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

Projektarbeit

PhotoHelix - Organizing and Viewing Photos in a Tabletop environment

Dominikus Baur
baurd@ifi.lmu.de

Bearbeitungszeitraum:	30. 5. 2006 bis 30. 11. 2006
Betreuer:	Dipl. Inf. Otmar Hilliges
Verantw. Hochschullehrer:	Prof. Dr. Andreas Butz

Abstract

In the course of my Project Thesis I designed and implemented an application for tabletop displays that allows the user browsing, organizing, viewing and sharing of his or her photo collection. The photos are shown along a curled, spiral time-line that is bound in its position to a (physical) rotary device on the table and can be navigated with it as well. The user can combine single photos to events and create enlarged, manipulable copies using simple gestures. In the following, I present goals, design, implementation and its aspects and the limits of PhotoHelix.

Zusammenfassung

Im Zuge meiner Projektarbeit habe ich eine Anwendung für Tabletop-Displays entworfen und implementiert, die dem Benutzer das Durchsuchen, Organisieren, Betrachten und Tauschen seiner digitalen Fotos erlaubt. Fotos werden entlang einer spiralförmigen Zeitlinie angezeigt, deren Position an einen (physikalischen) Drehgeber gebunden ist, der die Navigation der Spirale ermöglicht. Der Benutzer kann mit einfachen Gesten einzelne Fotos zu sogenannten "Events" zusammenfügen und vergrößerte, manipulierbare Kopien erstellen. Im Folgenden werde ich die Ziele, das Design, die Implementierung und ausgewählte Aspekte davon und die Grenzen von PhotoHelix darstellen.

Aufgabenstellung

Tasks:

- The construction of a prototype of the rotational device and its connection to the system.
- The design and implementation of an application for the Instrumented Room that allows the user to browse his photo collection using the rotational device and a circular interface, organize the photos in events, share photos with another user and present photos on the wall display. All user interaction is to occur on the SmartBoard table using the rotational device and a pen or finger. The application is to be written in Java. Photos are kept as JPEG-files on the harddisk; their organization (i.e., event structure, comments, etc.) is kept either in a database or in the file structure and IPTC/EXIF-tags.
- Giving a presentation of the status of my work during and afterwards in the Oberseminar.
- Writing a summary of my work, its underlying concepts and ideas. Its size should be about 40 pages and follow these rules: `https://wiki.medien.ifi.lmu.de/pub/Main/ProjektArbeitAufgaben/PA_DA_Richtlinien_161105.pdf`

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, August 4, 2008

. . . . . .

Contents

1 Introduction

Tabletop displays, especially in the living room of the future, allow a novel approach to classic tasks like the management of personal media libraries. With the transition from analog to digital and therefore virtual photos, a switch from paper (i.e., printouts) to a fully digital approach seems only natural.
Yet, regarding casual interaction with photos, most (PC-based) digital user interfaces have a too rigid structure and force the user into a certain usage scheme developed and optimized for office work which incorporates other goals (e.g., efficiency, speed). While the real-world photos can be grouped, classified, stored and manipulated in all ways provided by physical objects, their digital equivalents have to be meticulously categorized and even their most prominent advantage (non-destructive manipulation, e.g., scaling) can only be accessed in rather awkward ways pressing exactly the right buttons at exactly the right time (cf. market study in [1]). Creative ways of interacting with or presenting photos like the creation of personal albums, collages or other forms of paper-based art are not well supported.
But the restriction is not only limited to the output: By relying heavily on the mouse as the only way of input, in contrast to physical photos that allow bi-manual interaction as well and even encourage it, most applications are enforcing the classical desktop usage schemes, lumping work on photos together with tedious office tasks and taking all the fun out of a rather free-form activity.
Another difference is the social aspect of photos: As symbols of memory they commonly are shared between family members and friends, reminiscing about the moments when they were taken. They form mementos on walls and desktops, are given out as tokens of love or friendship and support or encourage telling the stories behind them. This plethora of social connotations is mostly ignored by our photo applications - apart from an abstract online sharing (e.g., via flickr (`http://www.flickr.com`) or e-mail) or the creation of such feasibilities via a hardcopy.
Disengaging the classical user interaction schemes and embracing a more real-world approach seems necessary (cf. [2]).
With Photohelix we tried to ease this passage from real photos to their virtual counterparts: The application is designed for a tabletop display, making use of bi-manual interaction and pen-based input, and still provides a coupling with the real-world by binding the user interface to a rotary (physical) device. With an emphasis on the sharing of photos, PhotoHelix allows casual organizing and viewing.

1.1 Goals of the Project Thesis

Anticipating the advent of interactive surfaces in everyday life we agreed on certain goals: In building a photo application for a tabletop, several constraints given by the software's scope had to be met as well as the limits (and possibilities) provided by this class of display. While some points were clear from the beginning, others developed during the design and implementation.

Usage scenarios :
Several usage scenarios were devised for Photohelix:

- Story telling, presentation and sharing of photos
- Organizing of a photo collection
- Casual browsing and sharing

We derived the following goals, that were the foundations for other requirements, from these scenarios:

- *Radial time-line:*
  Most applications use a time-line to show photos in chronological order, holding a timespan and reaching from the earliest photo (left-most) to the latest on the far-right (in the western culture area at least). The most basic idea (and the origin of its name) was using a radial time-line for Photohelix.
  While the primary thought was using a circle showing a certain section of the total photo set, we soon abandoned the idea in favour of a spiral showing the whole photo set (reasons see 2.1).
  Using time as only dimension seemed the natural choice, given indications from the literature that it greatly increases the efficiency of searching and browsing (cf. [3], [4]).

- *Hybrid User Interface:*
  The application's interface should not only consist of virtual knobs but also provide a tangible element, to alleviate bi-manual interaction and provide the user with a connection between the virtual and the real (cf. [5]). This tangible device should control the navigation on the time-line and the placing of the whole interface. Another requirement was for the device to be easily seizable and usable with one hand.

- *Gesture-based Interaction:*
  With Photohelix we strove for a minimalistic interface, that contains only information absolutely necessary for the user. Basically, the approach was to make only user related data (i.e., photos and grouping) visible and let go of classical interface controls like buttons etc. All handling should be performed by simple gestures that can be sequentially combined to perform more complex actions (cf. [6]). Two-step interactions, like menus at the end of a stroke (e.g., in [7]) should not be used.
  In the course of this minimalistic approach, the gestures should, of course, be as easy to learn and remember as possible, probably using some kind of metaphor.

- *Browsing instead of searching:*
  A benefit many of today's photo applications cannot provide is an easy way for the user to browse his or her whole photo collection. Another downfall is the supply with an overview of all photos, to identify patterns (are there certain periods when many photos were taken? How many photos were taken in a certain time span?) or simply browse one's collection casually to maybe stumble upon old forgotten photos which is naturally possible with paper photos (cf. [2], [1]).
  One strong requirement for Photohelix was therefore the provision of an easily-accessible browsing capability. In combination with the hybrid interface this is one of the strongest points of the application: The user has at any time an overview of the whole collection an can naturally browse through it.

- *Orientation independent:*
  While classical desktop interfaces can safely assume that the user sits in front of the screen and has it turned the right way up, tabletop displays do not offer this security. Because any side of the table offers the same interaction and view the user is no longer restricted to using one special position to operate applications (cf. [8]).
  Photohelix should reflect this by allowing the user to use it from any position around the table without any constraints. Because of the rectangular shape of the screen, different styles of usage (see 2.1.1) can emerge and transitions are fluent.

- *Multiple users:*
  PhotoHelix was from the beginning conceived as an application that could be shared not only passively but also actively by more than one user at the same table. Having more than one user's collection visible at the same time and exchanging single pictures or whole

collections with another user without much effort was one of our goals during the design phase.

The way these requirements were fulfilled in the course of this work is described in the section Photohelix (2).

1.2 Related work

Organizing, viewing and sharing photos are tasks that are commonly found in conjunction with digital photography. Because of the enormous growth of this market in recent years (hitting an estimated 89 million sold devices in 2006 ([19])) software publishers as well as researchers have been interested in getting the user's attention and making these common tasks of the real world easier with their digital counterparts.
An abundance of applications for each of these tasks is available, but are usually very similar in their features and characteristics. In the following, we will show the distinctive differences between the current state-of-the-art and PhotoHelix.

1.2.1 Desktop-based

The first type of application we will focus on is based on a standard desktop PC system, mostly running Microsoft Windows. The target audience of such a software is the end user with a commonly middle-sized photo collection, mostly consisting of holiday and family shots (cf. [1]). These applications typically rely on a classical WIMP (Windows, Icons, Menus, Pointer) interface and let the user organize photos by categories, edit and print them and share them via e-mail.
Arguably the most advanced (commercial) solution for these tasks is Google's Picasa 2 (`http://picasa.google.com`). It combines a clear interface and a broad array of abilities in the fields organizing and editing with internet-related features like easy online publishing and backups and tops it off with convenient automated services like background scanning for new images. Yet, one could say that Picasa is more or less nothing but a tool suite: Providing only functions other software features as well without creating additional benefits from synergies.
From an academic background come two applications, that were written from scratch with a certain goal in mind: Shneiderman's PhotoFinder ([9]) and PhotoMesa ([10]) try to give the user control over his photo collection by either relying on flexible annotations and queries based on them(PhotoFinder) or browsing by zooming (PhotoMesa). Especially PhotoMesa shares with PhotoHelix not only a similarity in naming but also the idea of browsing the whole collection, thus giving a permanent browsing context, and showing only a small section of the whole in detail.
Still, PhotoHelix clearly differs from all the above examples in leaving the classical desktop design paradigms behind, inviting multiple users at once and making use of a hybrid user interface on a tabletop display. The sharing of photos is readily available and not only restricted to classical, indirect ways like e-mail or common folders (either on the internet or a LAN).

1.2.2 Tabletop-based

More akin to PhotoHelix are applications that only work on a tabletop as well. Since the spreading of tabletop displays in common households is still in its infancy they mostly stem from R&D facilities or universities.
One such application is the Personal Digital Historian (PDH) by Mitsubishi Electric Research Labs ([11]). The system tries to alleviate an user's access to his or her multimedia data. Based on the central usage scenario of casual storytelling, the application shows media sorted along

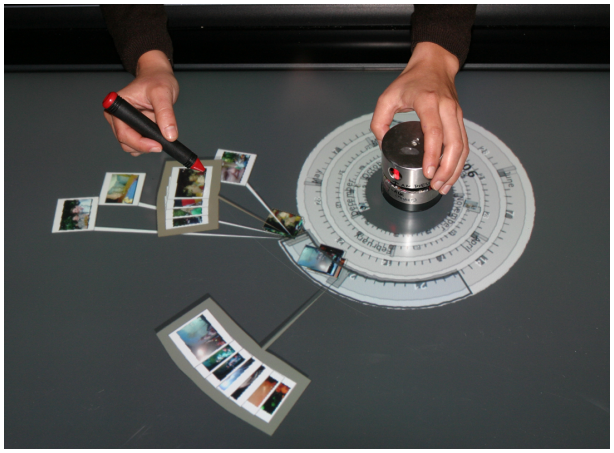


Figure 2.1: Photohelix

different axes (Who, When, Where, What) with rudimentary filtering options.
Teamsearch ([12]) heavily focuses on a shared experience, letting a group of users search for photos by creating collective boolean queries using small physical tokens.
Sharing the same platform, PhotoHelix combines virtues of both applications in providing browsing and sorting options for the single user similar to PDH, but at the same time allowing multiuser interaction for exchanging photos.

2 PhotoHelix

2.1 Overview

Photohelix's design approach was strongly intertwined with its implementation: Certain features were concretisized in an iterative design process. The individual design considerations will be discussed in this section.

2.1.1 Concepts

Presentation of time Photos are arranged in a linear fashion deriving from the time they were taken. While PhotoHelix uses a classical timeline approach - presenting time as points on a linear scale - the timeline is wound up and shown as a spiral.
This idea originated from the calendars of the Mayan culture of Middle America:

People with a Western background think of time as a sequence of moments with a fixed beginning and a possible end. The Middle American natives had a circular notion of the phenomenon: Their time was separated into nested periods that repeated themselves after certain amounts of time (similar to our hours, weeks and months). They represented their calendars accordingly using circles instead of lines (cf. [13]).
PhotoHelix was influenced by this depiction. Initially, time in the application was planned to be shown as a sector of a circle, representing a part of the total amount of time. The user could change the position of this part on the (global) time-line and thereby navigate the photo collection.

The major downside of this approach is the lack of an overview of all photos, which violates our design goal of *Easy Browsing*. Also, finding a certain photo is constrained by the missing orientation otherwise provided by the total of the search space.

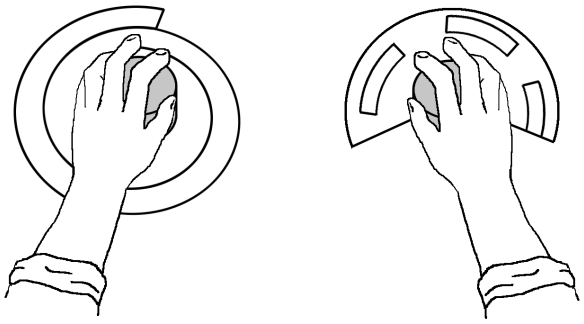


Figure 2.2: Alternative time-lines: Circle versus spiral

For these reasons, we opted for a spiral that shows the whole time-line plus thumbnails coiled around the rotary device. Not only does this provide a distinct connection between the real and the virtual and makes the application's feedback more direct as soon as the user spins the physical device, but it also gives a permanent overview of all photos and avoids the disadvantages of only showing a limited time-frame.
The overview is of course hindered by the user's hand, that for the most time rests on the rotary device and thereby occludes the vision. Yet, by releasing the device, the user can quickly glance at the whole time-line and afterwards continue navigating.

Events and organizing While a machine's intrinsic never-ending ticking might give it an adequate approximation of human time experience, it can only mimic the complex mechanisms our mind uses to mold change into an understandable concept.
Our memory for certain moments is not organized in distinct time units - it relies on its network capabilities, linking each instant to earlier or later ones and the magnitude of sensual impressions (plus their associations) we caught at this point in time.
So, an accumulation of image data on a hard disk might be "seen" by the computer as nothing more than a stream of data plus meta-data - while it allows the human operator to delve into his or her own past - recalling an earlier state and reviving moments long gone.
The analytic, linear view of time has therefore not much in common with the emotional, meshed sight. Yet, the latter makes it difficult to arrange all moments in a linear fashion despite the undoubted sequence of single instants. Free associations break a clear sense of temporal order, making it necessary to bind every moment to an abstract calendar.
PhotoHelix tries to combine the best of both worlds - while it originally relies on the saved dates of the photos the user has the chance to arrange photos as he or she perceives them.
The vehicle for this are **events**. PhotoHelix Events closely cling to the scientific meaning of an event - being *"... something that takes place at a particular place and time"* ([20]).
In the PhotoHelix-sense an event is a noteworthy happening, where, of course, pictures were taken. Examples of events are holidays, vacation trips, dinners etc. with the loose definition allowing of course also more personal categories. This understanding leads to certain constraints for a series of photos to count as an event:

- All photos in one event were taken "in a row", meaning they form chains of points in time. Therefore, no two events can overlap.
- Due to the documentation of a 'noteworthy happening', even a single photo can form an

event.

- Events are limited to a certain length in time for else they would lose their state of distinctiveness. PhotoHelix's internal time limit for an event are two weeks.

Events have an intrinsic social nature: They are practically always shared by more than one person, with possibly multiple people taking pictures. Whole chains of activities spring from such events, with informal meetings to view or share photos and reminiscing these happenings, constructing gifts and more general bonding.
PhotoHelix' sharing ability allows the easy exchange of photos and its usage in the future living room is able to take the place of physical pictures or slide projectors.
The user can group one or more photos to an event - forming a visually and functionally coherent entity, but just as easily dissolve one.

Spatially separated work and navigation In the course of its bi-manual interaction paradigm PhotoHelix follows the (spatial) separation of work and navigation (cf. [14]). Binding the latter to the non-dominant hand and the former to the dominant one allows the user to keep a constant view of the whole interface and prevents him from having to use awkward movements.
It also limits occlusion between elements of one of the two sections. Having mostly thumbnail images in the navigational section of the interface makes it vital to keep them from their larger work counterparts.
We coiled PhotoHelix's navigational part in the form of the time-spiral around the rotary device and placed the (main) work area right above it. This way, the two stay visually connected and allow the user to immediately see the effects of the rotation of the time-spiral and still have enough distance not to interfere with one another.

Initial placing and radial independence To prevent the asymmetry resulting from the separation between navigational and work area from violating the goal of a *orientation independent* interface, we had to add a mechanism to rotate the whole interface. Another issue to address was the initial placing of the interface. If the rotary device is not yet on the table, how to determine the position of the spiral?
At first, we thought about rotating the interface by turning the bottom part of the rotary device. Unfortunately that would have required to somehow gather the alignment of it - a task needing at least two detection points by the tabletop. But since we in total only had two points available (see 2.4.3), we decided for another way:
We solved the issues by using a two-stage placing method: Initially, the screen is blank. After the user puts the rotary device on the tabletop, the interface appears around it, but is almost completely transparent. He or she is now able to rotate the interface by turning or translate it by moving the rotary device. Leaving either attribute untouched for two seconds leads the spiral to fade fully in and restore the actual function of the rotary device. Pushing it around the table results in any case in a translation of the interface.
If the user is unhappy with the result of the placing, he or she can at any time lift the rotary device and repeat the process by putting it back on the table.
This mechanism ensures the radial independence of the interface, but also leads to the emergence of (mostly) two different usage styles: One horizontally (with the spiral at the left or right border and the rest of the table as work area) and the other vertically (e.g., for sharing photos with one user on the one side and the second one on the other).

Figure 2.3: Rotary Device - Prototype 1

2.2 Rotary Device

In search of a fitting real-world counterpart to the time-line spiral we were looking for a device that fulfilled several criteria:

- As small as possible, yet easily graspable and revolvable with one hand
- Flat on the bottom to make it placeable on the SMART Board
- Wireless connection
- No need for DIY

Unfortunately and contrary to my initial estimate, no such device currently exists. The closest one is the **PowerMate** by Griffin Technology (`http://www.griffintechnology.com`), an all-round, programmable multimedia controller, but it is connected to the computer via USB, which made it useless for our purpose.
The only option left was to build at least a prototype ourselves. Primal suggestions were the connection of a rotary encoder with a wireless kit (which would have taken more work in the programming section) or combining a digital compass with a wireless signaler. Gyroscopic systems were proposed as well.
Several problems with these ideas (like the disturbance of the compass by the magnetic field of the tabletop and the general difficulty of transmitting and processing the signals) led to us using an easy (and cheap) solution instead: Taking a wireless mouse, converting a rotary movement mechanically to a simulated movement (in one axis) and transmitting it using the built-in wireless mechanism (see figure 2.4).
After the initial construction of a rather bold device (see figure 2.3), the second one was a lot smaller and fulfilled all of the above criteria.

Prototype 1 The first prototype we built was constructed from an 8 centimeters waste pipe as body, a thin wood axis, a ball bearing, foamed rubber for the base and a wireless notebook mouse (see figure 2.4).
The axis is pivoted and can be rotated using the thumb and the forefinger, with the hand resting on the pipe. The axis' rotation is interpreted by the mouse as a movement and transmitted to the receiver (and converted back to degrees in the application). The device is asymmetric, with the

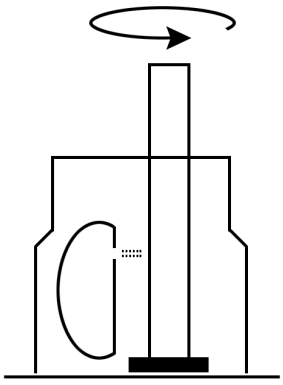


Figure 2.4: Prototype 1 - assembly

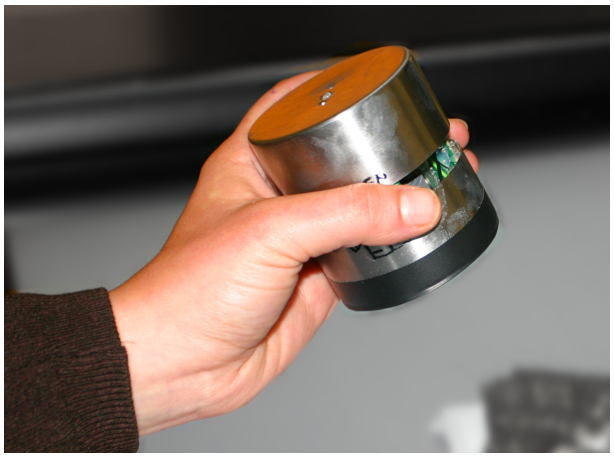


Figure 2.5: Rotary Device - Prototype 2

axis being nearer to one side than the other, due to the lack of space for the mouse. Yet, we did not have to deconstruct the mouse and could fix it lightly to make it easily removable (to engage a connection between the mouse and the receiver a button on the bottom of the mouse has to be pressed).
While our first prototype was suitable for the implementation and initial tests it has its weaknesses in size (grasping with one hand is possible, but moving it around on the table is rather difficult) and the rotary mechanism (we aimed for a more direct spin using the upper part of the device and not an additional knob), which lead to the construction of a second prototype.

Prototype 2 The second (and final) prototype was based on an aluminium kitchen timer that can be rotated on a level axis, making it much smaller and cancelling out the two disadvantages of the first version. Unfortunately, to reduce the size we had to dismantle the mouse and do some further work to secure the wireless transmission and make the connection between mouse and receiver possible. The distance between the mouse's sensor and the axis is carefully adjusted to make the recognition as precise as possible.
The basic functional principle is the same as above: The rotation of an axis under the mouse's laser pretends movement and is then transferred back to degrees in the application. For details see 2.4.2.

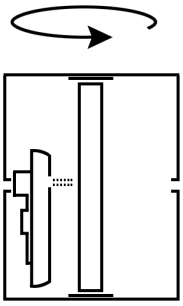


Figure 2.6: Prototype 2 - Assembly

2.3 Interface

PhotoHelix's interface tries to be as minimalistic and functional as possible - we cut short on graphical gimmicks and focused on information needed by the user. The same applies for the mechanisms of interaction, relying on simple and direct gestures.

2.3.1 Overview

The threefold functionality of the application is reflected in the structure of the interface:
The spiral and therefore the navigation is bound to the rotary device and fixed in one spot. The detail view for organizing photos is right above it, in reach of the user's free hand. Finally, viewing photos is possible on the whole screen - photos can be arranged without restrictions.
In the following, we will describe the single parts of the interface in detail and in connection with the possible user interaction.

2.3.2 Navigation

The spiral time-line provides a quick overview of all photos and is the center of navigation in PhotoHelix. All parameters, such as inner and outer radius, number of convolutions, spacing etc. can be adjusted, while not yet in the GUI, at least in a configuration file.

Spiral shape The type of the spiral is logarithmic, making the spiral arms wider on the outside and giving it a more natural look (nautilus clams and spiral galaxies have for example logarithmic spirals as bases of their shapes). Points on a logarithmic spiral are constructed using the following formula (in parametric coordinates):

$$x(t) = \alpha \cdot cos(t) \cdot e^{\beta t} y(t) = \alpha \cdot sin(t) \cdot e^{\beta t}$$

We derive the spiral from three given parameters, namely the radius of the circumcircle, the radius of the incircle and the number of convolutions. Each convolution has 360°, giving a spiral with

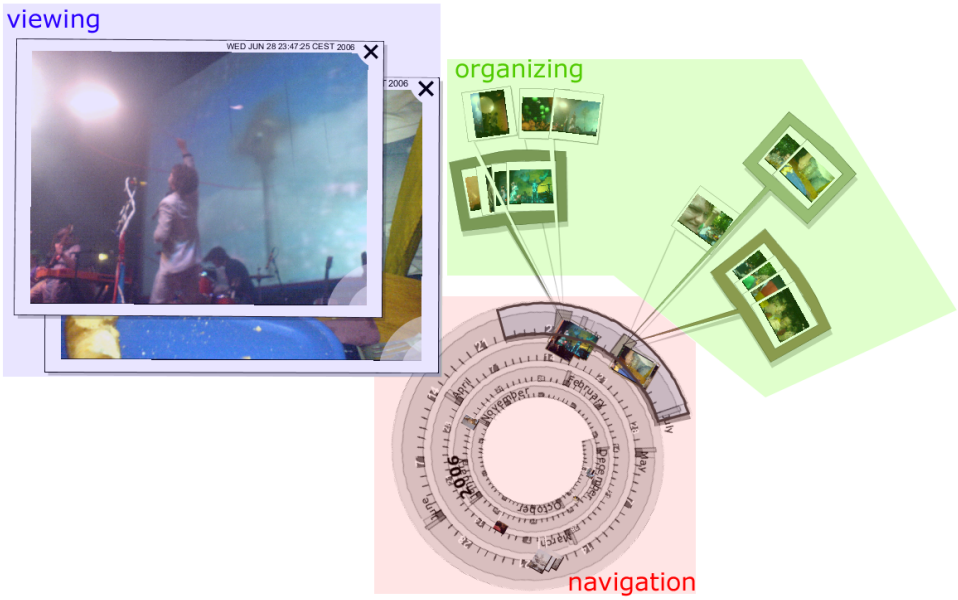
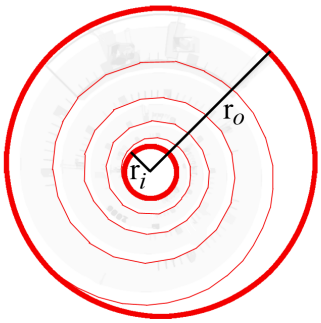


Figure 2.7: PhotoHelix - Basic interface

four convolutions (the default value in PhotoHelix) a total of 1440°, counting from the inside to the outside.

The values for α and β are calculated as follows:
$\alpha = r_i$

$\beta = \frac{log(\frac{r_o}{r_i \cdot cos(d)})}{d}$



with
r_i : *radius of incircle in pixels*
r_o : *radius of circumcircle in pixels*
d : *number of degrees (convolutions times 360°)*

The spiral is drawn not as a collection of short lines (number depending on the granularity-value of the spiral), with their starting and endpoints calculated with the above formula and the fitting values for α and β. The positions on the spiral, where lines begin or end depend of course on the number of lines: The default value as a trade-off between look and performance is 1000 lines.

Calendar The time-line shows dates for easier navigation, thumbnails of the photos taken at a certain moment and events. The length of the timespan depicted on the spiral depends on the age of the photos: The innermost part shows the oldest photos while the outermost part has the youngest photos. By using dynamic limits, we avoided the problem of an either too long (letting all photos collapse in one point) or too short (leaving the oldest photos outside) time-span.
The calendar itself shows values related to date in descending order: Years are the biggest and most prominent, in bold-black color, overlying other chronological labels. Months follow in size, being black as well, yet not as large as year. Last in line are days, with every seventh being shown in white (see figure 2.8).
The distinctive sections of time are shown as well as sections of the spiral: The outer part of the spiral in light-gray symbolizes years, with a dark stroke as separator for different years.

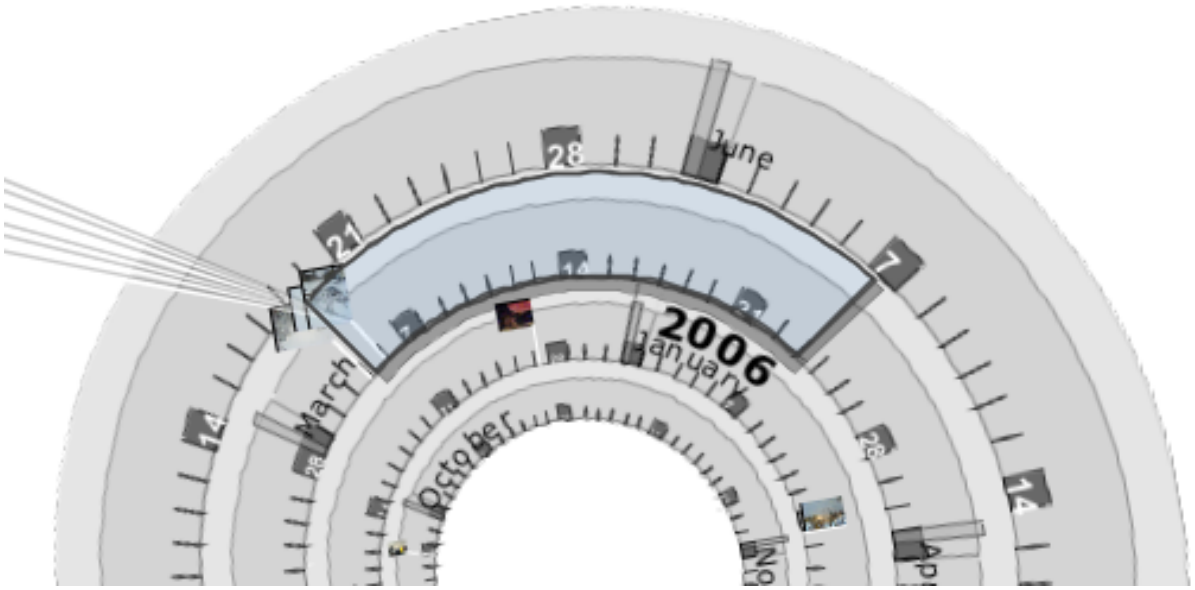


Figure 2.8: Calendar and lens in detail

The middle and inner part has a darker gray and stands for months with a dark gray but wider separator. Days are shown as short black lines on the inner part of the spiral, with every seventh day as black box with a label of its value.
This labeling makes it easy to gather a specific date from the spiral at first sight. Linking thumbnails to the time of their taking is therefore no problem and the fairly rough estimation based on the calendar normally suffices for the user's tasks.

Lens The "lens" is a translucent area rendered on top of the spiral, whose contents are shown in the detail view right above the spiral. It resembles a real-world lens trying to confer the idea of a tool to enlarge something, in our case a certain time-span. The position of the lens relative to the interface is fixed, i.e., it always stays above the center of the spiral, only changing its size according to its position. The visual impression while turning the rotary device is therefore that the spiral is spinning under the lens.

Thumbnails and events Single photos and events are represented on the time-line as thumbnails. Each single photo is placed according to the time it was taken. The thumbnail's size depends on its position on the spiral, leading to older photos having smaller thumbnails.
Photos that belong to an event are shown as stacks and are positioned relative to the event's date. Events are symbolized by colored circular arcs that span the time from the oldest to the youngest contained photo on the spiral, thus allowing the user to estimate the length of the event. The used color is the events average color (see 2.3.3).

Figure 2.9: Detail view

Interaction With the spiral (and the detail view right above it) being most of the time in the center of the user's attention, it is bound to the rotary device's position for easy retrieval.
The most basic interaction is therefore the moving of the rotary device on the tabletop to change the spiral's location. Changing the rotation is only possible in the initialization phase (see 2.1.1).
Turning the spiral and thus changing the position of the lens and the detail view happens by simply twisting the rotary device. Its format allows the comfortable resting of one hand on it and an easy manipulation.
The mapping between rotations of the device and rotations of the spiral is not linear: Depending on the position of the lens the spiral turns slower or more quickly, to allow finer grained navigation near the center of the spiral (which is smaller and harder to see than the outer part), and quick browsing on the outside.
Due to the limitation of the rotary device to these simple manipulations, that can be performed without much concentration or strain, the user can focus his or her attention on the rest of the display to perform other tasks while navigating the spiral without much thought.

2.3.3 Organizing

One of these other tasks is the organizing of photos to events. The user controls the creation and deletion of events in the section of the interface dubbed "detail view".

Photos and events The current contents of the lens on the spiral are shown enlarged in the detail view, a trapezoidal borderless section of the interface right above the spiral. Each photo is shown as a thumbnailed version with a border in the style of cartoon speech balloons. Single pictures have white borders and "umbilical cords" that connect the thumbnails to their actual positions on the spiral.
Events contain one or more photos whose thumbnails are stacked and fanned out. The pictures are sorted by age, with the oldest contained photo on the far left and the youngest on the right (similar to the spiral). Each event has a certain color, which is the average color of all its photos. The event's balloon is colored accordingly as well as its representation on the spiral and the event's umbilical cord, which points not to the positions of the contained photos but their average date.

Single photos and events are positioned relative to one another depending on their age,

with the oldest on the left and the youngest on the right. The main focus of the placing lies however on the optimal utilization of the available space and not necessarily the chronologically order. If, for example, the lens holds a section of the spiral that stands for one week, but all contained photos were taken on one evening, they spread over the whole width of the detail view instead of concentrating on one side.

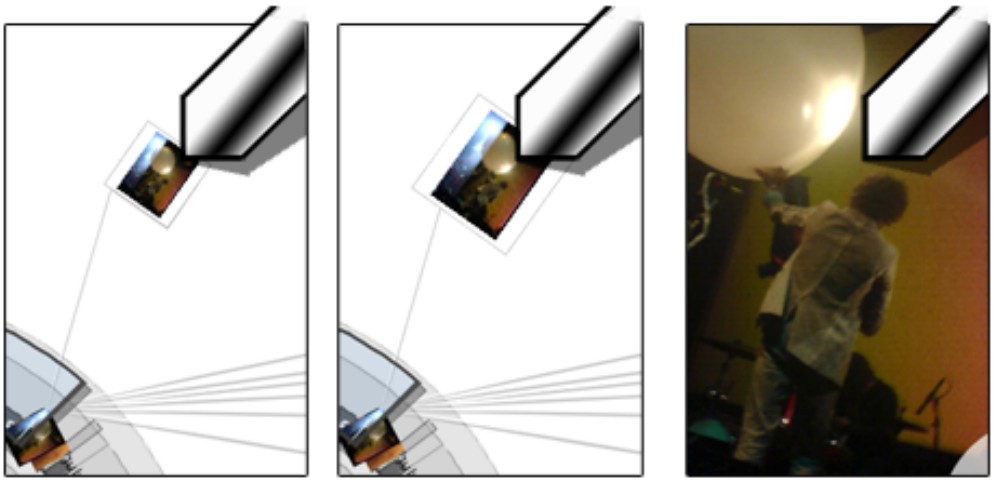
While the spiral gives the correct representation of chronological attributes, the detail view's focus lies on working with and organizing of the photos and tries to make this as convenient as possible for the user.
By turning the rotary device the position of the lens as well as the photos and events is changed: Turning it clockwise causes the balloons to move counterclockwise and vice versa.

Interaction The detail view allows the creation and deletion of events and the browsing through a certain part of one's photo collection. All these activities are triggered by a pen or finger using simple gestures of the free hand.

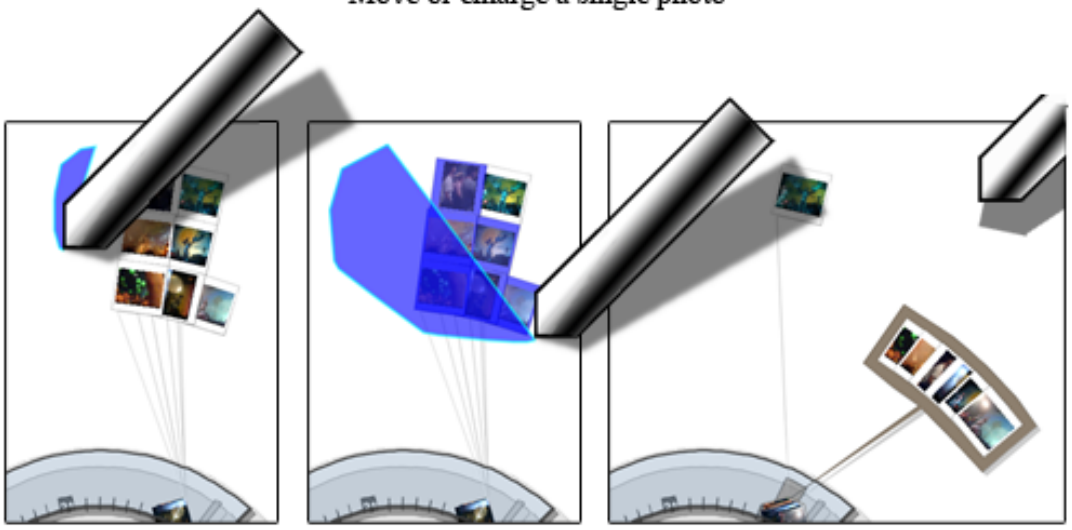
- *Single photos* Though the algorithm for the placing of the photos typically shows good results, the user might want to change the position of single photos. To do this, he or she simply has to touch it and drag it around. The new position is not limited and can be anywhere on the table. The photo's umbilical cord is updated accordingly.
  By touching the photo for a short moment (in the current version 500 milliseconds) an enlarged copy of it is created at the same position (see 2.3.4). In either way, by touching the photo, the thumbnail is slightly enlarged to ease identification.

- *Events* When PhotoHelix is launched for the first time, all photos are unbound and eventless, giving the user the chance to create his or her own desired structure from scratch. The user can create events by simply drawing a circle around photos. It does not matter whether these photos are already part of another event: All marked photos become part of the newly created event and are deleted from their former (if any) events. Events that loose all photos this way are deleted as well.After creating an event the detail view and spiral are updated accordingly.
  To ease the creation of events and prevent the user from the need to draw too elaborate shapes, single photos (see above) as well as events can be freely positioned. After touching an event's border, the user can drag it around.
  When touching one of the event's photos, its thumbnail is enlarged. If the user leaves the pen or finger on the display and moves it along the row of photos, he or she can flip through the stack of photos and see closeups of the contained thumbnails. By dragging a photo a certain distance perpendicular to the event's axis (away from the spiral) an enlarged copy of it is created (see 2.3.4). This point is indicated for the user by switching the photo border's color to red, as soon as the dragged distance exceeds a certain value.
  To delete an event without creating a new one the user draws a single line across the event's umbilical cord - thus striking out its connection to the spiral. If more than one event's connection is hit with the stroke, only the foremost is deleted (events are brought to the front by touching them).

These basic gestures allow a rich bouquet of interaction with photos and events and can be easily learned in a matter of seconds.

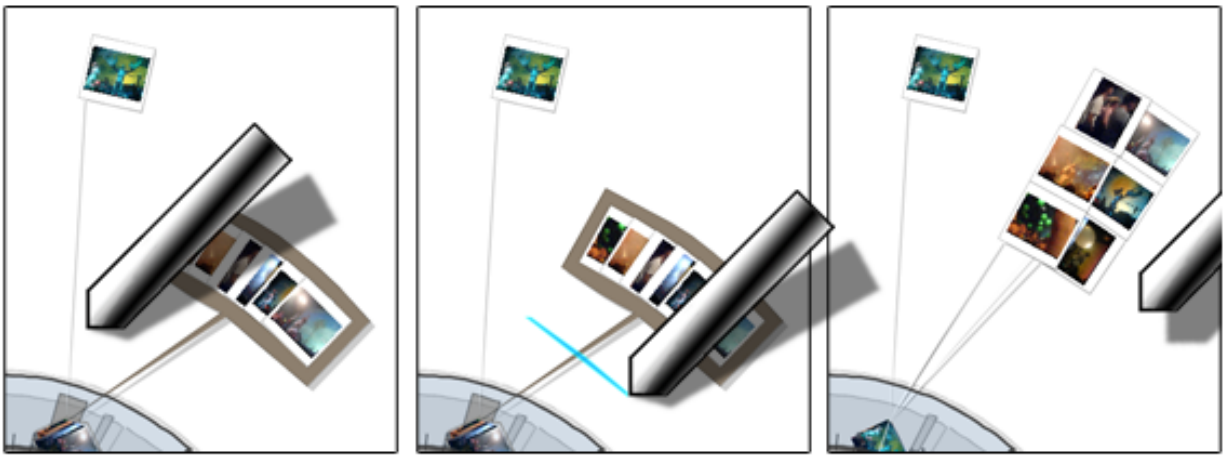


Move or enlarge a single photo



Create a new event

Flip through an event and create a larger copy



Delete an existing event

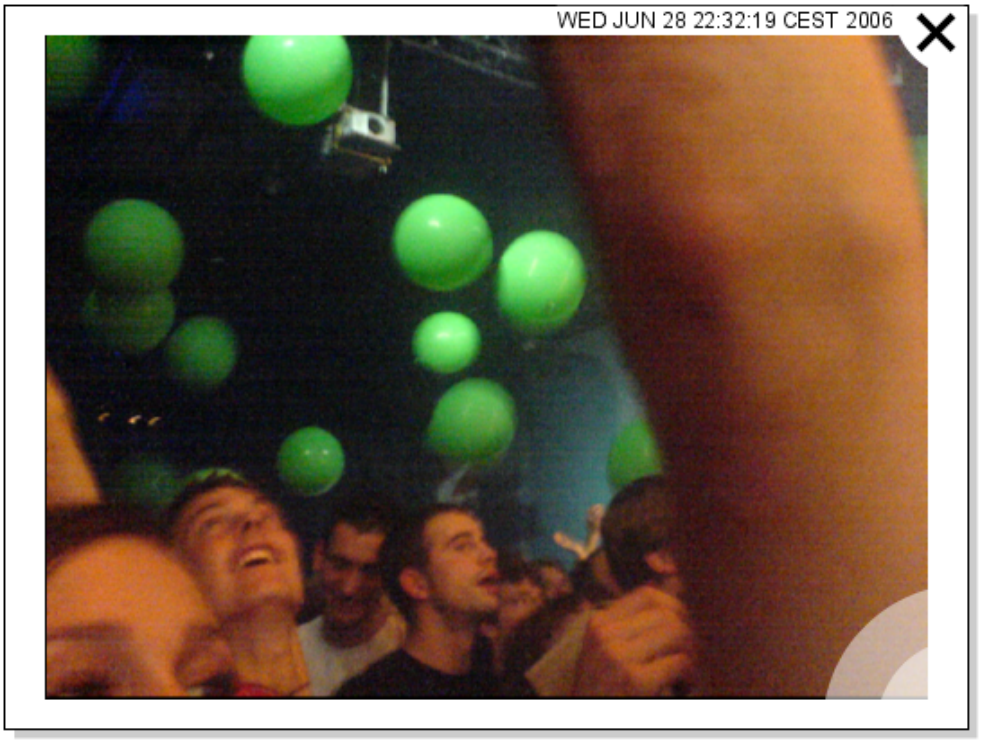


Figure 2.10: Enlarged photo

2.3.4 Viewing

Despite the advantage of having a quick overview of the whole collection, the thumbnail-sized versions of the photos lack the feeling of "real" photos, due to their size.
Therefore, the user always has the option to enlarge photos in the detail view (see above) and create a manipulable copy of them.
"Ripping" the photo from the event or touching a single photo long enough creates a large duplicate of it, that immediately occupies the front of the interface, i.e., occludes all other interface elements. Most of the time, enlarged photos are only used for a quick glance at a picture, e.g., to show it to another person, and are then closed again, so it does not matter if the rest of the interface is partly inaccessible. In this mode, the big photos are in the center of the user's attention, which is represented in the interface's layering order. Yet, even with the enlarged photos on top, they can still be manipulated to be used as decoration (see below).

Presentation Each enlarged photo has a white border (similar to their smaller counterparts), that provides information and affordances for manipulation. The title contains the exact date the picture was taken (with the spiral providing only a rough estimate). The photo itself is enlarged only to a certain percentage of its original size for the sound use of screen estate. The right part of the border has means for closing, rotating and scaling the photo (see below).

Interaction After creating an enlarged copy, it can be dragged around the screen with a touch-and-drag gesture anywhere within the photo or the border. Touching a photo also leads to it being moved to the front, if occluded by others. In the upper-right corner of the border is a small cross, reminiscent in position and appearance of the crosses on windows in windowing operation systems, that immediately closes the photo with a touch. Initially, photos were closed by drawing a short stroke through them because of our design goal of *Gesture-based interaction*, but we decided to let go of this gesture, due to its duration (tipping the cross is much faster than drawing a line, however short), its collision with the translation gesture (To close a photo the user has to start the gesture *outside* of the photo or it is interpreted as movement. The other way round, starting outside and closing the photo while actually only wanting to shift it, is even worse) and the familiarity of the users with graphical operating systems and therefore this symbolism.
The lower-right corner of the picture gives the user the ability to scale and rotate the photo. By touching and dragging one of the quarter circles (the inner one is for rotation, the outer one for scaling) the size and angle of the photo can be changed. The particular amount is determined by the degree of the movement.
The enlarged photos stay in the same position, even if the rotary device is lifted from the tabletop and are only removed by the 'Close'-operation. This, plus the scaling ability, allows the user to decorate the interface using enlarged photos more or less as "background" in unused areas of the table.

2.3.5 Sharing

The above interactions give the user the chance to conveniently organize and present his or her photo collection. Especially in this last context - showing pictures to other people - an immediate sharing of photos without greater effort (like quitting the application, accessing the file system and sending an e-mail) would be desirable. For every user with a rotary device (and therefore an own instance of PhotoHelix) such a simple sharing via Drag-and-drop is possible.

Two-user operation After putting another rotary device on the tabletop, a second version of the user interface appears, complete with another time-spiral and detail view. Interaction is possible

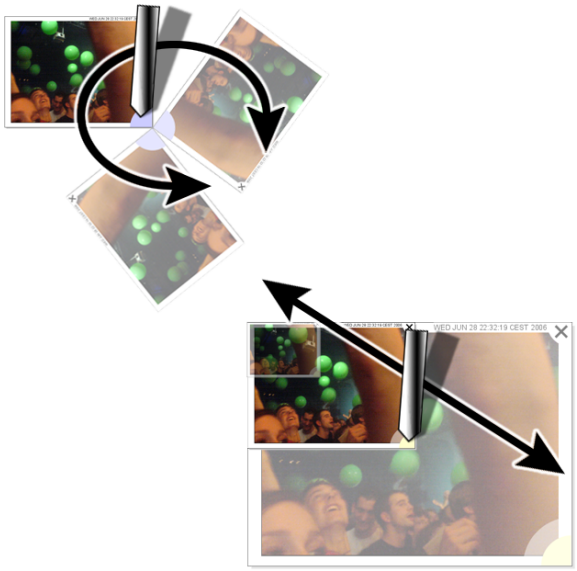


Figure 2.11: Rotating and scaling enlarged photos

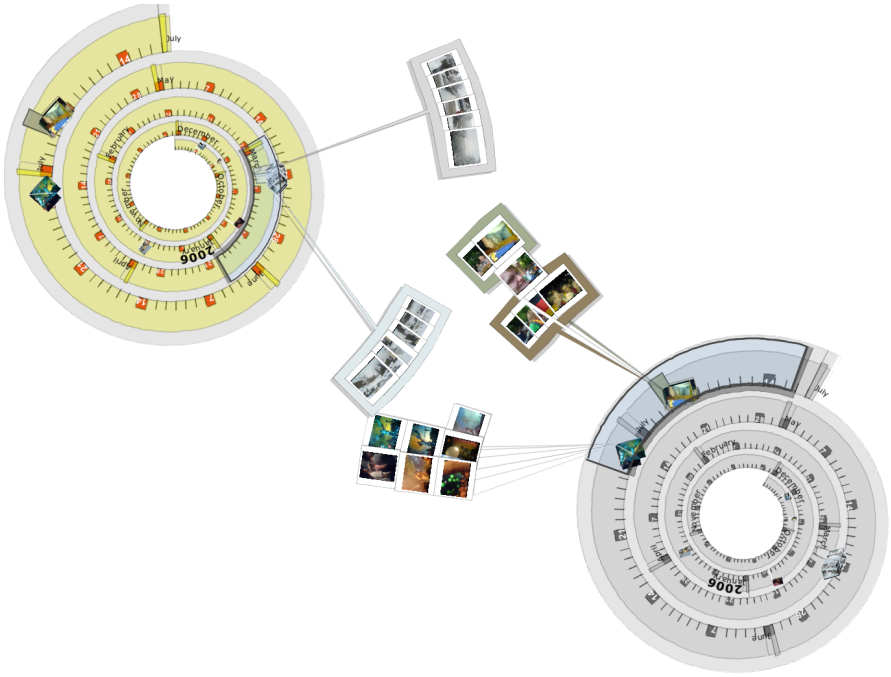


Figure 2.12: Simultaneous two-user operation

in the same ways as before - working with events, navigating the collection, etc. still are triggered using the same gestures - the only difference is that there are now two complete and independent spirals. The system does not differentiate between the two users, i.e., each user can manipulate the other's interface. Yet, the photos and events are still distinct, prohibiting, for example, the combination of photos from two time-lines into one event, due to the logical problems (to which time-line should the event go?). Such distinctions are possible via the internal representations of the visible objects. Still, an exchange is possible.

Sharing between two users By dragging a photo or an event from the second detail view to the own, a user can copy elements of the other interface to his or her time-line. As soon as the user pulls the photo or event across the border of the own detail view, its color changes to red. If the user now lets go, a copy is created and the spiral and detail view are updated. The original photo or event snaps back to its actual position on the other detail view.
Concurrent with our design goal of *Multiple users*, this technique allows for the quick exchange of photos or even whole events between two users. Unfortunately, we could not implement this feature to PhotoHelix, because of the limits imposed on us by the hardware. Sharing as shown above would have required the differentiation of at least four input sources (two rotary devices, two pens), but with the Smartboard distinguishing only two (see 2.4.3) we had to abandon this design for the current implementation.

2.4 Implementation

PhotoHelix was implemented using Java 1.5 on a Windows XP Professional system. For display and input recognition we used a horizontally mounted Smartboard display by SMART Technologies (`http://www.smarttech.com`).

2.4.1 Overview

The class structure of PhotoHelix is roughly based on the Model-View-Controller-paradigm. We used the University of Maryland's *Piccolo.Java* Framework ([21], [15]) to benefit from its easing of the translation, rotation, fading and animation of Swing Components. Therefore, most of the visible components are either derived from PNode (the most abstract Piccolo Node), or a more concrete node like PFrame (the JFrame equivalent of Piccolo) or PPath (a Piccolo Shape). Event-processing is also partly based on Piccolo, using PInputEvent for mouse and keyboard events, but also relies on the Smartboard-SDK for Smartboard-related events like touches and drags.
The classes in detail:

- **PH**: Launches PhotoHelix, only contains the static *main*-method
- **PH_Main**: The main class; initializes the other classes, starts event-processing. Manages connections between model, view and controller.
- **PH_Parser**: Used to convert photos and event structure contained in an XML-file to their PhotoHelix-internal counterparts and vice versa.
- **PH_Configuration**: Holds all kinds of settings for other classes, e.g., the size and position of helix and detail view, different colors for text, background etc.
- **PH_Support**: Provides methods for all kinds of tasks that are very general and can be used by all kinds of classes. Examples are numerical conversions, the calculation of points on the helix etc.
- **HelixSector**: Describes a helix sector and implements the Java Shape-interface.

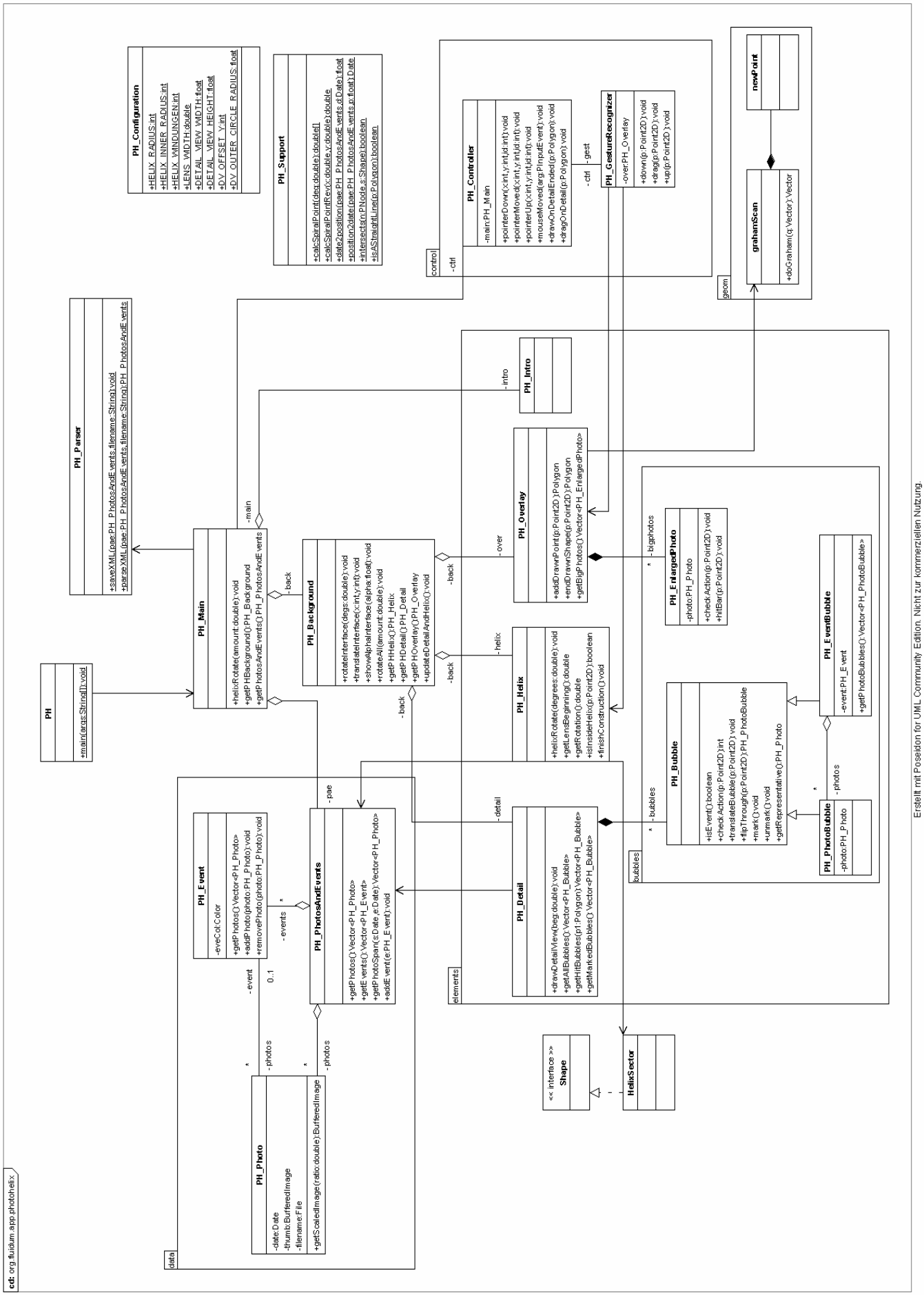


Figure 2.13: Class structure of PhotoHelix

- **PH_Background**: Main part of the view; manages the positions and relations of all visible elements.
- package **elements**: This package contains all elements of the interface, which are held in PH_Background.
  - **PH_Helix**: The representation of the navigational spiral. Provides methods for rotation and positional information like the current rotary value or the position of the lens.
  - **PH_Detail**: PH_Detail draws the contents of the detail view, mainly "bubbles", i.e., either photos or events and provides information about them (e.g., which bubbles are currently contained or which bubbles are marked etc.)
  - **PH_Overlay**: This class is responsible for the enlarged photos and the lens, i.e., elements that lie above detail view and spiral.
  - **PH_Intro**: Minimal class that shows a loading screen (which is faded out afterwards).
  - package **bubbles**: Minor interface elements are located in this package.
    - **PH_EnlargedPhoto**: Represents the enlarged version of a photo. Provides several kinds of possible manipulations.
    - **PH_Bubble**: Abstract base class for photos and events on the detail view.
    - **PH_PhotoBubble**: Represents a photo on the detail view (derived from PH_Bubble). Provides means for manipulation.
    - **PH_EventBubble**: Represents an event on the detail view (derived from PH_Bubble). Provides means for manipulation.
- package **data**: The package data contains the abstract internal representations of photos and events in PhotoHelix (i.e., the "model").
  - **PH_Photo**: A photo.
  - **PH_Event**: An event, which consists of photos.
  - **PH_PhotosAndEvents**: Holds all currently contained photos and events and provides methods to retrieve them (e.g., based on a certain time-span).
- package **control**: These classes control user interaction and forward it to the affected classes in model and view.
  - **PH_Controller**: Manages all incoming input events either from the mouse, a keyboard or the Smartboard. Causes corresponding actions in the model and view classes.
  - **PH_GestureRecognizer**: Is used by PH_Controller if the user starts "drawing" with a pen or finger on the Smartboard. Interprets the movements and returns the estimated action.
- package **geom**: This small package is used for the Graham-Scan-algorithm, for smoothing out the user's drawings with their convex hull. Its code was taken from [22] and adapted to our needs.
  - **grahamScan**: Uses the Graham-Scan on a polygon and returns it.
  - **newPoint**: Extends the Java Point-class and is used solely by grahamScan.

2.4.2 Selected aspects

In the following, we highlight several processes, that are commonly used in the application as well as certain parts we want to describe in more detail.

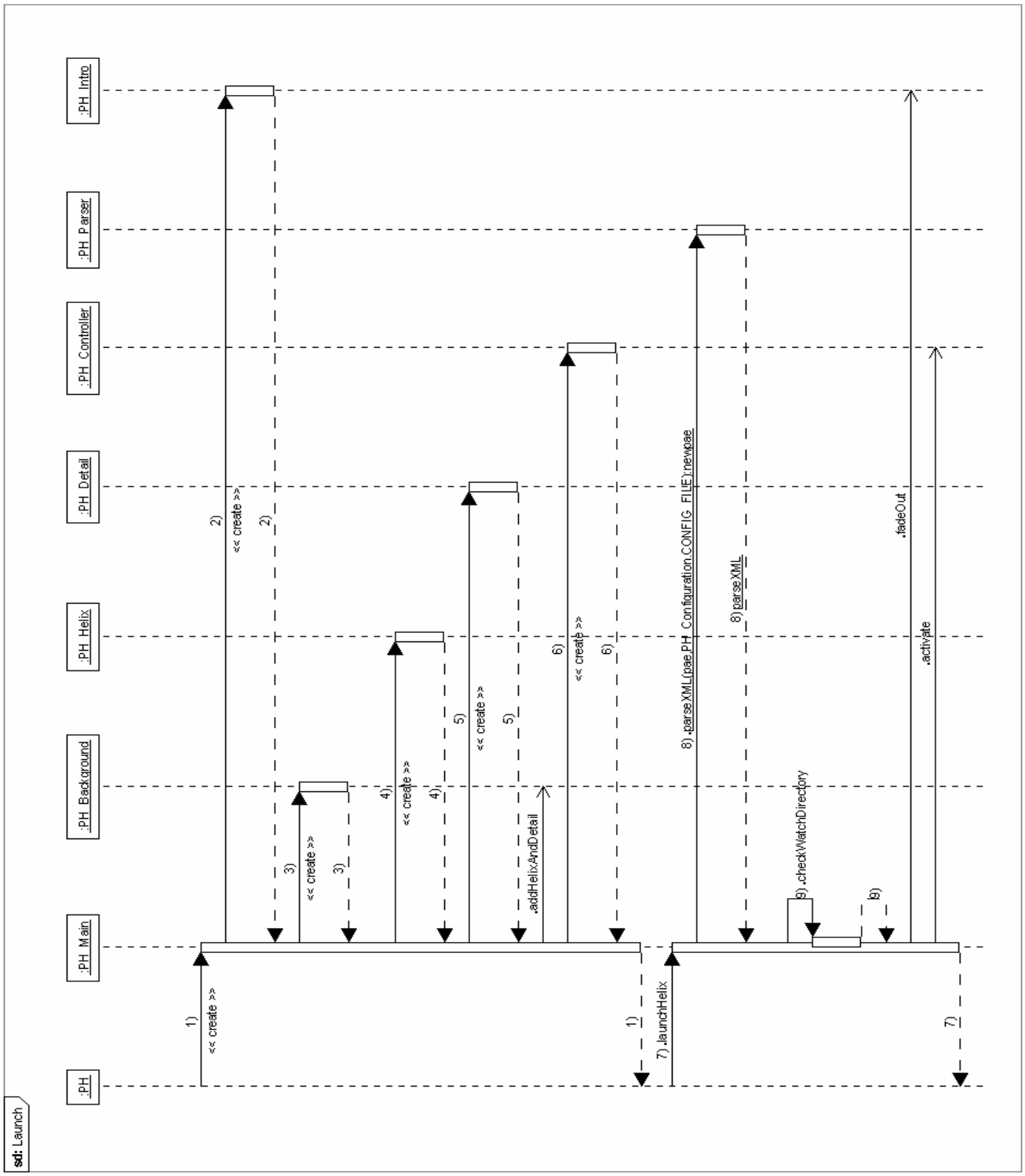


Figure 2.14: Sequence Diagram: Launching PhotoHelix

Launching After starting PhotoHelix, the launch sequence is initiated: The user sees the splash screen provided by PH_Intro, while the different elements of the interface (all classes in the package elements and elements.bubbles) are generated and PH_Controller is initialized.
Photos come to PhotoHelix in mainly two forms: Either by the XML-Configuration-file that contains paths and event structures from the last run or the scanning of a watch directory (which can be set in PH_Configuration), that is scanned each time the application launches. While converting the JPEG-files to their internal counterparts, thumbnails are generated which takes some time.
As soon as all photos are loaded and the interface is ready, PH_Intro disappears and the user is left with a blank screen.
By touching it with (preferably) the rotary device, a transparent version appears, that can be translated and rotated, using PH_Background's methods triggered by PH_Control. Every time the user stops moving and rotating the rotary device, a timer is started that finishes the initialization mode upon completion.
The application switches afterwards to normal usage, that stays active as long as at least one detectable object is on the table. If all objects are lifted, the screen again changes to blank and PhotoHelix repeats the initialization (while of course preserving the current state of the interface).

XML-Configuration The already mentioned XML-Configuration file contains a textual representation of the user's photo- and event-structure, to maintain the state after shutting down the application.
It is based on the following XML DTD:

```
<!ELEMENT photohelix (photo* | event*)>
<!ELEMENT photo (path)>
<!ELEMENT path (#PCDATA)>
<!ELEMENT event (photo*)>

<!ATTLIST event name CDATA #Required>
```

Originally, we intended to give the user the ability to name each event, but dropped it after pondering the use of optical character recognition with the pen and the necessity of such a feature, holding the event's position and its color as sufficient for identification. Still, the name-attribute remained in the XML.
Using XML for the structure of photos and events was not planned in the beginning as well (a very early version used the structure of the file-system with directories as events), yet we decided on it because forcing the user to leave the file-system untouched or manipulating the file-system itself to represent the current internal PhotoHelix-structures seemed unhandy.

Converting horizontal movement to rotation Using a mouse, which is constructed to acknowledge horizontal and vertical movements only, to measure rotation leads to a necessary conversion between the two dimensions. Both our prototypes are based on some shaft rotating beneath the mouse's optical sensor and thereby creating movements along the horizontal axis. These impulses are transmitted wirelessly and reach PhotoHelix via the usual event listener-mechanism (in Piccolo via a PInputEvent-object). The transformation is performed in the *mouseMoved*-method of PH_Controller: The difference between the current and the last measured horizontal position of the cursor is taken, multiplied by a certain (fixed) transformation factor and another position-dependent one and taken as degrees. The latter factor is derived from the lens' position on the spiral and is linear to its position (making the rotation finer controllable towards the inner (and smaller) end of the spiral). Rotation is of course only possible as long as the lens is inside the spiral - if it hits one of the limits the rotation simply stops.

Two marginal adjustments had to be made to use the mouse cursor as rotary encoder: First, we had to make the pointer invisible by using the *java.awt.Window.setCursor*-method with a transparent cursor. Second, the mouse stops at the edges of the screen, even if the user keeps on moving the device in the particular direction. The same applies for our rotary mouse and would have made a continuous rotation impossible, so we reset the cursor each time a movement is detected to the middle of the screen using the java.awt.Robot-class.

Gesture Recognition Gesure recognition is performed mainly in the PH_GestureRecognizer-class. It contains only the three methods *down*, *drag* and *up*, which are called by PH_Controller for the corresponding pen-events from the table.
PH_GestureRecognizer's *down*-method derives the kind of interaction the user plans to perform by the position of the pen and the parts of the interface it touches and this state is kept during the calls of *drag* and *up*. These two methods forward necessary changes in appearance or structure to the affected objects or the controller. After calling *up* the state of the class is reset to its initial one.
Because the internal mechanism of Piccolo which returns the foremost visual element at a certain position proved to be not as reliable as desired, all visual elements are checked for containing the current pen position. The ones that do are then analyzed according to the hierarchy of gestures, making manipulations of enlarged photos the most prominent, with actions on the detail view and gestures on the background following behind.
Depending on the type of manipulation, different courses through the class structure are taken in *drag* and *up*:

- **Enlarged photos**, being only visible elements and without effects on the internal states, perform manipulations by themselves based on the user input. PH_GestureRecognizer only forwards the current position of the pen to their *checkAction*-methods, that determine the type of manipulation, perform the necessary changes in size and rotation (an easy task using Piccolo's built-in methods) and update the display.

- **Photo- and event bubbles** are also only visible elements and provide merely the abilities to translate them, create enlarged versions or flip through them (for events). The first and third options are handled by the *translateBubble*- and *flipThrough*-methods respectively. PH_GestureRecognizer manages new larger copies using the *addBigPhoto*-method of PH_Overlay, that are being triggered by either a timer which is started on *down* for photo bubbles or the return value of the *flipThrough* call for event bubbles.

- Drawing on the **background** causes the creation or deletion of events. Each position that reaches PH_GestureRecognizer via the *drag*-method is added to a polygonal shape, which is shown and stored in PH_Overlay (*addDrawnPoint*) and converted to its complex hull using the Graham scan-algorithm from the class of the same name. While drawing, each bubble that is contained in the shape is marked and shown in a different color (by the *dragOnDetail*-method of PH_Controller, which is called by *drag*). Releasing the pen from the table causes PH_GestureRecognizer to return control to *drawOnDetailEnded* of PH_Controller, that first determines if the user wants to create or destroy an event by calling *isAStraightLine* of PH_Support with the drawn polygon as parameter. Depending on the form of the shape it then either takes all marked bubbles and forms a new event out of them (removing them from their former events and deleting those, if necessary) or takes the foremost (hit) event bubble and deletes it. Changes are propagated to the internal representation in the PH_PhotosAndEvents-object and shown by calling the *updateDetailAndHelix*-method of PH_Background, that refreshes the most prominent display elements.

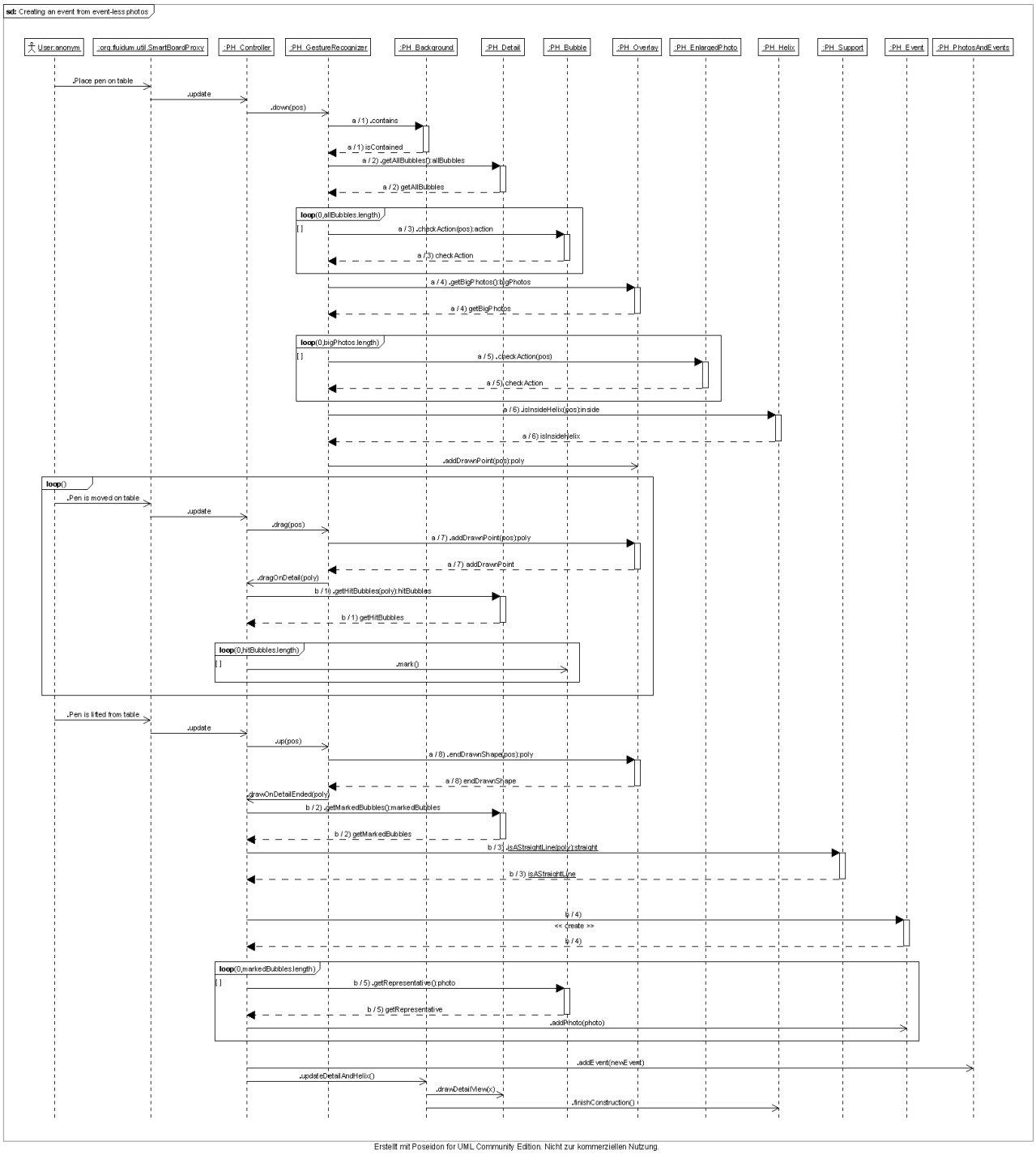


Figure 2.15: Sequence Diagram: Creating an event from event-less photos

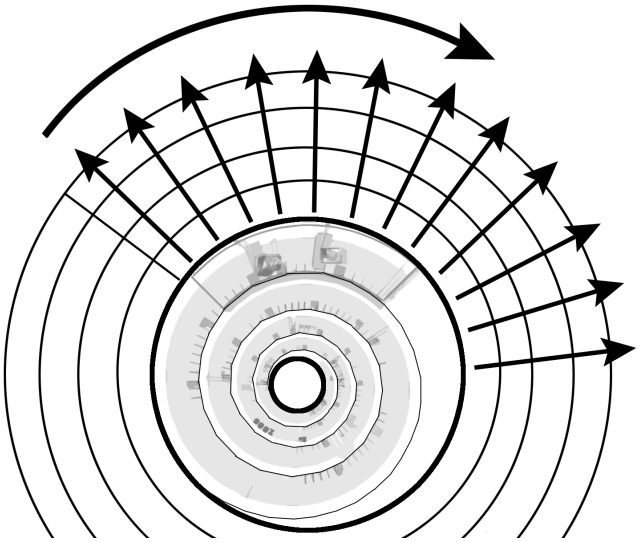


Figure 2.16: Placing algorithm for the detail view

Drawing the detail view Every time the lens is moved or gestures change the structure of photos and events, the detail view is redrawn. Although this might seem unnecessary, especially when performing only small changes like the connection of two photos to an event, due to the generally small number of visible photos, an update is fairly performant.
The algorithm (which is located in the *drawDetailView*-method of PH_Detail) first takes all currently visible photos and events from the PH_PhotosAndEvents-object and checks, which of these are already there and moves them to the left or to the right, depending on the direction of the rotation, or not at all if a change in the structure was made. This way, a rotation is made visible, yet the position of the bubbles, which may have been changed by the user, is preserved and the cost for only minor changes is reduced greatly.
As a second step, all newly surfaced photos and events are positioned and shown. The positioning works along a grid, which consists of several circles that surround the spiral and are separated into adjacent sectors. The first (and preferred) position for an object lies on the innermost circle. The preferred sector depends on the object's date relative to the dates of the other visible photos and events, thus keeping the inherent order. First, all events are placed (because they take more space and gaps between them can be easily filled with single photos) starting with the oldest one. If its favoured position is already taken (e.g., by an event that remained from the last drawing), it is shifted away from the spiral to the outermost circle until a free position is found. If the sector is completely taken, the one to its right is tried, repeating the same process and so on until a position is found or the circle is complete. This is repeated for every event until all are placed and then for all single unplaced photos (again sorted by date).
Performing this algorithm, the usage ratio of the available space on the detail view can be maximized, yet leaving all photos and events completely visible and accessible. If the user is unhappy with the placement or needs the photos in another order (e.g., to create an event out of them with a single gesture), he or she can always change the placement by dragging them around.

2.4.3 Limits

Unfortunately, several limits where imposed on us by the chosen hard- and software, that prevented us from implementing the complete original design and made our implementation restricted in certain areas.

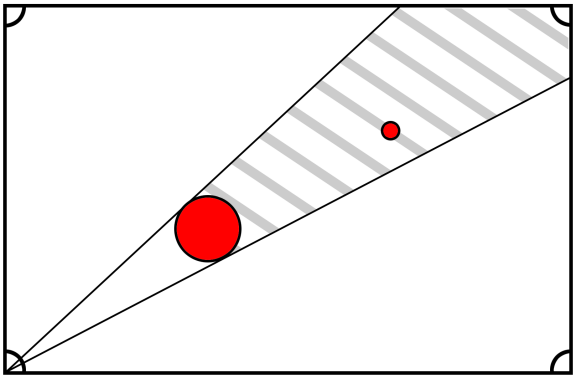


Figure 2.17: Shadowing effects of the rotary device

Smartboard tracking and sharing The sharing of photos and concurrent multiuser-mode necessarily depend on at least four sources of input: Two rotary devices plus two fingers/pens. The tabletop therefore has to provide the application with a minimum of four positions. Unfortunately, the Smartboard we used was not able to fulfill this requirement: For the sake of accuracy of the calculated locations its four cameras are used to determine only two points, which made it impossible for us to integrate the sharing feature into PhotoHelix as designed.
While we first thought about using an alternate, sequential mechanism (like a proxy area for storing photos after a session has ended), we soon dropped the idea, because only casual sharing would have made sense within our application - a forced sequential process, that required the users to engage in an awkward waiting alternation would have decreased the chances of utilization of the feature heavily and made it an interesting quirk at best.
The planned integration of sharing has to be delayed for a future version with a supportive hardware that can track four or more signals (for an overview see [16]).
So, although limiting our use of input sources to two concurrent ones, we still encountered problems: The Smartboard guarantees a reliable identification for every one, yet it tends to fail and mix up sources when one of these stays on it for a longer time (like our rotary device). The tabletop is actually constructed to be used as a whiteboard (vertically mounted, only used for drawing by at most two persons) and has difficulties within our usage frame. To circumvent the random switching of device IDs (which leads to inaccuracies in the gestures of the user and ugly flickering) we used a plausibility method, that acts on the assumption that the rotary device stays in one place and the pen moves only a certain distance at a time, thus countering the negative effects (as long as the user does not move the rotary device and the pen simultaneously).
Another problem with the rotary device on the table is its obscuring the area behind it. The Smartboard's tracking is based on four cameras and to provide a high accuracy, a signal has to be found by more than one. This works well as long as the input sources are small (like fingers or pens) and only stay a short time on the board. Yet, the rather bulky rotary device causes a large shadow and renders one camera almost unusable when placed in a corner (see figure 2.4.3), thus prohibiting a certain area of the screen for gestures.

Photo collection Which and how many photos the user can load into PhotoHelix is bound to several constraints that derive from the chosen system and the limited time we had for the implementation:

- **Missing profiles**: PhotoHelix has only one usage profile, i.e., one configuration-file and one watch directory, which restricts one instance of it to a single user. The photo collec-

Figure 2.18: Limits of the placing algorithm

tion is thereby fixed and independent changes in the event structure are not feasible. With some kind of profile-system being a necessary prerequisite for the sharing of photos and the concurrent use of PhotoHelix, both features have to wait for a later version (see 3.2).

- **File types**: At the moment, only photos in the JPEG-format are supported and loaded. This stems from the heavy reliance of PhotoHelix on metadata, which is the foundation for the order of the photos and is more reliable than the time of the last file change. JPEG pictures were the natural choice, because they are generally small in terms of memory, very widespread and produced (and provided with metadata) by most of the current digital cameras. Also, existing photo collections tend to be already in this format ([1]).
  Other photo formats like PNG or TIFF were for these reasons neglected and also because they either provide metadata in a completely different way or not at all.

- **Number of photos**: While launching, PhotoHelix produces thumbnails from the user's photo collection to show them in the navigational part and the detail view. Unfortunately, this plus the extraction of metadata takes a while and lengthens the duration of the start, with a large (500+) collection causing an almost unacceptable waiting time. Exemplary launch times (all taken on the same machine) are 16 seconds for 20 photos, 2 minutes and 30 seconds for 350 photos and 4 minutes and 30 seconds for 700 photos. Solutions to this problem would be either a reduction of showing thumbnails on the spiral and the loading of them only when needed or accessing a prefabricated thumbnail library like the OS-created "thumbs.db" in Windows XP.
  Another problem with a high number of photos is their placing on the spiral and subsequently on the detail view: With the time-line adjusting dynamically to the dates the photos were taken, it is well possible that a lot of photos that were taken on one day appear as a whole in the detail view, leading to a meaningless wall of pictures (see figure 2.4.3). The problem can partly be solved by the user restricting the time range of the photo collection to a smaller scale, but with this only being a workaround an actual solution still has to be

found.

Performance Our implementation of PhotoHelix has certain decelerating factors, that were of course not clear in the beginning, but inhibit a fluid performance. A general delay is always there with further lags at certain points (e.g., while loading a photo from the hard disk to create an enlarged version of it). Major reasons for this lack of performance are of course flaws in the implementation, but also the use of certain technologies:

- **Smartboard tracking**: The finger- or signal recognition in general adds another layer of complexity to the application and increases the response time. Our application can be started in a desktop-environment as well, with the mouse as pointer and the keyboard as replacement rotary device, and comparisons between the two versions show a palpable delay on the tabletop one, which can only derive from the tracking algorithm.
- **Java**: Java in general tends to run application slower compared to other languages that are closer to the operating system like C++ or C#. This comes from its being interpreted in a virtual machine in contrast to direct execution and is unavoidable (cf. [17]). Its effects can only be marginally countered (cf. [18]).
- **Piccolo**: The Piccolo framework allows convenient geometric manipulations of graphical objects and animations, but at a cost: Its calculations (plus certain workarounds) additionally decrease PhotoHelix's performance.

3 Summary

3.1 Comments

With PhotoHelix we presented an application for tabletop displays that allows a user to comfortably work with his or her photo collection. The hybrid interface that connects the real world to the virtual via a handy rotary device allows concurrent navigation and operation without breaking the user's concentration. Features are the constant overview over the whole collection, gesture-based organizing of photos to events, quick enlarging of single photos and manipulations of these copies plus an unobtrusive feasibility for sharing photos with other users.
All in all, we can say that we are pleased with PhotoHelix in its ideational form as well as its concrete implementation, which is an aesthetically pleasing and fun application that makes use of all advantages a tabletop display provides.

3.2 Future Work

PhotoHelix still has room for improvements and there are several ideas for future versions.
On top of the list is the integration of the sharing feature. As an integral part of the original design its absence is unfortunate and should be corrected in the next iteration in combination with a profile-system and the support of a second rotary device.
Other issues concern the user interface: The detail view has limits displaying very large photo collections (see 2.4.3), which should be solved by either using some kind of hierarchical display, with an automated grouping of photos and a deliberate display of only a certain part, the auto-adjustment of the lens' width to the number of currently shown photos or some kind of zooming mechanism.
Regarding events a more sophisticated structure would be nice, that allows the hierarchical grouping of events (e.g., top-level "vacation in Turkey" and below that "Arrival", "First trip", "Day at the beach" etc.). Another point would be the thematic connection of distinct events, to allow for example the easy retrieval of all Christmas events or all holidays. Naming events is imaginable as

well as letting the user scribble annotations (not necessarily via OCR) for events or photos, with a fixed metadata-system as the final step.
Minor shortcomings like the missing configuration ability in the GUI or the restriction to one watch directory could be fixed more easily.
Our ultimate vision for PhotoHelix is the integration of all programming logic plus photos to a light-weight, easily transportable rotary device. Future houses with an adequate level of ubiquitous computing and large tabletops that automatically register the device and start the application as soon as its placed on a surface (maybe via Bluetooth in a virtual sandbox) would be a convenient way to work with one's photos or show them to friends and family. Concurrent use with other similar devices or even digital cameras, mobile phones or directories on some remote computer and the exchange of photos and events between them would top it off.

Inhalt der beigelegten CD

```
/src
  /org
    /fluidum
      /app
        /photohelix
          - Source code and necessary files (images etc.)

/lib
  - Necessary libraries for compilation
  metadata-extractor-2.3.1.jar
   - Java library for the extraction of metadata from JPEG-pictures.
     Homepage: http://www.drewnoakes.com/code/exif/
  /piccolo-1.2
    - Java Piccolo library by the University of Maryland
      Homepage: http://www.cs.umd.edu/hcil/jazz/

/Quellen
  - References available on the available in Adobe PDF-format.
    Numbered according to the "References" list

/Ausarbeitung
  /Bildvorlagen
   - Pictures used in this work in Adobe PSD-format
  /LaTex
   - Source code of this work in LaTex-format
  PhotoHelix.pdf
    - This work in PDF-format

/Folien
  /Zwischenvortrag
    Zwischenvortrag.ppt
    - Slides for the Zwischenvortrag in MS Office 2003-format,
      held in the Oberseminar on the 5th of July, 2006

inhalt.txt
  - Contents of the CD (German)
```

References

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

[2] D. Kirk, A. Sellen, C. Rother, K. Wood: Understanding Photowork. *Proceedings of the SIGCHI on Human Factors in computing systems* (2006), 761 - 770

[3] D. F. Huynh, S. M. Drucker, P. Baudisch, C. Wong: Time Quilt: Scaling up Zoomable Photo Browsers for Large, Unstructured Photo Collections. *CHI '05 extended abstracts on Human factors in computing systems* (2005), 1937 - 1940

[4] A. Graham, H. Garcia-Molina, A. Paepcke, T. Winograd: Time as Essence for Photo Browsing Through Personal Digital Libraries. *Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries* (2002), 326 - 335

[5] G. W. Fitzmaurice, H. Ishii, W. A. S. Buxton: Bricks: laying the foundations for graspable user interfaces. *Proceedings of the SIGCHI conference on Human factors in computing systems* (1995), 442 - 449

[6] G. Robertson, M. Czerwinski, P. Baudisch, B. Meyers, D. Robbins, G. Smith, D. Tan: The Large-Display User Experience. *IEEE Computer Graphics and Applications* (2005), Vol. 25, No. 4, 44 - 51

[7] A. Agarawala, R. Balakrishnan: Keepin' it real: pushing the desktop metaphor with physics, piles and the pen. *Proceedings of the SIGCHI conference on Human Factors in computing systems* (2006), 1283 - 1292

[8] R. Kruger, S. Carpendale, S. D. Scott, S. Greenberg: How people use orientation on tables: comprehension, coordination and communication. *Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work* (2003), 369 - 378

[9] H. Kang, B. Shneiderman: Visualization Methods for Personal Photo Collections Browsing and Searching in the PhotoFinder. *Proceedings of IEEE International Conference on Multimedia and Expo (ICME2000)* (2000), 1539 - 1542

[10] B. B. Bederson: PhotoMesa: a zoomable image browser using quantum treemaps and bubblemaps. *Proceedings of the 14th annual ACM symposium on User interface software and technology* (2001), 71 - 80

[11] C. Shen, N. Lesh, B. Moghaddam, P. Beardsley, R. S. Bardsley: Personal digital historian: user interface design. *CHI '01 extended abstracts on Human factors in computing systems* (2001), 29 - 30

[12] M. R. Morris, A. Paepcke, T. Winograd: TeamSearch: comparing techniques for co-present collaborative search of digital media. *First IEEE International Workshop on Horizontal Interactive Human-Computer Systems* (2006)

[13] E. M. Reingold, N. Dershowitz, S. M. Clamen: Calendrical calculations, II: Three historical calendars. *Software - Practice and Experience* (1993), Vol. 23, No. 4, 383 - 404

[14] W. Buxton, B. Myers: A study in two-handed input. *Proceedings of the SIGCHI conference on Human factors in computing systems* (1986), 321 - 326

[15] B. B. Bederson, J. Grosjean, J. Meyer: Toolkit Design for Interactive Structured Graphics. *IEEE Transactions on Software Engineering* (2004), Vol. 30, No. 8, 535 - 546

[16] S. Boring: A Wall-sized Focus and Context Display. *Diploma Thesis Ludwig-Maximilians-Universität München* (2006)

[17] C. H. A. Hsieh, M. T. Conte, T. L. Johnson, J. C. Gyllenhaal, W. M. W. Hwu: Compilers for improved Java performance. *Computer* (1997), Vol. 30, No. 6, 67 - 75

[18] J. Shirazi: Java Performance Tuning. *O'Reilly* (2003)

Web-Referenzen

[19] InfoTrends: Maturing Digital Camera Market Expected to Show Growth in New Worldwide Regions. http://www.infotrends-rgi.com/home/Press/2006/3.20.06.b.html, accessed November 25, 2006.

[20] Wikipedia: Event. http://en.wikipedia.org/wiki/Event, accessed November 25, 2006.

[21] B. B. Bederson et al.: Piccolo Home Page. http://www.cs.umd.edu/hcil/jazz/index.shtml, accessed November 25, 2006.

[22] M. F. Hulber: CG DRAWING BOARD 1.0. http://www.cs.rpi.edu/~hulber/cgeom/, accessed November 25, 2006.