clutter. Other IR techniques could be used to inform following information selection processes, such as deciding which images to keep vs. which images to delete.

The student has to develop visualization of a digital image collection that allows, browsing, organizing, and sharing (as in story telling) and of course viewing of digital photos. Further a matching interaction technique, on the interactive table, has to be developed. Finally an evaluation of the prototype needs to be conducted.

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig angefertigt, alle Zitate als solche kenntlich gemacht sowie alle benutzten Quellen und Hilfsmittel angegeben habe.

München, July 18, 2008

. . . .

Acknowledgements

I would like to thank all the people who made this work possible:
Many thanks go to my tutor, Otmar Hilliges, who I could always rely on and who was a permanent source of creative and constructive input and never grew tired of discussing everything from theoretical background to implementation details.
I would also like to thank Peter Kunath and Alexey Pryakhin for their preparation and help with the image analysis tools and for sacrificing their Friday evening to investigate weird OPTICS results.
Andreas Butz, for good ideas and undismayed help when I wondered about odd things, like how to best draw metaballs.
The members of the focus group, who bore with me for an exhausting two and a half hours on a hot August afternoon and all the time stayed in good humour and had productive ideas.
Additional thanks go to the members of the Fluidum team and the other members of the Media Informatics Group for their comments and input on my project.
A great Thank You to my family who supported me not only financially and morally, but also helped me keep things in perspective.
And last but not least I want to thank Bettina Zech for helping me through difficult times and, of course, for building piles.

Contents

1 Introduction

During the 1950s, when high quality cameras became affordable for everybody, photography took its place in modern society as a constant companion in all situations. From this early one-camera-per-family ratio with father capturing family trips, vacations and holidays to our today's world, where the option to take a picture is available in the most unlikely of gadgets, we grew with the pictures and they became an integral part of our lives. From private moments to reports of catastrophes: Photos are one of the most powerful objects of our time. They allow us to see events from thousands of kilometers away, immerse ourselves within them, feel like we know those foreign people pictured only by looking at their grief-stricken faces. They show us one relevant moment that tells us more than a thousand words ([2]). A good photo can summarize years of war or the suffering of a whole nation and has the power to stir millions of people into action.

In our personal lives photos save milestones and are the cornerstones of our very own time lines. They let us keep track of the past and cherish moments that are no longer gone forever but last as long as the printed version of them. Photographs are a powerful tool of communication as well: Our tales become coloured, all those names get faces and grow to be something more than abstract, unreal characters. The places we have visited are no longer dots on a map, they come to life, are suddenly equal to our own realities.
We can rely on photos as dependable memory hooks that can keep more and more of our lives as technology develops, probably culminating in a camera that constantly films our lives (similar to the Dew Camera prototype by NEC ([79])).

While the prospect of such a device (maybe in an even more portable and smaller form) will stay just that for a few years to come, its implications are overwhelming. The possibility to treat life's situations like scenes in a film that can be repeated indefinitely, rewinded and fast-forwarded and even published and shared with other people opens new horizons.

Still, as technology continues along this path one central problem becomes more and more pressing: In which ways can we organize such massive amounts of data? How can one get rid of bland and tiring scenes (which life provides enough of) and only show the highlights, the most aspiring moments? And even if this filtering can somehow be established: How to reach the one record one wants to see as quickly and conveniently as possible?

1.1 Motivation

The ever increasing speed of innovation in the sector of digital cameras made those devices available for everyone in the last ten years. And because of these technical advances and the general advantages of digital cameras compared to their analogue counterparts, sales are rising to such a degree that some fabricants are stopping their analogue product lines altogether (e.g., Nikon ([80]).
In terms of absolute numbers, by now analogue cameras lead a niche existence with a total of 520.000 sold in Germany in 2006, compared to 7.850.000 of their digital competitors (further dropping from 970.000 in 2005) ([83]). At the same time prices for storage space are falling as

Figure 1 HDD storage density is improving at 100 percent per year (currently over 100 Gbit/in2). The price of storage is decreasing rapidly and is now significantly cheaper than paper or film.

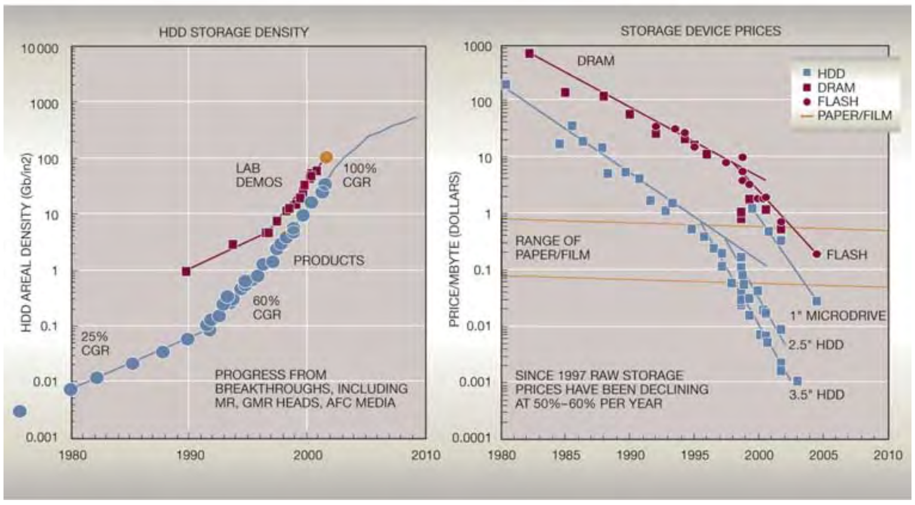


Figure 1.1: Data density and price per megabyte for IBM hard disks (from [71])

well. Especially apparent is this development for hard disks: Sizes for such components are at the moment in the half a terabyte range and are going to rise even further (for the development of price per megabyte see figure 1.1).

These two developments led to a change in consumer behaviour: While taking an analogue picture normally has been preceded by careful consideration whether the motif is really worth it, because of the costs for film, the work of bringing it to the photo lab and the price of development, a digital camera does not demand these expenditures. Just like in the original Kodak slogan ("You press the button, we do the rest" ([75])), there is not really more to do with amateur digital cameras than pressing the trigger to create a new photo, because of autofocus and even more advanced automatic features. Coupled with the ease of use of these new devices and their high level of mobility (by being lightweight and handy), large numbers of photos are taken in almost all places and situations.
In addition, more and more storage space is available on the cameras' own memory cards and the users' hard disks, so deleting a photo because of space considerations is no longer necessary.

All in all, the progress in digital photography and the missing economic limits lead to an ever increasing number of digital photos on end users' PCs. Still, while the production of those digital data has become so cheap, the other end of the chain has not been advanced in the same amount: Applications that support the organization of digital photographs can do so only if the user invests a lot of work, which leads to the rise of "digital shoe boxes", directories on the hard disk with myriads of forgotten digital photos. Additionally, a lot of consumers without a solid background in computing are simply unable to produce a meaningful organization structure, thus rendering all their digital photos useless.

Printed photos provide a number of ways of organization borrowed from physical objects in general: They can make up a photo album, be left in the folder from the photo lab and accompanied with notes and comments or be put on the wall. All these means can be performed only in an abstract sense in the digital realm - buttons and menus have to be learned, workflows memorized. The natural accessibility of physical photos is lost.

This may be the reason that while the number of developed analogue photos is sinking, that of digital ones is on the rise: Photo service provider CeWe Color for example developed in the first quarter of 2007 280.7 million digital photos and 271.3 million analogue ones, which is a rise of 46.8 percent or a fall of 28.7 percent respectively from the previous year ([66]). The numbers of photo kiosks and online photo labs are rising as well.

This need for a physical version of a photo is not only caused by the more flexible ways of organization (it is not necessarily easier), but also by the social component of photos: They are inherently connected to the event where they were taken, so a connection between the shown persons and the photo exists as well. The sharing of photos is a common part of family reunions, parties with friends and other social occasions, where people who were separated for longer times might meet again and try to give their friends and acquaintances a summary of their recent lives. But using digital photos for that does not seem to be too popular. Frohlich et al. found in a study ([18]) that "many participants reported being 'turned off' by the notion of looking at digital photos on a computer screen when sharing with friends and family. To these participants, images on a computer screen were too abstract, lacking the tangibility and manipulability of physical photographs".

These options of "tangibility and manipulability" are often neglected by photo applications. But even if they are somehow supported - watching a photo on a vertical screen is more of a shoulder-to-shoulder than the face-to-face activity of watching its physical counterpart. The latter eases communication (the so-called "photo-talk") and gesturing-supported inquiries that are essential in the sharing of such a personal object as a photo.

The recent interest in horizontal, touch-based display spaces, the so-called tabletop computing, promises to bring back some of the physical feel to digital photos while keeping their advantages (easy storing, advanced manipulation). So tabletop interfaces might bridge the gap between the two photo worlds and are an auspicious field for photo applications.

1.2 Short overview of the thesis

The thesis is separated into seven chapters:

1. The **Introduction** gives the real world background and the motivation for the development of the photo application *flux*.

2. In **Informed Browsing**, I give a short overview of studies that concern themselves with the ways people use digital and analogue photographs, show approaches to solve the problem of organizing many photos and present Informed Browsing as a possible way to overcome existing problems.

3. The **Related Work** chapter contains examples for three different types of applications: First, desktop photo applications that target either amateur or professional photographers or were developed by the scientific community. Second, tabletop applications with an emphasis on

photo manipulation and sharing. And third, one recent example of an Informed Browsing application.

4. In **Design** I describe the design process behind the application and the final outcome. Parts are the creation of the requirements, the initial designs, the description of a focus group that was conducted to proof the ideas behind them and the final, refined version that was ultimately implemented.
5. **Implementation** gives an overview of the main aspects of the implemented version of *flux*. It shows the hardware background, describes a prototype version and then explains the basic elements of the application, exemplary data flows and algorithms and existing technology that was used.
6. **Limits and criticism** contains a list of features that were dropped last-minute, a list of known program errors and the situations where they arise and finally an evaluation of the implemented version in the context of the requirements.
7. **Summary** sums up the work, contains my personal comments and points out ways to improve the application.

2 Informed Browsing

In the following, I will introduce the idea of Informed Browsing, a combination of information retrieval and interface design. It is motivated by the background research of the ways people work with photos.

2.1 Photowork

As the above described problems concerning the overwhelming numbers of digital photos came apparent, researchers began to analyze the ways in which people work with photos.
This can be a variously complex process and encompasses all actions performed with digital photos from pressing the trigger to deleting them from the hard disk. The term "photowork" that describes everything that happens with a digital photo before its intended end use (more or less all the organizing) was introduced by Kirk et al. ([29]) who gathered it from a field study. They separate the process into three parts (see figure 2.1):

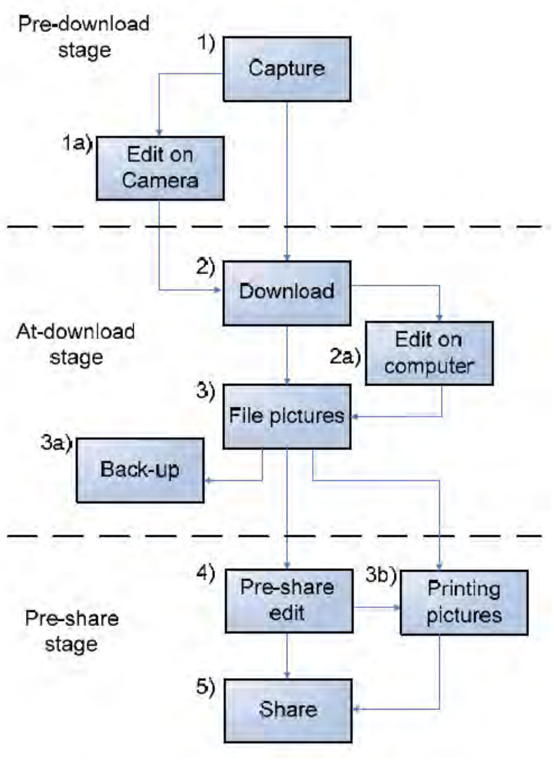


Figure 2.1: Kirk's Photowork (from [29])

The Pre-Download Stage only concerns actions on the camera like primitive editing and deleting. The At-Download Stage describes all actions taken around the time of downloading, like organizing and editing. The Pre-Share Stage lets the users prepare the photos for sharing, for example by editing them again.

Frohlich et al. examined the problem with interviews of their own ([18]) for the four different actions of archiving, sending, co-present sharing and remote sharing. They then derived guidelines for future photoware. Crabtree et al. lie their focus on remote sharing ([14]) and present ideas to support so-called photo-talk and gesturing at a distance.

Rodden and Wood analyzed people's behaviour in this regard as well ([43]), but also let them use their tool Shoebox, which had some advanced multimedia functions like speech annotation.

Their results are reassuring as well as sobering: They conclude that the two most important features for a photo application are sorting photos chronologically and showing many thumbnails at once. This enables users to find the photos they are looking for simply by browsing. High-level features were not used in the long run.

This work and other concerned with the same topic (e.g., [53]) hint at four general actions a photo application should support:

- **Browsing**: The most central action of a user describes all instances in which she goes through her collection. Browsing can happen while searching for a certain photo, but also while sharing a set of photos (e.g., of a certain event) or simply to reminisce on one's own. It can of course also be coupled with filing (with a short browsing part to keep the organization collection-wide consistent) or selecting (to see if one has already a better photo of the same motif).
- **Selecting**: Deleting photos is by now no longer necessary because of memory bottle-necks (only in special cases like on vacation or when making photos in the RAW format), but can be useful to make a collection more concise and clear. Doing a series of shots of one motif is not uncommon and keeping all resulting pictures (where one is in most cases simply the best) with equal status can increase a collection's complexity. So an application should provide means to browse such photo series effectively but also enlarge photos enough to let the user decide about its quality.
- **Filing**: All actions that belong to the organization of photos are summed up under filing. The general idea behind filing and thus differing between photos is to part the whole collection into manageable chunks based on the user's own mental image. There are different ways to organize a collection and they can revolve around hierarchical directories in the file system, additional metadata or keywords / tags. Whatever the implementation, the organization should be consistent on a global level to make the user find what she is looking for.
- **Sharing**: Whenever the user makes photos available for another person we call it sharing. Classical, co-present sharing happens with analogue just like with digital photos and has all concerned people at the same location. But digital data can also be consumed at a distance, so remote sharing via special applications, asynchronous sharing via email or even sharing of photos with an unknown audience on web photo portals is possible. Quickly accessible methods to transform (enlarge, rotate) and browse photos can support this action.

Editing might be seen as a central part as well but is not as tightly coupled with these other four activities as they are among themselves. Additionally, photo editing is mostly performed with a specialized application like Adobe Photoshop, because of the restricted feature set of a solution for organization ([43]).

2.2 Approaches for filing

The organization of the collection can be performed in many different ways, which are often predetermined by the choice of the application. Several different approaches have been created:

- **Tagging and direct search**: One way to structure a collection is to give every photo some keywords or tags that describe its content. Those keywords can be the names of shown persons, objects, the circumstances the photos was taken under, etc. So, they can have different levels of complexity and it is mostly up to the user to decide what tags are used and how complex they are. A fully-tagged collection allows for extremely convenient interaction like filtering (e.g., showing only photos with a certain person) or direct search for keywords. Unfortunately, bringing a whole photo collection to this state is a lot of work which most users (except for an excerpt of their collection in web photo portals or if under professional circumstances) do not want to perform ([43], [53]). Additionally, users do not have for every search a clear idea what photo they are looking for. If they want to find one that expresses, for example, friendship they might be at a loss with tags.

- **Hierarchical structures and naming convention**: A standard approach that can even be performed using only the file system is putting all photos into directories with a firm naming convention (e.g., "Date - Event - [Keywords]" for directories as well as for photo files). Direct search is possible via the operating system's corresponding facilities, but is mostly not as elaborate as with tags. Time-based browsing can be performed by sorting the directories based on their date of creation or their name (if the name has a format like "YYYY-MM-DD"). This approach is not only application-independent but can also be performed without too much work, so it is quite common. Yet, by being more or less a hybrid between tagging and browsing it only partially reaps both of its benefits.

- **Pure Similarity search**: Describing visual information with keywords can become difficult, so similarity search relies on the user's impression of a photo. By giving the system a red photo, it automatically retrieves pictures that have a similar look. The problems with this approach are immediately obvious: The computer can only operate with low-level visual features like colour histograms so its idea of "similar" might be radically different from the human version. But even if computers would think of similarity the same as we do, there are still logical flaws: First, two people might find a photo similar while a third would disagree, so such "perfect" similarity would only work for most, not all people. Second, similarity can also be firmly coupled with semantic background information in our minds, so two photos that look radically different but contain the same person or where taken on the same day might be interpreted as similar. Last but not least, it appears that users find it harder to retrieve the one red photo they want from within twenty other red photos than not uniformly coloured ones ([42]).

All in all, browsing is an essential part of photo applications, even if they support some kind of direct search: Searches are mostly ambiguous and the user has to browse the results of the search in any case. Almost all of the above mentioned studies reach the conclusion that supporting the browsing-action is essential.

2.3 Informed Browsing

Still, browsing on its own does only work for a very limited number of items. As the number of items increases an average browsing action takes more and more time and becomes tedious for the user. Therefore, an application that provides no direct search and only relies on browsing has to couple it with another, superordinate organization structure (e.g., hierarchical directories in the file system, hierarchical tags) to reduce the number of browsable objects beforehand.
This solution, however, bears a trade-off between the time the user invests into the organization of her collection and the time it takes to find a photo. In general, to optimize finding a photo via browsing all thumbnails should be visible on the screen at a time, so the number of (meaningfully) displayable items forms the upper boundary for the lowest level of the organization. Next-higher organization structures should provide the user with enough information of their contents to be able to judge whether the browsing should be performed within them (e.g., for directory structures with thumbnails of the most representative photos), which again provides an upper limit to stay meaningful. A standard search action is performed by drilling down from the highest organization level to the lowest one, performing a browsing operation on each level.

To make this approach feasible, two conditions have to be met:
First, the user has to create enough organizational structures to make browsing in the above described way possible.
Second, higher-level representatives have to be meaningful enough to let the user successfully judge its contents.

Both functions can of course be delegated to the user by letting her manually create for example hierarchical directories and thumbnails and names as representatives, which mostly bears the best results but is also the most labour-intensive. Additionally, once the hierarchical structures have been created based on one aspect (e.g., time) they can only be changed to another with a lot of difficulty, even if this other aspect was better suited for the search task at hand.

A solution to alleviate the necessary work for the user and make hierarchies flexible is using metadata. Some of these can be derived from information provided by the digital camera (e.g., time a photo was taken, camera settings, GPS-data), other explicitly given by the user (e.g., keywords, directory in the file system) and still other by automatic methods (e.g., similarity analysis, quality analysis).
Using this approach, the system can flexibly adapt to the user's current aims: When planning on deleting duplicate photos the user could first choose a time period from a calendar, then sort contained photos by similarity and finally let the system show photos arranged by their (perceived) quality.
Making use of all metadata (and not only of a certain category like in former systems) could give the user a starting point with camera- and automatically created metadata that lets her browse the collection and perform simple search operations. This might serve as an incentive to work with the photos and add own organizational elements like keywords to further improve its usefulness and finally arrive at a hybrid database.

An approach based on this notion is *Informed Browsing*. It tries to keep the user's amount of work as minimal as possible while trying to maximize the benefits of the collection by using information retrieval techniques and combines them with proven results from the field of interaction design. It is important to note in this regard that Informed Browsing does not use every available type of metadata, but constrains itself to certain parts. Combining as many information as possible can lead to eventually rendering all data useless, so it is used sparingly, depending on the current situation. Also, the user is able to see and control what data is currently used.

Informed Browsing also utilises the given information to reduce the user's mental load (and the complexity of the interface) by automatically creating summaries of objects if their numbers become too large to allow for a convenient usage.

The three main ideas taken from interaction design are *Overview at all times*, *Details on demand* and *Temporary structures*.

- **Overview at all times** can be achieved by letting the system summarize the whole collection or parts of it based on metadata and thus more or less creating the organization structures mentioned above on-the-fly. Additionally, the system can flexibly create a combination of metadata based on the current task. Plus, the user should be able to quickly get an overview even if she is deep within the collection.
- **Details on demand** means that the user can get further information about an object (e.g., when was the photo taken) in a quick and convenient way.
- **Temporary structures** allow for the user to create throw-away copies of objects for sharing

and storytelling that she can use without disrupting the actual structure of the collection and quickly get rid of again.

	Filing	**Selecting**	**Sharing**	**Browsing**
Quality	Global automatic clustering	Sort photos by quality, delete bad ones automatically	Only show the best pictures	Let the user toggle bad pictures
Colour	Global automatic clustering	Find under-, over-exposed photos	-	Filter the global view for certain colors
Texture	Global automatic clustering	-	-	Only show photos containing some texture
Object	Global automatic clustering	-	Filter for a certain object	
Face Detection	Global automatic clustering	Delete photos containing a failed portrait	Share only stories/photos with people/without people in it	Filter the global view for photos of people
Face Recognition	Global automatic clustering	-	Share photos containing a certain person	Filter the global view for photos containing a certain person

Table 2.1: Informed Browsing: Analysis methods for different photowork tasks

Two situations that often appear in media collections and should be supported by an Informed Browsing application are *Satisficing* and *Sidetracking*. Satisficing means that the user is looking for something but does not try to find the best solution but only a sufficing one. Sidetracking describes a situation where the user is performing a certain task but suddenly changes her mind and does something differently (e.g., because she saw a long forgotten photo).

Informed Browsing appears to be an approach to take the best of the two worlds of image analysis and interaction design to produce an application that supports users in all tasks concerned with a photo collection (see table 2.1).
Working with photos regularly generates situations where switching between orders can be beneficial. A user could stumble on a photo of a good friend and then decide (*sidetracking*) to browse more photos of him by arranging the whole collection after persons. This order can also be useful when sharing photos and wanting to put the emphasis on photos of the present persons. The range of affected photos can be limited by previously sorting the collection by time and only choosing, for example, recent photos.
Informed Browsing can also automatically reduce the complexity of a large number of photos by grouping them by time and/or similarity and showing only the most representative ones of each group.

Of course, the concept of Informed Browsing is not without its downsides: While the general idea, letting the computer perform the user's work to reap the benefits of higher-order organization, is a nice one, it always depends on the quality of the used algorithms. If the error rate of the automation becomes too high the system becomes useless. But such problems could be countered by allowing the user to change automatically produced values and generally rely more on the user-generated data than the system-generated one.

Another downside is the individuality of users: While one might fully agree with the classification performed by the system, another could find it completely wrong. So finding a general classification scheme that works for most of the users becomes a challenge, just like in a pure similarity search. So IB applications should not rely on one type of data (in this case: the similarity value) alone.

Flux is an example of an Informed Browsing application.

3 Related Work

In the following section, I will describe related applications and systems that encompass the three sections Desktop, Tabletop and Informed Browsing.

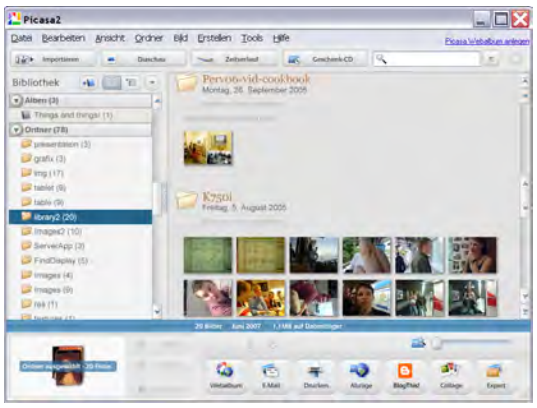
3.1 Desktop Photo applications

"Desktop photo applications" is a generic term for all software systems that let a user organize and work with her photo collection on a PC. Especially for the hobby photographer an abundance of software, both commercial and free, is available to support her in all tasks from uploading photos from a camera to the hard disk to performing high-level visual manipulation. In the following, I only present software that provides the four basic activities in photowork defined above (filing, selecting, sharing, browsing).
These applications all share the following attributes: Being based on the PC platform, they mostly adapt the classical WIMP GUI paradigm and provide options bound to menus and/or buttons. They also support the multitasking approach of modern operating systems in being executed in a window parallel to other applications. Furthermore, they work under the assumption that there is only one user working with the system (in concord with e.g., Microsoft Windows that supports only one mouse pointer at any time). A usage with more than one person is only available in a controller-spectator-scheme, where one user interacts with the system while the others are watching or even with spectators only, when the system performs some non-interactive function (e.g., a photo slide show).
Desktop photo applications are targeted at different types of users and I will divide the exemplary ones into the three different categories *Aimed at hobby photographers*, *Non-commercial with an academic background* and *Aimed at professional photographers*.

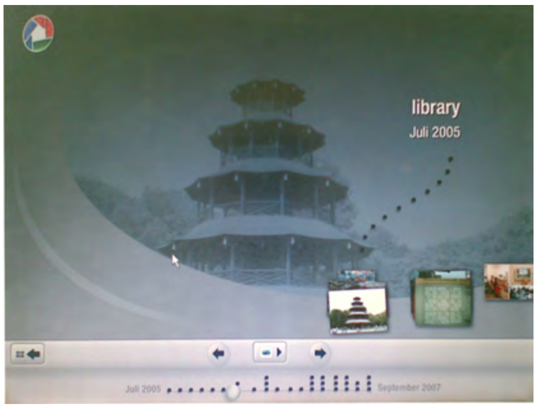
3.1.1 Applications for hobbyists

The larger part of all photo applications probably falls into this category. Such software aims at a user who does not take more than a thousand pictures a month and does not deviate too much from the classical usage of downloading photos from the camera, deleting bad ones and sorting the rest into some organization structure. Photo manipulation is only marginally supported (e.g., red eye removal) and most available functions can be performed in the same way using only the file system (but of course not in such a convenient manner).
So the Windows Explorer (Microsoft Windows) or the Macintosh Finder (Apple Mac OS) allow for the most basic organization of photos, by making the corresponding image-files accessible on a file system level. A higher category of photo applications is the packed-in software of digital cameras: Even the cheapest product comes with some basic type of that, while the most sophisticated incarnations (ZoomBrowser EX/ImageBrowser by Canon ([65]), Pixela Image Mixer for Sony cameras ([86]) or EasyShare by Kodak ([74])) provide multilayered organization schemes, tagging and advanced manipulation options (but this functional range mostly comes with the inability to disengage from the manufacturer's proprietary format).

Two exemplary, free applications are Picasa 2 by Google ([69]) and Photoshop Album Starter Edition 3.2 by Adobe ([56]). They both provide basic image manipulation capabilities, that let the user improve blurry or otherwise failed snapshots, online sharing via photo portals like flickr ([90]) and convenience options like ordering prints directly from the interface. Still, they both approach the organization aspect of a collection from two different sides: Picasa builds on the directory structure of the file system and lets the user combine photos to so-called albums. Those albums can then be named, commented on, uploaded to Google Webalbum. All directories that contain photos are shown sorted by time on the left side of the screen (the user can restrict the

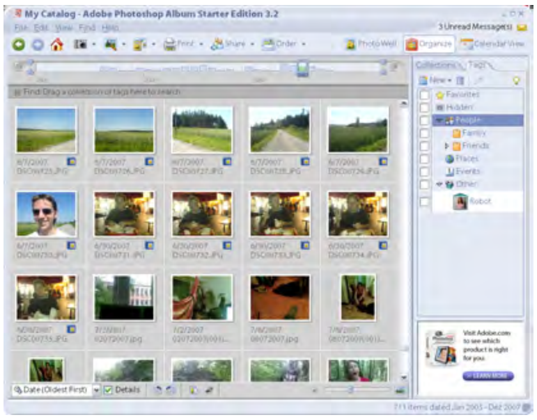


(a) Main View

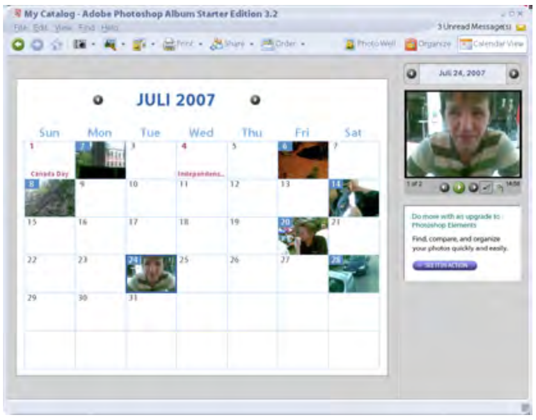


(b) Time View

Figure 3.1: Google Picasa 2



(a) Organize View



(b) Calendar View

Figure 3.2: Adobe Photoshop Album

shown directories to those belonging to a certain time period), with a larger view of their contents on the right side (see figure 3.1). A different view (which seems more like a gimmick) lets the user browse her collection based on time to switch to a slide show view.
Photoshop Album has no such hierarchical structures; it relies on the user's motivation to either add a photo to a flat collection (similar to a Picasa album) or to tag it. Tags are organized hierarchically and the whole collection can be filtered to display only photos with a certain tag or without it (see figure 3.2). As a second option, Photoshop Album lets the user see all photos month-wise on the calendar view, where for every day of the month a slide show can be triggered.
These two approaches for photo organization, using a hierarchical, folder-like structure or a flat structure with tags, are state-of-the-art for photo applications. Most other applications do not rely this heavily on one of the two and favour a more hybrid solution: HP Photosmart Essential ([70]), IMatch ([89]), Apple iPhoto ([61]) and Adobe Bridge ([57]) all use a combination of file system and tagging.

3.1.2 Academic background

The organization of a photo collection has also been the target for academic research. Some of these applications try to employ the proven Overview-at-all-times and Details-on-demand (see above) approach by providing the user with the ability to zoom in and out of the collection.

PhotoMesa ([7]) is a zoomable image browser that tries to use the available screen real estate as good as possible by using so-called Quantum treemaps. PhotoFinder ([28]) aims for a more accessible user approach to searching with the easy formulation of boolean queries, but only works well with a fully tagged collection. MediaBrowser ([17]) tries to combine some of the above and other approaches and has a convenient user interface, automated extraction of various metadata, allows the user a rapid tagging of pictures and powerful filters. Unfortunately, it does not support more than 600 photos. TimeQuilt ([27]) and the Calendar Browser ([20]) are both based on the notion that users do not want to put too much work into their collections but want to have them organized anyway. So they both rely on the temporal data that every digital camera saves with a photo and sort the photos based on it. A semantic zoom lets the user get an overview on the highest level and zoom in to get details from periods of time. TimeQuilt has its emphasis on using the available screen space as well as possible while still communicating the temporal structure. Calendar Browser creates different summaries for differently sized periods of time.

3.1.3 Software for professionals

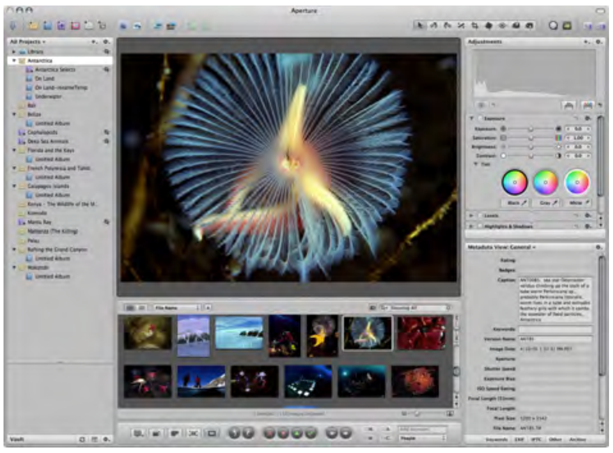


Figure 3.3: Apple Aperture 1.5 (from [60])

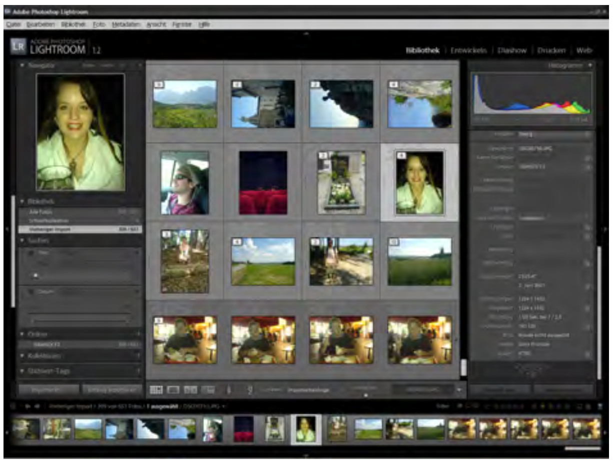


Figure 3.4: Adobe Photoshop Lightroom 1.2

Professional photographers that work with a large number of photos every day have to be able to quickly organize and manage their far larger than average collection. Larger also in the physical sense, because saving a photo in the RAW format (i.e., as uncompressed, unedited data,

which is why the RAW format is often referred to as a "digital negative") costs much more space. The advantage of saving a photo in RAW is more flexibility in post-processing, which is elsewise performed with default values in the camera for the JPEG format.
So, applications for professionals mostly support a myriad of post-processing options based on the RAW format, but also provide enhanced organization functions to allow the users to keep an overview. Apple Aperture ([60]) and Adobe Photoshop Lightroom ([58]) are the most proficient offers in this software category.
Both applications support not only hierarchical structures internally and in the file system but also tagging and rating (based on the perceived quality) of photos, with the accompanying filtering options. But in addition to those known options, photos can also be "stacked" on a lower level. Stacks (which are available in both applications) contain all photos that were taken during a short period of time (the threshold is adjustable), to automatically merge all shots of one motif and increase the clarity of the collection's structure.
Thanks to this feature they are, at least in part, representatives of Informed Browsing, though this complexity reduction is only an optional function.

3.1.4 Disadvantages

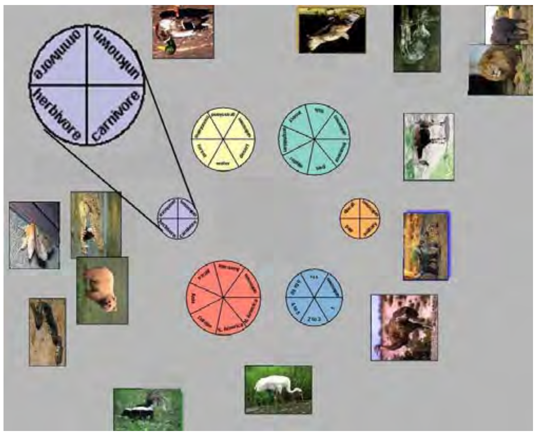
Desktop photo applications can support the user in a multitude of scenarios and provide astonishing possibilities in photo manipulation and organization.
Still, they also have their weaknesses: They aim in their operation at a single user in front of the screen and thus are awkward for presenting and sharing photos (mostly because of the problem of gesturing ([14])). PC applications also employ a firm division between digital and analogue photos: Using the former is radically different from using the latter, because it misses all tangibility and other physical qualities and forces interacting with photos into the WIMP paradigm.
Last but not least, while some applications provide automatic support for organization (like automated stacking), bringing a digital photo collection into an accessible shape is generally a lot of work and mostly tiresome. And leaving photos untagged or putting them all in one folder annuls even the most sophisticated filtering and organizing functions.

3.2 Tabletop Photo applications

Tabletop software is a relatively new field in computer science. It encompasses all applications that run on tabletop displays and the theoretical and sociological background. Typical tabletop hardware consists of a horizontally placed display that detects touch events of one or more users. Optional features are the recognition of the users' positions, distinguishing of users and recognition of objects on the tabletop.
The one aspect in which tabletop applications differ the most from regular PC ones is the inherent support of more than one user. Tables automatically invite people to gather round them to work together or separately what makes their interactive kin an ideal platform for multi-user applications. New possibilities arise from the abilities of a tabletop: First of all, interaction with the system is mostly direct, so the direct manipulation approach can be fully implemented ([26]). Additionally, multi-touch tabletops allow the user to utilize both her hands and perform actions in the virtual world similar to the real one in a bi-manual fashion. A fundamental (psychological) work in this regard is Guiard's Article from 1987 ([21]), that separates all bi-manual actions into symmetric and asymmetric ones and describes human hands as motors in a kinematic chain.
An interesting field in tabletop computing is the support of collaboration between users: Challenges that appear are the management of objects and the separation between private and public spaces. Shen from MERL and Ringel-Morris from Stanford University published several works in this regard, exploring different techniques for sharing documents ([38]), showing ways to mediate between users by system design ([40]), comparing centralized and user-replicated control schemes

in TeamTag ([41]) and the influence of group and table size on the interaction ([46]). A general overview of their research results can be found in [49].
Hinrichs et al. solved the problem of sharing documents on larger tables in Interface Currents ([25]) with a "Lazy Susan"-like interface. Kruger et al. explored the roles of orientation in tabletop interfaces ([31]) and subsequently proposed a way to integrate rotation with translation ([32]) to let users quickly rotate objects on the table. Pinelle et al. examined so-called highly integrated collaboration in real world scenarios ([36]).

3.2.1 Photo applications



(a) TeamTag (from [41])

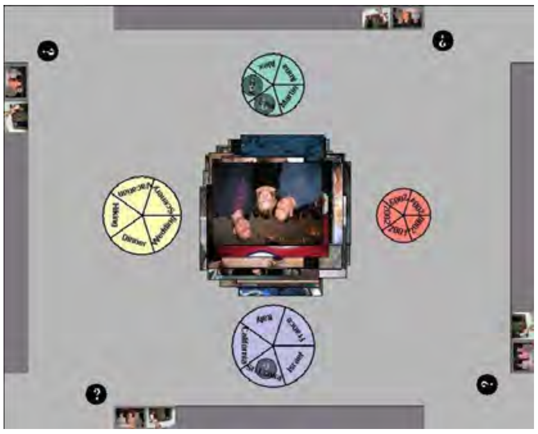
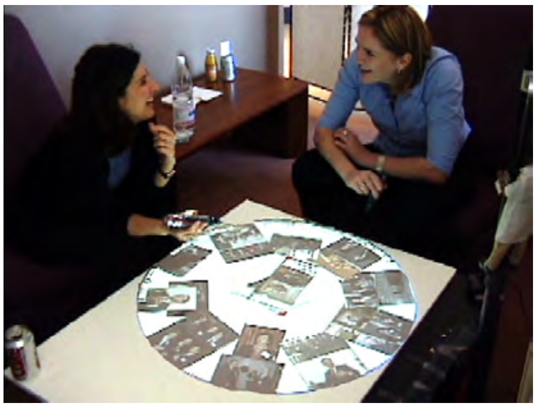

(b) TeamSearch (from [39])

Figure 3.5: Collaborative tagging and searching


(a) PDH (from [48])

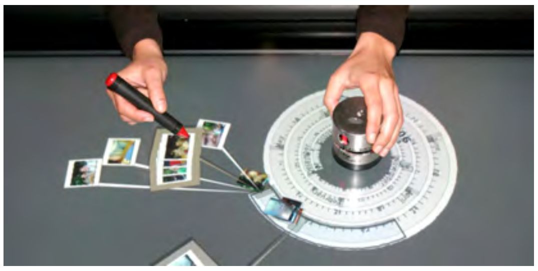

(b) PhotoHelix (from [24])

Figure 3.6: Media and Photo organization

While the above work concerns itself with general aspects of tabletop interfaces, several systems have been proposed to support the work of users with photos on a tabletop:
TeamTag ([41]) and TeamSearch ([39]) both let a team (as the name implies) tag photos and search for them in a collaborative fashion (see figure 3.5). TeamTag places all photos on the screen (either in the middle or near the border to test the influence of replicated versus central controls) and user can tag them by touching them and then subsequently one of the available tags. The system aims at making tagging photos a collaborative experience and exploiting the group memory. TeamSearch is based on TeamTag and lets the users search collectively by formulating boolean queries via drag and drop and then manipulating the resulting photos. Both Team-applications are very limited in

their usages and support only the task they were built for. They are also independently from one another, so switching from one to the other is not possible. They are not feasible (and were not meant to be) tools for a full-fledged photo organization.
SharePic ([5]) was built to support photo sharing for the elderly and has a minimal and easy-to-learn set of gestures to translate, scale and rotate photos. Apted et al. wanted to create an application that could be used by everybody by imitating some of the physical aspects of printed photos. The following evaluation showed that their two-handed scaling gesture was difficult to do for senior citizens, while younger participants had no problems. SharePic works with a physical metaphor for photos, but has no advanced organization capabilities. Still, the interface approach itself is promising.
The Personal Digital Historian ([47]) is an ongoing project of Mitsubishi Research that aims at creating a central interface to a user's media collection (see figure 3.6 on the left). Its platform is a circular tabletop display and it supports building a personal history complete with accompanied media and boolean queries ([48]). Photos are of course an important part of such a history and they are subsequently the system's focus. The user can arrange the photos along the four dimensions *who*, *when*, *where* and *what* or generate a boolean query and then zoom into her collection. While PDH provides a convenient interface, it only support sharing. Organization is not possible and where the keywords for the required partially annotated collection come from is left out. The movement of photos, however, is based on their physical versions.
PhotoHelix ([24]) is a hybrid interface with a physical handle that lets the user browse his photo collection on a spiral time line (see figure 3.6 on the right). Photos can be grouped to non-hierarchical "events". A part of the collection (which can be chosen with the physical handle) is displayed enlarged on the table and the user can interact with those photos, further enlarge and rotate them by using a button. A more "natural" interaction with a combination of translation and rotation is not supported. Additionally, the system does not scale well, because automated organization is not supported and it is only able to sort photos along the time dimension.

3.2.2 Disadvantages

While tabletop applications can take credit for inherently supporting multiple user collaboration, alleviating face-to-face communication and providing in general more physical and tangible interaction, they also have their downsides:
First of all, because of their origins in the academic sector, they mostly just exist to proof a point or to try an idea out. So they are in general unfinished and not meant to be used for longer periods of time.
Second, they do not sport the full range of features that their desktop counterparts have and support only the one thing their produced to check. Again, they are not meant for an end-user.
Third, hardware restrictions often make using them tedious, because they can only be used with special pens or only with fingers or are difficult to use because recognition works only sparingly.
As a last point, similar to the desktop versions, tabletop photo applications normally do not provide the user with automatic organization aids and therefore have to fight with the same restrictions and are forcing the user to do a lot of work.

3.3 Informed Browsing applications

With Informed Browsing being a new concept, *flux* is one of the first that implements it. Still, its general ideas appear in several instances.
Depending on how close one draws the line, every application that depends on metadata can be seen as an IB one - so even the file manager becomes an example if all files are sorted by date.
A more explicit example not for photos but for music is AudioRadar ([23]) (see figure 3.7). AudioRadar is based on metadata extracted from songs, that describes their character in a four-

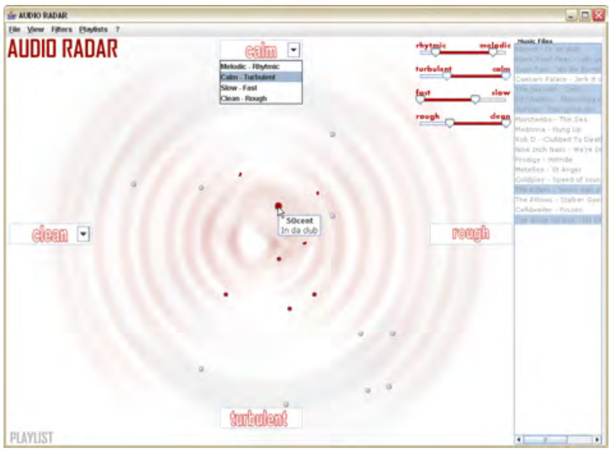


Figure 3.7: AudioRadar (from [23])

dimensional feature space (slow, fast, calm, turbulent, etc). While listening to a song the system shows its neighbours, not based on the filename but its musical character, so the user can cross her collection not based on abstract values like filename or place in the directory hierarchy but its actual audible qualities. Additionally, playlists can be automatically generated based on a mood (e.g., only listen to rhythmic songs now). AudioRadar thus uses automatic extraction algorithms to create similar results like a system that is based on extensive tagging of music.

4 Design of flux

In the following section I will present the requisites of *flux*, its design evolution, the focus group that was performed to support its underlying assumptions and its final version.

4.1 Hardware prerequisites and target group

Tabletop displays Touch-based tabletop displays could be the basis for organization and interaction with future media collections. Microsoft's Surface ([77]) is the first step to releasing this technology from the academic and industrial sectors and making it available for smaller businesses and even individuals, thus launching the feedback loop between more and more consumers and sinking manufacturing costs and prices.
Maybe some not too far away future will bring one (or more) tabletop to every household or a comparable technology that allows for the displaying on and interaction with larger surfaces.
Within the classical Ubiquitous Computing approach, where the three main display sizes *pads* (inch-sized), *tabs* (foot-sized) and *boards* (yard-sized) were first defined in Xerox PARC ([54], [55]), board-sized displays were used "in the home, [as] video screens and bulletin boards; in the office, [as] bulletin boards, whiteboards or flip charts" ([54]). But placing a large display horizontally broadens its possibilities of usage: With devices that can display information and are touch-sensitive replacing non-interactive tables, for example in the living room, casual interaction becomes common-place. The user does not have to get up to work with the system like in the PARC approach, but can sit and thus not only extend the interaction phase because of the missing exhaustion but also interact infrequently with the system, in short interaction bursts (a kind of "parallel" interaction with the system running all the time). Furthermore, tables invite people to place things on them so a connection between the tabletop and these real-world objects is the next logical step (like pioneered in Tangible User Interfaces ([52])). As a last point, multiple people can gather around one table, which makes it a prominent starting point for multi-user interaction (see above 3.2).
Tabletop displays therefore seem to be ideal to support all four central notions of photowork (see above 2.1), especially *sharing* and so this platform was chosen for *flux*.
The standard hardware prerequisite for the system is a tabletop-display that is able to recognize at least two concurrent sources of input. How this input happens (e.g., does it only work with fingers or special pens?) is not important.

Target group As for the target group, the hardware requirements almost automatically lead to a younger, more computer-savvy audience. Additionally, taking photos regularly seems necessary as well.
My ideal candidate for the system is between 20 and 35 years old, has a few years of computer experience and takes up to one hundred photos a month. The background in computing is useful because of a few concepts (e.g., hierarchical structures) that are easier to grasp with knowledge in this field. For the interaction, however, it is probably even advantageous to have no computer experience whatsoever, because of the radical break between using a standard desktop system and a tabletop interface, which lightens getting used to it for a computer novice.
The user's photo collection should span from about 1000 to 10000 photos from a period of one to a few years.

4.2 Usage scenarios

In the following section I will present three exemplary scenarios that show the basic operations that should be possible with the system. As for the background of those scenarios:
Julia, an active twenty-something, lives in Berlin in the near future. Because of her constant

nagging, her parents finally bought her the sleek new tabletop-display that now stands in her living room. She uses it for casually checking her emails or surfing the web while sitting on the couch and watching TV, but also to support her in the day-to-day access and organization of her ever-growing media collection. Photography, her current favourite pastime, centres around the tabletop and its interface. For devices to take snapshots, she owns an easy-to-use ten megapixels compact camera and her ever trusty cellphone with a five megapixels camera.

4.2.1 Alone at home to organize or randomly browse through a collection

Julia returned from a spontaneous weekend trip to Norway with her friends which not only brought her a sore knee from falling while hiking in the fjords but also about four hundred photos she took during the trip (about three hundred of the beautiful landscapes with the compact camera and a hundred from the Oslo nightlife with the phone). After dropping her baggage next to the front door, she drags herself to the couch and puts her feet up. She starts checking her emails and then connects her phone to the table to view the photos she took. The table automatically accesses the phone and adds the photos to Julia's collection. Although the system tried to make some sense of them, they appear a bit untidy in the otherwise organized lot.
Sighing, then holding her knee with a grimace, she lifts herself up and focuses the system with a short gesture on the new photos. By spatially separating the heap with a few quick movements she gets a general idea of the different topics she captured on digital film (the system assists her by automatically grouping similar photos in handy piles) and remembers the evening in Oslo, with the short warm-up in the not so glamourous bar, the few hours in the disco and the slightly inglorious return to the hotel. Julia decides to group all photos to one event called "evening in oslo" and further separate this event into the three sub-events "shady bar", "partying at the sikamikaniko" (luckily, she made a note of that!) and "triumphant return". Four gestures later she flips through those last photos with a skeptical expression: Although they are not as awkward as expected, most of them are blurry or underexposed. Before going through every one of them herself, she lets the system focus on it and separate them as it sees fit. After shortly browsing through the bad ones, she deletes them all except one that might come in handy to tease her friend Lea.
Content with her work she sits back and refocuses the system on all photos of the last few months, when the photos of an older party catch her eye. She shortly flips through them with a smile before freezing: Who is that man who stands behind her on that photo? She knows the face, she's seen him in Oslo! Julia frantically focuses on the one photo and the new event and browses through the contained photos in "partying at the sikamikaniko", hoping (or better: fearing) that one of them shows the person as well.
Having found the relevant photo a sigh escapes her lips: It was not the same man, partying people just all look the same. She resets the system to the soothing, flowy screensaver mode and lies back on the couch.

4.2.2 Showing photos to friends and family

A few weeks later, Julia is still hectically stuffing things into her unused third room, when her parents are at the door for the announced Sunday afternoon visit. After hugging her mother and receiving a fatherly tap on the shoulder she leads them into the living room to give them an update on what has happened in her life in the two months since they last visited. To support her tales, she starts up the photo application and displays the pictures of this time frame in the overview mode. She focuses on the events from two months before and flips through the photos, enlarging one once in a while, not only to illustrate her narration but also to help herself remember all the little details, that a photograph can just keep so much easier than the mind. Additionally, her own organizational structure plus the titles of the events act as a guide for her. When Julia's mother asks about one person on a photo, she easily retrieves a few more of him before returning to her current

point in the narration, dropping all the temporary copies. While her mother listens to her intently, her father, a self-appointed computer pioneer back from when the things still had cables, is at the other side of the table, not really giving her much attention, and rummages through another set of photos (but luckily not the embarrassing ones).
When Julia sees his face taking on a skeptical expression, she already knows what had happened, before he starts apologizing. But with a sigh and a few rewind-gestures she restores the organization her father just messed up.
While her parents browse through a few older galleries from their last vacation together, Julia goes to the kitchen and fetches coffee and cake. As she puts the dishes on the table, the system subtlety lets the photos flow around them, so cups and plates do not cover something, and even crumbs are spared. Her father is fascinated and finally vows to quit joking about the cuts he had to make in his budget to afford this monstrous table.

4.2.3 Exchanging photos with a friend

The same evening Lea visits Julia to get ready for a night out at the clubs. While Julia is still in the bathroom, Lea already connects her digital camera to the table to upload a few photos she took last week. They appear in a certain corner of the screen, spatially and visually separated from the others. When Julia joins her at the table, both women shortly leaf through the new photos and Lea tells about them. Finally she shows one with a smirk, where Julia is not quite looking her best. After a playful back and forth between the two, they finally agree to do a switch: Lea's embarrassing shot against Julia's Norway photo, that she has been using for one month now to annoy her friend.
They take place at opposite sides of the table, put their fingers on the two concerned pictures and move them on three, Julia hers to her private region and Lea to the little symbol representing her connected camera. While Julia is still grinning triumphantly, her friend quietly focuses the system on one of the last of the new photos and enlarges it. Julia loses her grin, as she sees herself looking even dumber than on the other one. "I guess you know what to expect this month", says Lea while disconnecting her camera.

4.3 Requirements

Several requirements for the system were derived from the ideas of photowork, Informed Browsing and the above usage scenario:

4.3.1 Support the photowork tasks

An obvious requirement for a photo application is supporting the user in the four main tasks with photos defined above: *Filing, Selecting, Sharing, Browsing*. While it is also imaginable to "outsource" some parts of it to the file system, having it all under one application's roof so the user can perform all necessary actions without changing the program context is probably the most convenient solution. For the single tasks:

- *Filing* should the user allow to give her photo collection some kind of structure. Letting the system do all the organizing would of course be a very pleasing alternative but it is unrealistic that such an automated solution could provide the same degree of accessibility and speed of use like one that was built by the user in the shape of her mental image of the collection.
- *Selecting* combines all activities concerned with the quality-wise arrangement of photographs and their deletion. While it can of course be supported by automatic methods, the last part (deleting a bad photo) should always be left to the user.

- *Sharing* means showing or giving parts of the collection to other people. This therefore sums up all activities surrounding convenient presenting as well as making the digital information accessible and copying objects within the application.
- *Browsing* is an essential part of all the other activities: The system should provide the user with the ability to easily cross the collection and reach photos she has in mind. The chance to browse well is of course coupled with a plausible organization structure.

Supporting these tasks is the main requirement for the application.

4.3.2 Support multiple concurrent users

In contrast to classical desktop setups, on a tabletop having more than one concurrent user is the normal case and not the exception. The application should therefore adapt to this situation and provide every user with a convenient experience. With the application centering around photos, every user should be able to bring his own collection into the application and share it with others (move and copy photos between different collections). Additionally, the designer has to anticipate that users are not standing still at the table but change their positions and even leave and return after periods of time.

4.3.3 Work with the characteristics of the hardware

A touch sensitive tabletop display gives the application designer new freedom in the way the users are to interact with the system. While such a setup can of course be used to build a classical desktop application based on windows and menus, new forms of interaction, maybe in a variation on Direct Manipulation ([26]), can be tried and used to better support novice users and let experts perform their tasks more quickly. The design should also try to bridge the gap between digital and analogue photos by employing an interaction scheme that borrows many aspects from the real world.
Additionally, if the hardware does not support it, a tabletop application can never be sure where the user is currently standing. So the designer should not rely on one fixed position of the user, but allow for an easy rotation and make use of general circular symmetry.

4.3.4 Support a regular-sized photo collection

An important demand for the application is working with realistically sized numbers of photos. While some interfaces might yield good results with collections with a thousand or less objects, even a hobby photographer's home collection is probably larger than that. So it is necessary that the application works just as well (or even better) with a few thousand and up to ten-thousand photos. In such dimensions different rules apply for the interaction with as well as the presentation of photos and the designer should take that into account. Informed Browsing should provide solutions for handling such numbers of media, so its main tasks, namely *Overview at all times, Details on Demand* and *Temporary structures*, should be supported.

4.3.5 Prevent the users from getting lost

Especially within such huge numbers of objects it is probably easy for a user to get lost, so the application should account for such a case and provide countermeasures. Giving the user simple ways to regain overview and also make sure that all objects are accessible at any time (and do not disappear in some dark corner of the interface) should have a high priority. A clear organization structure (see above) might help, so the user should have enough freedom to create a fitting one.

4.3.6 Reduce visual clutter

Visual clutter ([45]), a relatively new concept, describes the amount of visual distraction in an interface. As an extension to the demand of making the user not feel lost the application should be designed to reduce the amount of distracting visual input, which is mainly caused by unintentionally gaining the user's attention with blinking or moving objects. But another point that follows is to present important information as clean and concise as possible by cutting away unnecessary parts. In short: Keep it simple!

4.4 First Design

4.4.1 Initial thoughts

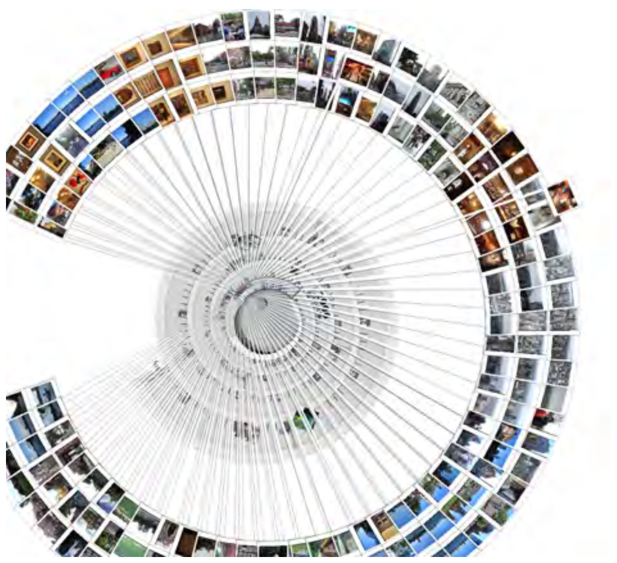


Figure 4.1: PhotoHelix with too many photos

Information overflow When starting to work on *flux*, I still had the main shortcoming of my previous application, PhotoHelix ([6]), in the back of my head: When placing many photos on the screen it brazenly ignored all imperatives of clarity and usability and simply placed them next to one another until the helix was surrounded by several photo rings, that led its usage ad absurdum (see figure 4.1). So my main design goal became the elimination of this shortcoming in building an application that could support a realistically sized photo collection of several thousand photos. I researched several existing concepts, asked friends how they do it and found that most of what the literature says is obviously true: People think that tagging is useful but too much of a hassle for their whole collection ([43]), though they try to do it when publishing photos on internet platforms like flickr ([82]). On the other hand, almost everyone I asked had come to the conclusion that some organization was necessary and relied on a homebrew solution based on a naming convention and directory structure in the file system. Giving directories a date-event-keywords name seemed to be enough, special photos (of a memorable event, of personal relevance or excellent quality) were additionally marked by some. The standard photo workflow was:

1. Attach the camera to the PC
2. Copy all photos to a temporary directory
3. Create and name directories based on the contents
4. Delete bad photos and move the rest to their directories (and optionally rename them)

Hierarchical structures So I gathered that giving the user a way to quickly put photos into some hierarchical structure was more important than letting her tag them. It was also preferable to allow a quick diving into these structures, which became clear during the PhotoHelix evaluation, where we compared PhotoHelix to the Windows Explorer and some participants praised "that I don't have to click into a folder in order to retrieve pictures" ([24]).

Giving the user the possibility to sort objects into some sort of hierarchy especially makes sense in the context of Informed Browsing, which, as the name implies, also relies on the background knowledge of the user. Letting the system do all the organizing necessarily discards information the user might have associated with a photo and which is not directly contained in some way in it (e.g., who took the photo, what were the circumstances, who is missing, etc) and also does not reproduce the place of the photo within the user's mental organization: After uploading pictures, she always has to view the system's newly created organization to match it with her own. So, even the perfect media organization system of the future probably cannot work without some kind of personal user input to make her able to retrieve an item again (not at least until computers are empathetic enough to simulate a user's emotions and thoughts so perfectly, that the generated hierarchy completely reflects the user's internal one).
PhotoHelix did not support hierarchical structures ("events" (clusters) could only contain photos - no other events), but especially in a photo collection hierarchies make sense: The standard photo opportunity of the hobbyist are the vacations ([53]), which already inherently carry a hierarchical structure, even if it is only "Holidays" - "Arrival", "Fourteen days at the beach", "Departure".
Additionally, hierarchies can be adjusted to fit the user's needs. The typical approach of course is to build the classification based on time, with sequential segments that are further divided. But users might also sort their photos by topics, persons that are on them, or even colours. Yet, here some basic problem becomes apparent: Taxonomies force the user to put an object into one fixed category as well as link the categories in one pattern. So a carefully constructed "People on photos" organization falls apart if one has photos with more than one person in it (does the photo of Simon and Tanya belong to the category "Simon" or "Tanya" or both?) and even if one then decides to put copies in all relevant categories, where do these categories lie (is "Simon" a subcategory of "People" (in an actually flat hierarchy), of "Acquaintances" or even "Simon & Tanya")? Additionally, if one searches for the category "friends" does the system also return "acquaintances" and all the subcategories of "friends"? Taxonomies have their limits (cf. [19]), but can be useful as an overarching structural element.
In a hierarchy there has to be some highest category, some place where all other categories fit in and which has to be there to keep it all together but which would be of only little use to the user ("photos"?). The second-highest classes would be more essential.
With the backgrounds of tabletop computing and multiple concurrent users I developed the notion of "streams" as high-level containers that consist of multiple photos from the same source and are sorted by time. Sources could be different cameras with which a photo was taken, for example, a digital camera or a cell phone. But there might of course be other sources as well: Scanned analogue photos, photos from another digital camera, relevant photos one got from a friend or found on the internet, different directories in the file system.

If we draw a higher line of abstraction, each source might be a different user of the system, with the previously mentioned personal sources as categories on a second level, which supports more than one user and by copying photos from one stream to another the sharing of photos between them as well.

Merging physical and digital An important point I tried to reach was to combine the advantages of analogue and digital photos: In using a tabletop hardware the first step away from the classical desktop metaphor is already performed - users approach such a device differently than a regular computer, not only because of its (momentary) novelty value but also because of the distinct way

one operates it (either standing or sitting but in any case looking down instead of staring ahead and using direct touch instead of a mouse) and the number of concurrent users. It breaks down the gridlocked expectations of people who anticipate certain standard interaction patterns because of their previous experience with computers and allows the designer of an interface to establish completely new modes of communication. Additionally, the original WIMP-approach is a product of the late seventies, adapted to the hardware of this time. A paradigm for the possibilities of current hardware may fail on standard PCs, but the tabletop platform could get away with it. One approach is to rely on existing, proven methods like direct manipulation ([26]) and combine them with a physics engine. Agarawala et al. built a standard desktop application but made all visible objects behave like physical ones ([1]), which made the interface not only visually more appealing but also (based on their initial user evaluation) more fun to use.
I also tried to use the advantages of physical objects, giving the interface a more playful feel and rely on the user's background from decades of experience with them. Photos should at least in part behave like their analogue counterparts, making it possible to shift them around on the table with a finger and rotate them easily. Picking up a photo and analyzing it in detail is obviously not possible, but at least some of disadvantages of physical objects like aging, photos sticking together, etc. are removed as well. Giving users the possibility to treat their digital photos just like analogue ones, who they have learned to love and treasure from an early age heightens the ease of entry and increases the general usability, improves their impression of the interface and makes them more inclined to stick with a purely digital version of the pictures and not let them have developed.
After lessening the downsides of interaction with digital photos, the interface should also provide the users with the advantages: Digital photos are indestructible, can be copied ad infinitum, can be easily enlarged and shrunk and manipulated in highly creative ways without scissors or glue. Tabletop applications could enrich the way we interact with photos, allowing directly "touching" objects, cutting out portions of photos without destroying them, drawing on them, adding hyperlinked, hideable notes, putting them in multiple hierarchies, virtual albums etc. An abundance of applications is imaginable. Especially the task of building a photo album could be highly facilitated: A study by Frohlich et al. ([18]) shows that most of the participants thought that putting photos into an album "was ... the best way of arching conventional photos for future sharing". Still, hardly any of them was able to fulfill this goal, because of the tediousness and complexity of the task that "often fell to the wife or mother in the families". Exploiting the directness of multi-touch interaction and the robustness and repeated applicability of digital information could strip the task of building an album of its physical dullness, grind out the creative core again and maybe make it to an activity the whole family can share, sitting at their living room tabletop display.

Displaying a large number of photos A basic problem I had to overcome was how to display a larger number of photos. The limits of the screen space are not only given by the hardware but also (at the far end) by the way a person interprets a given picture and how much visual information the human perceptive system can absorb conveniently: Even if display and eye were capable of the high resolution, displaying thousands of pictures at the same time would only increase the user's stress and decrease her productivity ([45]). Several different methods exist to solve this problem:

- **Navigation in a three-dimensional space:** Placing objects in a three-dimensional world and letting the user navigate it seems at first the natural thing to do if the hardware supports it. We as human beings interact daily with physical objects and must have gathered some skill in it. Yet, many studies suggest that a 3D-interface has no clear advantages to a 2D one ([9]) and might even yield disadvantages in spatial memorization ([10], [11]) or general usability ([12]). Additionally, three-dimensional interfaces work only if the screen has a clear relative position to the users - if the users stand around the screen (like in the tabletop case) only one of them has a 3D-impression.

- **Scrolling:** The classical approach, used within most desktop-style applications: All objects are sorted by some attribute like filename or date of creation, shown as thumbnails or icons and put on an abstract space with a high vertical size with only a section of it visible. The user can adjust this section's position on the abstract space by using a scrollbar, a mouse wheel or some other method. This can happen either continuously or on fixed pages. The disadvantages of this method are the absence of the global context and the tediousness of scrolling through a large collection of items with only a fixed number of items visible at one time.
- **Panning and Zooming:** This method is more or less an updated version of scrolling and lets the user adjust the number of currently shown items on screen (zooming) and adds movement in two-dimensions (panning). The advantages are clear: The user can control how much information she wants to have at any given moment, can see the local and global context of an object and first get a quick overview and then enlarge an interesting object without having to scroll through a number of pages. Systems that implement this notion of Pan and Zoom are for example Photomesa ([7]) and MediaBrowser ([17]). The main flaw of such an interface is its scalability: Zooming out and getting an overview of the whole collection only works up to a certain number of items, before their representations shrink to unrecognisability.
- **Semantic zoom and summaries:** As an extension of Panning and Zooming, a semantic zoom interface adjusts the complexity of visible information based on the zoom level. An example from "Pad" ([35]) shows only the years in a calendar on the highest level and gradually fades in months and days as the user keeps on zooming in. Other exemplary interfaces are Time Quilt ([27]) and Calendar Browser ([20]) (see above: 3) that visualize photo collections by using this approach. Semantic zoom alleviates the main problem of a zoom-interface while preserving its advantages, but the automated summarization process necessary to do so can of course fail and leave the user no choice but to scroll laboriously through the whole collection. In the context of Informed Browsing, semantic zoom can mean the combination of similar objects to one with a clear representative.

4.4.2 Design evolution

In the following section I will show the way *flux* developed in the course of time. I will describe three different stages, from the initial design via the first actual one to the second and final one, that was mostly shaped by the experiences during implementation and the results of the focus group. The differences between the second and third one are, with one exception, only drops of features, so I will present the second design in detail and depict changes to the implemented one later.

First design In my initial design, all streams were placed vertically on the screen, with two users sitting face to face at the sides of the tabletop, each having a personal workspace on their side of the table (4.2). Streams could be made visible by toggle-buttons on one side of the display, to let the users narrow down the current focus. All photos were sorted along a time line and multiple photos were combined by similarity with one representative if the available space did not suffice. Single sections of the time line could be scaled independently for both users with a two-fingered gesture - streams on each side of the screen would be arranged according to the corresponding time line (the vertical position of a stream could be changed by dragging it with one finger). The vertical axis would allow several different scales (e.g., brightness, number of shown persons) with the horizontal one being fixed to time as the most popular scale ([43]).

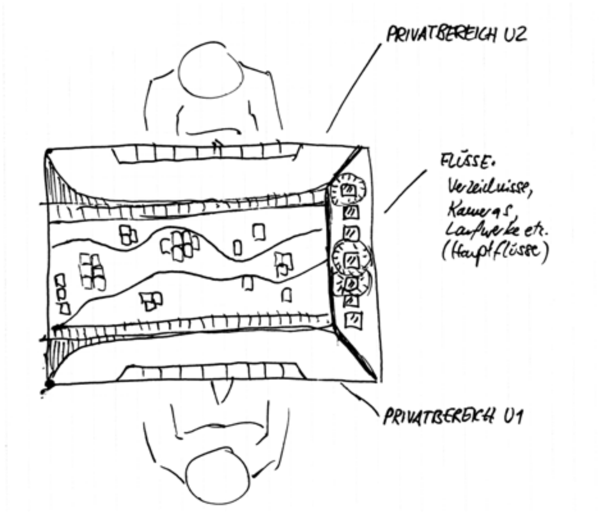


Figure 4.2: Initial design of flux - Overview

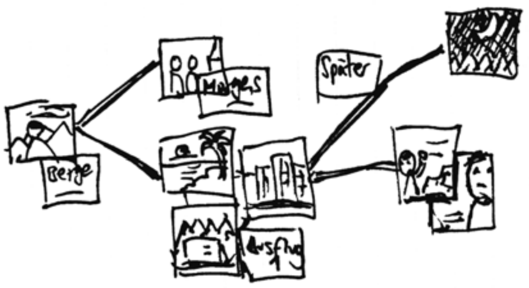


Figure 4.3: Initial design of flux - Photoarcs

Substreams were created by circling single photos, which could be moved afterwards from one stream to the next by simple drag and drop. Those substreams could also be moved to the personal workspace and handled in detail. Changes made in one workspace would be propagated to all instances of a photo.
As an additional way of organizing photos and especially conserving all the little details that would be lost otherwise, photos on the lowest level on the stream hierarchy could be connected and furnished with additional comments and titles (4.3), similar to PhotoArcs ([4]). Those details would only become visible if the corresponding stream was enlarged enough (in a form of semantic zoom).
Collaboration was supported by letting the users combine their workspaces to a large one in the middle of the screen, placing a small copy of the stream-view on their sides (4.4).
Both workspaces were scalable, so the need for space could be dynamically adapted, going as far as switching to a pure one-user operational mode by shrinking the second workspace to its minimal size.

The way to the refined version This first approach already had many of the later features and all basic ideas in it and seemed promising. Yet, several points appeared unfavourable:

- The fixed screen layout with its buttons on one side and the opposing workspaces forces the users into a certain way of usage and I wished for more flexibility. I especially wanted to support the spontaneous starting and stopping of interaction episodes and letting the users

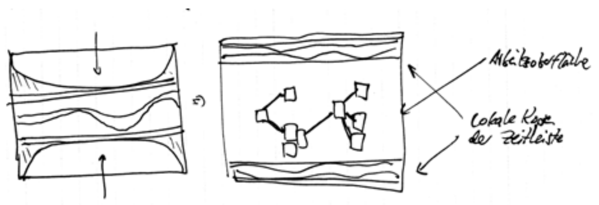


Figure 4.4: Initial design of flux - Combining the two personal workspaces

choose where they sit.
Additionally, all the fixed elements (especially the stream-buttons) would take up a lot of screen space even if they were not needed too often.

- Writing on the tabletop display is tiresome. So an elaborate system to add written notes to photos and events would probably have been hardly used in practice. The problem of "forgetting details of people and events depicted in old photos" ([18]) still remains pressing, but forcing the users to write long messages on a touchscreen is no elegant solution.

- Photoarcs do not have necessarily have to be built by hand - an automated system could also treat one stream of photos by clustering them based on their date of capture. Different, complex branches are uncommon and would at most appear between photos from different sources (which would not have been supported by the system anyway). This plus the above tediousness of text input made the idea seem less intriguing.

I started to think about different aspects of my sketch and it seemed to do much things right, but have an overall patchy nature: Some things would probably work but did not fit in really well. In aiming at a unified design, I first defined some general premises, that *flux* should follow:

- **Flexible Interface**
  Almost all parts of the interface can be freely manipulated and arranged with simple gestures

- **Adaptability**
  The system provides elements that the user can arrange and use to create a very personal organisation

- **Use of screen estate**
  The system aims for a maximum utilisation of screen estate, filling it with photo thumbnails that are as large as possible

- **Usability**
  Ease of use, consistency (for gestures etc.) on all levels, global undo-gesture etc.

- **Collaboration**
  Flux tries to guess the user's intentions and optimize the visualization based on it (adjusting the similarity-measure, enlarging parts of the interface, etc.)

- **Simple rules, complex actions**
  The interplay of several simple rules plus the flexibility of the system leads to the emergence of complex actions

- **Co-located use**
  More than one user can interact with the system at one time. The number of users is flexible and its fluent change is supported by the interface.

Having these concepts in mind, I refined the initial design and produced the first actual version.

Root	
Background	Workspace
Cluster	
Pile	
Photo	

Table 4.1: Object hierarchy in flux

4.5 Main concepts, screen elements and interaction

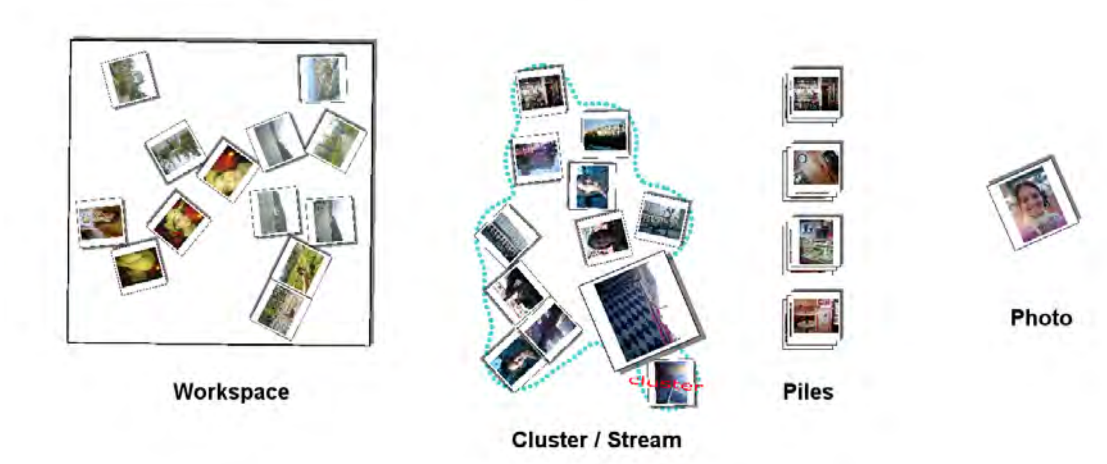


Figure 4.5: Flux - Interface elements

The *flux* interface consists of four main elements:

- **Photos**: Manipulable thumbnails of JPEG-files
- **Piles**: System-generated, instable combinations of more than one photo
- **Clusters / Streams**: User-generated, stable combinations of more than one photo or pile
- **Background / Workspaces / Views**: Views on the photo/cluster collection

Each visible object belongs to a certain category (see table 4.1). Higher objects have to contain lower objects (no higher object can exist without containing at least one lower object).

Intermediate steps can be skipped (e.g., a photo that is actually contained in a pile in a cluster can be shown in a new workspace ignoring the intermediary objects). The lowest objects on the hierarchy, photos, can be contained within each other object, i.e., photos don't have to belong to a cluster; they can be freely positioned within the background. Objects belong to two categories: They are either created and manipulated by the user (workspaces, clusters, photos) or dynamically by the system (piles).

At the top of the object chain lies at any time either the background or a workspace. Both of them are just views onto an underlying abstract model, that contains the following objects:

- **Abstract photos**: Data structures representing a JPEG-file and its metadata
- **Abstract clusters / streams**: User-generated, stable combinations of more than one abstract photo

All visual objects except piles and views are based on abstract objects. The latter are unstable, i.e., they are not constructed by the user and carry no additional information so they do not have to be represented on an abstract level. Piles are built by the system if needed, so a photo might belong to a pile on the one view while it is unpiled on the other, which is why they have no abstract version either.
Every user interaction with the system can thereby classified into one of two categories: It is either **local**, meaning its effect is restricted to the current view and does not influence the model, or **global**, where the model is changed and these changes are propagated to all visual representations to keep them consistent.
As an example, changing the scale of a photo on one view is a local action and has no impact on the size of other photos (because the display size is a purely visual attribute and is not saved in the model), while combining three photos to a cluster is a global one, modifying the model as well as representations of the photos on other views.
Because of the possibly destructive nature of global actions I initially thought about making a virtue of necessity and using only two-finger gestures for them. The rationale was that the SmartBoard (the target hardware of the system (see 5.1)) could interpret at most two input points, which meant that no user could perform a global action without the consent of the other, who had to lift her finger and not interfere. Thus, it would be clear that every global change was an agreement between all users and no user could change e.g., a cluster while the other was still working on it.
Although it is a nice idea, it has its downsides: First of all, the other user's consent is not in any case necessary. If she is currently not working on something, maybe doing things away from the tabletop and not paying attention to the happenings on screen, global changes could slip by unnoticed and unconfirmed.
Second, limiting the interface to a certain hardware restricts its spread and outlook. Novel tabletop concepts like Microsoft's Surface ([77]) or existing ones like DiamondTouch ([16]) already support more than two concurrent input points and it is clear that this direction will be pursued by future hardware as well, which leads the proposed dichotomy ad absurdum.
Third, one local action, namely the scaling of a photo, is much more convenient and natural to do with two fingers (see below) than with one (the one-fingered alternative would have been using a button like in Photohelix or touching the border like scaling a window in desktop applications). Additionally, it allows holding an object with one finger while repositioning the other (more or less like lifting a mouse from the table board), which is handy in some situations where releasing the object would cause it to snap back (see below).

4.5.1 Overall interaction concepts

To reduce the user's need to learn and to make the interface more accessible, most of the interaction techniques are identical for all objects. That means that for example translating a photo is performed by the same gesture as translating a pile or a workspace.

Transforming an object Most interface objects can be transformed (i.e., translated, rotated and scaled). Transformation is achieved by using two different techniques, that combine either translation and rotation (**RNT**) or rotation and scaling (**RNS**).

- **Rotate'N Translate** (**RNT**) was proposed by Kruger et al. ([32]) and tries to mimic up to a certain point the interaction with real-world objects. When touching an object near one of its corners and dragging it in a constant direction, the touched point always stays beneath

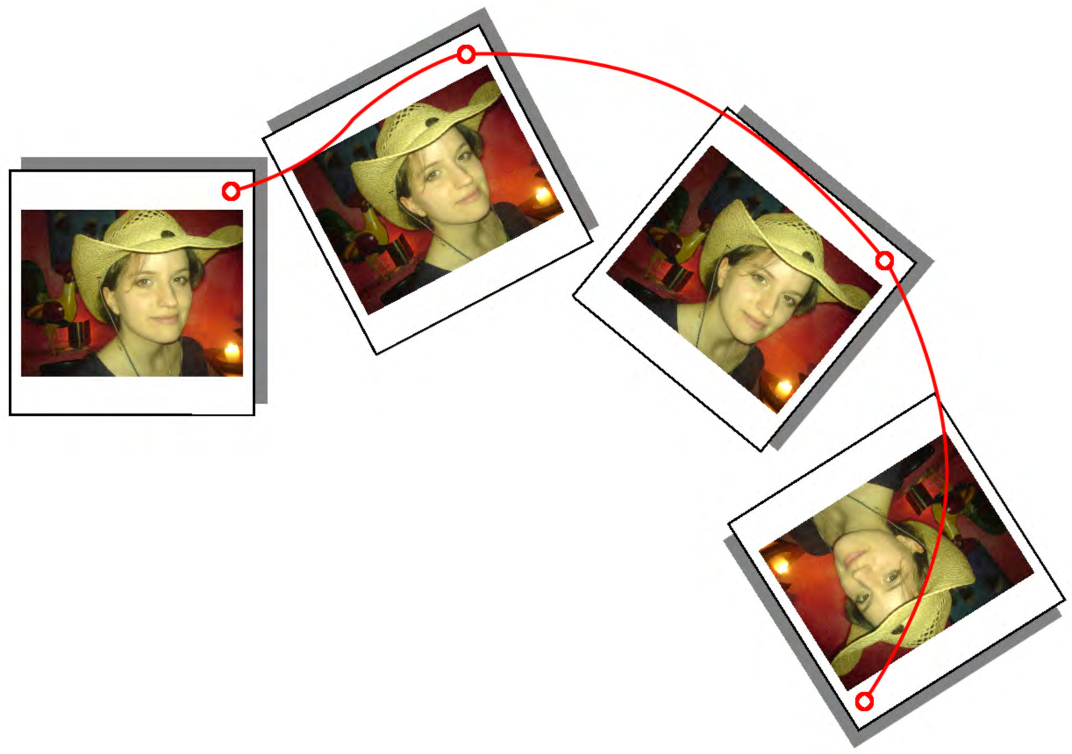


Figure 4.6: Rotate'N Translate

the cursor but the object additionally rotates as if it was influenced by some kind of friction or opposing current (see figure 4.6). This rotation is based on simplified "*pseudo-physics* developed and adjusted for interaction ease" ([32]).
A pure translation without rotating the object is possible as well: If the user touches the object near its center, the image is treated in the same way but not rotated (see figure 4.7). The region of the object where only translation occurs can be freely defined (Kruger names a value of 20% as optimal).

- The second transformation technique is called **Rotate and Scale** (**RNS**) and combines rotation and scaling. It is similar to *rosize*, used by Apted et al ([5]). The interaction mode switches from RNT to RNS as soon as the object is touched at a second position. In RNS, the object is scaled and rotated to keep the two input sources always at the same (relative) positions, i.e. if one finger lies at the center and the other near the upper-left corner, they stay there no matter how they are moved because the object is transformed accordingly (see figure 4.8).
  Scaling is unlimited but restricted by the size of region where input points are recognized (normally, the corners of the tabletop). But this restriction can be circumvented by lifting up one finger, repositioning it and continuing with RNS.

The translation between the two interaction techniques is fluent: Lifting one finger or placing it on the table causes the switch, allowing a quick and uninterrupted interaction.

Advanced interaction Transforming an object is the most basic activity in *flux* and probably used the most, which is why it is directly triggered if the user touches an object and moves the finger. Still, to provide more than just those general actions the interface has to support some kind of input mode change. There exist several approaches to this problem in pen-based interfaces (for an overview see [33]). Because of being the only method available on the hardware and its relative

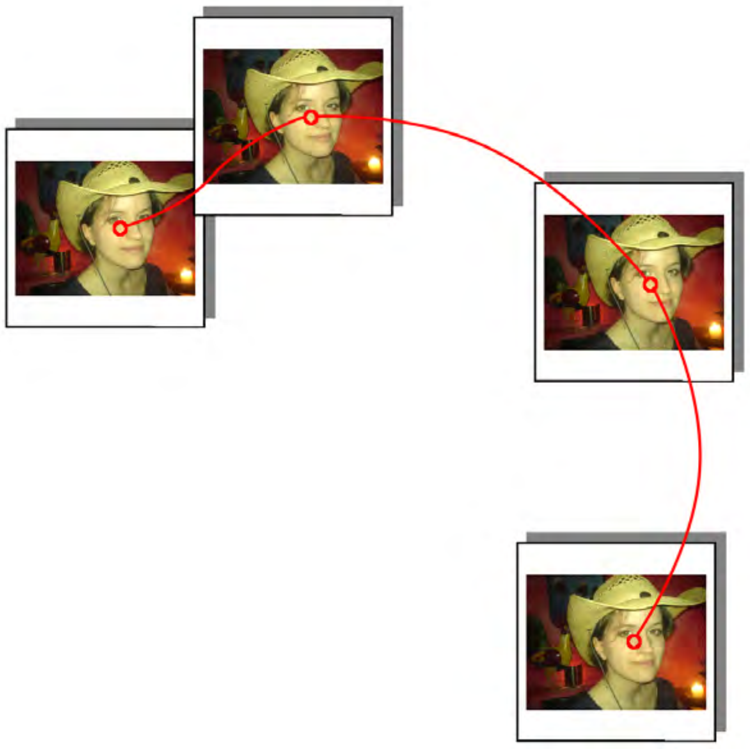


Figure 4.7: Translate Only

straightforwardness I opted for the so-called "Press and Hold" or "Press and Wait" ([30]) method, where a mode change is caused if the user touches a section of the interface and holds still for a certain period of time. This change is symbolized by a switching of the cursor's color.
The user is now able to perform more complex actions, namely creating new clusters by using the Circle-gesture and creating a new workspace by performing the Lasso-gesture (see figure 4.9).

- **Circle**:
  After the timeout, the user can draw on the table by moving the input source. The resulting polygon is drawn in red and automatically closed, covering a certain section of the screen. New points are added on moving the input and the display updated accordingly. Lifting the input source from the table is interpreted by the system as ending the gesture and the result (forming a new cluster from the enclosed objects) is shown.

- **Lasso**:
  This gesture is taken from Bumptop ([1]) and starts from the Circle-gesture. If the user closes the circle, the polygon's colour switches from red to blue symbolizing another mode change. Moving the input now lets the user draw the "tail" of the lasso. If the input source is lifted, the gesture is finished leading to the creation of a new workspace with its center at the end of the lasso and containing copies of the enclosed objects. If no objects are within the Lasso, an empty workspace is created.

A second level of advanced interaction becomes accessible if the user continues to hold the finger down after the cursor switched from red to blue. After waiting for the same amount of time again an object-specific marking menu ([30]) appears, that allows further manipulation of the affected object (marking menus are available for views and clusters). By moving the input source the user can choose an option from the marking menu and conduct it by lifting the finger (see figure 4.10).

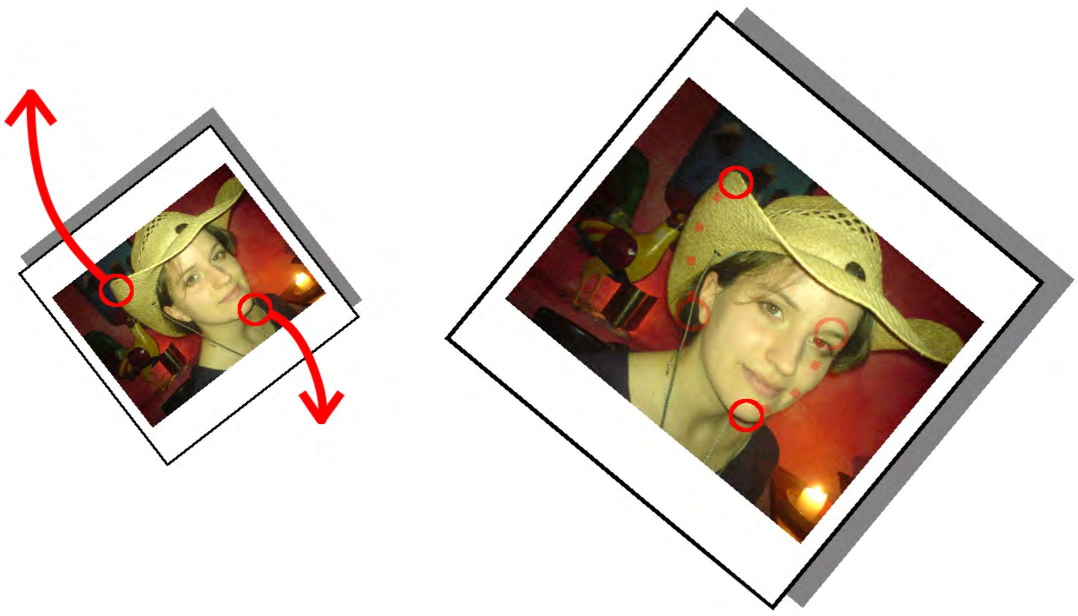


Figure 4.8: Rotate and Scale

History and Undo As mentioned above, global actions can be disruptive - they might include the deletion of a meticulously crafted cluster or even the removal of a JPEG-file from the file system. Therefore, giving the user a way to undo an unintentioned action is highly important. Yet, this undo-feature would not be as commonly used as e.g., moving a photo.
The different interaction techniques are scaled along the time-axis depending on their relative occurrence. The most common interaction is accessible without waiting, while the most uncommon one needs a longer waiting time to become available. After the marking menu (if any) appears, the user has to hold still for a last timeout to access the undo-mode (a reasonable time for one timeout are 500 milliseconds, so the third one takes only one and a half seconds of waiting). The marking menu disappears, a light gray veil covers the interface and the user can undo and redo the last changes by drawing either a counter-clockwise circle for the former or a clockwise one for the latter. One full circle encompasses more than one action and if the user draws more than one circle the speed of undo or redo is adapted to it, meaning that while the first drawn circle might undo ten actions, the third one could do the same to thirty, which lets the users "scroll" faster through whole portions of time. The number of actions that are affected by one circle depends on their general impact: Global actions weigh more than local ones and need more drawing time to be undone (e.g., one circle could undo two global actions or ten local ones). The undoing and redoing is performed gradually along the time axis, with currently relevant objects fading in (and thereby standing visually above the veil and expanding into the user's center of attention) and fading out again if they are no longer important. Abrupt actions like the deletion of an object are shown gradually as well to allow the user the estimation of its impact (ending the undo-mode in such an intermediate state leaves the system at the last fully active position in the history).
All actions are saved and can be undone - ranging from local, unimportant ones like changing a photo's position to global ones like opening or closing a workspace or deleting an object.
The undo-mode is restricted to the view it was started on, which means that on one workspace only the actions that were performed on it (up to its creation) can be undone. Only the background holds a truly global undo-mode that lets the users retrieve the whole interface history and manip-

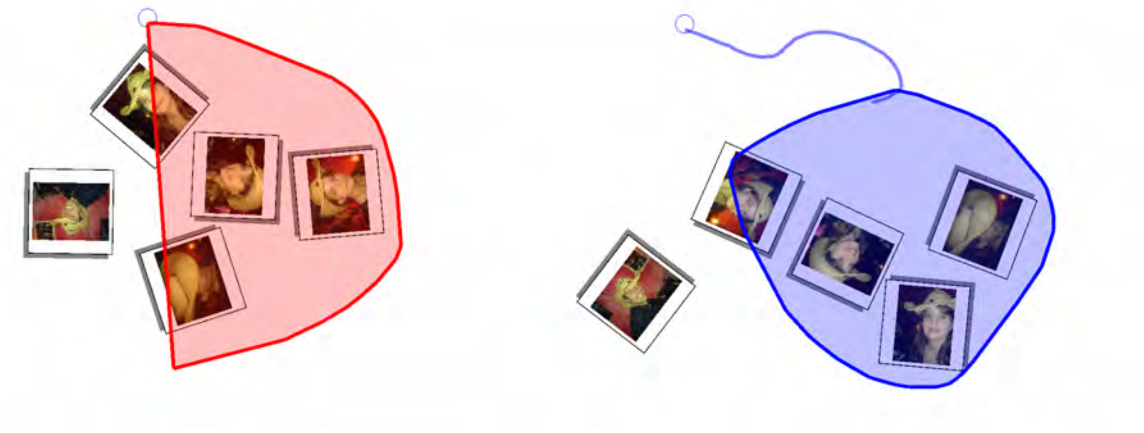


Figure 4.9: Circle and Lasso gestures

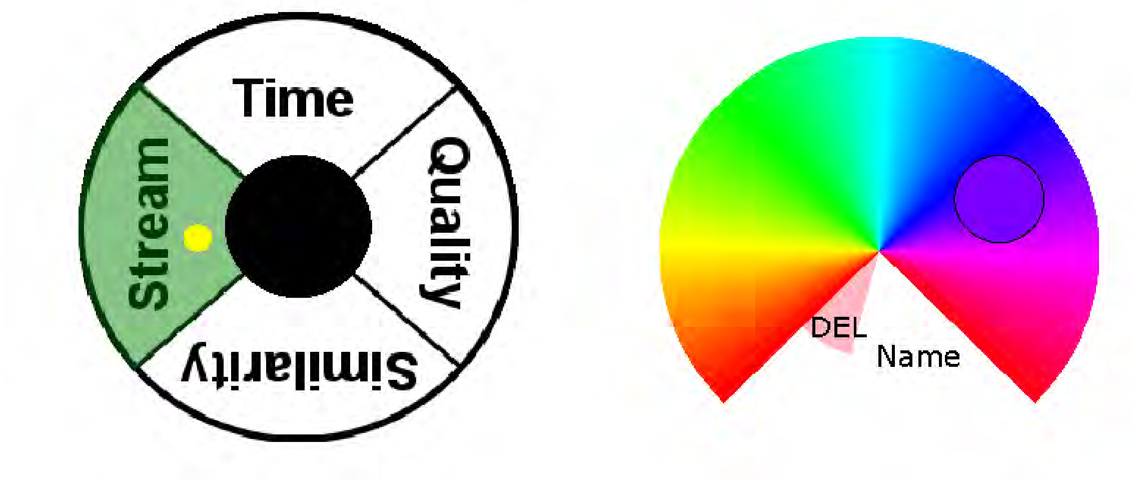


Figure 4.10: Marking menus (View and Cluster)

ulate it.
After lifting the input source again, the undo-mode is stopped and the system reset to the chosen position in history.

4.5.2 Types of objects and specific interaction

The techniques described above are relevant for the application as a whole. In the following, I will describe each interface element and its assigned interaction options in more detail.

Background, Workspace, View : The background and a workspace are two concrete variations of a view on the underlying model of photos and clusters so the term "view" is used as a synonym for the other two. The background is a special case of a workspace, because it cannot be closed or translated and shows the whole photo collection at any given time. Additional workspaces can be created by the user with the Lasso-gesture (see above) and are shown on a layer above the background. A workspace contains only a certain section of the collection (but can of course contain the whole as well) and can be manipulated by the whole range of transforming gestures. It is closed by scaling it to a small size with two fingers (it turns red in the process to signalize this to the user).

Objects within a view can be freely positioned, rotated and scaled but they behave like physical objects: If one object bumps into another, the latter moves accordingly. This physical behaviour not only leads to a more natural feel in interacting with, for example, photos (see above) but also has the side-effect that all objects are visible at any time and no two objects can slide below one another. Plus, users can quickly free a screen region by wiping around it with one object and pushing all other objects away.
This physical behaviour also extends to the workspaces (but not the background), which interact among each other accordingly, preventing them from slipping below each other as well (downsides are of course that users' workspaces might collide unintentionally breaking each other's concentration on the task at hand and that there is only a certain amount of screen real estate available because workspaces cannot overlap one another).
The concept of workspaces tries to fulfill the options proposed for *movable work surfaces* by Pinelle et al. ([36]).
A great number of objects might be on one view at the same time, so the system provides a way to arrange them according to a scale. By waiting for two time-outs the marking menu appears (see figure 4.10 on the left side for the current version with four orders) and lets the user sort all contained objects according to either time, similarity or quality. The objects lose their physical nature and are rearranged and in the process combined to piles. This piling becomes necessary if the screen real estate does not suffice to show all objects in satisfactory sizes. The criteria for piling depend on the chosen overall order as does the impact of the cluster structure on photos' positions.
The different available orders are:

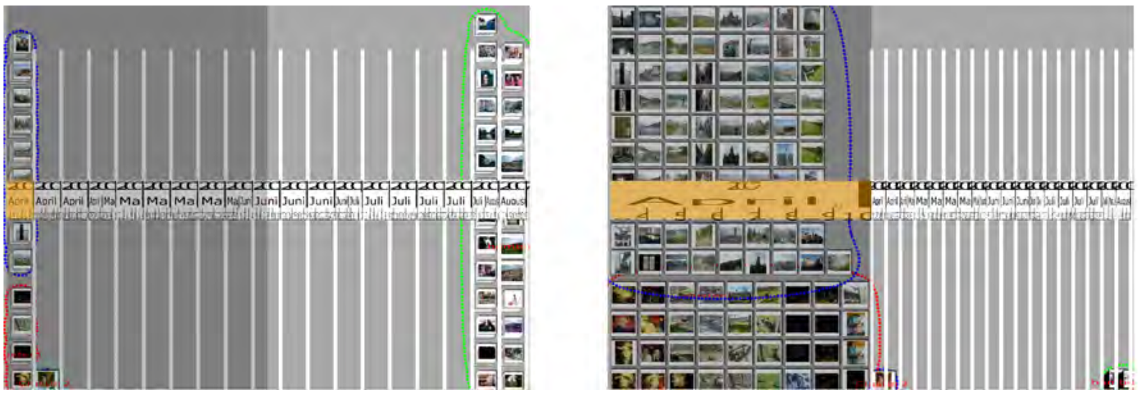


Figure 4.11: Focusing on a certain period of time

- **Time**: The view is separated into a number of vertical columns that represent one fraction of time. All photos that were taken during this period are arranged within it. Their position additionally depends on their affiliation with a cluster, because all main clusters are placed above one another. If there is not enough space available in the column, photos are combined based on their vicinity in time.
  One specialty of the Time-arrangement is that the user can adjust the width of the columns: By touching one section of the calendar strip that appears in the middle of the screen, she can size the corresponding column's space with either one or two fingers (one finger lets her change the position of either the left or the right border, while she can manage both borders' positions with two) (see figure 4.11). The other columns are shrunk accordingly and all photos are rearranged and either unpiled or piled depending on their new size.

- **Similarity**: In the similarity arrangement all main clusters place themselves on screen depending on their representational value (i.e., the more representative the contained photos of one cluster are, the nearer it lies to the screen's center). Additionally, they try to lie near other clusters that are similar to them. This grouping of similar objects is repeated within them: Contained subclusters act the same and contained photos are not only piled based on their similarity-values (most similar ones first), but also move near other, similar photos or piles and even in the direction of other main clusters, whose contents look like them.

- **Quality**: All photos are, independently from their clusters, placed along a horizontal axis depending on their relative quality (estimated by the system). The more qualitative one photo is the further it lies to the left. Additionally, all photos are marked by an either green or red border (the colour's opacity depends on the value the system attributed the photo quality-wise).

Photos : Photos are the main objects in *flux*. They are the representatives of JPEG-files from the user's file system. Each photo has attributive metadata, like its original resolution, the date it was taken, its quality value (determined by the application) etc. This information is centrally held within its model version. The possibly multiple visible versions show the photo and can be scaled, translated and rotated. They are squarish with a white border, that allows the users to quickly distinguish two photos and estimate the number of visible photos at first glance. Additionally, it reduces the visual clutter by giving all photos a uniform appearance.
Photos can be combined to clusters with the Circle-, or copied onto a new workspace with the Lasso-gesture. It is possible as well to drag a photo onto a workspace to create a copy of it there, or drag it from a workspace onto the background to remove it (no additional copy is created on the background).

Piles : Piles are not created by the user and have no counterpart in the model: They are fragile objects that can only be created by the system and are used to reduce the overall visual information. Piles are depicted as pseudo-three-dimensional packs of pages that allow the user to estimate how many objects are contained.
The user can interact with piles in the following ways: She can translate and rotate them with the standard RNT technique, but they are not scalable: If a pile is touched with two fingers one finger marks the position of the bottom end of the pile while the other marks the top end. All photos are placed accordingly between the two. The pile can be "closed" again by bringing the two fingers together. A pile can be dissolved by waiting two timeouts. In this case, all contained objects are shrunk so they take up roughly the size of the pile in total and are placed next to each other. Each object is then treated again like an unpiled one. Once a pile is dissolved, it cannot be restored again - only by re-arranging its view.
A copy of a pile can be brought to another workspace by either drawing the Lasso-gesture above it or dragging one of its ends onto the view.

Clusters : Clusters can be created by the user and are used to structure the collection. The top-level clusters are called streams and depict the source of the contained photos. Each cluster has a color and a name that the user can choose. Clusters can contain all other objects, even clusters, which means that they can be nested and support hierarchical structures (see above), but every object can only belong to one cluster.
The borders of a clusters are shown in its colour and allow the visual attribution of objects to the cluster. A cluster can be created by using the Circle-gesture around other objects. Its parent cluster is then defined by analyzing which cluster in the cluster-hierarchy is the lowest one that contains all affected objects, which then becomes the container for the newly created one. If no common container can be found (because the objects belong to different top-level clusters), the new cluster becomes a stream.
Clusters can be translated, rotated and scaled just like other objects. When their size is changed and

more room is available contained piles are dissolved, if there is less room, objects are combined to piles.
More advanced interaction with clusters is performed by using their marking menu (see figure 4.10 on the right): Here, the user can choose a colour for the cluster, open up a text field, where she can write a new name for it or delete it. By deleting a cluster, all contained objects become parts of the parent container (either another cluster or a view).
A whole cluster can be moved to another workspace by dragging it on (using the Lasso-gesture only moves the affected objects into the workspace).

4.5.3 Additional features

Three more types of objects were planned in my design, but the short available time forced me to drop them in the implementation. They should have enabled the user to interact more freely with her collection and adapt *flux* to her needs. In the following, I present these dropped features.

Lines Lines were planned as a means to add another, non-hierarchical layer of organization to the collection to address the strictness of hierarchical taxonomies. The idea was inspired by network visualizations ([22], [51]) that use lines to connect objects. Lines in *flux* would have connected arbitrary objects and would have had a user chosen colour and thickness. The user would have been free to use them as she would have seen fit. Exemplary uses would have been adding a second layer of organization vertical to the existing, cluster-based one (e.g., having all clusters representing a temporal order, while lines of one colour connected all subclusters of a certain topic), adding the PhotoArcs from the first design by hand or connecting a photo to a Textblock (see below) with annotations.
To reduce the visual clutter within the interface, lines would not have been visible all the time. They would have appeared if the containing object (i.e., cluster or view) would have provided enough space or if the user would have touched one of the two connected objects. In the second case, more and more neighbouring lines (i.e., lines that are connected to the affected objects) would have gradually appeared.

Textblocks Textblocks were freeform squarish objects containing text the user could enter with the finger. Their use was again not predetermined and they could have been used for example to annotate a photo with additional information (probably connecting the Textblock to the photo with a line), add a textual background to a cluster or create tags. Especially the creation of tags would have been supported by another order for workspaces ("Text"), that would have worked akin to the similarity order but would have used the titles of clusters and the text from Textblocks to position the screen elements. The nearer a Textblock lied to a photo, the more relevant it would have been seen for it and Textblocks with identical texts would have been treated as the same. Thus, the user could have created her own photo tag cloud (a type of weighted list most prominently used by certain web pages like flickr [90] or del.icio.us [67]) by using Textblocks as tags for photos.
Textblocks would again only have become visible if there was enough space available or the user would have interacted with an object in their vicinity (with gradually more and more Textblocks appearing if the interaction lasts for some time).

Boxes Inspired by the advantages of digital photographs, I created the concept of Boxes for *flux*. A Box was supposed to be a container for visual information, just like a photo, but not derived from an existing file on the file system but newly created within the application. The user should have been able to create a rectangular object and paste other objects or parts of them into it. A Box would have provided an edit-mode, where it would have been placed behind the other objects and could have added the visual information of overlapping parts to itself, just like a regional screen-shot. Additionally, users would have been allowed to draw on a Box with different colours and

tools.
Usages for Boxes were again free to the users' imagination, but imaginable ones were creating collages and panoramas from more than one photo (Boxes could have had a resolution that dynamically adapts to the contained objects), creating screenshots from parts of the application, adding personal notes to clusters or for drawing on existing photos.
The user's work would have been saved as a regular JPEG-file in the file system to preserve it and make it available outside of *flux*. Boxes would have been treated just like regular photos, which means they could also have been put into piles, arranged according to their similarity to other photos, etc. Maybe the options for manipulation of a Box would have been taken over for photos as well in the end, to spare the user the shortcut of creating a new Box and pasting the photo into it to make it manipulable.
Boxes would have been visible the whole time (at least if they did not disappear in a pile).

4.6 Focus Group

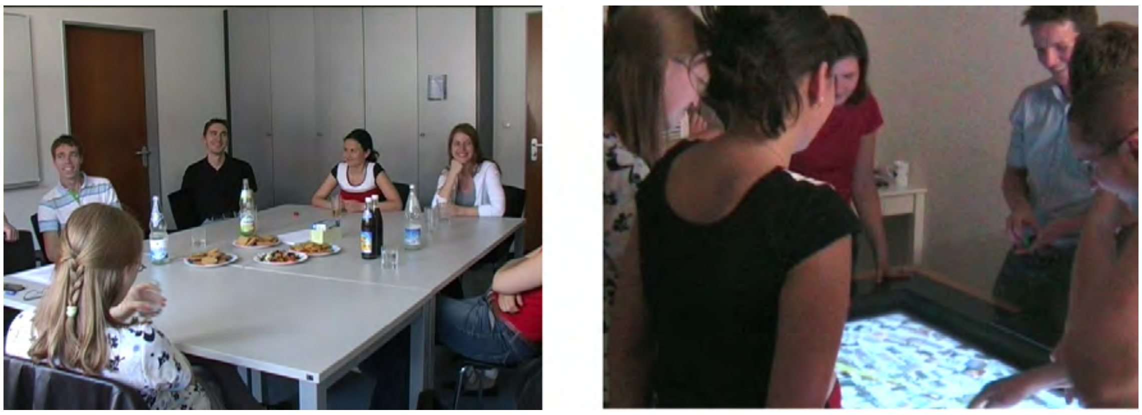


Figure 4.12: Impressions from the focus group

On August 2nd 2007 I performed a focus group session (see figure 4.12). My aim was to gather whether the thoughts I had about Informed Browsing and *flux* were valid or if maybe there existed some general flaw that I overlooked. Another reason was that I am not part of the target group, because I was not making enough photos, so I was interested what people who worked with a larger number of photos would think about it.
The focus group had 3 male and 5 female participants. Everyone of them had expert background experience with computers, with 6 of them being students of Media Informatics, 1 PhD-student of Business Informatics and a master in Computer Science and 1 with a diploma in Media Informatics. I invited only computer-savvy people to get more concrete suggestions and criticism and rely on their existing knowledge. People with a different background might become swamped by the new impressions and direct their criticism (if any) not at the given system but the hardware or the whole setup.
Each participant additionally had a background in photography. Their experience ranged from mainly amateur photography with a digital compact camera to several years as a professional photographer.

Structure The focus group lasted from 5PM to 7:30PM and took place in the Amalienstraße 17 in Munich in meeting room 507 and partly the Instrumented Room in the basement. The whole event was filmed with two cameras and had roughly this structure (for my full script see Appendix A):

1. Introduction (of project and the participants).

Ways to organize photos	
Analogue	**Digital**
Shoebox (maybe still in the folder from the photographic laboratory)	Unsorted photo directory
Photo Album	Subdirectories following a naming convention (date, topic, location)
Scan photos to digitalize	Verbal annotations (possible with newer digital cameras) and automatic tag creation via speech recognition
Organize negatives	Tagging
	Software: packed-in camera software, ACDSee, Adobe Bridge, iPhoto, Picasa
	Webportals: flickr, Picasa Webalbum
	Let digital photos develop or print them out

Table 4.2: Focus group - Organizing brainstorming

2. Photos: How are photos (analogue and digital ones) organized, how and why are they accessed again.
3. Photowork: Presentation of the general photo workflow by Kirk et al. ([29]).
4. Informed Browsing: Presentation of the concept, what metadata could be used.
5. Flux: Short introduction, demo, then participants' comments.

4.6.1 Outcome

Photos After a short introduction of myself and the project, a short presentation of PhotoHelix ([6]), which was deemed as more or less the predecessor in spirit, and two rounds where the participants introduced themselves and their connection to photography, we started.
The first block of questions addressed the way the participants worked with photos. We developed different ways to organize analogue and digital photos and compiled them on a whiteboard (see table 4.2).

For the topic of tagging, the participants understood the benefits but did not perform it for their whole collection because of the amount of work. But while it was not used on the local collection, online portions of it were tagged (e.g., on photo portals like flickr).
One summed up the whole subject as follows: "For this tagging and stuff: You do that sometime, but you don't keep it up in the long run," and added quietly "although it actually needs be" (Note: All quotes were translated from German by the author).
A popular organization scheme was creating a directory on downloading pictures from the camera and giving it a specific name in the format *date-location-keywords*. Some participants confirmed to use that and one even said that he additionally "[has] directories for topics that are connected by file system links [to the photos]", as a second, topic-based organization. Some of the participants used explicit photo software and here mainly the packed-in one coming with the camera (e.g., ZoomBrowser / ImageBrowser by Canon). One emphasized that users with less computer experience might have a higher tendency to use the provided software, because "they don't know it differently. They don't know that they have their photos in the [Windows] Explorer in 'My Photos' or in some other photo folder. They don't know that the Explorer exists!".
Web portals were used by some, but sparingly and were seen as no alternative for a local collection. They were regarded as an opportunity to "publish" the best (and probably retouched)

Available metadata for Informed Browsing:
Similarity (colour histograms)
Object recognition (special case: Face recognition)
Portrait / Landscape format
Quality
Usage history (invocation frequency, common actions)
Interior, exterior shots
Exif-information: Date, camera status, etc.
Location via GPS
Supervised learning, feedback from the user

Table 4.3: Focus group - Metadata Brainstorming

photos and make them available for a world-wide audience. Privacy concerns were shortly touched in this context, with one user complaining about people uploading private party snapshots to unrestricted websites like flickr or lokalisten ([76]), while another praised the finely adjustable privacy settings of facebook ([68]), that allows the user to make photos available to a restricted group of people only.

The usage of the photos was not only restricted to sharing with friends or family, but also spanned the online publishing or browsing the collection on one's own.
Failed photos were kept by some in a special directory for a possible future use, or if the photo was the only version of some motif one had. One said that she deleted photos "only if one cannot use them at all, which is true for a really very, very small section [of all photos]". Selecting happened partly directly on the camera for photos that were obviously botched (all black, eyes closed) or then on the PC after downloading from the camera.

I gave two concrete tasks to make the participants evaluate their methods of organisation. While the first one (find a photo from a party that was exactly one year ago) seemed easy for them with their date-based directory structures, the second one (find a good photo of a parent or the partner) was harder to do without having a concrete photo (and its date) in mind. Without this help it seemed difficult to do (one participant added that this will be the case for every photo "when we're eighty"). Tagging was named as a solution, but it was, as already mentioned, seen as tedious and not available on the file system-level (only marginally by putting tags in the directory name) but application-dependent (which many did not use).

Photowork and Informed Browsing Closing with this unsatisfying situation (and a first glimpse of the problem), I presented the concept of photowork and asked for the participants' opinions, who agreed to its validity.
I continued by explaining the ideas behind Informed Browsing and its emphasis on metadata. We then compiled sources of metadata on the whiteboard (see table 4.3).

Afterwards I explained the concepts of overview-at-all-times, details-on-demand and temporary structures, with overall acceptance.

4.6.2 Focus Group Prototype

As a next step, I introduced *flux*, explaining the original concept of streams, then piling to reduce the user's mental load, different orders and the advantages of a tabletop display (which resulted in a short discussion about the general merits compared to a setup where all people sit on a couch and

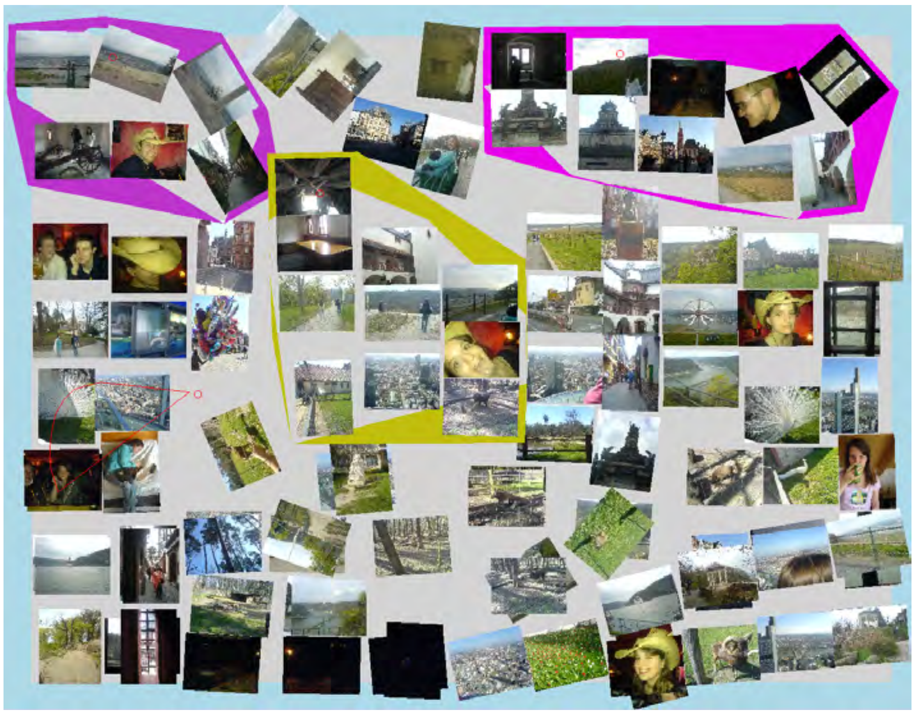


Figure 4.13: The prototype from the focus group

watch the photos on a TV set). After describing the interaction techniques for photos (moving, scaling, creating clusters) we went to the Instrumented Room and played around with the system. The prototype I showed during the focus group (see figure 4.13) had all basic features described above except additional workspaces, similarity and quality order and the possibility to change a cluster's colour or name. All photos could be sorted by time or placed randomly and were arranged in a rectangular grid on the background. If the available space did not suffice, the system built piles from either temporally near or similar photos. The two-finger open-and-close gesture of piles was not available, but all photos in one pile were connected by an elastic joint, which meant that by dragging one of the photos all the other followed in a short distance, allowing for an alike effect. Piles were dissolved by touching them for half a second. Marking menus were not implemented as well, so changing the order of the background was performed pressing a key on the keyboard. The physics engine on the other hand was fully operational and moved all objects in a realistic fashion (one participant commented "It looks like water, like they are lying in the water.").

4.6.3 Feedback from the participants

The first reaction of the participants was very positive, they especially liked the physical behaviour of the photos and the two-finger scaling (one said "The joy of use is something you shouldn't underestimate"). Collisions when two users tried to interact with the system at the same time were accepted and managed by ad-hoc social protocols. Participants also asked spontaneously why a certain pile contained only two photos and had a third, similar one, not in it (because the number and height of piles was connected to the available screen space and the number of objects (photos and piles) visible at the same time) and if piles would be created if the screen space shrunk, for example because a photo was enlarged. Also, piles were difficult to distinguish from single photos, since they only contained two photos, because of the number of total photos and the screen real estate.
After showing the prototype, we returned to the meeting room and discussed the system in detail.

General criticism The first point was the piling of photos. Two participants opted for a more radical grouping, that combined all similar photos and not necessarily used the whole available screen space ("If [the system] is able to build a good pile from some pictures, it should group them and even when there's more space there, which, I believe, is always a good thing, because you actually want to always enlarge photos and you can only do that now if you push other pictures away"). Dissolving a pile was not seen as optimal, because "you send all this nice automation to hell"; so piles should, once opened, be combined again after the user stopped interacting with the contained photos or the system should at least group photos if there is not enough space available. It was again mentioned that it was seen as a serious flaw if some obviously similar photos were not grouped into a pile.
Another point was the inability to gather from the looks of a pile the number of contained photos (a proposal was putting their number as an overlay on the pile). One said in conjunction with the scaling, that it should not be possible to scale a photo to more than its actual resolution or give each photo a fixed maximum size and allow unrestricted resizing of parts of it only within this border, probably because more than one participant had difficulties to use the two-finger scaling without enlarging the object to the whole screen size.
One participant brought up the topic of 3D visualization which resulted in a discussion about its downsides (which perspective to use if users are gathered around the table, missing context information and generally confusing with the ever-changing piles) and advantages (more space, pushing photos to the side). She continued by proposing "planets" at different coordinates, that hosted photos and could be reached by navigating the 3D-space, which sounded interesting but was a too radical break from the actual design.
The importance of an additional organization scheme (like clusters) was summed up by a participant who said: "If I have everything on one table, every photo that I took at some moment then I won't look through at some point. So, I have to add some kind of organization in any case, because else you have a thousand photos combined to some similarity piles and to find something there I guess I wouldn't manage."
Some were skeptical as well whether the system would be of any use when working with more than a thousand photos, leading to the question if the main aim was organizing photos or supporting multiple users.

How to change a view's order I asked what possible ways the participants saw for changing the order of the photos (which was done using a keypress in the prototype). Suggestions were:

- Buttons for every user or a menu bar at the side of the table (similar to the task bar in Windows)
- Marking menus that appear on touch
- Distinguishable pens or distinguishing between finger and pen (not possible with the hardware)
- Hide functions on the back of the photos and make them turnable
- Gestures (have to be symmetric because of the uncertainty of the user's position)

Would you use it if it was available? This last question should provoke the participants to really think about the disadvantages of the system. One said, that it would be a cool way to show new photos from a vacation to friends, e.g., if the table stood in a café. Another stated that she'd use it to watch and search photos (probably because of the easy access to and interaction with them) but not organize them, because it was too chaotic. She was backed by another participant who said:

"That I always have everything in sight, I find that totally exhausting. I'd just do it the other way round: Make piles first, than drag them apart."

4.6.4 Lessons learned

All in all I was content with the results from the focus group. My basic train of thought was confirmed and I could take with me many valuable suggestions.
I also gathered experience from the event as such and afterwards drew conclusions as to what was not optimal.

- **Duration**: The main criticism was the duration of the event. Most of the participants probably had a long day behind them anyway as they arrived at 5PM and so the focus group took much too long with almost 2 1/2 hours. Additionally, the weather outside was warm and very close, so towards the end some participants were clearly tired and contributed only marginally to the discussion.
  The first part of the topics regarding the ways to organize and work with photos was definitely interesting but could have been cut from one to half an hour. Also, the theoretical part about photowork and Informed Browsing was a bit too extensive as it was too novel a concept anyway to provoke spontaneous discussion.
- **Concrete tasks**: Some questions where asked too vaguely to yield any utilisable results. This was especially the case for the discussion about *flux* right after the demo, where my first question ("What do you think of it?") was exemplary. While this brought some interesting suggestions after some time it also made it harder for the participants to answer (because they were unsure whether their contribution would be fitting) and not to drift off too far from the actual issues.
- **Involve quiet participants**: Something I had anticipated but was difficult to avoid anyway during the stressful moderation was to encourage more quiet participants to speak their minds. As a warm-up I had two rounds at the start of the focus group, where everybody had to introduce him- or herself, but the effect of it lasted only about half the duration of the event. Maybe regularly interspersing tasks for all participants would have been helpful.
- **Be more energetic**: A more personal point that I gathered from talks afterwards and watching the resulting videos was that I should have acted more encouraging and energetic to influence the participants into saying more and feeling more involved in the focus group, maybe battling the influences of the time of day and the weather. One incident that one participant told me after the focus group seems symptomatical: "So you showed us this thing that you built (PhotoHelix) and told us something like 'Yeah, I built this but it's basically crap' and I thought the whole time 'What is he talking about?! This is absolutely great!'"

4.7 Second, refined and final design

After the focus group, I once again started to revise my design to include the received suggestions as well as ideas I gathered from talks with my tutor and other people. I also had to work with the limits of the hardware I had discovered in the meantime.
So, after some contemplation I finalized the one design which would be the eventual shape of *flux* (see figure 4.14).

4.7.1 Conceptual changes

The new background Because of the comment of one participant of the focus group, that the interface was nice for viewing but unfeasible for organizing, I rethought the aspect of the background. For the sake of a consistent interface, the background was originally only insofar different

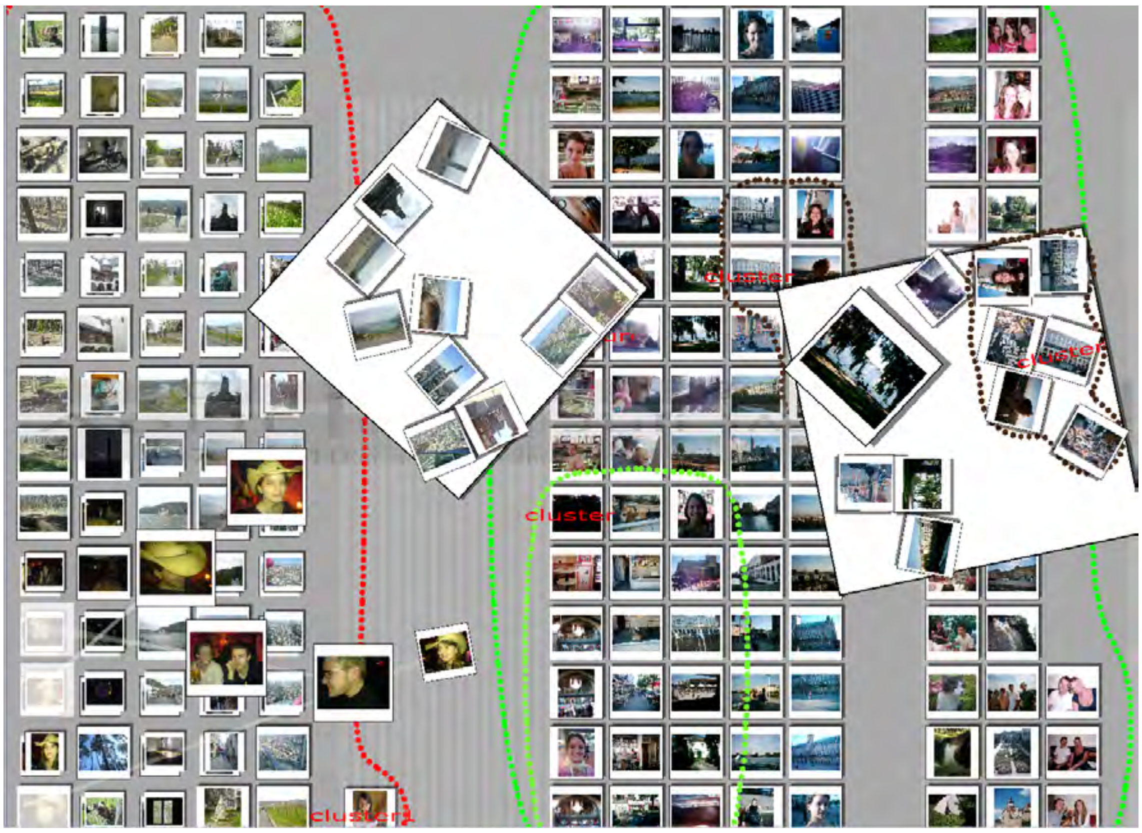


Figure 4.14: Flux - final design

from the other workspaces as it could not be closed and translated (which would have made not much sense). It should have worked for users as a concise starting point into the collection, where they could have gained a rough overview of the collection and delved deeper into it on a separate workspace. But it probably could not provided that in its current state.
The main flaw was the physics engine: While it certainly looked nice and was fun to use, it was not practical for this special use case.
First, the "swimming" photos generated a lot of visual noise - it was hard to concentrate on the collection with the constant colliding and drifting in the corners of the field of view. What the piling of photos did for reducing the user's mental load, the floating objects destroyed again.
Second, users tended to collide with one another while interacting with the system even if they worked in different corners of the screen. This happened because collisions quickly propagated through the whole background even on translating a photo only a small distance. With the scaling of photos it became unbearable.
Third, the physical movement of objects made an order useless because it was always short-lived. Shifting one photo a bit triggered a chain reaction that moved most of the other ones too. So underlying time scales etc. became superfluous, because the photos were quickly drifting apart and the users could not derive any meaningful information from their positions. Furthermore, while the problem might not be as pressing for one user who might remember where she moved her photos and can quickly reset the order, it becomes much worse for multiple users who can leave the table and miss changes and return to find a totally different setup. Letting the users freely translate the objects on the background leads ordering them ad absurdum.
But because I was sure that orders where useful and the background's primary benefit was providing the users with a clear overview, I overhauled its interaction scheme.
In the current design, objects on the background are no longer affected by the physics engine. They are fixed to their positions and cannot be moved, not even by the users. If a user wants to

interact with photos, she can touch them to create enlarged copies (see figure 4.15). Those copies can be manipulated using the standard RNT and RNS gestures but are dissolved again once the user lets them go. If an object has an active copy it gets a whitish overlay to symbolize that.

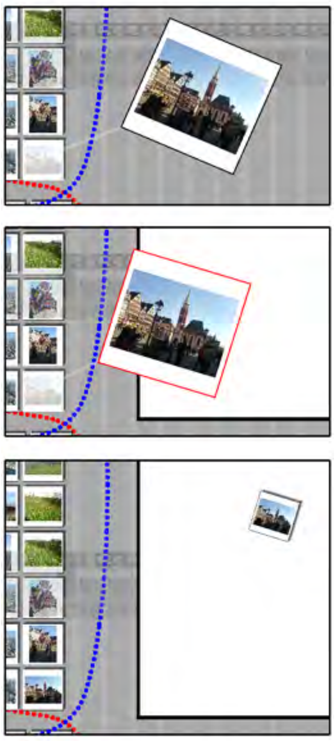


Figure 4.15: Photos: Moving, Drag to new workspace

Piles on the background can no longer be dissolved (because the additional photos would make the interface cluttered without the ability to move them) and interaction is only possible as well via a copy (see figure 4.16). A pile's copy is created by touching the pile, dragging the finger along the screen while the contained photo are fanned out along the path and letting go. After the user lifts her finger from the tabletop the pile's copy is not dissolved but stays active until the original pile (that is also marked with a white overlay) is touched. While the copy is active the single photos can be manipulated just like the unpiled ones.
The rest of the interaction stays the same. Clusters can be compiled with the Circle-, workspaces with the Lasso-gesture and copies of photos or piles can be moved to an open workspace by dragging them above it and letting go.
By fixing the positions of the main screen elements the above problems are compensated and all important actions are still possible, even playing around with physical photos, because the physics engine is still running for all workspaces (and even amongst them).

Streams and the Stream-order Having drifted away from the idea of streams I returned to it after some talks with my tutor, who especially liked about it the notion of the inherent separation of collections by different users (simply as single streams for each of them).
While I might have distanced myself from it conceptually, they were not hard to bring back implementation-wise as top-level clusters (before, all photos were initially unclustered like in the focus group-prototype) stemming from the configuration file (i.e., they have to be defined beforehand).
The reason why I dropped the concept in the first place was that in first tests in the Time-order

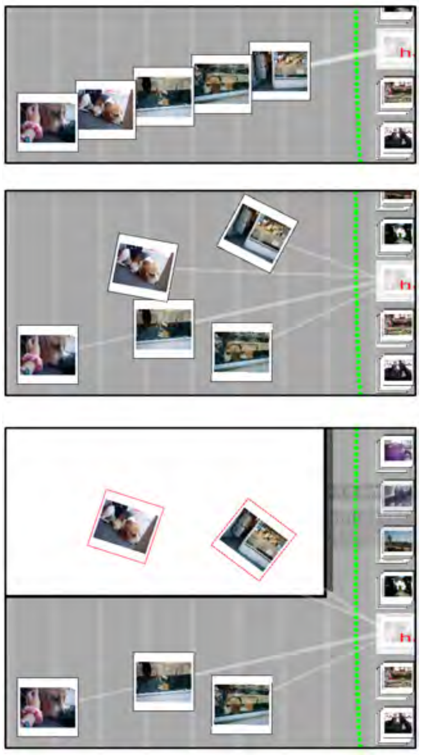


Figure 4.16: Piles: Fan out, manipulate photos, drag to new workspace

(where streams are distinguished) high-level clusters did not become as obviously visible as hoped for. The prerequisites, that every column uses the vertical space as extensively as possible and even empty columns have the same width like the others, led to a situation where the largest part of the screen was filled with whitespace and a few packed columns, which was more or less the complete opposite of my expectations. The problem, of course, was that the linear time line was global, spanning all streams and mostly condensing them in one or two columns.
While it might not have met my expections, it certainly fulfilled the function of letting the users see temporal correlations between different streams and see patterns (see above) in the structure of the photo collection.So I did not want to drop the Time-order in favour of a new one and made the new order (Stream) an addition to the existing ones.
The Stream-order separates the screen vertically and gives every cluster room according to the number of contained photos. Those are aligned along the horizontal axis depending on the date they were taken on, so every cluster fills the whole width of the background. Piling is special, as the system does not pile photos based on the time they were taken alone but their visual similarity as well. The rationale behind it is that different bursts of photos are not necessarily separated by time (e.g., taking a few pictures of one motif, then turning around and continuing with another). The effectivity of such an approach has already been shown by Cooper et al ([13]). Additionally, in the Stream-order there is no minimal photo size, so the user can be sure that every pile contains only similar photos (without a minimal photo size the photos can become very small in certain configurations.
Clusters within the main cluster are singled out and shown above its parent cluster with their own internal time line as well.

4.7.2 Features dropped due to time constraints

With an abundance of planned features and only a limited amount of time it was clear that I had to drop some of them sooner or later. After the focus group when I could roughly estimate the duration of their implementation, I streamlined the design and reduced it to the essential.

Additional user-created interface elements As one of the first parts I dropped the earlier mentioned Lines, Textblocks and Boxes. While they would have certainly enhanced the users' experiences, their implementation would have resulted in a lot of additional work regarding the orders of views and underlying structures. Still, the ideas as such where good and could probably be included in a future version (see 7.2).

History and Undo A powerful and elegant feature, Undo, was just not manageable in the available time. Especially the "winding back"-metaphor plus the corresponding visualization with a focus on the currently active objects (inspired in parts by TimeMill ([50]) would have been not only useful but also enjoyable to watch. With additional features like separation between users (which the current hardware does not support) as an extension to the separation between workspaces a fine tuned history management would have become possible. And with a new gesture to see the changes without altering the current system state another browsing-dimension along the time line would have become available to see not only one's collection but also the way one interacted with it.
This Undo-feature also has to wait for the next version (see 7.2).

Reducing the complexity of the similarity order The last point is no complete drop of a feature but a simplification. The Similarity order as described above proved to be difficult to implement and would probably not have been too useful. Especially the "representativeness" of a cluster seemed to be a very abstract concept, as every cluster tends to contain all kinds of photos, so placing one in the center as the *most* representative one must appear forced in most configurations. Also, positioning similar photos next to each other does only work if they all belong to clearly defined sections of the collection. As soon as ambiguity slips in and a few photos share the same marginal similarity, a placement in two dimensions becomes difficult and must seem random at best. Furthermore, following a chain of similar photos to browse the collection is unlikely (see [42]).
So the similarity order was simplified by separating the screen space vertically and giving each cluster room based on its number of photos (like in the Stream-order). To fill this space most effectively each cluster forms piles of similar photos. This new approach was easier to implement and still as useful as the old one.

4.8 Flux in the context of Informed Browsing

The presented design satisfies the conditions of an Informed Browsing application. It makes use of the general concepts and provides requested features.

Overview at all times is provided with the background: The background shows every object within the collection at any time and is easily reachable even if covered by additional workspaces. Within the Time-arrangement, where the user can enlarge certain sectors of the time line, the collection as a whole is still visible, because no column can completely disappear. This fisheye-like effect was chosen deliberately to provide the user always with a fallback solution, and to keep all objects visible at any time.

Details on demand are on the one hand supported by the different arrangements, that visualize for example the time or similarity value of a photo and on the other by the gestures to transform photos. These provide quick access to their visual features and also allow the examination of a pile, so that even if photos are buried within it, they are still only two gestures away from a full screen view.

Temporary structures that support the users in their work are workspaces and the copies of objects from the background. Workspaces allow for a manual limitation of the concerned photos to focus on a certain section of the collection or working with one object in particular. All contained objects are only copies and are created and can be deleted again without affecting the overarching order. The copies that are created of background objects again preserve some organization, but this time not that of the collection as a whole, but the visual order from being destroyed by the users.

Informed Browsing workflows that become possible by switching between different metadata are supported as well. If the user wants, for example, to choose the best photos from recently made pictures for sharing, a passable way would be to first switch the background to the Time-order and enlarge the time column containing the photos to get a good overview. Then, the designated photos can be moved to another workspace, either as a whole cluster, if they are already organized that way, or one by one. The order of the workspace can then be changed to Stream, which sums up all similar photos in piles and arranges them based on the time they were taken, thus giving the user an idea of the time structure of the photos. She can then open a new workspace and put the best shots there as a preparation for sharing and finally sort them again by Time (and close the first, temporary workspace).
Another exemplary workflow would be deleting failed photos from several shots of one motif. Those should be sufficiently similar to let the system group them to one pile each when changing the background's order to Similarity. Now the user can unfold each pile and delete bad photos directly or move a pile to a new workspace and change its order to Quality, to see the system's opinion on the matter.

The creation of clusters and subclusters gives the user the chance to structure the collection on a basic level herself, to increase the speed of interaction and the clarity of the visualization. But this is not obligatory - working with a completely unstructured collection is feasible as well thanks to the gathered metadata.

Last but not least, satisficing and sidetracking might appear as well.
Satisficing can happen in all circumstances where the user is looking a photo but has a hard time finding it. While the standard search process in *flux* is not linear from the first to the last element but more like backtracking (e.g., gaining an overview on the Time-order, drilling down in one time column, opening the most auspicious piles, moving up to the next higher level if nothing was found) and thus more promising to reach a good solution, the user can conclude at any point and stop when an acceptable result was found.
Sidetracking also happens often, because *flux* optimizes the display for showing as many photos as possible and thus always confronts the user with maybe long forgotten or interesting ones.

5 Implementation

The following part describes the development of the implementation. I first provide the hardware background, then give a summary of a prototype version I built, followed by an overview of the class structure, of important aspects of it and exemplary data flow. The chapter ends with a description of used existing third-party technologies and how they were embedded and an explanation of selected algorithms in detail.

Flux was implemented in the course of June to August 2007. This long duration is in part due to the intertwined design refinements: Some ideas had to be tested, as it is often the case, in the live system and even major parts were dropped as soon as it became apparent that either their gain was overestimated or an implementation (especially regarding the graphical aspect) could not be completed in a performance-friendly way.

In the following, I will present the latest version of *flux* that is currently running on our table hardware. The system was developed using Microsoft Visual Studio 2005 Professional Edition in Visual C# and in conjunction with the Microsoft .NET Framework 2.0. The used graphics platform was Microsoft Direct3D 9.0.

5.1 Hardware

The system's target hardware where it was deployed eventually is an interactive table that was built for the Fluidum-project. It consists of a 42-inch LCD display mounted within a customly built cabinet. The display's native resolution is 1360×768. Its whole surface is surveyed by an overlaid, touch-sensitive DVIT [84] panel, that tracks objects (e.g., fingertips, pens) on it. While the DVIT-panel is originally constructed for vertical operation (e.g., in an interactive Whiteboard), it works as well in a horizontal position. The advantages of such a setup in contrast to other, similar tabletop-systems are:

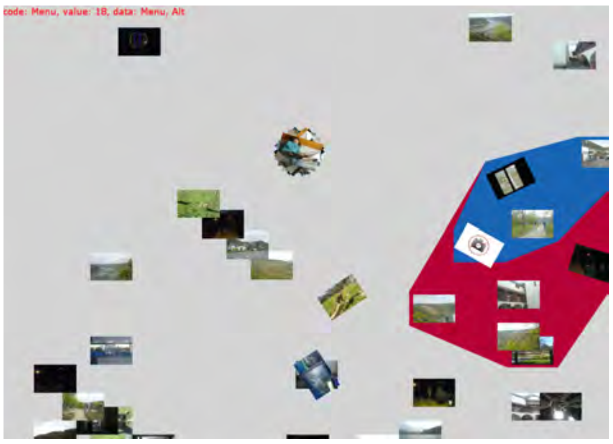
- Every object that blocks light can be used as an input device, in contrast to other technologies like DiamondTouch [16] or SmartSkin (FTIR) [37] that rely on the user's hands as input devices. Advantages are a more precise input by using something thinner than a finger (e.g., a pen) and the possibility for more unconventional uses, like fixing an object on the table by putting a small physical object on it.
- Awkward sitting positions are not necessary - legs can be placed under the table and the system thus operated for longer periods of time without fatigue.

Yet, while using an optical recognition system has certain merits, it also includes several drawbacks:

- Lighting conditions in the room become important - using bright infrared emitters can disturb the recognition.
- Everything is tracked, thus making it necessary for users to literally "roll up their sleeves" before interacting with the table.
- Objects occlude one another by blocking their light from other cameras with a large object rendering the recognition of other inputs almost useless.
- Because the recognition works based on triangulation, with four cameras it is only possible to track up to three objects, with a trade-off in accuracy for each additional one. The Smart-SDK that I used only permits two inputs to keep the flickering of recognized objects as low as possible. Additional objects on the table are ignored, but of course disturb the tracking of existing inputs (see previous point).

Especially the last point, limiting the number of concurrent inputs to two, drastically shrank the opportunities for multi-user interaction: Social protocols become necessary, because several awkward situations arise by this limitation. While two users can work in parallel if they both only use one-touch actions like drawing or rotating and translating a photo, as soon as one of them tries to use a two-finger action she either succeeds (because the other user's finger was currently lifted) and thus breaks the other user's ability to interact, or she does not and has to negotiate for a time window where the other user pauses.
Other hardware does not have those limits, which is why I eventually decided not to make this two-input limit a firm constraint (see above: 4.5).

5.2 Prototype



```
code: Menu, value: 18, data: Menu, Alt
```

Figure 5.1: The first prototype of flux

The first version of *flux* (see figure 5.1) that was finished by June aimed at providing a testbed for different kinds of interaction techniques. Basically, it loaded several photos from one folder, placed them randomly on a 2D-plane and let the user play around with them.

5.2.1 Interaction

The general elements of *flux* - photos, piles and clusters - were already contained. Photos could be rotated and translated with a one finger touch and scaled with two fingers. Piles were created manually, by dragging photos onto one another. Piles could then be translated and rotated as well with one finger (while all contained photos were arranged in a circle around the foremost one) or be spread out by crossing them with the finger. On lifting the finger up, the pile snapped back again (see figure 5.2).
Clusters were created, just like in the final version, by touching the background for a short time, then drawing a surrounding shape and lifting the finger up again. Hierarchical clusters were not possible, because internally a photo could belong to several clusters (see figure 5.3).

5.2.2 Results

I initially created the prototype to get used to Direct3D and C#, guessing that there would be a number of difficulties creating a basically two-dimensional application in three-dimensional surroundings. Having built the basics that drew the world, caught inputs from the mouse as well as the tabletop and loaded the photos, I started implementing the interaction, namely, movement and scaling for photos. Afterwards, the above interactions for piles and clusters were created.

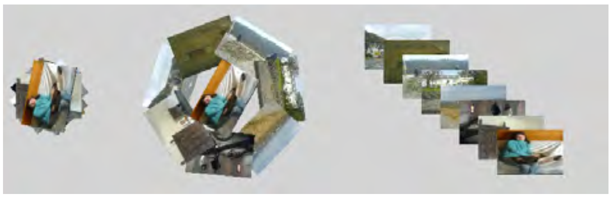


Figure 5.2: Pile interaction in the flux prototype

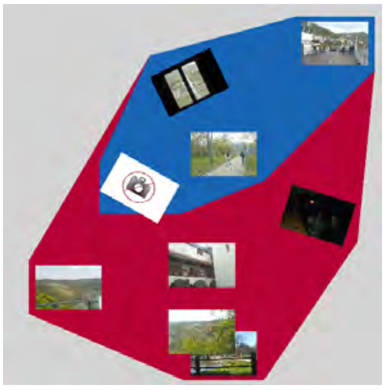


Figure 5.3: Two connected clusters in the flux prototype

At this point, I found myself well enough prepared to write the software's actual version, with a stricter and tidier organization and a less "grown" architecture. Still, I reused several parts of the prototype code that were by then stable enough and adapted them, most prominently the methods for Rotate and Translate and Rotate and Scale (4.5.1).

5.3 Basic program structure of flux

Flux is based on the Model-View-Controller software pattern: It operates on an internal model of abstract photos and clusters that are displayed by several view-classes. Interaction from the user is caught by the controller and results in changes to the model that are transferred to their visual representatives by calls to their update-methods (this part uses the Observer software pattern).
The most time, the software courses through the render-loop in the Main-class, that redraws the Direct3D-world, in turn updating some objects.
This rough structure makes up the main (and visible) part of *flux*. The remaining available machine time is taken up by threads that run independently from the main interface and produce the results of user interactions in the controller, perform maintenance operations like loading photo textures, recalculating cluster borders or arranging the contents on a view.

5.3.1 Class structure

(For a class diagram see figures 5.4 and 5.5)

Main and initialization classes

- **flux.cs** This simple class creates a new instance of Main.cs and runs it.
- **Draw.cs** Draw contains all methods for the creation and drawing of vertex buffers and textured 3D objects. These methods are static and can thus be called by all other classes.

- **Main.cs** Main performs all initialization processes like creating a Direct3D-Device, loading the configuration file, initializing the model and view and creating and starting the controller. It furthermore runs the render loop that redraws the application.

Model

- **World.cs** World is the abstract counterpart to VisibleWorld. It holds references to all photos and clusters. If a photo or cluster is deleted from World it is treated as no longer existing.
- **IModelObject.cs** This interface symbolizes that an implementing object is a part of the model.
- **IContainable.cs** This interface derives from IModelObject and additionally indicates than an object is containable (Cluster and Photo implement it).
- **Cluster.cs** A cluster in the model contains one or more other clusters and photos.
- **Photo.cs** A photo in the model holds meta-information about a photo in the file system, like absolute file name, date of capture, dimensions etc. It can be part of a cluster, but does not have to be.

Control and input

- **Controller.cs** The controller manages the application's interaction with the user. It has two big tasks: First, it works as a handler for keyboard, mouse and SmartBoard events, that converts the information into a general input format. Second, it analyzes the differences between the current and the previous input state, finds differences (i.e., some input was changed in some way) and makes corresponding changes in model and view (like creating a new cluster or workspace, showing a marking menu, etc.). Additionally, it catches expired input timers and reacts accordingly (for more information on the handling of inputs see below).
- **Input.cs** An instance of Input is a generalized version of any input source.
- **InputState.cs** InputState holds several Inputs and provides methods to access them by their ID. Controller uses two InputStates: One from after and one from before the latest input event.

View

- **VisibleWorld.cs** VisibleWorld is the view's counterpart to World.cs. It is, contrary to its name, not visible but contains references to all visible objects (mostly indirectly via Views). It also controls and manages the connection to the PhysX engine via PhysXSimulator.cs (see below).
- **View.cs** A View is a workspace - it derives its geometric features from Square (see below) and contains IVisibleObjects, namely VisiblePhotos, VisibleClusters and VisiblePiles. It also contains methods to change the arrangement of contained objects that can be triggered by Control.
- **Background.cs** Background is a View with an updated drawing-method, that shows a calendar in the time-based arrangement and allows its manipulation. Background cannot be scaled, rotated or translated and the user is only able to manipulate copies of its contained objects.

- **IVisibleObject.cs** The interface is implemented by all objects that can be displayed. It requires a Z-value for the object, to keep a visible order of objects within a container. IVisibleObjects also have to have a method DrawMe, that triggers the drawing of the object.
- **IHasMarkingMenu.cs** An interface that indicates that an object has a marking menu.
- **IContainer.cs** An IContainer can hold other visible objects. It provides convenience methods for Controller, that return objects at a certain coordinate or objects that intersect with a polygon. This interface is implemented by View, VisibleCluster and VisiblePile.
- **IContainable.cs** IContainables can be situated within IContainers. Additionally, such objects can create proxy objects on the overlay (if they are on the background). Implementing classes are VisibleCluster, VisiblePile and VisiblePhoto.
- **ISelfPropelled.cs** An ISelfPropelled object can move on its own, has a speed, a target position and scale. Its method NextFrame calculates its next position and should be called before drawing it.
- **IModelBased.cs** IModelBased indicates that an object of the view has an equivalent in the model.
- **VisiblePhoto.cs** A VisiblePhoto is the view's counterpart to Photo. It has a position and dimension and can be contained within a VisiblePile, VisibleCluster or View (Background). It also is ISelfPropelled and can move on its own.
- **VisibleCluster.cs** A VisibleCluster is the viewable version of Cluster. Its PlaceYourselfInGridX-methods force an instance of it to build piles and position itself to use a portion of the screen as good as possible. A VisibleCluster's border consists of metaballs that have to be recalculated in an own thread as soon as the positions of contained objects change.
- **VisiblePile.cs** VisiblePiles contains photos and can in turn be contained by Views or VisibleClusters. They are unstable structures that are normally destroyed, as soon as the View's current arrangement is changed.
- **Overlay.cs** Overlay displays objects on top of the 3D-world using an orthogonal projection. It contains two private classes, OverlayPile and OverlayPhoto that are used by background objects to display transformable copies of themselves. Other objects on the overlay are marks for input points and marking menus.

Connection to the PhysX engine

- **PhysXSimulator.cs** This class forms the connection between the VisibleWorld and the physics engine. It holds an internal (physical) version of the VisibleWorld and updates the visible objects if changes by collisions etc. occur. To prevent racing conditions all changes (like adding or removing objects) are added to a task list that is processed before the actual physical calculations take place. This step-method is called by VisibleWorld each frame.

Geometry helper classes

- **ITranslatable.cs** An ITranslatable object has a position and is translatable.
- **IRotatable.cs** An IRotatable object has a rotational value and is rotatable.
- **IScalable.cs** An IScalable object has width and height and is scalable.

- **IObject.cs** An IObject is translatable, rotatable and scalable.
- **Square.cs** A square implements the IObject interface and all the required methods. It is the base class to many of the view's classes to provide methods that are for all visible objects identical.

User profiles

- **Profile.cs** Profile holds configuration values that control the behaviour of the application. Those can be general ones like screen dimensions or the location of the configuration file but also specific ones that control border sizes or thresholds for algorithms.
- **ProfileFactory.cs** ProfileFactory creates concrete instances of Profile for different purposes.

Helper classes

- **Topology.cs** This class calculates convex hulls and metaballs and triangulates polygons in static methods.
- **CfgLoader.cs** Loads the configuration XML-file and converts its information into a Photo-Cluster-structure.
- **Convertor.cs** Provides miscellaneous conversions for vectors, matrices and colour spaces.
- **Counter.cs** Counter returns unique IDs for different classes.
- **MathHelp.cs** MathHelp has different static methods, that perform general operations from linear algebra, like converting points between coordinate systems, rotating vectors etc.

Texture loading and management

- **ImageLoader.cs** ImageLoader loads photos from JPEG-files and converts them to Direct3D-Textures. Photos can be loaded in different levels of quality (to go easy on the texture memory). ImageLoader has a task list of photos to be loaded.
- **ResourceManager.cs** ResourceManager holds all the textures in *flux*. It keeps track of loaded photo-textures, prevents duplicates and initially generates the marking menus.
- **PhotoTex.cs** PhotoTex holds the texture of a photo with its quality level.

5.3.2 Important aspects of the implementation

The render and physics loop The central part of *flux* is the Render-method, located within Main.cs (see figure 5.6). In this endless loop all internal adjustments of the world are performed, like e.g., user-independent movement of objects, and the display is refreshed.
After checking the Direct3D device for availability (if this check fails it is reset), the camera position is set and the DrawWorld method of VisibleWorld called. This method consists of two steps: First, the physical model is updated by calling step from PhysXSimulator. Here, all tasks that have accumulated since the last run are processed (e.g., adding or removing physical objects), then the simulate method of the PhysX engine is called to perform the actual calculations and finally the results are used to update the positions and rotations of the visible objects. The second step is the recursive drawing of all visible objects. VisibleWorld calls the DrawMe-methods of the contained Views, which in turn call those methods of their contained objects like VisiblePhotos, -piles and -clusters.

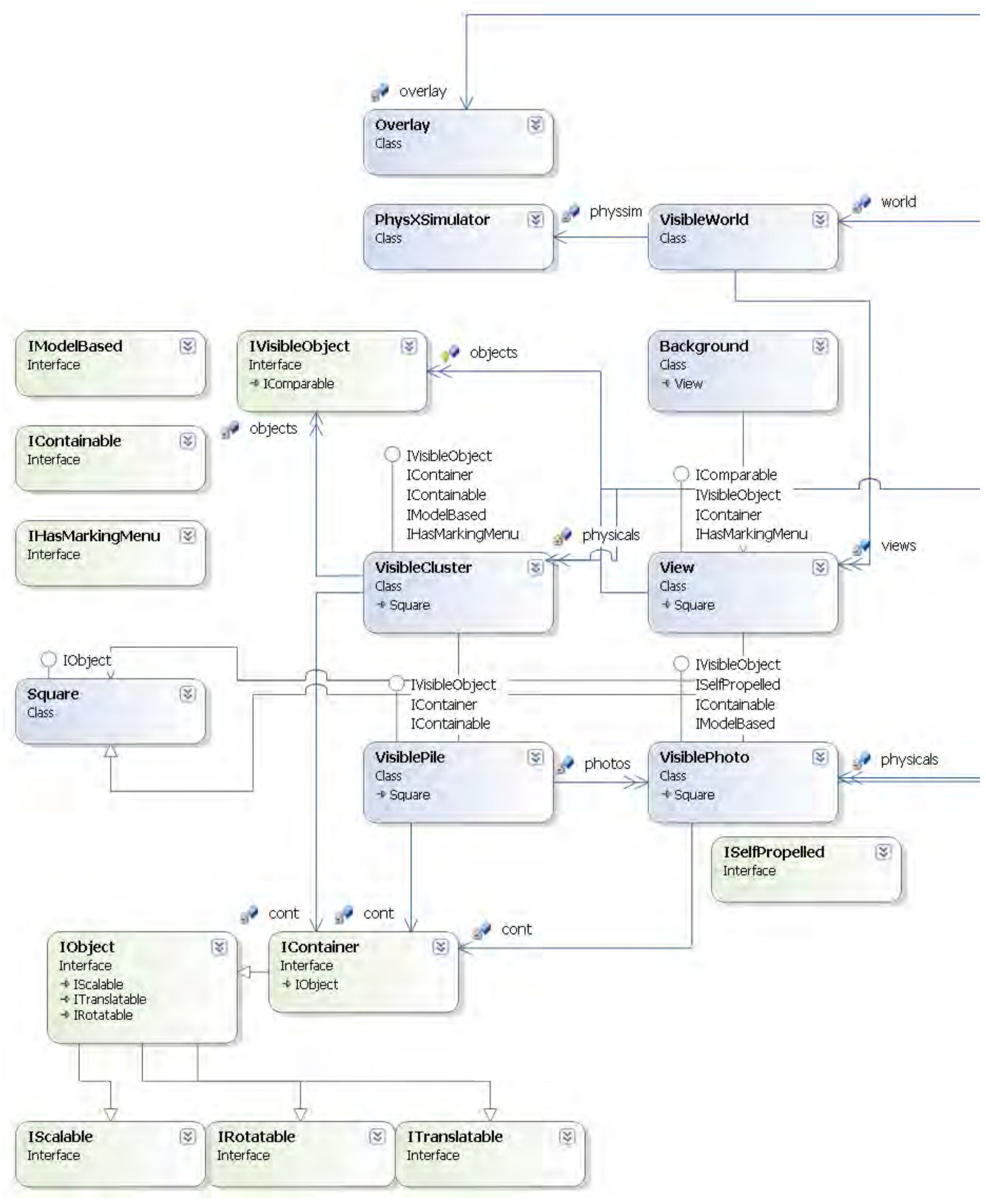


Figure 5.4: Class diagram of flux - left side -

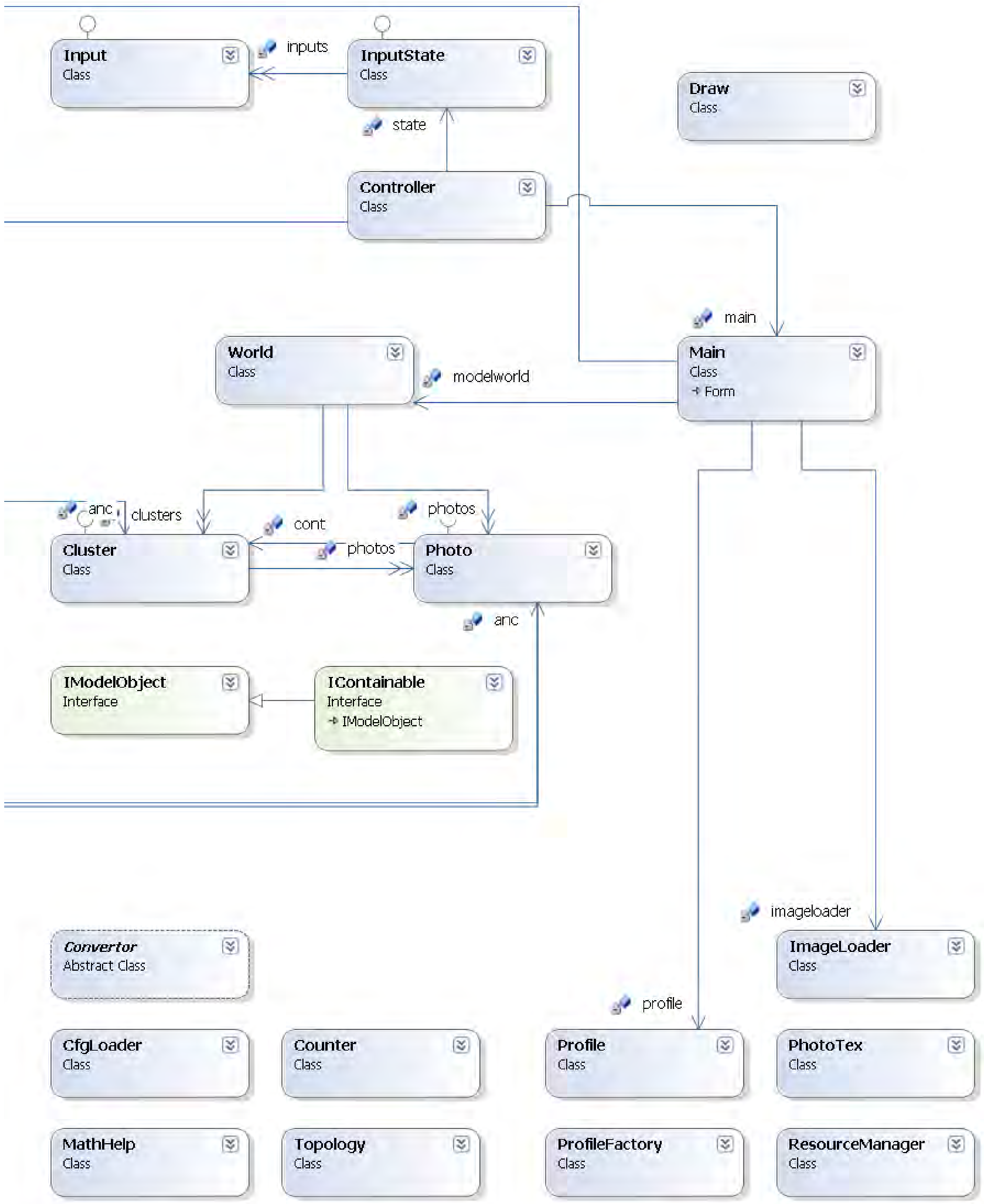


Figure 5.5: Class diagram of flux - right side -

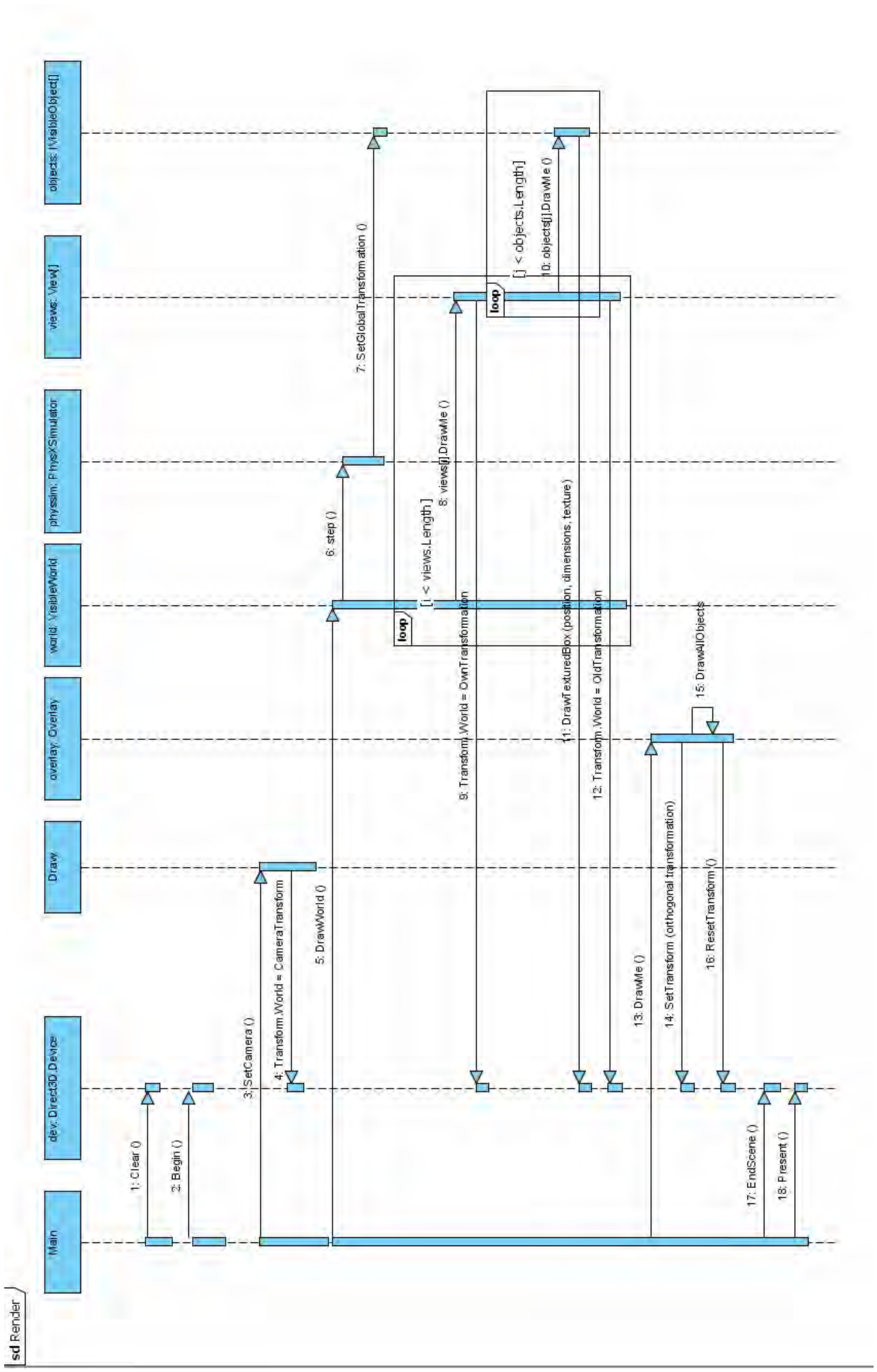


Figure 5.6: Render loop in Main.cs

Now that all visible objects are redrawn, the overlay is refreshed. It consists of polygons, circles, marking menus and copies of the photos and piles of the Background. Objects on the overlay always lie above the rest, so their drawn last and after one another.
Having drawn all objects, the methods for finalizing the Direct3D scene are called and the loop restarted.

Connections between visible and model objects Not every visible object has a counterpart in the model. This is due to the fact, that View and Background represent only (as the name implies) views on the underlying cluster and photo-structure. They can be dissolved easily without destroying it. Piles also are only created in need and do not have to exist necessarily on each view that contains the affected photos. So they are realized as view-only structures that can easily be removed by the controller without affecting the general structure.
VisiblePhotos and VisibleClusters, however, have counterparts in the model. They add themselves in their constructors as listeners to the model versions. As soon as a change takes place in the model (examples are moving a photo to another cluster, renaming a cluster or removing it), the ModelUpdated-methods (or ModelMinorUpdate for minor changes in the cluster) of all listeners are called. The VisiblePhoto version checks whether it is still within the same container as its model counterpart. If not, it searches for the new VisibleCluster in its View via the GetVisibleCluster-method. This method either returns the cluster (all visible clusters and photos are unique within one view) or, if it is not yet contained, first creates and then returns it. The VisibleCluster version checks whether it contains the same objects like the model version and adds them if necessary, using the GetVisibleCluster and GetVisiblePhoto-methods from the View. It also checks whether it is still in the right container, but does not force the View to add it if not: This keeps the system from adding the complete Cluster-Photo-structure to each view as soon as the ModelUpdate method is called, which would take whole complexity reduction by additional workspaces idea ad absurdum.
If the addition of uncontained objects or the removal of contained objects fails because the VisiblePhoto ModelUpdate already executed it, it is ignored. Also, VisiblePiles that contain VisiblePhotos but have no model counterpart are special objects that have to be treated as such: When looking for surplus or missing photos, the VisibleCluster has to check the photos in a pile separately.
Using the observer-pattern for the connection between model and view has the effect that one change in the model automatically leads to updates for all visible counterparts in all views. Thus, the creation of a cluster, for example, propagates itself to all Views and even adds missing photos (i.e., photos that are in the cluster but not in the particular view) via the GetVisiblePhoto-method.

Different coordinates systems Each object that implements the IContainer interface has its own coordinate system. This facilitates repositioning the whole container with all objects, because only one position (namely, the container's position within the parent's coordinate system) has to be changed. When redrawing a container, first the Direct3D coordinate system is set to its own, by setting the device's Transform.World matrix to the container's transformation matrix. Then, all objects are drawn at their current positions, which are based on the container's coordinate system. When moving an object from one container to another in different hierarchies, the object's coordinates are first transformed to the global coordinate system (which corresponds with the Direct3D one) and then back again to the local ones of the new container. Converting between coordinate systems is handled by the GlobalToLocal and LocalToGlobal methods of every visible object. Because objects can be contained hierarchically, this conversion is performed recursively, multiplying the transformation matrices of all containers within the hierarchy.
Every visible object additionally provides different methods for getting its corners (if based on geometry.Square) in either local or global coordinates to use for collision detection with e.g., user-drawn polygons or other input points.

Realization of occlusion It is important to show an object with which the user is currently interacting in front of others. While Direct3D provides this feature inherently by letting objects which are nearer to the camera occlude other objects that are further away, I did not use this option in *flux*, because the renderer does not use orthogonal projection, so nearer objects are displayed larger as well (which in turn would create the necessity to shrink objects based on their distance from the plane to always keep them the same size) and managing the Z-coordinates of objects would have been necessary in any case.
So I created my own version of it, with a long-attribute Z for every IVisibleObject that determines which objects are drawn first and therefore occluded by others (a higher Z-value means a later redraw means less occlusion). Internally, all objects within one container have a unique Z-value ranging from 0 to (number of contained objects - 1). If an object is added or removed, the Z-values of the others are adjusted accordingly. Each container provides the method BringToFront, that moves a visible object to the front of its container by giving it the largest Z-value.

Connections between the physics engine and visible objects Visible objects and their physical counterparts do not share the same transformative information - every position, rotation and scale is stored twice. The reason is the same as in the model-view-dichotomy: Not every visible object has a physical counterpart. All objects that lie on the background have no physical analogy, so their information has to be held separately anyway. On top of that, the visible objects use other data structures for saving position and rotation (DirectX vectors and float respectively) than their PhysX-versions.
The creation and removal of objects is only one-way: If a new object appears or an old one becomes redundant, VisibleWorld calls the respective methods to update the physical model. Updating position or rotation, however, normally works both ways. Visible containers can block an update of their contained objects by setting the rearranging-flag (most prominently used by Views when changing their order). The physics engine then updates its internal representations accordingly.
If the update of the visible objects passes, their UpdatedByPhysX flag is activated, which can be checked by their containers to react (e.g., redraw the borders of a VisibleCluster if the contained objects are moved).

The Overlay object The Overlay is a multi-functional class that is used for direct user feedback, showing copies of visible objects and marking menus. It basically consists of lists of objects that are redrawn by Main's render loop and added and removed by the Controller. Unlike other visual objects, those in Overlay are in two-dimensional screen coordinates and the rendering is subsequently based on an orthogonal projection.
The first type of objects in Overlay are circles and polygons. Circles mark the recognized positions of input sources and give the user a direct feedback where the system thinks she is currently pointing at (which is convenient because of the SmartBoard's unreliability in this regard). Additionally, they turn from red to blue as soon as the input's timer expires (via the TimerCallback method in the Controller) and thus notify the user of the mode switch. If the user starts drawing the outlines of the drawn shape are made visible by Overlay as well in the form of a polygon. Overlay keeps track of the drawn points and returns them to the Controller to determine which objects are affected when the user lifts the pen again.
Photos and piles on the overlay are created by their respective visible complements. They keep track of position and rotation and update the overlay versions accordingly. User manipulation is handled via proxy objects, that are basically general Squares and created temporarily. As soon as the user is finished, their values are fed back to the responsible VisiblePhoto or VisiblePile and in turn to the Overlay.
OverlayPiles themselves are not visible - they only consist of a number of OverlayPhotos that are

positioned accordingly and hold the information necessary to interact with them as piles and not as photos (e.g., prevent them from snapping back if the user lets go).
The third important category of objects within the Overlay are the marking menus. Both VisibleClusters and Views provide them and they can be called by waiting after the first input timer callback for the second one. While the View marking menu only provides access to the four arrangement types (which are triggered after releasing the input above one of the sectors), the cluster menu allows changing its color and name or deleting it. The name change is handled by a new frame, which also appears on the overlay and works similarly to a marking menu, that feeds back the user-made strokes to the Tablet PC SDK, which tries to recognize written words.

Input sources *Flux* was built with the support for multiple input sources in mind. In my implementation the number of inputs is only limited by the hardware - with the SmartBoard and the computer's mouse we have a maximum of three inputs in the current setting. All inputs are merged in the Controller and converted into the generalized Input class. This class saves for each input a unique ID (provided by the Controller), its current and last position in screen coordinates, its state (DOWN, MOVE, UP) plus a reference to the visible object the input is currently interacting with (if any) and a timer that starts as soon as the DOWN-state is set, has a callback method in Controller (TimerCallback) and is stopped if the state changes before it expires.
The Controller's task is to handle the events triggered by different input means (e.g., mouse, SmartBoard), convert their information to an Input, save these inputs in the current state and check for resulting actions. The sum of all inputs is hosted within InputState, which is basically a list of Inputs with convenience methods for getting an input via its ID. Controller uses two instances of InputState, one from before the latest input event and one from after. The new one is basically a copy of the old one incorporating the latest change. The Controller then compares the two states via the method UpdateState, finds differences based on the state and performs necessary changes in model and view.

5.3.3 Data flow in concrete situations

In this section I describe the application's internal data flow using several concrete examples.

Copying photos to other workspaces Having created a workspace, the user opens a pile, drags one photo into the workspace and closes the pile again.

1. The first step is selecting and opening the pile on the background (see figure 5.7). The centre for this action is the Controller: After catching an input event (be it from the mouse, the SmartBoard or any other source), its position and ID is transferred to the Down-method, that updates the current input state, saves the previous one and calls UpdateState (again, within Controller) with the two states as parameters.
   This generalized method compares the two states, finds the differences (in our case the new DOWN-state of one input) and gets the selected object by first querying the overlay if an object at the input's position exists (the overlay always has precedence) and then the VisibleWorld via GetVisibleObject (see figure 5.8). This method steps through all views, checks if the position is within and if so, lets the view return its foremost object at this point (by recursively traversing the hierarchy of contained objects and stopping at the lowest one).
   After getting the foremost object (in the current example a VisiblePile), it is saved within the Interac attribute of the Input for easy retrieval. When moving the input source, the VisiblePile's proxy object is manipulated by the Rotate-And-Translate or Rotate-And-Scale methods (UpdateState first checks whether the Interac-object is a VisiblePile or a VisiblePhoto and if it lies on the background (and not another view). In this case a proxy object is created the pile or the photo via CreateProxy and used to conserve the actual positional and

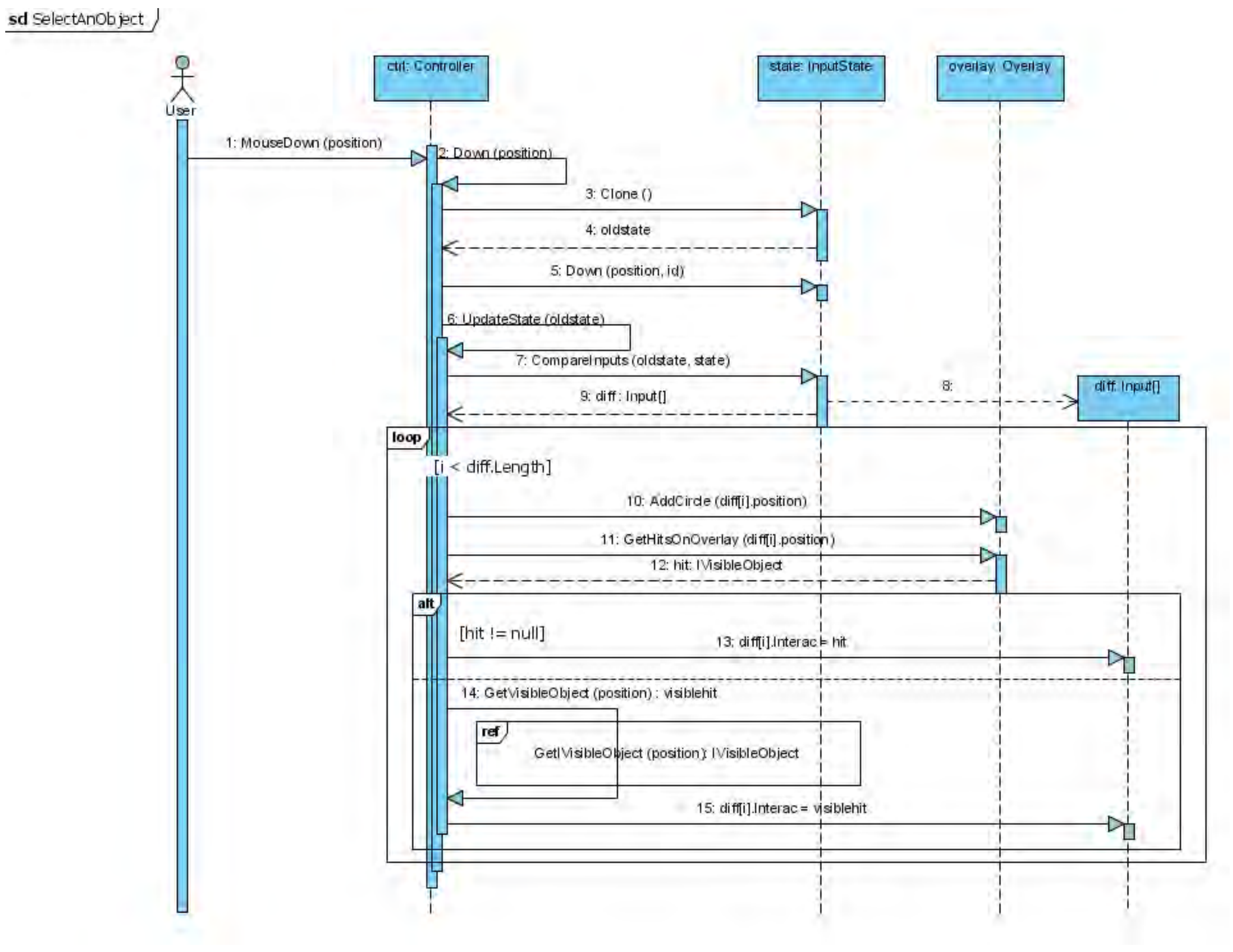

sd SelectAnObject
User
ctrl: Controller
state: InputState
overlay: Overlay
1: MouseDown (position)
2: Down (position)
3: Clone ()
4: oldstate
5: Down (position, id)
6: UpdateState (oldstate)
7: CompareInputs (oldstate, state)
8:
diff: Input[]
9: diff : Input[]
loop
[i < diff.Length]
10: AddCircle (diff[i].position)
11: GetHitsOnOverlay (diff[i].position)
12: hit: IVisibleObject
alt
[hit != null]
13: diff[i].Interac = hit
14: GetVisibleObject (position) : visiblehit
ref
GetIVisibleObject (position): IVisibleObject
15: diff[i].Interac = visiblehit

Figure 5.7: Selecting an object on the background via mouse

rotational values). The pile then updates its copy on the overlay via the MoveOverlayPile method. If the pile's copy does not exist yet, it is created and all contained photos are added as overlay photos to the Overlay's object list and placed in a row via ArrangePhotos. Every time the input source is moved (without lifting it) the position of those overlay photos is updated. If the user finally lifts it the pile's internal copy is removed and only the overlay photos remain.

2. Now that the pile is opened, the user lifts the input source up and puts it down again on one of the overlay photos. The GetHitsOnOverlay method returns its parent object (a VisiblePhoto) and UpdateState again moves its proxy when the input is moved. UpdateState also passes on the View the proxy is currently above, which is transferred from the VisiblePhoto to the Overlay together with the updated coordinates and rotation and is used by the overlay photo to signal this with a red instead of a black border. Additionally, the proxy's new view is kept within the VisiblePhoto to use it as soon as the input is lifted. If that happens, Controller checks if the view already contains an instance of the photo and if not, creates and adds it (its position and rotation is taken from the overlay photo) via the AddPhoto method (which takes the photo object from the model as parameter).

3. The user can close the fanned out pile again by touching its actual position. UpdateState checks for each input whose state changed to down whether its interaction counterpart (determined by the process above) is a VisiblePile and whether its proxy object is still visible.

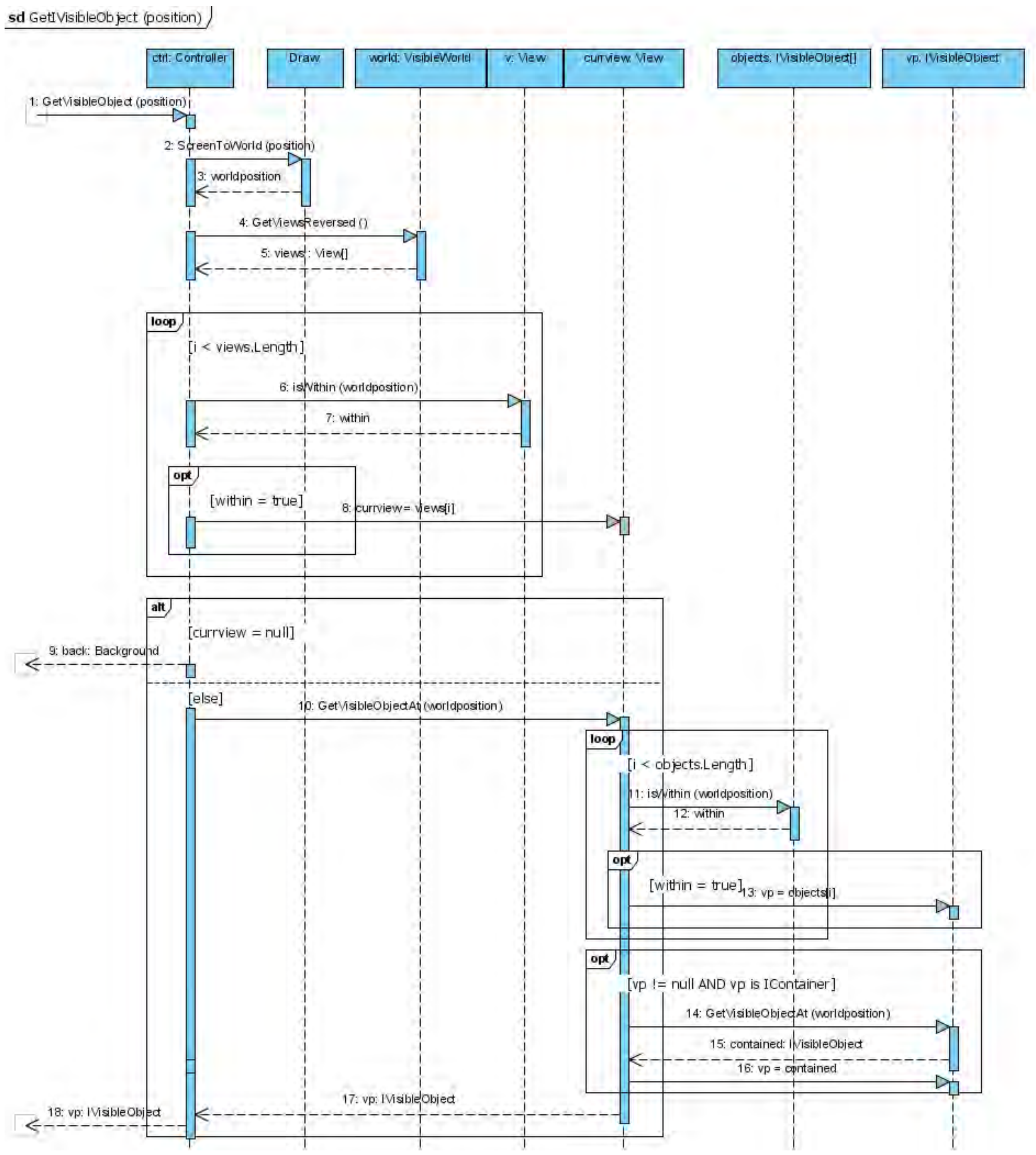


Figure 5.8: Finding an object at a certain position in all visible objects

If so, the piles RemoveProxy and subsequently the overlay piles method of the same name are called. The latter removes all created copies of photos and then itself from the Overlay.

Creating a new cluster or workspace Both actions share certain key elements with greater differences only in the end.

1. By putting down a pen on the table or pressing the mouse button the user triggers a cascade of Down-events that drip down from Controller to InputState to Input. If the Down method is called in Input it starts a timer with TimerCallback in Controller as callback function. Changing the state of the input leads to the cancelling of the timer. TimerCallback does not much more than counting how many times the timer has already expired and adapting the user interface accordingly. After one expiration the colour of the circle that represents the input source on the Overlay is changed from red to blue and after two expirations the marking menu is shown if the affected object has one.
   UpdateState checks for each moved input if its timer has already expired once. If so, it no longer moves the visible object below it but lets the user draw a polygon. This polygon is created on the Overlay, has a fixed ID, which is held within the Input object, and contains all points the user has drawn so far. It also checks on adding a point whether it closes the polygon (i.e., it lies near the first drawn point) and changes the polygon's drawing mode to lasso (again symbolized by a colour change), which means that the polygon itself is closed and all additional points are added to the tail of it. Each polygon is redrawn within the Overlay's DrawMe method. Switching between standard and lasso mode lets the user choose between creating a new cluster or a new workspace.
   If the input source is lifted up again, UpdateState calls either FormGroups or FormWorkspace, depending on the mode of the Input's polygon.

2. Both of these functions begin by querying all Views for visible objects that lie within the polygon's body. This is done via each View's GetOverlappingObjects method, which work similar to GetVisibleObject (see above) but has not a single point but an array of points (the corners of the polygon) as parameter and returns possibly more than one object.
   FormWorkspace then creates a new View, loops through all objects within the polygon and creates copies of contained photos with AddPhoto and of contained clusters with AddCluster. The View's size is adjusted to the number of photos, its position is set to the last point of the lasso and it then is added to the VisibleWorld via AddView. Fortunately, creating a new workspace does not require changes in the model section of the application and only adds new visible objects.

3. FormGroups (see figure 5.9) first fetches all affected objects, then dissolves all piles within these objects to retrieve their VisiblePhotos. Dissolving piles is performed via the ResolvePile method, which every pile-compatible container (View and VisibleCluster) provides and that restores the VisiblePhotos' status as independent objects by reattaching them to it. This is necessary because the later creation of visible objects is triggered by their model counterparts only and VisiblePiles do not have these. Afterwards, the method checks whether all photos are contained within the same (model-based) container - if an object implements the IModelBased interface it provides the Anc.get method, which returns an IModelObject (either a Cluster or a Photo). The result of this check is either the model-version of the container or null if none was found.
   Next, all affected objects' model counterparts are removed from their containers, which leads to a multitude of automated calls: Every object in the model calls its listeners' (i.e., all visible versions of it) ModelUpdated methods. This method causes a VisiblePhoto or VisibleCluster to notice that its container (normally a VisibleCluster) is no longer valid, which

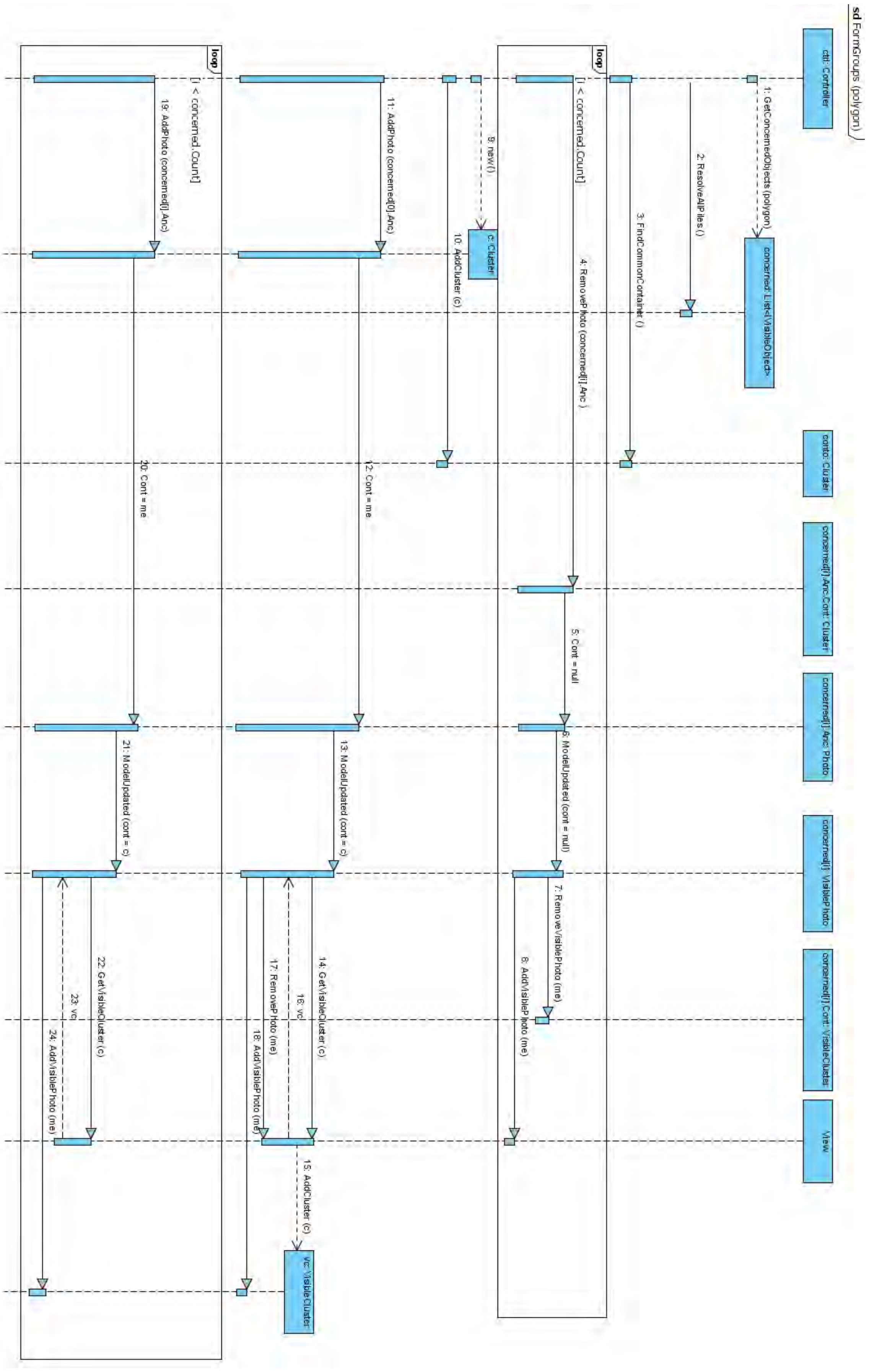

sd FormGroups (polygon)
ctrl: Controller
concerned: List<VisibleObject>
contc: Cluster
concerned[i].Anc.Cont: Cluster
concerned[i].Anc: Photo
concerned[i]: VisiblePhoto
concerned[i].Cont: VisibleCluster
View
1: GetConcernedObjects (polygon)
2: ResolveAllPiles ()
3: FindCommonContainer ()
loop
[i < concerned.Count]
4: RemovePhoto (concerned[i].Anc)
5: Cont = null
6: ModelUpdated (cont = null)
7: RemoveVisiblePhoto (me)
8: AddVisiblePhoto (me)
9: new()
c: Cluster
10: AddCluster (c)
11: AddPhoto (concerned[0].Anc)
12: Cont = me
13: ModelUpdated (cont = c)
14: GetVisibleCluster (c)
15: AddCluster (c)
vc: VisibleCluster
16: vc
17: RemovePhoto (me)
18: AddVisiblePhoto (me)
loop
[i < concerned.Count]
19: AddPhoto (concerned[i].Anc)
20: Cont = me
21: ModelUpdated (cont = c)
22: GetVisibleCluster (c)
23: vc
24: AddVisiblePhoto (me)

Figure 5.9: FormGroups in Controller.cs

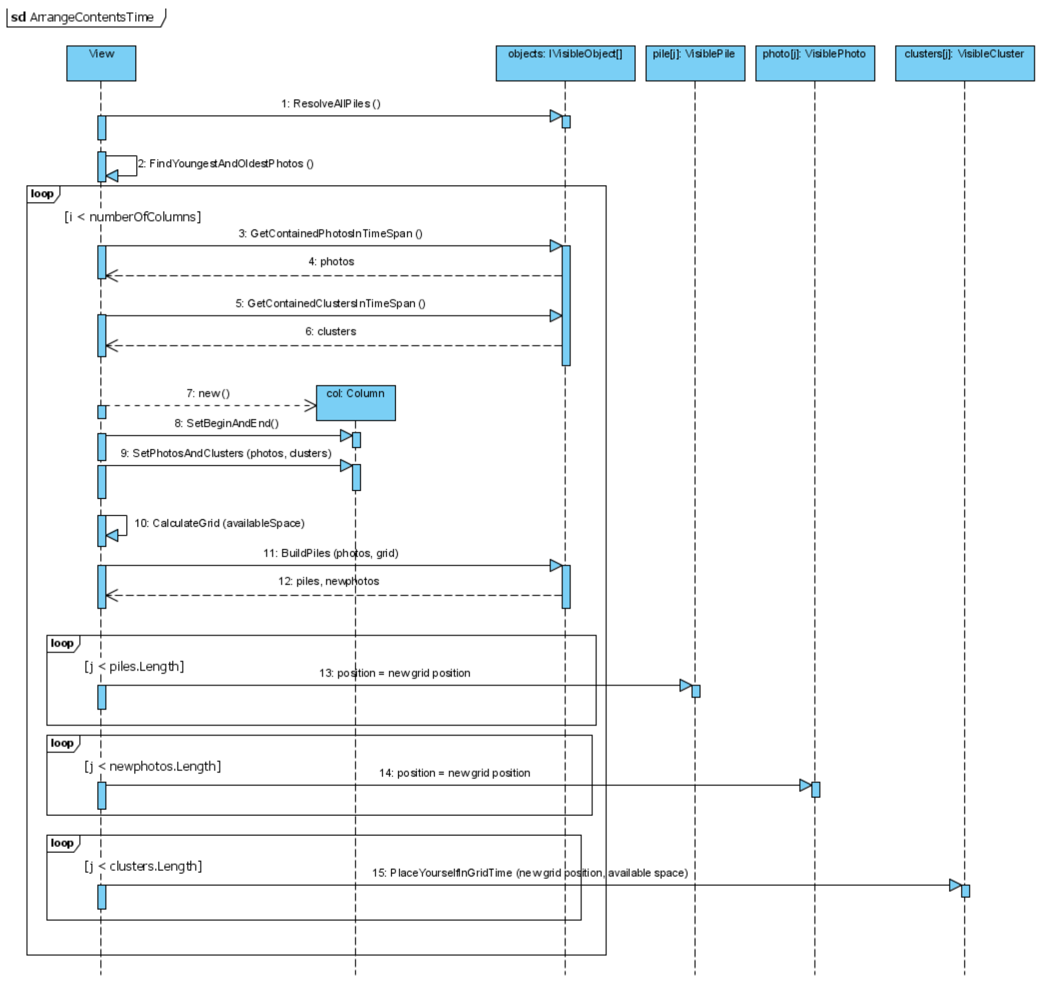

sd ArrangeContentsTime
View
objects: IVisibleObject[]
pile[j]: VisiblePile
photo[j]: VisiblePhoto
clusters[j]: VisibleCluster
1: ResolveAllPiles ()
2: FindYoungestAndOldestPhotos ()
loop
[i < numberOfColumns]
3: GetContainedPhotosInTimeSpan ()
4: photos
5: GetContainedClustersInTimeSpan ()
6: clusters
7: new()
col: Column
8: SetBeginAndEnd()
9: SetPhotosAndClusters (photos, clusters)
10: CalculateGrid (availableSpace)
11: BuildPiles (photos, grid)
12: piles, newphotos
loop
[j < piles.Length]
13: position = new grid position
loop
[j < newphotos.Length]
14: position = new grid position
loop
[j < clusters.Length]
15: PlaceYourselfInGridTime (new grid position, available space)

Figure 5.10: ArrangeContentsTime in View.cs

leads to its removal from it and its reattachment to the highest View in the hierarchy (consequently converting its position to the View's coordinate system). Both affected objects (the VisibleCluster and View) then have to update their internal object lists and make other adjustments (like updating its border in the cluster's case).
A new cluster in the model is created and either added to the common ancestor of all affected objects if one was found or to the root of the model's world.
Taking advantage of the connections between model and view is then repeated by adding all model objects to the new cluster, which again leads to calls to the ModelUpdated methods. A visible version of the abstract cluster is created automatically by the first VisiblePhoto that searches for its new container in its View: It calls GetVisibleCluster, which causes the View to produce and add it, based on the model version which is given as parameter. In subsequent calls from other VisiblePhotos this new VisibleCluster can be returned. All VisiblePhotos' coordinates are thus added to the new VisibleCluster, including adapting their positions to the new coordinate system, and the consistency between model and view is restored. Changing the model leads to changes in all views (see 5.3.2).

Arrange photos within a workspace in order 'time'

1. This process, again, starts with the Down and UpdateState methods in Controller. Down sets the state of the corresponding Input object and triggers its timer. The callback method, TimerCallback in Controller, checks how many times the timer already expired and on the

second expiry causes the displaying of a marking menu if the object below the input supports it (implements IHasMarkingMenu) and sets the Markingmenu flag. The DrawMarkingMenu method, which is provided by View and VisibleCluster, first creates an instance of its own version of Overlay.MarkingMenu (ViewMarkingMenu or ClusterMarkingMenu respectively) and then adds it to the Overlay via AddMarkingMenu.
On every Moved-event UpdateState now evokes DrawMarkingMenu because of the Markingmenu flag. On subsequent calls of the method, only the cursor's position within the marking menu is updated - the menu itself stays put. The object also checks which sector is currently active, underlays it with a color and saves the selected choice. If the user lifts the input source again, the parent's RemoveMarkingMenu is called, that removes the menu from the Overlay and interprets the user's choice, e.g., leading to a call of ArrangeContents in View.

2. ArrangeContents first dissolves all piles (which are obsolete in the new order) and then calls the corresponding ArrangeContents*Order* method (in our example ArrangeContentsTime) in an own thread. ArrangeContentsTime (see figure 5.10) gathers all contained objects, counts the total number of photos. After building the column structure based on the available space each column is filled with VisibleClusters and VisiblePhotos of the respective time frame. It is then divided along the vertical axis among its contents. Normally, there are more photos than available space, so they are combined to piles via BuildPiles. CalculateGrid produces a grid based on the minimal size of a photo which is then filled with the affected objects (first photos and piles, then clusters). The VisibleClusters place themselves using their PlaceYourselfInGridTime method.
   Finally, an update of all clusters' borders is triggered and the Rearranging-flag is set, to allow the VisiblePiles to fetch its photos (photos only move on their own if the pile's container is rearranging).

5.3.4 Interaction with persistent data

ImageLoader I wanted to keep the waiting time for the user at launch as short as possible, so I decided to do the image loading and conversion to textures (which makes up the bulk of maintenance tasks) after launch and in a parallel thread with the rest of the application. If a texture is not yet available, a placeholder image is used. To quicken the time it takes for all photos to become visible, the first priority of the ImageLoader is to load a low-quality version of each photo (which is acceptable, because all photos have a small size in the beginning) before doing anything else. As soon as this task is completed it has free capacities to reload better versions.
In its original implementation, ImageLoader tried to keep all photos on the same level and proceeded straightforward, by first loading all photos on the lowest resolution, then go on with the next quality and so on. The disadvantages of this approach were the radical memory usage (with an unbearably slow application with all photo textures in full resolution) and the time it took for a photo to become available in its actual quality level (especially a nuisance when scaling it).
The second ImageLoader has a task list that is accessible for all other classes and worked off in a First-In-First-Out manner. After all photos are loaded in Main, tasks are created for each of them so they are initially loaded in a low quality (in fulfilling the first constraint). After those tasks are finished, the ImageLoader stops and checks every second if a new task has arrived. If so, it performs it, otherwise it keeps on sleeping and looping. The VisiblePhotos themselves are now in charge of triggering the loading of higher-quality versions of themselves. If their sizes (or their copies') exceed a certain threshold, they add a task to ImageLoader to reload their textures. To lighten the stress on the texture memory, they add tasks to switch back to the original low quality, if they are shrunk again.
Thus, the application's responsiveness and its ease on resources is maximized.

XML Configuration file *Flux* keeps all photos and clusters within an XML-based configuration file. It is based on the following DTD-structure:

```
<!ELEMENT flux (cluster*)>

<!ELEMENT cluster (name, color, contains)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT color EMPTY>
<!ATTLIST color r CDATA #Required>
<!ATTLIST color g CDATA #Required>
<!ATTLIST color b CDATA #Required>
<!ELEMENT contains (photo* | cluster*) >

<!ELEMENT photo (filename, date, quality, deriv, similarity)>
<!ELEMENT filename (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT quality (#PCDATA)>
<!ELEMENT deriv (#PCDATA)>
<!ELEMENT similarity (distance*)>

<!ELEMENT distance (other, value)>
<!ELEMENT other (#PCDATA)>
<!ELEMENT value (#PCDATA)>
```

Main nodes are **cluster** and **photo**. A cluster has a **name**, a **color** (defined by the three values of **r**ed, **g**reen and **b**lue) and contains a number of other clusters and/or photos. A photo has a **filename** (an absolute path in the file system), the **date** it was taken (saved in .NET ticks), a **quality** value ranging from 0 to 1, the second **deriv**ation of its similarity values (which was at one point in developed used for building piles) and finally the **similarity** values themselves. Those consist for convenience of a list of **distance**s to other photos in the feature space, represented by a filename (**other**) and the normalized distance **value** (again: 0 to 1).
An exemplary XML-configuration file can be found in the appendix (B). For more information on the similarity values see below (5.4.3).

5.4 Used Technology

I relied on a number of existing technologies to realize certain aspects of *flux*.

5.4.1 Microsoft .NET framework and Direct3D

From the Microsoft-side I first of all used C# and the .NET-Framework. Furthermore, *flux* is not based on Windows Forms and GDI but runs within the space provided by Direct3D (see figure 5.11). Being actually a pure 2D-application, situating it within a three-dimensional space was not necessary, but I decided to do it because of my background with OpenGL and the lack of experience with Windows Forms, to avoid barring the return into 3D (possibly in a future version) and to use the easy transformation of objects via matrix multiplication (the decision to base the rendering on D3D was reached early on).
Disadvantages were the need to create a texture of each string to be displayed within the world (e.g., on the calendar strip) and the general performance overhead of rendering in three dimensions instead of two.

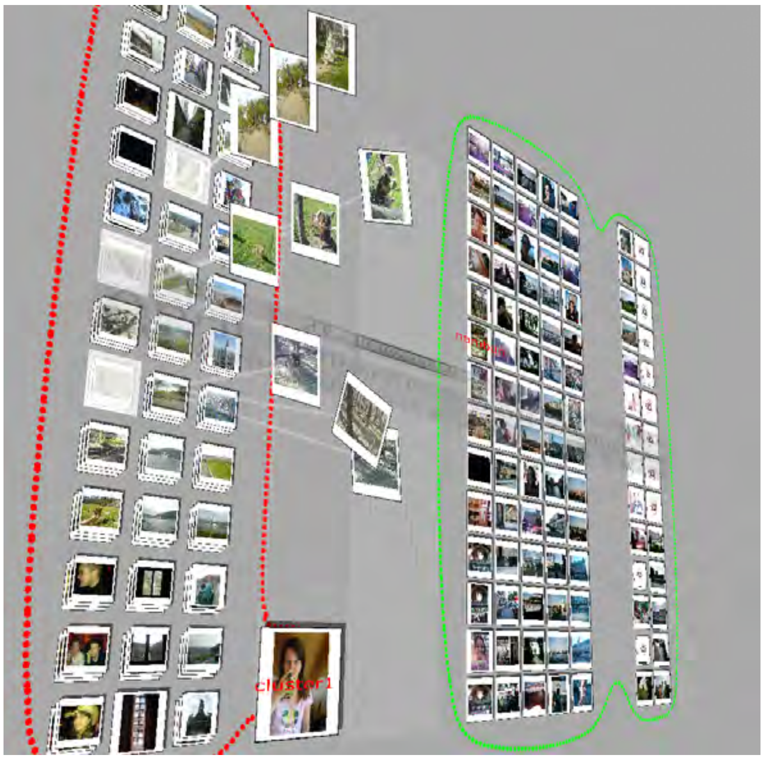


Figure 5.11: The flux plane is situated in three-dimensional space

5.4.2 Physical simulation via Ageia PhysX

Preventing photos from overlapping one another and keeping them from disappearing below others was realized by using a physics engine.

Simulated objects All visible objects in *flux* are situated in one of three layers (see figure 5.12): The bottom layer is the background, with all photos and clusters visible but not movable. The middle layer contains additional views that contains only certain photos and clusters and cannot overlap one another. The highest layer is the overlay, which contains foremost UI-elements.
Only objects on the second layer have physical counterparts, because of their tendency to (and ban on) overlapping. All objects are limited in their movements to the boundaries of their Views and those are in turn limited to the boundaries of the background. Photos and Views are kept from overlapping one another by treating them all like physical objects and letting them collide, thus providing a permanent occlusion-free view independent from the users' actions.

Decision process There is an abundance of engines available, which made deciding for one especially difficult. I based my decision on the following requirements:

- It should be available for free or at least be low-cost for academic use
- To incorporate it in the application it has to be written in C# or another managed language (which fortunately almost all object-oriented languages from C++ to Python are).
- Clear and (preferably) complete documentation should be available. Digging through the code to gather what a function does is no option.
- The engine should be high-performance and eat as little computing time as possible.
- It should provide two-dimensional physics without much tweaking (preferably none).

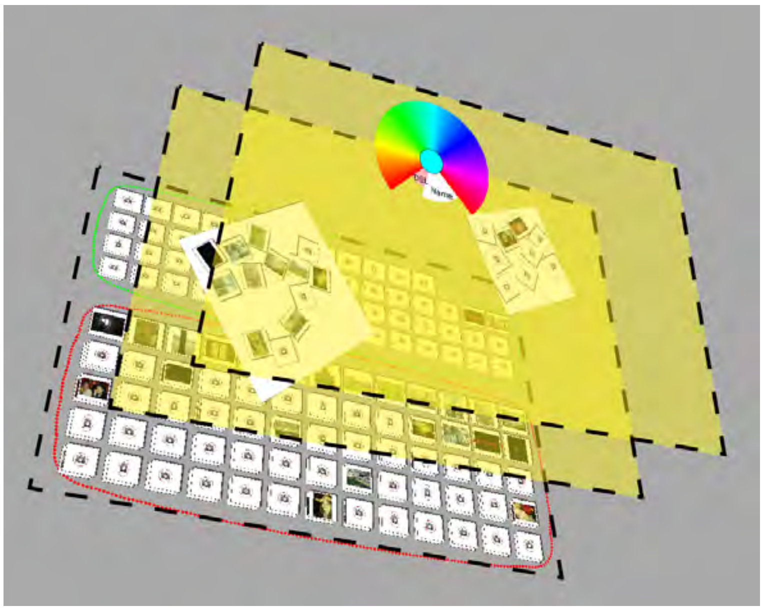


Figure 5.12: Three main layers of flux: background, views and overlay

Searching the internet I found four engines that were shortlisted (see table 5.1).

The most important points for the final decision were the quality of the documentation and the ability to simulate two-dimensional physics: While Physics2D.NET ([64]) is written in managed code and especially optimized for 2D calculations it fails on documentation, making the development of software based on it tiresome and tedious. Newton ([72]) and ODE ([85]) are well documented and have a very active developer community but support objects in one plane only via joints between the object and the plane which can break if the engine miscalculates or the forces become too great, sending the affected object deep into the third dimension. PhysX ([59]), which is actually only the SDK for the physics processor (PPU) of the same name by Ageia, provides not only commercial-grade documentation but also an extremely adjustable limitation system for objects, making it possible to restrict objects to an arbitrary number of dimensions or axes and thus became the solution of choice.

Adjustments To adapt PhysX for my purposes, I had to make several adjustments:

- First of all, PhysX is actually built for computer and video console games which is why the developers attached great importance to its engine being high-performance. It is only consequently written in C++ and supports the execution as managed code not natively. Because of all its qualities it is, however, also popular with hobby developers who prefer more accessible languages like C# or Java, which led Jason Zelsnack to write .NET wrappers for several versions and even adapting the data structures (vectors, matrices, etc.) to Direct3D ([91]). Unfortunately, he obviously stopped the development so the latest supported version of PhysX is 2.5.1, which has prevented me from using the latest features (see below) from the engine.

- The central method of PhysX is simulate, which proceeds with the simulation for a certain period of time. In its course it normally manipulates most of the objects of the current physics scene and changes their attributes. Simulate is called in PhysXSimulator in step (). To make it thread-safe, I implemented a task list, that is worked off before starting the simulation. External, user-triggered threads can add tasks (like adding or removing an object or changing its size or position), which are then brought to the simulation in an orderly manner and noted.

	Open Dynamics Engine (ODE)	**PhysX SDK**	**Newton Game Dynamics**	**Physics2D.NET**
Author	Russell Smith	Ageia Technologies Inc.	Julio Jerez	Jono Porter
License	BSD License	Free for non-commercial use	Free	MIT License
Current Version	0.8	2.7	1.5.3	1.4
Language	C	C++	C	C#
Managed code	via .NET Wrapper	via .NET Wrapper	via .NET Wrapper	native
Documentation	Detailed user guide for version 0.5, Wiki and mailing list	Very detailed user guide and tutorials, commercial support	Detailed documentation, Wiki	Comments in code
Computational effort	Middle	Low (can be cut via additional PhysX hardware)	Middle	High
2D physics	via Joints	native support	via Joints	native

Table 5.1: Final candidates for the choice of physics engine

- While fixing objects to a two-dimensional plane was easy, using the built-in FrozenRot*X* and FrozenPos*X* flags, making them stick to a certain section of the plane was not as simple. To connect two physical actors, PhysX provides so-called joints that are available in several types. The PointInPlaneJoint is used to attach a point of one actor to a certain plane. It is also possible to limit this plane's dimensions, giving it fixed borders. Using this joint *flux* connects objects to their corresponding Views, but the objects are able to jut out beyond the Views' borders, because only their centers are fixed to it and the application additionally has to keep track of size changes in the View and update the limits of the joint accordingly.

- Originally, I planned on implementing clusters as irregular "blankets" on which contained objects would lie and which would also work like their physical equivalents in making it possible for the user to split a cluster into two by ripping the two halves apart. Unfortunately, such interactions were not possible with the .NET-wrapper, so I dropped it and made the clusters completely independent from the physics.
  Another alternative version of a type of visible object was implemented, however, and then dropped because of other issues: Visible piles were at one stage no single boxes within the physics simulation but a number of elastic joints that connected each photo to its neighbours in the stack. By picking and dragging a photo, all others followed automatically and the pile could even be split by holding two neighbouring photos and dragging them apart until the joint broke. While they were neat to watch, those piles also caused a lot of calculations, making a performance leak and nullifying the advantage of piles, namely reducing the complexity for system and user. Another problem was the number of maximum piles: PhysX can distinguish internally up to 128 groups of objects that do not collide, which would have meant a total maximum of 128 piles for all views. So, piles were created in their box form.

- To increase the friction between objects I had to increase their elongation along the Z-axis.

PhysX's internal solver has problems calculating collisions between two very flat objects (the solver works in three dimensions) and assigns them, if any, only a very small friction, which led to a constant permeating of objects. By increasing the depth of the simulated versions of objects I could give them a more stable feel.

5.4.3 Automated Feature Extraction and Quality analysis

Flux demanded, as a system based on the concepts of Informed Browsing, the means to automatically categorize photos based on their "similarity". Now, current systems providing such categorizations mainly depend on low-level features like colours or edges (an overview can be found here [15]) which makes it difficult to emulate the human notion of it, which more relies on high-level features such as contained objects or even the message of the photo ([2], [44]).
For the automatic feature extraction from photos I relied on a software called FeatureExtractor provided by Peter Kunath and Alexey Pryakhin from the LMU's Database and Information Systems research group, which produces 52 different low-level features (for a full list see Appendix C).

Feature Extraction Finding the best combination and weight of these features turned out to be a challenge. A few points were clear: The more features are considered the more difficult it would be to group photos, because even photos that are highly similar in one regard might differ hugely in another, thus increasing the likelihood of a separation for each additional feature. Secondly, I gathered in some tests that colour descriptors were especially important when searching for similar photos.
Using tips from my tutor and Peter and Alexey, I finally decided on the following features all incorporated in equal measure: Color histogram in the YUV colour space, mean, standard deviation, skewness and kurtosis of the colour moment in YUV, the histogram of the gray values and the Haralick texture features 5 and 12. This composition yielded good results and was used in the final version to realize similarity measuring between photos.
One downside of the system is the required calculation time: It takes about half a minute to extract all low-level features from a 4-megapixels-photo on a current PC, which made it impossible to incorporate live extraction in the application, thus preventing on-the-fly import of images. Before using photos with *flux*, they have to be extracted manually.

Building a configuration file The first step to build a new configuration file (or gather the needed metadata for a number of photos) is launching FeatureExtractor. It takes one directory with JPEG-files, extracts low-level features and saves the results to 52 ARFF-files. The Attribute-Relation File Format is a file type that was introduced by WEKA, a software for data mining via machine learning methods ([88]). Its purpose is to save a number of data entries with a fixed number of typed attributes.
ConfigMaker takes one or more of these ARFF-files (in our case the corresponding files for the above features) and uses the contained information to calculate the distance values for each pair of photos. Each low-level feature can be seen as a high-dimensional vector space with each photo having a position within (as an example: the YUV colour space has 36 values/dimensions and each photo is described by a 36-dimensional vector). The distance between two images is determined simply by calculating the Euclidean norm of the distance vector. All distances form one distance matrix, which is then normalized.
To combine multiple features, a distance matrix is produced for each and then the distances are simply added. To let a feature weigh more heavily, its normalized values (which lies between 0 and 1) can be multiplied with a fixed factor before adding them to the rest.
At first, I experimented with OPTICS ([3]), a system for the automatic extraction of hierarchical clusters from a given data set, but it proved to be not only difficult to integrate in a .NET-setting

(OPTICS was written in Java), but also to perform slowly for a greater number of photos because of its average case performance of O(n^2). I dropped OPTICS and used the final normalized distance matrix instead, which contained all distance values directly. ConfigMaker takes these values, additionally gathers the time the photo was taken using its Exif-metadata (see below) and saves all results in a configuration file.

Quality analysis Quality analysis is performed by another system I got from Alexey Pryakhin. This software uses Support Vector Machines (SVMs) to classify objects in high-dimensional vector spaces. It is trained by a number of examples and tries to mimic its patterns for other data with the same features. I use a set of twenty images and two features (namely, the R-t (roughness value for the total height of the profile) and Haralick texture feature 12) to train the algorithm and let it find blurred and out of focus photos. Unfortunately, it only allows an absolute classification of either 0 (bad quality) or 1 (good quality), allocating borderline cases to either the one or the other. Results from the quality analysis are saved in the configuration file by ConfigMaker as well.

5.4.4 Other technology

Exif-Extractor The Exchangeable Image File Format is a standard for metadata in photos made by digital cameras. It contains among other things information about the camera type, its manufacturer, exposure time, focal length, color space and the time and date the photo was taken. This information is much more reliable than the file system's in determining a photo's age.
Microsoft provides access to Exif-data in the .NET framework, but has no automatic typing or labeling of values which is a constant source of errors. So I used the Goheer EXIFExtractor ([63]), that takes a Bitmap-object and allows easy accessing of Exif values by their names.
ConfigMaker uses the "Date Time"-tag, creates a new DateTime-object from it and saves the ticks (i.e., the number of milliseconds since the .NET epoch) to the configuration file.

Microsoft Tablet PC-SDK The names of VisibleClusters can be changed by their marking menus. The Overlay shows a text field, where the user can write and end the text input with the touch of a button. The system then tries to gather meaning from the writing and sets the cluster's name accordingly.
The handwriting recognition is performed by Microsoft's Tablet PC Software Development Kit. One feature of Tablet PCs is their ability to be used without a keyboard by simply writing on their surface. The algorithms performed by Windows XP Tablet PC Edition to achieve that were released as the Tablet PC-SDK ([78]). In combination with a recognizer pack, the SDK provides simple, fairly accurate recognition of written words.

5.5 Selected algorithms

In this section, I present several algorithms that I used in more detail.

5.5.1 Moving objects

Two main techniques are used to move different objects in *flux*: One is the one-touch Rotate-and-Translate and the other the two-touch Rotate-and-Scale. Both work on photos, piles and views, making them consistent beyond object-boundaries.

Rotate-and-Translate The Rotate-and-Translate (RNT) technique is used (as described above 4.5.1) to give interaction with objects a more natural feeling.
RNT uses three points (see figure 5.13) in determining of the outcome of the user's interaction with the object: O is the position the user touched the object in the last step, C is the center of

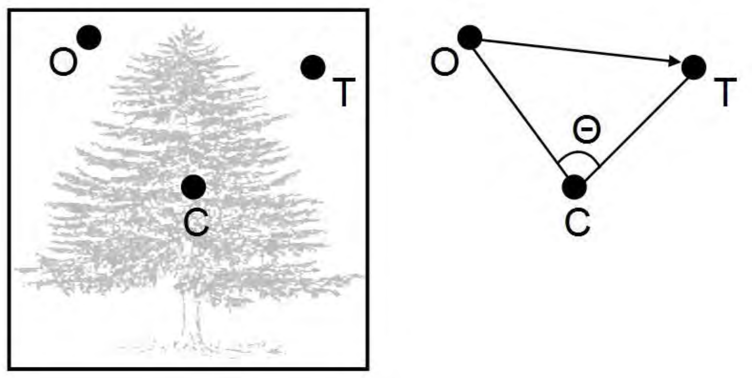


Figure 5.13: The three relevant points in RNT (taken from [32])

the object and T is the position the user currently points at. During one step of the algorithm, the object is moved by the vector OT, and rotated around C by an angle of θ, which is defined by O, C and T. Finally, T becomes O for the next iteration (cf. [32]).
RNT is useful to combine two transformations but can become hindering if the user wants to move an object without rotating it, so Kruger et al. declared the center region of the object "translate-only", meaning the object is only translated if the user touches it there. So, if the user newly touches an object, the input's distance to the center is calculated and if it lies below a certain threshold (in our case one-fourth of the object's width) the translate-only mode is activated. Here, the object is not rotated around C, but only moved by OT.

Rotate-and-Scale The second kind of interacting with objects again combines two transformations in one gesture and appears in several applications that support more than one input point (examples are Jeff Han's legendary TED Demo ([87]) and recently the iPhone ([62])).
Rotate-and-Scale (RNS) is actually a quite simple algorithm which is used if the user touches the object at two points (P1 and P2) (see figure 5.14). The basic idea is to adjust the rotation and the scale of the object if the user moves an input to match the original spatial relation between the points and the object. That input events are processed sequentially helps in reducing the algorithm's complexity: Only one point can be moved at any time, so RNS takes three parameters: The static point (P2), the point before its movement (P1) and afterwards (T1).
First, the object is rotated by the angle defined by P1, P2 and T1 θ around P2. Then the object's scale is changed by $factor$, which is the ratio between the distance between T1 and P2 and P1 and P2.

5.5.2 Metaballs

Metaballs is the common name for blob-shaped, two- or three-dimensional objects in computer graphics. They were first described by James Blinn in 1982 ([8]), but soon became a common sight in computer demos and were also used for example in Brad Paley's Mind'Space installation ([81]).
A metaball is an n-dimensional function whose value at a certain point can be used to decide whether the point lies within or outside of the boundaries of the metaball system. If a point within a common, two-dimensional metaball system will be filled can be determined by the following equation:

$$\sum_{k=0}^{n} \frac{s_k}{|m_k - p|^g} > r \qquad (1)$$

with n being the number of metaballs, p being the point's position, m_k being the kth metaball's center and s_k its size, g being the "goo"-factor of the metaballs, i.e., how sticky the metaballs are

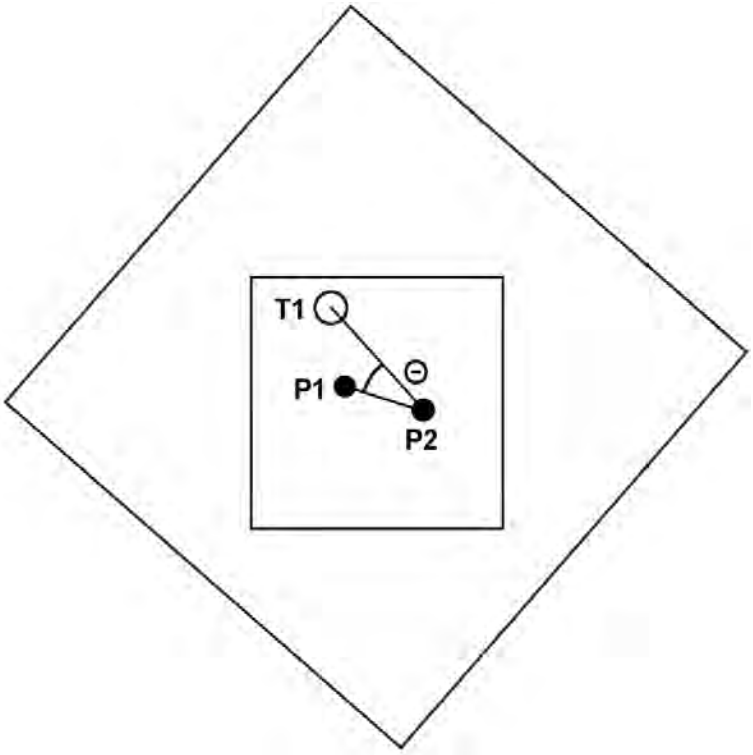


Figure 5.14: The three relevant points of RNS

and r being a value specifying the observer's "height" within the field.
A trivial rendering of metaballs can be performed by taking a rectangle of screen space and solving the above equation for each pixel to decide whether it is filled or not. This approach is unfortunately quite computationally expensive.
A more sophisticated way of doing it is via Marching Cubes ([34]) (or Marching Squares, as the algorithm's 2D version is called) that is able to find the contour of an object in space. It works by taking eight positions (four in the 2D case) that form a cube and checking for each of its vertices if it lies within or outside of the shape. By using symmetries the actual 256 cases can be reduced to 14, which can be saved within a lookup table to quickly generate polygons for the surface. To perform illumination on the surface the normals of these polygons can be calculated as well. Marching Cubes finds the surface for a arbitrary scalar field and thus can be used for metaballs as well.
I finally decided to use an algorithm which is similar to Marching Squares and can be found here [73].

It is based on the above equation and takes advantage of the metaball field being a scalar field, which means that the normal at any point can be simply described by the gradient (i.e., a vector made up of the partial derivatives of every dimension):

$$normal = \nabla \sum_{k=0}^{n} \frac{s_k}{|m_k - p|^g} \tag{2}$$

or

$$normal = \sum_{k=0}^{n} -g * s_k \frac{m_k - p}{|m_k - p|^{2+g}} \tag{3}$$

The algorithm works as follows:

1. We start at the center position of any metaball and try to find the border of the metaball by moving along the normal from the above equation. It is possible to use a fixed step size and

reach the border after a small amount of steps. The process can be accelerated, however, because the force within the metaballs system is inversely proportional to the distance from the center of the metaball to the power of g ($f(x) = \frac{s_{min}}{x^g}$ with s_{min} being the smallest metaball in the system). So, the step size can be derived from these three equations:

$$f(p) = force \tag{4}$$

$$f(border) = r \tag{5}$$

$$p + stepsize = border \tag{6}$$

By modifying the last one we get:

$$stepsize = \frac{s_{min}}{r}^{\frac{1}{g}} - \frac{s_{min}}{force}^{\frac{1}{g}} \tag{7}$$

We cannot use this step size directly, because it is an asymptotic approximation and reaches the border after an infinite amount of steps. By adding a small constant (e.g., 0.01) we can overcome this problem.
Another problem is the possibility that the tracker gets stuck in a local minimum that is larger than r and continues indefinitely. In my implementation, trackTheBorder saves all tried positions and stops, as soon as it lands at an already visited one. The corresponding metaball is then dropped and I rely on the others.

2. After finding the border, we can follow it by stepping along the tangents. The tangent is orthogonal to the normal and has therefore a value of

$$tangent_x = -normal_y \tag{8}$$

$$tangent_y = normal_x \tag{9}$$

The step size can be adjusted by the user via parameter to control the quality (*flux* uses a value of 10). Stepping along the border is stopped if we reach our starting point again.

3. Repeat until all metaballs have finished their tracking.

This algorithm runs at acceptable times even for larger numbers of metaballs. I use it to calculate the border of a VisibleCluster and improve its runtime by using only one metaball for a pile of photos.

5.5.3 Arrangements

The four different ArrangeContents (see figure 5.15) all share the same basic structure: First, all affected objects (all objects directly within the View) are identified, then the horizontally and vertically available space is determined and split among the objects based on its weight (number of photos). Next, piles are built from photos until the available space is met. Then photos and piles are placed, followed by contained clusters (that repeat the process within themselves).
Two important methods in these steps are CalculateGrid, that takes the available space and builds a grid from it, and BuildPiles that forms piles from photos based on given criteria.

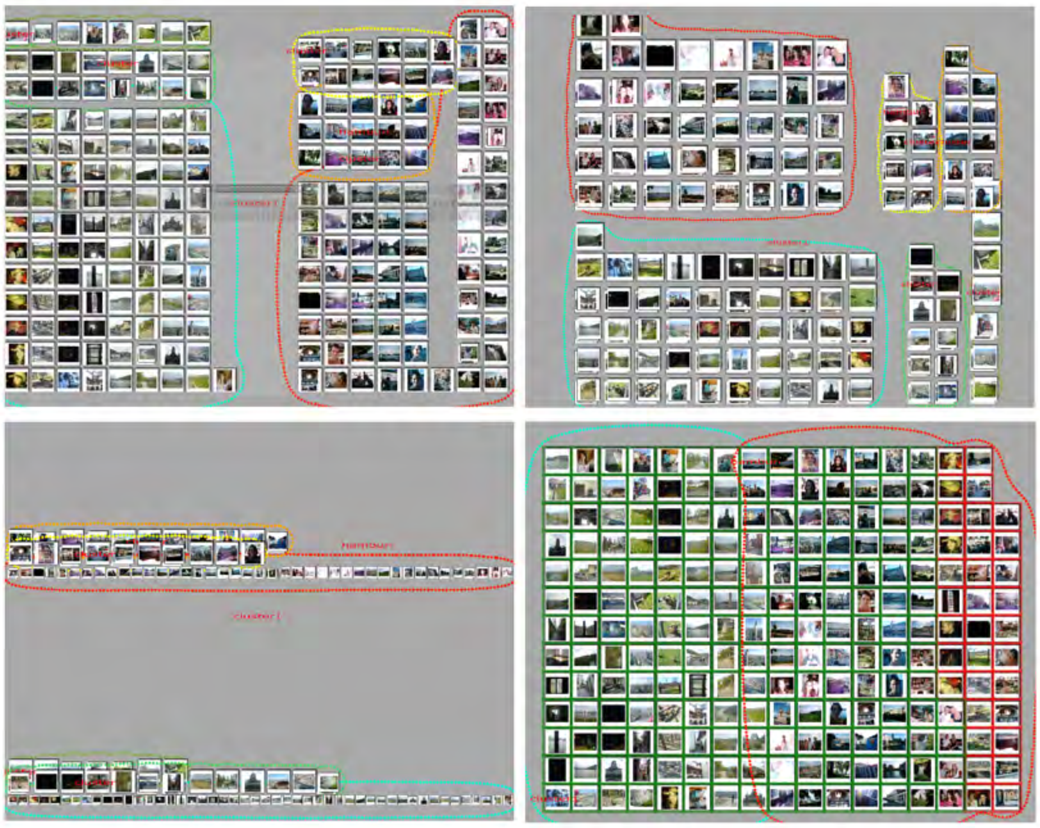


Figure 5.15: The four different arrangement types: Time, Similarity, Stream and Quality

CalculateGrid CalculateGrid takes available horizontal (*width*) and vertical (*height*) space, the desired number of cells (*maxcount*) and the minimal edge length of one cell and returns the possible edge length and the number of horizontal (*boxeshori*) and vertical cells (*boxesverti*).
It starts with $boxeshori = boxesverti = 1$ and sets the edge length to *width*. As long as $boxeshori \times boxesverti < maxcount$, *boxeshori* (or *boxesverti* if $height > width$) is increased and edge length and *boxesverti* (*boxeshori* respectively) are set correspondingly. If a suitable number of cells is found, it is checked if the edge length is still greater than the minimal value. If not, *maxcount* is reduced by one and the whole process repeated.
CalculateGrid returns always at least a one-by-one grid, even if there is actually not enough space available. Based on the number of available cells, ArrangeContents can continue to build piles.

BuildPiles BuildPiles takes a list of photos, the number of available grid cells and a type of order and combines photos to piles until the space is sufficient. Piles can be built either based on time or on similarity, but their workings are analogue: They try to find an especially promising candidate (one with a high number of photos that are in the time or similarity neighbourhood, which is defined by the corresponding threshold) and then add its direct neighbours to the pile until the minimal pile size is reached.

Similarity ArrangeContentsSimilarity arranges the cluster structure as follows: All main clusters are placed vertically, with each getting space according to the number of totally contained photos. Photos and piles are placed horizontally from left to right and subclusters end up on the right side of the main cluster next to each other (they internally repeat this placing method). Clusters in workspaces use the BuildPiles method explained above to build as many piles as necessary. Clusters on the background use a slightly different version to sort photos by similarity. PlaceYour-

selfInGridSimpleSim method is a variation of the standard BuildPiles approach: While the latter aims at using the available screen real estate as well as possible and show as many photos unpiled as fit in the space, the former tries to combine all related photos to piles even if it means that all photos of one cluster collapse into one pile.
PlaceYourselfInGridSimpleSim starts with a maximum similarity threshold and builds piles where all photos' similarity values to one representative exceed this threshold. If no more piles are possible, the threshold is lowered by a certain step and the process is repeated until some lower threshold is reached (both thresholds can be set in the profile). If the photos are not similar enough and the algorithm cannot build enough piles, some overlapping happens (see below 6).

Time ArrangeContentsTime does not aim at utilising the available space fully as well, but tries to let the user recognize temporal patterns in her photo collection (i.e., at which times are the most pictures taken, are there any longer periods of time where no photos are made, etc). That is why it starts by the taking the oldest and youngest photo and separating the space of time between them in equally long sections.
Each of these columns gets the same amount of vertical screen space and they are placed next to one another. All clusters that contain at least one object from the according period of time have to place these objects within the column and are arranged themselves above each other (in the same order as in the Stream- and Similarity-arrangements). Subclusters are placed internally on the top end of a cluster. If photos lie next to each other in the same column they are again sorted by time. The vertical space is divided depending on the number of photos a cluster has, which means that the whole column is filled if there is only one cluster or left empty if none is available. In a normal collection where photos are taken only at irregular intervals but then many of them (burst-like), the Time-arrangement therefore tends to leave many columns empty, which gives the user enough space to enlarge columns she is interested in.

Quality ArrangeContentsQuality ignores all cluster-borders and treats all photos equally. A grid the size of the View is constructed and filled with photos column-wise from left to right and depending on the quality of the photos. Though all photos only have a quality of either good or bad, the system arranges them from highest to lowest quality (which means that with a different quality recognition process that rates a photo more flexibly ArrangeContentsQuality would still work). Additionally, all photos display a border which is either green or red, giving the user a quick overview.
Because all photos are arranged next to each other regardless of their cluster affiliations, this order is more useful for a workspace that only contains a fraction than the background with the whole collection.

Stream The last type of order, Stream, combines the benefits of Similarity and Time and arranges all main clusters vertically above one another. It does not work with single photos which makes it most useful on the background. Every main cluster first checks that all single photos are sorted by time, then goes through them and builds a new pile every time the similarity value between one photo and the next lies below a certain threshold. All such piles are placed from left to right. Subclusters are placed on top of the main cluster and perform the same process.

6 Limits and criticism

In the following, I will present the shortcomings of my implementation of *flux*, by first describing all features that were dropped in the last minute, then known bugs that still exist within the application followed by an examination whether the requirements and the original goals were reached.

6.1 Missing features

- **Copying of items between clusters**:
  It is not possible in the current implementation to copy an object from one cluster to another. Moving is possible by forming a new cluster that contains the old target cluster and the object (the cluster is dissolved and all objects are put into a new one).

- **Pile interaction**:
  Piles can be dissolved in two ways, either by waiting two timeouts or touching them with two fingers. The two-finger fold/unfold gesture was not implemented.

- **Deleting photos**:
  Together with the Undo-feature, the deletion of photos was dropped as well. It would not have been possible to make the removal undone in another way and I did not want to add some substitute interaction (e.g., pressing a key on the keyboard). With the SmartBoard's inaccurate recognition, deleting a photo with one gesture alone and without a way to take it back would have certainly led to user frustration.
  A way to perform selecting anyway is creating a new cluster from photos than can be deleted.

- **Consistency of changes**:
  While the cluster structure is imported from the configuration file, the application does not paste changes back to it. So all global manipulation is lost after quitting *flux*.

- **Background rotation is deactivated**:
  Because the SmartBoard tends to have problems with two input sources, it produces many incorrect recognition events in this case, which normally leads to erroneous interaction. In testing I discovered that an unintentional rotation of the background happened quite often and was very distracting, so I disabled the feature.

- **No rotational independence for text field**:
  Entering the name of a cluster is done via a text field that can be activated with the cluster's marking menu. Unfortunately, this text field is always aligned towards one side of the screen and the user has no way to rotate it. So entering the name of a cluster is only (conveniently) possible from one side of the table.

6.2 Known bugs

Several errors still exist within the implementation but the situations where they appear can be circumvented. Mostly, they are just nuisances; all the critical bugs that rendered the program unusable have already been removed.

- **Joints break when scaling workspaces**
  When scaling a workspace sometimes the internal joints, with which the objects are bound to the workspace break and all objects lapse into jittering motion. While this cannot be prevented and seems to happen randomly, the joints can be fixed again by carefully scaling the workspace. Another solution is creating a new workspace from all contained objects with the Lasso-gesture and closing the old one.

- **Dissolving piles on a workspace can lead to photos jamming**
  When dissolving a pile on a workspace (it is not possible on the background) the contained photos are rescaled and placed in a ring around the position of the pile. Most of the time they simply stay this way, but sometimes two or more photos jam and start a slow rotation, because the physics engine tries to resolve the collision but cannot. A simple way to separate the objects again is to grab to foremost and move it swiftly back and forth. The inertia usually divides the objects again.

- **Quality analysis**
  The results of the quality analysis are not satisfying and it is common that acceptable photos are recognized as poor quality and vice versa.

- **Photos disappear when dissolving a cluster**
  When forming a new subcluster, changing the arrangement on the view and dissolving the cluster again the contained photos disappear. This bug can be resolved by changing the view's arrangement again, so the photos appear within the parent cluster.

- **Visual collisions when arranging objects**
  Certain configurations of objects, minimal photo sizes and clusters can lead to one row or column of photos lying outside of the borders of the view after changing its arrangement. Additionally, clusters can overlap in the Similarity-arrangement.

- **Manipulation of time columns in the time-arrangement**
  The user is able to move the left border of the leftmost time column and the right border of the rightmost time column towards the middle, so the screen space is no longer optimally used. All objects are, however, still shown and by re-arranging all objects or repositioning the column the space can be used again.

- **Loading time of XML-file**
  Because of its being in plain text and additionally containing tags, the configuration file can become quite large and thus take some time to load when launching the application. Similarity distances between photos are generated beforehand and saved for each photo, so the file size could be about halved by saving distances only in one photo of a pair.

- **Insufficient unfolding**
  When pulling open a pile on the background, the user has to make sure that the photos are spread far enough so the original pile is still reachable to be able to close the pile copy again. If not, the user has to move all occluding photos out of the way manually. An easy solution would be to let the application check whether the original pile is still reachable before spreading out the photos.

- **Application crashes when enlarging too many photos**
  The application reloads all photos that are scaled large enough in a higher quality version. So, if the user has enlarged too many photos *flux* can crash as soon as the texture memory is filled.

- **Application becomes slower when many photos are shown**
  The higher the number of shown photos, the slower the application becomes. Working with it is at the moment possible with up to around 1000 photos, even though the underlying concepts allow (and were planned for) a lot more. The main reason for it is the missing support of hardware acceleration: Although the whole interface is built in Direct3D, running its rendering on the graphics card is considerably slower than using a pure software mode, more or less the complete contrary to what one would expect. The error lies probably within one of the drawing routines of the visual objects and finding it would most certainly

increase the system's speed (and subsequently the number of maximal photos). Other speed optimizations like displaying only visible photos in a pile that are not completely hidden by others or recalculating the clusters' metaball borders only if a contained object is moved farther than a certain threshold would probably allow the system to display and let the user work with more than 5000 photos.

6.3 Evaluation of the fulfillment of the requirements

In the following, I will describe for each of the defined requirements in how far the implemented version of *flux* fulfills it and what may be missing.

6.3.1 Support the photowork tasks

All four *photowork* tasks are supported by the system. In detail:

- Deliberate *filing* is possible with the hierarchical cluster structures with changeable names and colours.
- *Selecting* is not yet supported to a full extent, because photos cannot be deleted. But letting the system sort them quality-wise and building clusters to separate higher and lower quality photos is possible.
- *Sharing* of photos in the sense of sharing visually is supported very well with quick gestures for rotating and enlarging photos. Copying photos between different clusters/streams cannot be performed.
- *Browsing* is an essential part of *flux* and is supported on multiple levels. The background view shows after changing its arrangement an overview of the whole collection as a starting point for a browsing process. Opening piles, changing the size of time columns in the Time order or copying objects to a new workspace lets the user access different parts of the collection, which can then be treated the same way or browsed linearly.

6.3.2 Support multiple concurrent users

Due to the hardware restrictions, having more than two users at a time makes not much sense because concurrent work is hardly possible with only two input points. It is already difficult for only two users to interact with the system without thwarting the other unintentionally, because two finger actions like scaling photos or workspaces are commonplace. But while concurrent use is unfavourable, collaboration (e.g., while sharing) works well, because then the users are paying more attention towards one another and are also communicating most of the time, so possible conflicts can be anticipated and managed with social protocols.
Every user can bring her own collection to the system by introducing it as an own stream (streams have to be created using the configuration file), but photos cannot be copied from one stream to another.

6.3.3 Work with the characteristics of the hardware

Flux makes extensive use of the touching capabilities of the SmartBoard as every interaction is performed using direct touch. Keyboard or mouse input is not necessary (and only possible for debugging operations). Objects are manipulated by touching them and multiple objects are selected by drawing a circle around them, so buttons etc. are not necessary. The only additional interface elements are marking menus and information carriers like borders of clusters or the time line in the Time-order (which can be manipulated as well).

The whole interface is rotationally independent and can be used from any side of the table (with the above mentioned exception of the text field for renaming clusters), because all objects can easily be rotated.

6.3.4 Support a regular-sized photo collection

Giving the actual design of the application, supporting even large collections is possible. The automatic piling of photos leads to a situation where even large numbers of objects can be visualized and still handled well. Unfortunately, the current implementation does not provide an acceptable speed for more than a thousand photos (see above), so it could only be proven that the concepts were promising for a few hundred photos.

6.3.5 Prevent the users from getting lost

The following aspects make sure that users do not become lost within their collections: The background that contains all photos is always easily reachable and gives users a clear overview of the collection. Workspaces cannot overlap one another which forces the users to optimize and close unneeded ones (which again improves the clarity of the interface). The options for organization help the users structure their collections and piles make it more concise visually.

6.3.6 Reduce visual clutter

By cutting down on the number of needed interface elements, the application automatically gets a clearer look. Photos, which are the conceptual backbone of it, also become the most prominent visible objects. Additionally, information carriers are very subtly visualized, with a semi-transparent time line in the Time-order and only a chain of small circles to symbolize a cluster. Marking menus only become visible at all if triggered by the user.
One source of visual clutter, the physical movement of objects, has been reduced by the second design to the workspaces, which are only present in some cases (and whose objects only move if being manipulated by the user). Trimming down even more of the interface would only have been possible while at the same time forcing the users to learn more gestures.

7 Summary

In this work I introduced *flux*, the design and implementation of a tabletop photo application. I first described the current situation, where consumers are switching from analogue to digital camera systems and are left without feasible means for organization and physical interaction. After analyzing the things people do with digital photographs, the so-called photowork that was divided into the four actions *Filing*, *Selecting*, *Sharing* and *Browsing*, Informed Browsing was presented.
Informed Browsing represents the idea of lifting the organizational overhead of existing photo solutions from the user and replacing it by a more lightweight, metadata-based structure. This metadata is gathered not only from existing sources like Exif, but also generated with image analysis algorithms for extracting similarity or quality values. Additionally, Informed Browsing is based on interaction techniques from the HCI community, mainly *Overview at all times*, *Details on demand* and *Temporary structures*.
Within the Related Work chapter I presented applications from the fields of desktop photo software (divided into software for amateurs, software from the academic sector and software targeted at professionals), that have advanced methods of organization but lack the ease of communication and the feel of physical photos. They were followed by applications for tabletop displays, that stem from the scientific community and are generally not as technically mature as their commercial counterparts. Still, tabletop photo applications are promising in lowering the borders for interpersonal communication while sharing digital photos and representing those in a way akin to their analogue versions thanks to touch-based manipulation. The last presented type of application was the first implementation of the Informed Browsing idea, AudioRadar, that allows the user to browse her music collection based on the character of songs.
The Design chapter contained the different design iterations coupled with the ideas behind them. From the hardware-setup, via exemplary usage scenarios to concrete requirements the user side was developed. Then the first design iteration was presented in detail, followed by a description of the focus group I performed whose results were reflected in the second and final design.
The Implementation chapter again contained the way the implementation took from the prototype version to the final one. The class structure and the data flow in exemplary situations was shown. Certain aspects of the implementation were described in detail, as well as specific algorithms. Finally, existing technology and the way it was embedded was presented.
In the Limits and criticism chapter I detailed the limits of the application, namely features that were dropped due to time constraints and bugs that could not be fixed and confronted the final implementation with the requirements.

7.1 Comments

With *flux* I created an application that aims at combining the best of two worlds: The organizational capabilities of desktop photo software and the tangibility and multi-user environment of tabletop applications. Additionally, Informed Browsing concepts support the user in organizing the collection, which makes it usable from the start.
Unfortunately, the time constraints only allowed me to barely finish with the implementation and a test with real users and subsequent refinements were not possible. Also missing are optimizations to allow a usage with a realistically sized collection which would have been important to prove my point. Still, the underlying concepts are valid for a few hundred photos and could probably be extrapolated to larger numbers.
But while the organization part is not fully functional, the interaction is promising: Transforming photos works flawlessly and very quickly and sometimes I found myself shoving photos on a workspace just to bump them into other ones. So, the physical feel of the photos and rapid in-

teraction works well which could be a starting point for other tabletop applications that concern themselves with photos.

7.2 Future Work

A first and essential step to improve *flux* would be implementing the optimization methods mentioned above. In the current version it is simply not possible to operate the application with more than a thousand photos which is a pity given the original motivation.
The next step would be to add the features that were dropped during the implementation, to support for example copying of objects between streams to allow advanced sharing and deleting and undoing for the selection of photos.
Additionally, some of the interaction concepts could be modified: Time lines and coupled manipulating capabilities for the Stream-order would be useful, an improved interaction technique for modifying time columns in the Time-order and clearer feedback for the users as well. Also, the different automatic arrangements should be fixed and maybe changed.

To support the user in giving her collection a more complex organization and working more creatively with photos, the dropped Lines, Textblocks and Boxes could be added again or some other, similar features that manage to give the collection another dimension additionally to the hierarchical clusters.
It is also imaginable to provide the users with more freedom regarding the usage of metadata: While it is in the current design only possible to arrange objects by one dimension (e.g., time, quality), a future version could support a "weighted" arrangement that lets the user combine different features and sort objects according to it (for example: Show only the most recent, most high quality photos).

One last point is the "fluidity" of the interface: With the removal of the physics engine the interface became much calmer, but it also lost a lot of its appeal. Some kind of independent photo movement could be added, probably in the form of an attraction between similar objects, to make the interface tidy up itself. This movement of objects should of course be so slow that it does not pose as a source of visual clutter, but fast enough that it yields benefit for the user. It could probably be effectively coupled with some measure of the need for room on the interface: If there are many large objects (e.g., because the user just enlarged a photo) the movement speed of the photos grows, while it shrinks as soon as enough piles were formed and the interface is no longer crowded.

A Focus group script

Here the complete script I prepared for and used during the focus group. It was left in German.

1.Einleitung

1.1. Kurze Vorstellung Dominikus (Diplomarbeit, Verbindung zu Fotos)
1.2.Als Einführung ins Thema: Video von PhotoHelix
1.3.Vorstellung der einzelnen Teilnehmer. Insbesondere: Allgemeines wie Name und Beruf, dann Verbindung zu Fotos (beruflich / privat, woher stammt etwaige Faszination etc.) und Computern (PC/Mac)

2.Fotos
2.1.Nochmal Runde: Diesmal speziell auf Fotonutzung bezogen. Jeder soll angeben, was für Kameras (digital, analog) er benutzt, wieviele analoge/digitale Fotos er etwa hat, welches Medium (analog/digital) er präferiert und warum
2.2.Damit das ganze nicht wieder zu einer Runde degeneriert: Mit Meldung: Wie genau werden Fotos organisiert? Zuerst für analoge Fotosammlungen, dann für digitale.
2.2.1.Analog: Schuhkarton? Alben? Oder etwas ganz anderes?
2.2.2.Digital:
2.2.2.1.Werden Fotos auf PC/Mac übertragen? Mit einer speziellen Software oder mit dem Betriebssystem?
2.2.2.2.Wie wird verwaltet? Alles in einen einzigen Ordner? Ordnerschema? Innerhalb einer speziellen Fotosoftware (Picasa, iPhoto)?
2.2.2.3.Werden Fotodateien umbenannt, vielleicht nach einem speziellen Namensschema? Werden Tags benutzt? Onlineportale (flickr)?
2.3. Ab jetzt nur noch Fragen zu digitalen Fotos.
2.3.1.Wie finden die Teilnehmer Fotos wieder? Nutzen Sie die Desktopsuche (OS-intern, Google Desktop Search, o.ä.), vielleicht in Verbindung mit einer Namenskonvention? Oder hangeln sie sich entlang einer Ordnerstruktur?
2.3.2.Was passiert, wenn sie den genauen Namen eines Fotos vergessen haben?
2.3.3.Oder suchen sie manchmal auch einfach so, ohne festes Ziel oder nur mit einer vagen Idee (ein Weihnachtsfoto, ein Urlaubsfoto)? Wie läuft das dann ab?
2.3.4.Wann genau und warum wird auf Fotos wieder zugegriffen?
2.3.4.1.Bzw. was passiert mit den Fotos, wenn sie einmal auf der Festplatte liegen?
2.3.4.2.Werden sie Familie und/oder Freundin gezeigt? Wenn ja, wie? Hinter dem Bildschirm? Mit einem Beamer? Oder ausgedruckt? Oder entwickelt als Dias?
2.3.4.3.Wird die Sammlung oft alleine/mit mehreren durchgegangen? Wenn ja, warum?
2.3.4.4.Stöbern die Teilnehmer ab und zu in ihrer eigenen Sammlung? Wie sieht das aus, wie gehen sie vor? Hilft ihre etwaige Organisationsstruktur/Fotosoftware?
2.3.4.5.Hand auf's Herz: Wieviele Fotos liegen irgendwo in einem vergessenen Ordner und werden nie mehr angeschaut?
2.3.5.Werden Fotos gelöscht?
2.3.5.1.Warum? Schlechte Qualität? Missfallen? Anschlußfrage: Werden mehrere Fotos von einem Motiv gemacht?
2.3.5.2.Wann? Direkt auf der Kamera? Erst daheim auf dem Computer?
2.3.5.3.Wie? Alle Fotos als Thumbnails nebeneinander? Oder werden sie linear durchgegangen?
2.3.6.Werden Fotos nachbearbeitet?
2.3.6.1.Wenn ja, womit? Einfache, integrierte Möglichkeiten in Picasa oder iPhoto? Oder mit komplexerer Software wie Photoshop?

2.3.6.2.Was genau wird gemacht? Kleinigkeiten wie Unschärfe, Farben? Oder auch größere Sachen wie Mash-Ups etc.?
2.3.6.3.In welchem Zahlenverhältnis stehen bearbeitete zu nicht-bearbeiteten Fotos?

3.Photowork und Informed Browsing
3.1.Gemeinsame Entwicklung des Photowork-Konzepts:
3.2.Entwickeln und Erklären des Diagramms am Whiteboard.
3.3.Diskussion mit den Teilnehmern:
3.3.1.Ist dieses Modell sinnvoll und komplett? Können sie sich darin wiederfinden?
3.3.2.Welchen Weg nehmen sie normalerweise durch diesen Graph?
3.4.Zusammenfassen der Tätigkeiten auf die grundlegenden vier:
3.4.1.Filing
3.4.2.Selecting
3.4.3.Browsing
3.4.4.Sharing
3.5.Diskussion: Passt diese Reduzierung? Wurde etwas vergessen?
3.6.Übergang zu Informed Browsing: Schwerpunkt auf Browsing, Verquickung mit automatischer Analyse
3.6.1.Gemeinsames Erarbeiten am Whiteboard: Welche Möglichkeiten der Automatischen Analyse gibt es (Qualität, Farbe, Textur, Kanten, Geometrische Formen, Gesichtserkennung)? Welche könnten sich die Teilnehmer vorstellen? Evtl. wie gut funktionieren sie im Moment?
3.6.2.Nutzung in IB: Reduktion der Komplexität, Erhöhen der Übersichtlichkeit. Mechaniken versuchen den Benutzer bei seiner Tätigkeit zu unterstützen.
3.6.3.Nutzer hat zu jeder Zeit vollen Überblick (overview at all times), damit keine Fotos verloren gehen
3.6.4.kann aber auch ins Detail gehen (details on demand), um einen bestimmten Bereich herauszugreifen oder zu vergrößern
3.6.5.und kann temporäre Strukturen erzeugen ohne die globale Ordnung durcheinander zu bringen
3.7.Diskussion:
3.7.1.Was halten die Teilnehmer davon? Ist das ein vernünftiger Weg? Wurde etwas vergessen?
3.7.2.Möchte man so etwas? Wie sehr soll der Computer eingreifen dürfen? Soll er möglicherweise die komplette Organisation übernehmen?

4.Flux
4.1.Konzeptzeichnung: Flüsse
4.2.Theoretische Vorstellung von flux (Abgewandelte Version des Oberseminarvortrags)
4.3.Dann: Gemeinsamer Gang in den Keller, einmal Demo durch Dominikus, dann dürfen die Teilnehmer damit herumspielen.
4.4.Kurze Pause danach, falls gewünscht
4.5.Diskussion (wieder im 5.Stock):
4.5.1.Meinungen? War's gut oder schlecht?
4.5.2.Reihum: Was hat besonders gut gefallen? (von jedem mindestens ein Punkt)
4.5.3.Reihum: Was war nicht so durchdacht? (von jedem mindestens ein Punkt)
4.5.4.Wie wird die automatische Analyse integriert? Wie können sich die Teilnehmer vorstellen, bestimmte Features zu- und abzuschalten? Halten sie diese in diesem Kontext überhaupt für vernünftig?
4.5.4.1.Marking Menus
4.5.4.2.Interface Currents
4.5.4.3.Gestenbasiert
4.5.4.4.Timer

4.5.4.5.Buttons
4.5.5.Was fehlt ihnen noch? Wie könnte man das System besser machen? Wurde ein gravierender Punkt vergessen (auch im Hinblick auf den vorher erarbeiteten Fotoprozess)?
4.5.6.Analogie zu analogen Fotos: Mit Freunden um den Tisch sitzen
4.5.7.Würden die Teilnehmer es benutzen, wenn sie so einen Tisch zuhause stehen hätten? Wenn ja, warum? Wenn nein, warum und was müsste man ändern, damit sie es nutzen würden?
4.5.8.Würde sie ein solches System bei ihrem persönlichen Fotoprozess unterstützen?

5.Zusammenfassung und Fazit
5.1.Die Zukunft: Jeder Teilnehmer darf sich einmal laut ausmalen, wie er sich das Arbeiten mit Fotos in zehn Jahren vorstellt. Tische? Handys? Tragbare Projektoren und Interaktion mit Wänden? Sprachsteuerung? Gedankensteuerung?
5.2.Nochmal reihum: Letzte Worte zu flux. Was war gut, was nicht.
5.3.Zusammenfassung durch Dominikus. Bedanken.

B Example for a flux-configuration file

```
<flux>
  <cluster>
    <name>cluster1</name>
    <color r="255" g="0" b="0" />
    <contains>
   <photo>
    <filename>D:\Flux\Photos/DSC00469.JPG</filename>
    <date>633113090430000000</date>
    <quality>0</quality>
    <deriv>0</deriv>
    <similarity>
      <distance>
        <other>D:\Flux\Photos/DSC00470.JPG</other>
        <value>0,924945705494861</value>
      </distance>
      <distance>
        <other>D:\Flux\Photos/DSC00471.JPG</other>
        <value>0,787342776603649</value>
      </distance>
    </similarity>
  </photo>
  <photo>
    <filename>D:\Flux\Photos/DSC00470.JPG</filename>
    <date>633113090590000000</date>
    <quality>1</quality>
    <deriv>-0,359114399381951</deriv>
    <similarity>
      <distance>
        <other>D:\Flux\Photos/DSC00469.JPG</other>
        <value>0,924945705494861</value>
      </distance>
      <distance>
        <other>D:\Flux\Photos/DSC00471.JPG</other>
        <value>0,80589289021725</value>
      </distance>
    </similarity>
  </photo>
  <photo>
    <filename>D:\Flux\Photos/DSC00471.JPG</filename>
    <date>633113090730000000</date>
    <quality>0</quality>
    <deriv>0,47776401216256</deriv>
    <similarity>
      <distance>
        <other>D:\Flux\Photos/DSC00470.JPG</other>
        <value>0,80589289021725</value>
      </distance>
      <distance>
        <other>D:\Flux\Photos/DSC00469.JPG</other>
        <value>0,787342776603649</value>
      </distance>
    </similarity>
  </photo>
  </contains>
  </cluster>
</flux>
```

C List of features extracted by FeatureExtractor

```
grayhistogram.arff - Gray histogram
colorhistogram_hsv.arff - Color histogram HSV
colorhistogram_hsl.arff - Color histogram HSL
colorhistogram_yuv.arff - Color histogram YUV
colormoment_hsv_mean.arff - Color histogram HSV Mean
colormoment_hsv_stdd.arff - Color histogram HSV Standard deviation
colormoment_hsv_skew.arff - Color histogram HSV Skewness
colormoment_hsv_kurt.arff - Color histogram HSV Kurtosis
colormoment_hsl_mean.arff - Color histogram HSL Mean
colormoment_hsl_stdd.arff - Color histogram HSL Standard deviation
colormoment_hsl_skew.arff - Color histogram HSL Skewness
colormoment_hsl_kurt.arff - Color histogram HSL Kurtosis
colormoment_yuv_mean.arff - Color histogram YUV Mean
colormoment_yuv_stdd.arff - Color histogram YUV Standard deviation
colormoment_yuv_skew.arff - Color histogram YUV Skewness
colormoment_yuv_kurt.arff - Color histogram YUV Kurtosis
haralick_01.arff - Haralick  1
haralick_02.arff - Haralick  2
haralick_03.arff - Haralick  3
haralick_04.arff - Haralick  4
haralick_05.arff - Haralick  5
haralick_06.arff - Haralick  6
haralick_07.arff - Haralick  7
haralick_08.arff - Haralick  8
haralick_09.arff - Haralick  9
haralick_10.arff - Haralick 10
haralick_11.arff - Haralick 11
haralick_12.arff - Haralick 12
haralick_13.arff - Haralick 13
roughness_rv.arff - Roughness Lowest valley
roughness_rp.arff - Roughness Highest peak
roughness_rt.arff - Roughness Total height of profile
roughness_rm.arff - Roughness Mean deviation
roughness_ra.arff - Roughness Arithmetic mean deviation
roughness_rq.arff - Roughness Root mean square deviation
roughness_rsk.arff - Roughness Skewness
roughness_rku.arff - Roughness Kurtosis
facet_min.arff - Facet Lowest
facet_max.arff - Facet Highest
facet_med.arff - Facet Median
facet_mean.arff - Facet Mean
facet_stdd.arff - Facet Standard deviation
facet_skew.arff - Facet Skewness
facet_kurt.arff - Facet Kurtosis
facet_area.arff - Facet Surface Area
polar_min.arff - Polar Angle Lowest
polar_max.arff - Polar Angle Highest
polar_med.arff - Polar Angle Median
polar_mean.arff - Polar Angle Mean
```

```
polar_stdd.arff - Polar Angle Standard deviation
polar_skew.arff - Polar Angle Skewness
polar_kurt.arff - Polar Angle Kurtosis
```

List of Figures

List of Tables

Inhalt der beigelegten CD

```
\Ausarbeitung
\LaTex
Enthält die Ausarbeitung im LaTex Format
\PDF
Enthält die Ausarbeitung im PDF Format

\Folien
Enthält die Folien des Oberseminar Vortrags

\Flux
\Focus-Prototype
Enthält den Prototyp, der in der Fokusgruppe gezeigt wurde
\Prototype
Enthält den in Kapitel 5.2 beschriebenen Prototype
\Final
Enthält die endgültige Version
\Tools
Enthält Hilfsprogramme wie ConfigMaker und FeatureExtractor

\Quellen
Enthält Quellen im PDF Format (soweit verfügbar)

Readme.txt
Kurzes HowTo zum Start von Flux
```

References

[1] A. Agarawala, R. Balakrishnan. Keepin' it real: pushing the desktop metaphor with physics, piles and the pen. In: CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems, Montreal, Quebec, Canada, 2006, ACM Press, New York, 2006, 1283 - 1292.

[2] R. U. Akeret. Photolanguage: How Photos Reveal the Fascinating Stories of Our Lives and Relationships. Norton, New York, 2000.

[3] M. Ankerst, M. M. Breunig, H. P. Kriegel, J. Sander. Optics: Ordering Points To Identify The Clustering Structure. In: SIGMOD '99: Proceedings of the 1999 ACM SIGMOD international conference on Management of data, Philadelphia, PN, USA, 1999, ACM Press, New York, 1999, 49 - 60.

[4] M. Ames, L. Manguy. PhotoArcs: a tool for creating and sharing photo-narratives. In: CHI '06: CHI '06 extended abstracts on Human factors in computing systems, Montreal, Quebec, Canada, 2006, ACM Press, New York, 2006, 466 - 471.

[5] T. Apted, J. Kay, A. Quigley. Tabletop sharing of digital photographs for the elderly. In: CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems, Montreal, Quebec, Canada, 2006, ACM Press, New York, 2006, 781 - 790.

[6] D. Baur. PhotoHelix - Organizing and Viewing Photos in a Tabletop environment. Projektarbeit, Ludwig-Maximilians-Universität München, 2006.

[7] B. B. Bederson. PhotoMesa: a zoomable image browser using quantum treemaps and bubblemaps. In: UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology, Orlando, FL, USA, 2001, ACM Press, New York, 2001, 71 - 80.

[8] J. F. Blinn. A Generalization Of Algebraic Surface Drawing. In: ACM Transactions on Graphics (TOG) 1:3, 1982, 235 - 256.

[9] A. Cockburn, B. McKenzie. 3D or not 3D? Evaluating the Effect of the Third Dimension in a Document Management System. In: CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems, Seattle, WA, USA, 2001, ACM Press, New York, 2001, 434 - 441.

[10] A. Cockburn, B. McKenzie. Spatial Cognition: Evaluating the effectiveness of spatial memory in 2D and 3D physical and virtual environments. In: CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems, Minneapolis, MN, USA, 2002, ACM Press, New York, 2002, 203 - 210.

[11] A. Cockburn. Revisiting 2D vs 3D implications on spatial memory. In: AUIC '04: Proceedings of the fifth conference on Australasian user interface, Dunedin, New Zealand, 2004, Australian Computer Society, Darlinghurst, Australia, 2004, 25 - 31.

[12] T. T. A. Combs, B. B. Bederson. Does Zooming Improve Image Browsing? In: DL '99: Proceedings of the fourth ACM conference on Digital libraries, Berkeley, CA, USA, 1999, ACM Press, New York, 1999, 130 - 137.

[13] M. Cooper, J. Foote, A. Girgensohn, L. Wilcox. Temporal event clustering for digital photo collections. In: ACM Trans. Multimedia Comput. Commun. Appl. 1:3, 2005, 269 - 288.

[14] A. Crabtree, T. Rodden, J. Mariani. Collaborating around collections: informing the continued development of photoware. In: CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work, Chicago, IL, USA, 2004, ACM Press, New York, 2004, 396 - 405.

[15] S. Deb, Y. Zhang. An overview of content-based image retrieval techniques. In: AINA '04: Proceedings of the 18th International Conference on Advanced Information Networking and Applications, IEEE Computer Society, Washington, USA, 2004.

[16] P. Dietz, D. Leigh. DiamondTouch: A multi-user touch technology. In: UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology, Orlando, FL, USA, 2001, ACM Press, New York, 2001, 219 - 226.

[17] S. M. Drucker, C. Wong, A. Roseway, S. Glenner, S. De Mar. MediaBrowser: reclaiming the shoebox. In: AVI '04: Proceedings of the working conference on Advanced visual interfaces, Gallipoli, Italy, 2004, ACM Press, New York, 2004, 433 - 436.

[18] D. Frohlich, A. Kuchinsky, C. Pering, A. Don, S. Ariss. Requirements For Photoware. In: CSCW '02: Proceedings of the 2002 ACM conference on Computer supported cooperative work, New Orleans, LO, USA, 2002, ACM Press, New York, 2002, 166 - 175.

[19] S. Golder, B. A. Huberman. The Structure of Collaborative Tagging Systems. In: Arxiv preprint cs.DL/0508082, 2005.

[20] A. Graham, H. Garcia-Molina, A. Paepcke, T. Winograd. Time as Essence for Photo Browsing Through Personal Digital Libraries. In: JCDL '02: Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries, Portland, OR, USA, 2002, ACM Press, New York, 2002, 326 - 335.

[21] Y. Guiard. Asymmetric Division of Labor in Human Skilled Bimanual Action: The Kinematic Chain as a Model. In: Journal of motor behavior, 19, 1987, 486 - 517.

[22] J. Heer, D. Boyd. Vizster: Visualizing Online Social Networks. In: INFOVIS '05: Proceedings of the 2005 IEEE Symposium on Information Visualization. IEEE Computer Society, Washington, 2005.

[23] O. Hilliges, P. Holzer, R. Kluber, A. Butz. AudioRadar: A Metaphorical Visualization for the Navigation of Large Music Collections. In: Lecture Notes in Computer Science, 4073, 2006, 82 - 92.

[24] O. Hilliges, D. Baur, A. Butz. Photohelix: Browsing, Sorting and Sharing Digital Photo Collections. To appear in Proceedings of the 2nd IEEE Tabletop Workshop, 2007.

[25] U. Hinrichs, S. Carpendale, S. D. Scott, E. Pattison. Interface Currents: Supporting Fluent Collaboration on Tabletop Displays. In: Proceedings of Smart Graphics 2005. Munich, Germany, 2005, Springer-Verlag, Berlin, 2005, 185 - 197.

[26] E. L. Hutchins, J. D. Hollan, D. A. Norman. Direct Manipulation Interfaces. In: Human-Computer Interaction 1:4, 1985, 311 - 338.

[27] D. F. Huynh, S. M. Drucker, P. Baudisch, C. Wong. Time Quilt: Scaling up Zoomable Photo Browsers for Large, Unstructured Photo Collections. In: CHI '05: CHI '05 extended abstracts on Human factors in computing systems, Portland, OR, USA, 2005, ACM Press, New York, 2005, 1937 - 1940.

[28] H. Kang, B. Shneiderman. Visualization Methods for Personal Photo Collections Browsing and Searching in the PhotoFinder. In: Multimedia and Expo, 2000. ICME 2000. 2000 IEEE International Conference on, New York, NY, USA, 2000, 1539 - 1542.

[29] D. Kirk, A. Sellen, C. Rother, K. Wood. Understanding Photowork. In: CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems, Montreal, Quebec, Canada, 2006, ACM Press, New York, 2006, 761 - 770.

[30] G. Kurtenbach, W. Buxton. User learning and performance with marking menus. In: CHI '94: Proceedings of the SIGCHI conference on Human factors in computing systems. Boston, MA, USA, 1994, ACM Press, New York, 1994, 258 - 264.

[31] R. Kruger, S. Carpendale, S. D. Scott, S. Greenberg. How people use orientation on tables: comprehension, coordination and communication. In: GROUP '03: Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work. Sanibel Island, FL, USA, 2003, ACM Press, New York, 2003, 369 - 378.

[32] R. Kruger, S. Carpendale, S. D. Scott, A. Tang. Fluid integration of rotation and translation. In: CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems, Portland, OR, USA, 2005, ACM Press, New York, 2005, 601 - 610.

[33] Y. Li, K. Hinckley, Z. Guan, J. A. Landay. Experimental analysis of mode switching techniques in pen-based user interfaces. In: CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems. Portland, OR, USA, 2005, ACM Press, New York, 2005, 461 - 470.

[34] W. E. Lorensen, H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In: SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques, ACM Press, New York, 1987, 163 - 169.

[35] K. Perlin, D. Fox. Pad: an alternative approach to the computer interface. In: SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques, ACM Press, New York, 1993, 57 - 64.

[36] D. Pinelle, C. Gutwin, S. Subramanian. Designing digital tables for highly integrated collaboration. Technical report HCI-TR-06-02, Computer Science Department, University of Saskatchewan. `http://hci.usask.ca/publications/2006/highly-int-collab-tech-report.pdf`

[37] J. Rekimoto. SmartSkin: an infrastructure for freehand manipulation on interactive surfaces. In: CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems: Changing our world, changing ourselves. Minneapolis, MI, USA, 2002, ACM Press, New York, 2002, 113 - 120.

[38] M. Ringel, K. Ryall, C. Shen, C. Forlines, F. Vernier. Release, relocate, reorient, resize: fluid techniques for document sharing on multi-user interactive tables. In: CHI '04: CHI '04 extended abstracts on Human factors in computing systems. Vienna, Austria, 2004, ACM Press, New York, 2004, 1441 - 1444.

[39] M. Ringel Morris, A. Paepcke, T. Winograd. TeamSearch: comparing techniques for co-present collaborative search of digital media. In: TABLETOP '06: Proceedings of the First IEEE International Workshop on Horizontal Interactive Human-Computer Systems, IEEE Computer Society, Washington, 2006, 97 - 104.

[40] M. Ringel Morris, A. Cassanego, A. Paepcke, T. Winograd, A. M. Piper, A. Huang. Mediating Group Dynamics through Tabletop Interface Design. In: IEEE Computer Graphics and Applications, 26:5, 2006, 65 - 73.

[41] M. Ringel Morris, A. Paepcke, T. Winograd, J. Stamberger. TeamTag: exploring centralized versus replicated controls for co-located tabletop groupware. In: CHI '06: Proceedings of the SIGHCHI conference on Human Factors in computing systems. Montreal, Quebec, Canada, 2006, ACM Press, New York, 2006, 1273 - 1282.

[42] K. Rodden, W. Basalaj, D. Sinclair, K. Wood. Does organisation by similarity assist image browsing? In: CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems. Seattle, WA, USA, 2001, ACM Press, New York, 2001, 190 - 197.

[43] K. Rodden, K. R. Wood. How Do People Manage Their Digital Photographs? In: CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems, Ft. Lauderdale, FL, USA, 2003, ACM Press, New York, 2003, 409 - 416.

[44] B. E. Rogowitz, T. Frese, J. Smith, C. A. Bouman, E. Kalin. Perceptual Image Similarity Experiments. In: Proc. SPIE Vol. 3299, p. 576-590, Human Vision and Electronic Imaging III, 07/1998.

[45] R. Rosenholtz, Y. Li, J. Mansfield, Z. Jin. Feature Congestion: A Measure Of Display Clutter. In: CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems, Portland, OR, USA, 2005, ACM Press, New York, 2005.

[46] K. Ryall, C. Forlines, C. Shen, M. Ringel Morris. Exploring the effects of group size and table size on interactions with tabletop shared-display groupware. In: CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work. Chicago, IL, USA, 2004, ACM Press, New York, 2004, 284 - 293.

[47] C. Shen, N. Lesh, B. Moghaddam, P. Beardsley, R. S. Bardsley. Personal digital historian: user interface design. In: CHI '01: CHI '01 extended abstracts on Human factors in computing systems. Seattle, WA, USA, 2001, ACM Press, New York, 2001, 29 - 30.

[48] C. Shen, N. B. Lesh, F. Vernier, C. Forlines, J. Frost. Sharing and building digital group histories. In: CSCW '02: Proceedings of the 2002 ACM conference on Computer supported cooperative work. New Orleans, LA, USA, 2002, ACM Press, New York, 2002, 324 - 333.

[49] C. Shen, K. Ryall, C. Forlines, A. Esenther, F. D. Vernier, K. Everitt, M. Wu, D. Wigdor, M. Ringel Morris, M. Hancock, E. Tse. Informing the Design of Direct-Touch Tabletops. In: IEEE Computer Graphics and Applications, 26:5, 2006, 36 - 46.

[50] L. Terrenghi, D. Patel, N. Marquardt, R. Harper, A. Sellen. The Time-Mill: An interactive mirror evoking reflective experiences in the home. Not published. `http://research.microsoft.com/sds/papers/Time-Mill%20paper.pdf`

[51] O. Turetken, R. Sharda. Visualization of web spaces: State of the art and future directions. In: SIGMIS Database, 38:3, 2007, 51 - 81.

[52] B. Ullmer, H. Ishii. The metaDESK: models and prototypes for tangible user interfaces. In: UIST '97: Proceedings of the 10th annual ACM symposium on User interface software and technology. Banff, Alberta, Canada, 1997, ACM Press, New York, 1997, 223 - 232.

[53] S. Vroegindeweij. My Pictures: Informal image collections. HP Laboratories, Bristol, 2003.

[54] M. Weiser. The computer for the 21st century. In: Communications, Computers, and Networks (Scientific American), September 1991.

[55] R. Want, B. N. Schilit, N. I. Adams, R. Gold, K. Peterson, D. Goldberg, J. R. Ellis, M. Weiser. An overview of the PARCTAB ubiquitous computing experiment. In: Personal Communications IEEE, 2:6, December 1995, 28 - 43.

Web-Referenzen

[56] Adobe. Kostenlose Software zur Bildbearbeitung auf Basis von Adobe Photoshop - Kostenlose Foto-Software von Adobe. 25.09.2007. http://www.adobe.com/de/products/photoshopalbumse/

[57] Adobe. Adobe - Adobe CS3 : Bridge. 25.09.2007. http://www.adobe.com/products/creativesuite/bridge/?xNav=DPBR

[58] Adobe. Adobe - Photoshop Lightroom. 25.09.2007. http://www.adobe.com/products/photoshoplightroom/

[59] AGEIA Technologies Inc. AGEIA. 14.09.2007. http://www.ageia.com

[60] Apple. Apple - Aperture. 25.09.2007. http://www.apple.com/aperture/

[61] Apple. Apple - iLife - iPhoto - New in iPhoto ’08. 25.09.2007. http://www.apple.com/ilife/iphoto/

[62] Apple Inc. Apple - iPhone. 11.09.2007. http://www.apple.com/iphone

[63] Asim Goheer. EXIFextractor library to extract EXIF information - The Code Project. 11.09.2007. http://www.codeproject.com/csharp/exifextractor.asp

[64] Bioslayer. physics2d - Physics2D.NET. 14.09.2007. http://physics2d.googlepages.com/physics2d.netscreenshots

[65] Canon. Canon Deutschland - ZoomBrowser EX / ImageBrowser. 25.09.2007. http://www.canon.de/for_home/product_finder/cameras/digital_camera/ixus/features/creative_software/zoombrowser.asp

[66] CeWe Color. Pressemitteilungen Detail CeWe - einfach schöne Fotos. 24.09.2007. http://www.cewecolor.de/index.php?id=157&no_cache=1&tx_ttnews[tt_news]=2540&tx_ttnews[backPid]=114&cHash=97a37c051b

[67] del.icio.us. del.icio.us. 25.09.2007. http://del.icio.us

[68] Facebook. Facebook | Welcome to Facebook!. 25.09.2007. http://www.facebook.com

[69] Google. Picasa. 25.09.2007. http://picasa.google.com/

[70] HP. HP Photosmart Essential 2.5 - Kostenlose Software zum Bearbeiten, Verwalten, Drucken und Weitergeben von Fotos. 25.09.2007. http://www.hp.com/united-states/consumer/digital_photography/free/software/index_ww_deu.html

[71] IBM. The evolution of storage systems. 24.09.2007. http://researchweb.watson.ibm.com/journal/sj/422/morris.html

[72] Julio Jerez. Newton Game Dynamics. 14.09.2007. http://www.newtondynamics.com

[73] Hannu Kankaanpää. « Hannu’s Plaza » Metaball math. 11.09.2007. http://www.niksula.cs.hut.fi/~hkankaan/Homepages/metaballs.html

[74] Kodak. KODAK EASYSHARE Software. 25.09.2007. http://www.kodak.com/eknec/PageQuerier.jhtml?pq-path=130&pq-locale=de_DE

[75] Kodak. Kodak: History of Kodak: George Eastman - the man: About his Life. 24.09.2007. http://www.kodak.com/US/en/corp/kodakHistory/eastmanTheMan.shtml

[76] Lokalisten Media GmbH. lokalisten - dein persönliches netzwerk. 25.09.2007. http://www.lokalisten.de

[77] Microsoft. Microsoft Surface. 25.09.2007. http://www.microsoft.com/surface/

[78] Microsoft Corporation. Download details: Microsoft Windows XP Tablet PC Edition Software Development Kit 1.7. 11.09.2007. http://www.microsoft.com/downloads/details.aspx?FamilyId=B46D4B83-A821-40BC-AA85-C9EE3D6E9699&displaylang=en

[79] NEC. Design : Advanced Design : Resonantware NEC. 22.09.2007. http://www.nec.co.jp/design/ja/advance/indexj.html

[80] Nikon. Nikon - Press room - Press Release. 24.09.2007. http://www.nikon.co.uk/press_room/releases/show.aspx?rid=201

[81] W. Bradford Paley. Mind'space at MoMa. 14.09.2007. http://wbpaley.com/brad/oldHome/mindspaceAtMoMA

[82] PEW Internet and American Life Project. 28% of Online Americans Have Used the Internet to Tag Content. 14.09.2007. http://www.pewinternet.org/pdfs/PIP_Tagging.pdf

[83] Prophoto GmbH. Prophoto GmbH - Deutschland / Europa / Welt - Photo- und Imagingmarkt auf konstant hohem Niveau - Marktwachstum durch digital Innovationen. 24.09.2007.http://www.prophoto-online.de/amateurphotomarkt/marktwachstum-photo-imagingmarkt-2007.html

[84] SMART Technologies Inc. DViT - Digital Vision Touch Technology - White Paper. 15.09.2007. http://smarttech.com/dvit/DViT_white_paper.pdf

[85] Russell Smith. Open Dynamics Engine - home. 14.09.2007. http://www.ode.org

[86] Sony. www.imagemixer.com. 25.09.2007. http://www.imagemixer.com/e/sony/index.htm

[87] TED Conferences LLC. TED | Talks | Jeff Han: Unveiling the genius of multi-touch interface design. 11.09.2007. http://www.ted.com/index.php/talks/view/id/65

[88] University of Waikato. Weka 3 - Data Mining with Open Source Machine Learning Software in Java. 11.09.2007. http://www.cs.waikato.ac.nz/~ml/weka

[89] Mario M. Westphal. photools.com IMatch. 25.09.2007. http://www.photools.com/products.php

[90] Yahoo. Willkommen bei Flickr - Fotosharing. 14.09.2007. http://www.flickr.com

[91] Jason Zelsnack. J(tt)Z - Ageia PhysX Novodex NET Wrapper. 10.09.2007. http://www.zelsnack.com/jason/JttZ/Novodex_NET_Wrapper