Camera supported user interface for presentation software

de Bruijn, R.

Award date:
2008

TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of Mathematics and Computer Science

MASTER'S THESIS

Camera Supported User Interface for Presentation Software

By
Richard de Bruijn

Supervisor:
C. Huizing

Eindhoven, November 2008

Abstract

The traditional way of interacting with a presentation requires a presenter to access the computer physically. When a presenter is pointing to parts of the presentation, there is no interaction between the pointing device and the presentation software. Furthermore, searching for a slide during a presentation is usually not an easy task and can be quite confusing for the audience. We propose a camera supported user interface in the context of presentation software that enables interaction with the presentation at a distance. We use a low-end digital camera in combination with a laser pointer to create an accessible environment that does not require exotic hardware. The system allows us to add interactivity to the pointing action. We design and implement our own presentation software to incorporate the new interactions and additional enhancements to the traditional presentation software.

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

Conventional presentation software implements a user interface that is engineered using the guidelines of a normal desktop application. Like most applications, the software can be controlled by a keyboard and a mouse. Even though, in some cases, there is support for a remote control, the conventional interfaces seem to lack desired features when a presenter is giving a presentation to an audience.

When looking from a presenters' perspective, it is clear that presentation software is used in a different way than an ordinary application. One observation is that a presenter does not sit in a chair, in most cases, when giving a presentation. The presenter is standing next to or in front of the computer, looking at the audience and sometimes looking at the presentation screen to point out certain parts of the presentation.

We believe that the conventional way of interaction with presentation software is unsatisfactory. As a presenter, you do not want to walk to your computer all the time to do something trivial as changing to the next slide. Having to walk to a computer can bring the presenter out of his concentration and might be a distraction for the audience.

The way in which a presenter can point to parts of a slide is usually very passive. A presenter can use something like a pointing stick or a laser pointer, but there is no real interaction between the pointing device and the presentation software. Furthermore, a pointing stick usually cannot reach the upper part of the projected screen and a laser dot is really small when compared to the elements of a presentation. Maybe a better way of interaction can be designed to allow pointing to a slide in a more interactive and clear way.

Controls for navigating a presentation usually consist of moving to the next slide and moving to the previous slide. This works well when the presenter sticks to the sequence of slides, but when he wants to deviate from the ordering of this sequence and he only has two navigational controls, undesired effects may occur. When the presenter is trying to find a specific slide using the next and previous controls, the intermediate slides will be displayed to the audience in full screen. This can be very confusing for an audience, because they do not know with how much attention they will have to look at the intermediate slides.

In this work we try to find solutions for the problems we believe are present in conventional presentation software as discussed above. Designing the solutions is only one part of this work, we also want to implement the solutions in a proof-of-concept application. This application is presentation software that is designed with our proposed solutions in mind and shows that our ideas of enhancing the conventional presentation software can be realized.

1.2 Environment

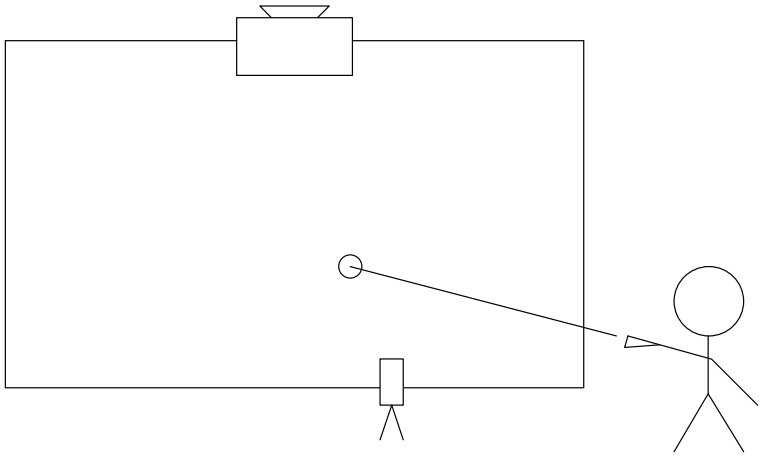

Figure 1.1: Setup of the projector (top), the screen, the camera (bottom) and the presenter with a laser pointer

Our proposal to enhance the conventional presentation software assumes a specific environment that provides an alternative to the standard input-devices like a keyboard and a mouse. We want to make the requirements on the environment easily achievable, therefore we want to use readily available commodity hardware. Below is a discussion of the elements of the environment that are used in designing and implementing the enhanced user interface.

1.2.1 Elements of the Environment

The presentation software is coupled to a projector to display the presentation on an external screen. It is assumed that the projector supports at least the screen-resolution of 1024x768, that is, 1024 pixels wide and 768 pixels tall. A DLP projector in which the so-called *rainbow effect* can occur cannot be used. Such a projector makes use of a color wheel to display the primary colors and the colors are displayed consecutively in time for each image. As we will see later on, we have to capture the image with a camera. When the projector and camera are out of sync, which is a desired state when the camera is calibrated, the individual colors of the color wheel are captured in a rainbow effect. So, if the projector has this rainbow effect, we cannot capture a consistent image from the projector. An LCD projector is best suited for our system, but any projector that creates an image with all of the colors simultaneously

will do.

A digital camera is used to receive images and those images are used to detect interaction events. The camera should be able to deliver images with a resolution of 320x240 at a rate of minimal 30 frames per second. One other requirement is the ability to change the parameters of the camera. It is important that the following parameters are available:

- Shutter speed control
- Gain control

The meaning of these parameters of the camera and the influence of these parameters on the system is discussed in Section 3.2.2. Most current webcams meet the requirements discussed above, because the requirements indicate a relatively low-end camera.

The quality and features of the camera will influence the interaction of the presenter with the software as we will see later on, but most current webcams meet the requirements discussed above.

A laser pointer is required that can be used to point to the external screen that is used by the projector. The laser pointer can be of any color, like red or green. The strength of the laser should be within normal safety ranges. Preferably, it should be easy to switch the laser on and off at the touch of a button.

1.2.2 Setup

The setup of the elements of the environment can be seen in Figure 1.1. The camera is focused on the presentation screen and the presenter can point the laser pointer to any part of the screen. In order for this to work, the camera should not be too far away from the screen and the presenter should not walk into the sight of the camera.

1.3 Previous Work

Using a laser pointer in combination with a camera to control an application is proposed in at least two papers. A paper by Olsen and Nielsen [1] discusses ways of interaction using a laser pointer. A paper by Laberge et al. [2] discusses a calibration and mapping method that enables the laser pointer to be mapped to the screen that is being projected.

Presentation software only has a few controls, so basic features like navigation and highlighting elements of the presentation are obvious ways to improve presentation software. A paper by Cheng and Pulo [3] that appeared in 2003 proposes such a way of interaction with presentation software. The system they describe uses special hardware to track an infrared laser pointer. Furthermore, they implement the system using a plugin for the existing presentation software Microsoft Powerpoint [4]. The system uses a simple gesture to highlight a piece of text permanently and enables the laser to highlight a piece of text temporarily and to navigate through the presentation.

Our work builds upon the proposal by Cheng and Pulo but takes a different approach by using readily available hardware, i.e. a webcam and a standard laser pointer. Cheng and Pulo use gestures only in a very limited way. We improve on this by enabling more complex gestures. We not only want to design new ways of interaction with presentation software using a laser pointer, we also want to re-design the presentation software itself to incorporate new features a presenter is missing when only using a keyboard and a mouse as input devices.

1.4 Layout

In the rest of this document, we introduce the problems that need to be solved and our proposed solutions. We discuss the technical solutions that enable interaction using a camera and a laser. Once interaction with the laser is enabled, we discuss how this interactivity enables the use of gestures in a user interface. We design and implement our own presentation software in which we incorporate our proposed solutions. Finally, our solutions are informally evaluated and conclusions are drawn.

Chapter 2

Goals and Solutions

2.1 Introduction

In solving the problems that we encountered in the introduction using conventional presentation software, we can propose solutions that have the following goals.

1. We want to avoid that the presenter has to walk to his computer during a presentation to interact with the software.
2. Pointing to elements of a presentation should not be passive anymore. There should be interaction between the presenter and the software to enhance the quality of pointing using visual solutions.
3. The traditional controls for navigation, like going to the next or previous slide, should be expanded to allow finding slides more easily without confusing the audience.

2.2 Interaction at a Distance

The environment we have chosen allows solutions to use a camera and a laser pointer as a means of interaction. If we can avoid using the conventional input devices like a keyboard and a mouse, we can interact with the computer at a distance using only the laser pointer in combination with the camera.

The user interface of the presentation software should be designed with the laser pointer in mind. Once the presentation is running, every feature that the presenter needs during the presentation should be accessible using the laser pointer. The software should be able to interpret the images from the camera and be able to detect the position of the laser. Once the position of the laser is found, a meaning should be given to the position and movement of the laser.

The process of finding the position of the laser can be done using image processing on the frames of the camera. One solution for controlling the user interface using the position and movement of the laser is to define a couple of gestures that have a meaning when applied to

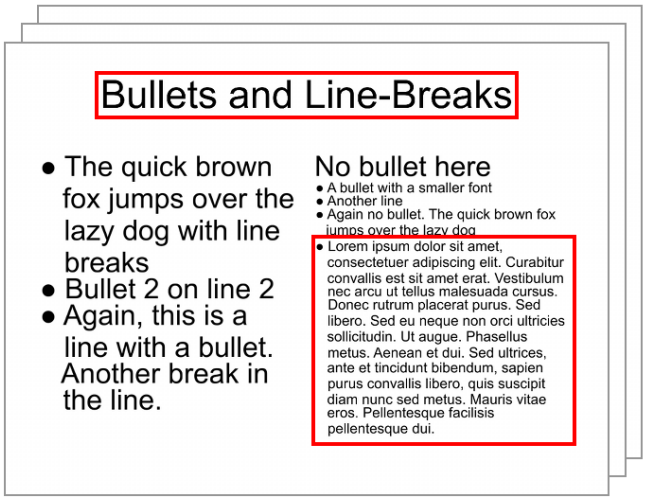

Figure 2.1: A presentation consists of slides with elements that have a bounding box

certain parts of the screen. This means that a user of the system will have to move the laser in such a way that the movement of the laser represents a specific figure, like a horizontal line or a circle.

Interaction at a distance can be done using alternative input devices like a wireless mouse or a remote control, but they cannot be used to point directly to the screen. Pointing a laser pointer to the actual screen is a much more direct way of interaction than having to coordinate a mouse-pointer on a flat surface, representing the area of the screen.

2.3 Pointing

Traditionally, when a presenter points to the screen using a stick or a laser pointer, the presentation software is unaware of this pointing action. In other words, when it comes to pointing, traditional software is passive. But when the position of the pointing device is known by the software, this information can be used to interact with the user interface. Interaction can be in the form of highlighting parts of the presentation or enabling sketching on the screen. The environment we have defined allows the software to be aware of the position of the laser pointer, thus allowing a more interactive way of pointing.

In this context, we define a *presentation* as an ordered sequence of *slides*, see Figure 2.1. A slide is a rectangular area that can be filled with text and images, which we will call *elements*. If we assume that the presentation software is displaying a single slide, a pointing action with the laser pointer results in getting the position of the laser relative to the slide.

Using the relative position of the laser, we can find out which element of a slide intersects it, if any. Intersecting elements can be modified in a visual way to make them stand out more. For instance, a box can be drawn around the element or the element can be made bigger for a short period of time.

Changing the element itself that is being pointed to adds interactivity to the software, but it also improves the quality of pointing in general. The audience has a better understanding of which element of the slide is being pointed to, because the slide is visually modified to incorporate the pointing action.

2.4 Navigation

To find slides more easily and at the same time expand the navigational controls, we propose a new feature in the presentation software, an overview screen. The overview screen visualizes the structure of the presentation by displaying all or some of the slides in the same image. Displaying more than one slide on the screen can be done by making the slides smaller instead of displaying them in full screen.

It is not our goal to display the entire structure of the presentation, but to visualize the current slide and the direct context of this slide. The context of the current slide consists of the previous slides and the next slides, if available. Showing the direct context of the slide helps the presenter find the slide he is looking for, if we assume the slides that are being searched for are recent or will appear in the near future.

The overview screen we propose uses techniques from the focus+context area. Our overview screen can be seen as a modified Perspective Wall [5]. The current slide is the focus and the previous and next slides are the context. The slides are linearly structured in the presentation, this means that the current slide has two direct neighbors. These neighbors also have two direct neighbors and so on. This observation can be used to scale the context slides proportional to the distance to the current slide.

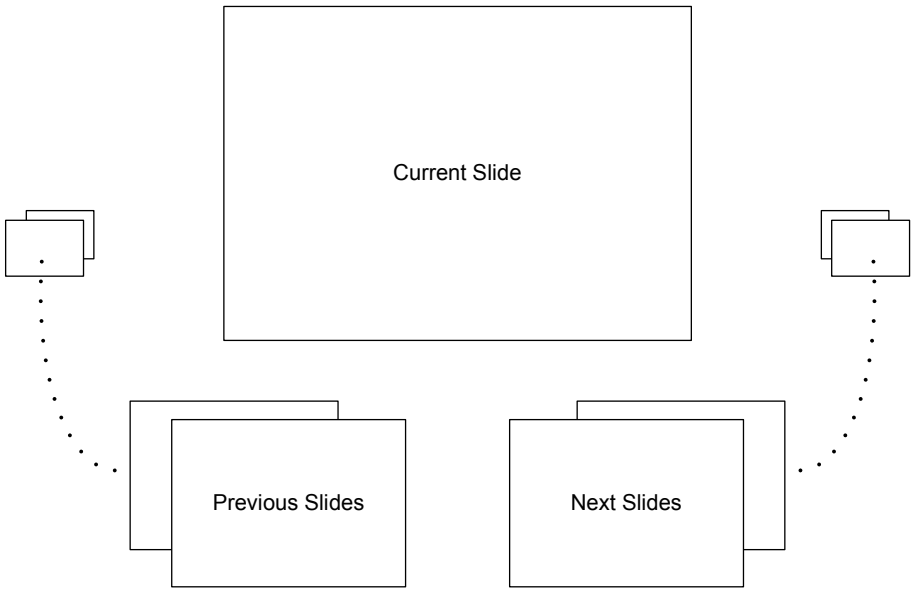

Figure 2.2: Design of the overview screen

Figure 2.2 is an illustration of the design of our overview screen. The current slide is centered and has the biggest scale of all of the slides that are visible on the screen. The previous and next slides in relation to the current slide are placed next to and below the current slide, but with a decreasing scale, dependent on the distance of the slides to the current slide. The neighboring slides are placed in a curve and can overlap each other, to fit more slides on the screen. The design can also be seen as placing the slides of the context on a three-dimensional circle, placed below the current slide.

Navigational controls can be added to the overview screen to allow navigating the presentation inside the overview screen. A next and previous slide control inside the overview screen allows the current slide to be changed. Changing the current slide changes the focus and also changes the context of the slide, thus revealing more of the structure of the presentation.

Using only the standard next and previous controls can be confusing for the audience when trying to find a slide. The overview screen removes the confusion, because the audience knows that the slides that are being displayed do not need any attention yet. The presenter has an overview of the slides and can see the direct context of the current slide, which will aid him in finding slides.

Chapter 3

Computer Vision

3.1 Introduction

We want to design presentation software that uses a camera as its main source of interaction. A camera can deliver a stream of images but it does not know what the images mean or represent. Our application should be able to interpret the stream of images from the camera and process it in some way to be able to understand some of the meaning of the images. Our application should understand enough of the image in order to allow it to be used as a means of interaction.

The environment (Figure 1.1) consists of a projector screen and a laser pointer. The camera is focused on the screen and will deliver the images of the screen to any application that requests the images. In order to allow interaction, the laser needs to be detected. Once the laser is detected, it should be possible to map the position of the laser in the captured image to a position in the projected screen. This can be done when the position of the laser relative to the projector screen is known. This means that the position of the projector screen should be detected as well.

So the steps that need to be taken in order to allow interaction using a camera, a projector screen and a laser pointer are:

1. Detect the laser in the image
2. Detect the projector screen in the image
3. Map the projector screen to the screen being projected and, using this mapping, map the laser to the screen

Getting information from images can be done using techniques from computer vision. The techniques that are used in order to process the images and the steps necessary to allow interaction are described in the next sections.

3.2 Detecting the Laser

3.2.1 Laser Properties

We want to detect a point of laser on an image that is delivered by a camera. In order to detect the laser, we need to find out the properties of the laser to distinguish it from the rest of the image.

A point of laser is very bright. It is a point on which a lot of light is projected, depending on the power consumption of the laser. The laser has a specific uniform color and has roughly the shape of a small circle.

The color properties of the laser can be expressed using HSV color values, hue, saturation and value. Bright spots have a large value, so the point of laser will have a value of almost 100%. The hue value of the laser is well defined, but the camera can have an influence on the hue of the laser on the image, which makes using this property unreliable. Furthermore, the environment states that the laser can be of any color. The color of the laser is intense and therefore the saturation of the laser is not low.

The properties of the laser in HSV space can be used to process images from the camera by filtering the pixels of the image. First we have to define ranges for the HSV values that determine if a pixel with a specific HSV value belongs to the laser. Once those ranges are defined, we can test every pixel of the image against the HSV-ranges and discard any pixel that is outside of the defined ranges. The pixels that are left in the image comply with the color-properties of the laser.

If we assume that every pixel that complies with the color-properties of the laser also belongs to the laser, we can find the entire laserpoint by calculating the bounding box of these pixels. Once we have the bounding box, we can define the position of the laser by determining the center of the bounding box. In this stage, we might also calculate the average color in the bounding box of the laser. The color we get can be used to increase the feel of interactivity in a feature like sketching on the screen.

3.2.2 Calibrating the Camera for the Laser

The quality of the image that needs to be processed depends on the quality and settings of the camera. The settings of the camera, like shutter speed and gain control, influence the image and hence the detection of the laser. Therefore, calibrating the camera is just as important for detecting the laser as processing the image.

The influence of the shutter speed settings of the camera can be seen in Figure 3.1. The figure shows three images from the camera and the images show a point of laser on a blank wall. In all of the images, the shutter speed is adjusted while the amount of gain is at a fixed value.

The image on the left shows the laser with a low shutter speed. The laser is indistinguishable from the background. In this case, no algorithm is able to see the laser. The data of the laser is lost because there is too much light. The image in the middle shows the laser with a medium shutter speed. The laser is visible, along with the blank wall. In the last image, only the laser is visible. The blank wall is black and the laser seems to be easily detectable. So,

Figure 3.1: Low shutter speed (left), Medium shutter speed (middle) and High shutter speed (right)

Figure 3.1 shows that the settings of the camera are of great influence when the laser needs to be detected. A lower shutter speed makes it impossible to detect the laser but a higher shutter speed makes it relatively easy to detect the laser.

In the same way we can show the influence of the gain settings of the camera. Figure 3.2 shows three different gain levels, while the shutter speed is at a fixed value.

Figure 3.2: Low gain (left), Medium gain (middle) and High gain (right)

Only in the first two images the laser is visible. Increasing the gain of the camera results in brightening fainter lights in the image, but this is undesirable because the laser is already a bright spot. Increasing the brightness of fainter lights results in clouding the laser.

From the results above we can deduce that the laser is most easily detectable when the camera has the following settings:

- High shutter speed
- Low gain

3.2.3 Initial Algorithm

Based on the results so far, we can design an initial algorithm to detect a laserpoint in an image.

Initial Algorithm:
1. Calibrate the camera: 30 fps, high shutter speed, low gain
2. **while** running
3. Get a frame from the camera
4. Filter the frame using the HSV-filter
5. Calculate the bounding box and determine the position of the laser
6. **end**

The algorithm assumes that every point in the image that is not discarded by the HSV-filter belongs to the laser. This assumption is only valid when no other bright lights are present in the room of the camera. Bright lights will generate bright pixels and if these pixels are detected as belonging to the laser, they will produce false positives. The environment in which a presentation is given is usually darkened, so realistically speaking, bright lights are not really a problem.

If we want to address the theoretical problem of bright lights, the problem can be solved by not only looking at a single image. Consecutive images give us information about the motion of the bright pixels in an image. The motion of the pixels can be used to further classify the pixels as belonging to the laser or belonging to a light source.

3.2.4 Removing Bright Lights

A possible solution to classify the pixels in a frame from the camera in the presence of bright lights is to look at consecutive frames and detect the motion of the pixels. If we use the extra information by using multiple frames, we can remove the number of false positives in the laser detection algorithm.

We make the assumption that someone holding the laser pointer is unable to keep it pointing on the exact same spot in consecutive frames. Using this assumption, we can state that pointing the laser on the screen for a few consecutive frames will result in movement of the laser pointer in these frames. The movement between two frames can be detected by subtracting the two frames and calculating a bounding box. The detection of motion in consecutive frames does not depend on the laser itself, therefore it does not matter if the laser is on or off during this process.

Once we have calculated the bounding box of the motion between the previous frame and the current frame, we only filter the pixels in this bounding box using the HSV-filter as described before. This results in pixels that were moved and comply with the properties of the laser. Bright lights that have a fixed position will not be recognized as belonging to the laser anymore and this results in less false positives.

Algorithm that uses motion:
1. Calibrate the camera: 30 fps, high shutter speed, low gain
2. Get an initial frame from the camera
3. **while** running
4. Get a frame from the camera
5. Calculate the difference between the current frame and the previous frame
6. Calculate the bounding box of the motion
7. Filter the frame using the HSV-filter in the bounding box of the motion

8. Calculate the bounding box and determine the position of the laser
9. **end**

While the algorithm removes the number of false positives in the presence of bright lights, it can introduce false negatives. When no change is detected by the algorithm, but the laser is actually on the screen, a false negative occurs. The assumption of not being able to point the laser on the same spot every frame might be reasonable, but this assumption does not hold when looking at the resolution and framerate of the camera. This means that the assumption might be true in the real world, but the camera is unable to pick up the changes in position of the laser.

There is always a tradeoff between the number of false positives and false negatives. Removing false positives can result in increased numbers of false negatives. If we assume that there are no bright lights, the initial algorithm will give the best results. In a room that has bright lights, only the algorithm that uses motion is able to produce satisfying results.

3.3 Detecting the Screen

3.3.1 Screen Properties

Detecting the projector screen amounts to detecting the four corners of the screen, therefore we assume that every corner of the screen is captured by the camera. The camera is already calibrated to easily detect the laser and we do not want to re-calibrate the camera. This means that we have to detect the screen using a camera with a high shutter speed and a low gain. Fortunately, the projector will project light on the screen which makes it a relatively bright area as can be seen in Figure 3.3. As with the laser, the settings of the camera will increase the detectability of the screen. Light that is not as bright as the screen and the laser is removed by the settings of the camera.

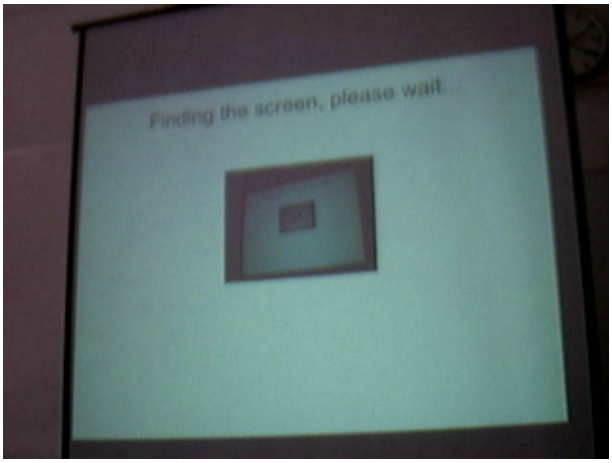

Figure 3.3: Projector screen captured with a camera using high shutter speed and low gain

As can be seen from Figure 3.3, the screen is brighter than the background. This difference in brightness creates hard edges between the screen and the background. If we can detect these edges, we are one step closer to detecting the screen.

Another property of the screen is that it is shaped like a rectangle in a perspective view. We know that the edges of a rectangle consist of straight lines. The angle between the edges of a rectangle normally is $\frac{1}{2}\pi$ rad, but the perspective that is produced by the placement of the camera can deform the angles.

3.3.2 Algorithm

We want to programmatically use the properties of the screen that are described above to automatically detect the screen. The difference in brightness between the screen and the background can be used to apply edge detection. An edge detection algorithm like Canny edge detection [6] can be used. The algorithm by Canny can be setup to only detect the hard edges and ignore the weaker edges.

If we apply the Canny edge detection algorithm to a captured image of the screen, we get the image in Figure 3.4. As can be seen, the detected edges are not necessarily straight lines. The rectangle we want to find consists of straight edges, therefore finding lines in the detected edges is the next step.

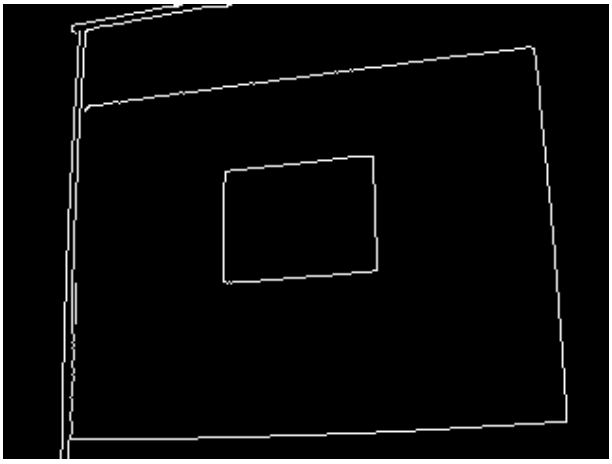

Figure 3.4: Canny edge detection to detect hard edges

The Hough transform algorithm [7] is able to find objects with a certain shape in an image that contains an approximation of such an object. This means that the algorithm can find a line in an image that contains an approximation of a line. If we look at Figure 3.4, we can see that the edges form approximations of lines. If we apply the Hough transform algorithm to the image of the detected edges, we are able to detect the lines of the rectangle.

Lines that are detected by the Hough transform algorithm are infinitely long. We need to process these lines in order to detect the screen. Figure 3.5 shows the result of applying the Hough transform algorithm to the image of the detected edges. We assume that no long edges and lines can be detected in the screen itself and that the screen is in the center of the image. Using this assumption, we can define the smallest rectangle in the center of the image, bigger than a certain minimum, that consists of the intersection of the lines produced by the Hough transform algorithm to be the rectangle of the screen.

We now made an abstract version of the original image of the screen. What is left is an image

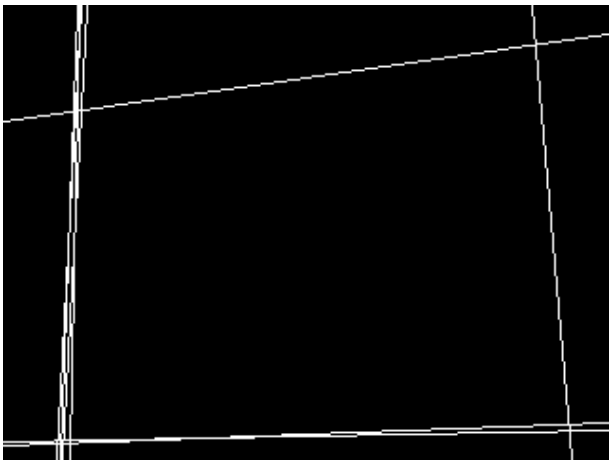

Figure 3.5: Hough transform algorithm applied to detected edges to find lines

containing lines only, see Figure 3.5. Using the assumptions described above, we can find the corner points of the screen by calculating the intersections of the lines mathematically or by processing the resulting image to find polygons that comply with the properties of the screen. The algorithm to find the screen can be formulated as follows.

Find Screen:
1. Get a frame from the camera
2. Apply Canny edge detection to find the hard edges
3. Apply the Hough transform algorithm to find lines in the edges
4. Find a minimal rectangle in the detected lines, bigger than a defined minimum
5. Extract the four corner points and check if the angles are valid
6. Return the four corner points

The results of the algorithm described above applied to the image in Figure 3.3 can be seen in Figure 3.6. The corners of the screen are found and visualized in a frame from the camera.

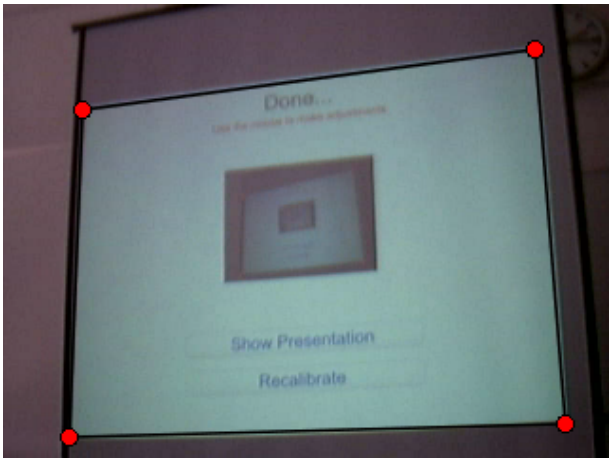

Figure 3.6: Applying the Find Screen algorithm

3.4 Screen Mapping

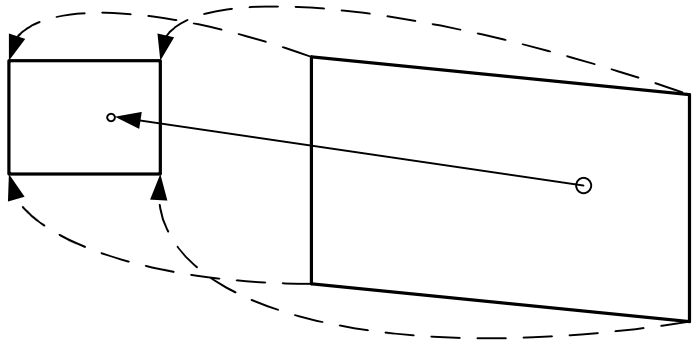

Figure 3.7: Mapping of the projector screen (right) and laser to the actual screen that is being projected (left)

When we can detect the laser and the projector screen, we need to find a correspondence between the screen that is being projected and the projector screen. Using this correspondence we can determine the position of the laser in the screen being projected. Finding this correspondence is described in a paper by Laberge et al. [2].

In the paper they use a homography to map the projector screen on the camera image to the original screen that is being projected, see Figure 3.7. Equation 3.1, a corrected version from [2], shows how the homography works.

$$\begin{bmatrix} x'w \\ y'w \\ w \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.1}$$

H is a homography matrix that can be calculated using the corner points of the projector screen and the screen being projected as is described in [2]. The vector on the right side of the equation represents the x and y coordinates of a point on the projector screen. Once the homography matrix is multiplied with the vector, you get the vector on the left side of the equation. x' and y' represent the coordinates of the point on the projector screen mapped on the projected screen. Getting these coordinates out of the vector is trivial.

Now it is clear that a detected laser point in an image from the camera can be mapped to the screen being projected using the homography. Once we have the projected coordinates, we can use them to interact with the presentation software.

3.5 Calibrating the Camera

It would be ideal if we did not have to change the settings of the camera at all and skip the calibration step, but one of our goals was to be able to use a simple webcam as our input device. Cameras with a higher resolution and better optical sensors can give a wider range of colors and luminosity which enables detecting the laser more easily in a camera with

default settings. A system that uses high-end cameras with a high resolution and framerate and provides a smooth interaction is described in [8]. A webcam usually has relatively poor optical sensors with a small range of colors and luminosity. Section 3.2.2 shows the typical images that a webcam produces. It also shows the importance of calibrating the camera. We need to standardize the frames of the camera to enable our detection algorithm.

In order to calibrate the camera, we had to setup the camera with a high shutter speed and a low gain setting. Our environment ensures us that we can manually control these settings of the camera, but it does not ensure that we can control these settings using our own applications. Changing these settings might only be possible by the user, using a control panel.

Another problem that we encounter is in determining the ideal value for the shutter speed and gain setting. We know that we need a high shutter speed and a low gain setting, but we do not know what kind of ranges of these settings produces the most reliable detection rate. The problem is worsened by the ever changing lighting conditions in different rooms. This means that a perfectly calibrated camera in one room can produce unreliable detection rates in the next room.

Combining these problems gives us the following considerations. The settings of the camera can only be adjusted by the user, while we need to adjust the settings of the camera to standardize the frames, but we do not know the ideal values of these settings. Furthermore, to measure the detection rate of the laser at a specific setting of the camera, we need to have more information about the laser than the detection algorithm can supply.

The problems and considerations we encounter do not lead to satisfactory solutions. If the settings of the camera can be controlled by our applications, a calibration algorithm can be developed that calibrates the camera with the help of the user. Without control of the settings, the user can only be instructed to adjust the settings of the camera to use a high shutter speed and a low gain. The user can be aided in this manual calibration process by giving visual feedback while settings are adjusted, but this does not necessarily lead to optimal results.

Chapter 4

Interaction

4.1 Introduction

Interaction with a computer usually consists of pressing a button on the keyboard or clicking with the mouse. These types of interaction generate well-defined interaction events. A laser pointer is not designed as an input device and only has two states. It does not generate well-defined interaction events. The laser can be switched on or off. In combination with a camera we can calculate the position of the laser, as we have shown in Section 3.4.

The laser and the camera provide us with the following information:

- State of the laser: on or off
- Position of the laser relative to the screen
- Changes in the state of the laser and the position of the laser over multiple frames of the camera

We only have access to this information and have to use it to enable interaction with applications using the laser. In the next sections we discuss how we use this information to allow interaction and what problems are encountered.

4.2 Overloading the Laser

We want to enable multiple types of interaction with the laser at the same time. Gestures and dwelling of the laser, as we discuss later on, and the position of the laser can be used to interact with our application. This means that we have to overload the function of the laser and use the information we have to simultaneously detect multiple ways of interaction.

If we only had to detect a single type of interaction with the laser, we could create an algorithm that only needs to classify the information we have about the laser as belonging to an interaction event or not, a binary choice. Multiple types of interaction makes it harder for an algorithm to classify the information and makes it more prone to false positives and false

negatives. If the wrong type of interaction is detected, unexpected things might happen and the application loses some of its value.

It might be the case that some of the interaction events do not interfere with each other, while others can only operate mutual exclusive. For instance, a gesture in the form of a horizontal line and a gesture in the form of a vertical line are mutual exclusive, but the vertical line gesture does not interfere with the raw position of the laser as an interaction event. Events that are mutual exclusive need to be differentiated as much as possible, to enable a smooth classification of the interactions. Differentiating the events can be done by increasing the minimum duration of the events before they are detected. A shorter minimum duration of an event makes the responsiveness of the application better, so differentiating the events results in a less responsive user interface, but it enables more ways of interaction and removes false positives and false negatives.

4.3 Detecting Interaction Events

The next step in enabling interaction with the laser is to use the information we have about the laser and find a way to detect interaction events. At every frame of the camera, we get a packet of information about the laser. A preprocessing step can be applied to smooth out the position of the laser by applying averaging. The flow of information is naturally represented by a stream of packets. The packets contain the position of the laser on the screen if the laser is turned on, and the state of the laser. Figure 4.1 shows the flow of information as can be represented by a stream. Packets arrive at the left side, while the current packet of information is present at the right side.

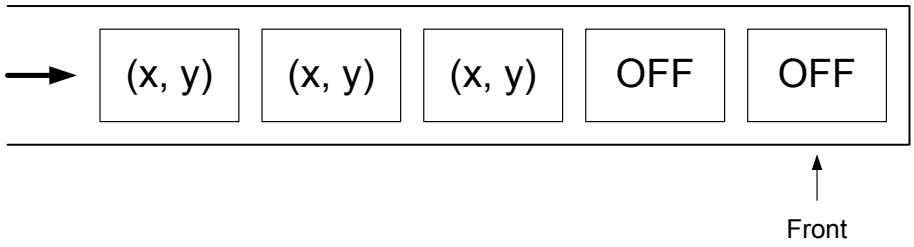

Figure 4.1: Flow of information

The problems encountered when detecting interaction events in the stream of information have been discussed by Cheng and Pulo [3]. The stream is continuous, but the interaction events are discrete. This means that we have to detect when an event begins and ends in the continuous stream. Cheng and Pulo wanted to detect gestures, but reasoned that a gesture detection library would be needed. The libraries cannot handle the continuous stream of data, therefore they have designed a simpler interaction event. They emulate a circular gesture by first dividing a bounding box into four parts. When the laser is pointed to the box and moved in a clockwise motion onto all of the parts of the box, an interaction event is detected. This emulated circular gesture asks to place interaction-boxes on the screen and is a very specific solution.

An other interaction event Cheng and Pulo [3] suggest is to allow navigation using a screen-

wide sweeping gesture. Again, they say it is infeasible because a gesture library is needed, but you do not know when a gesture starts or ends. We do want to design our interaction system with support for arbitrary and screen-wide gestures, without relying on a gesture library.

In the remaining of this section, we propose a possible solution to the problems discussed by Cheng and Pulo [3] and show a generalized algorithm to detect gestures using the stream of information as input.

4.3.1 Structuring the Stream

We need to find structure in the continuous stream of information, as is visualized in Figure 4.1. Finding structure amounts to finding the beginning and ending of an interaction event, as well as throwing away useless information that is not part of an interaction event.

We can find structure in the stream by first observing that an interaction event can only start at some time after the laser has been turned on and the event has certainly been ended when the laser is turned off again. This structuring of the stream of information generates prefixes of the stream of finite length, if we assume that the laser is turned off somewhere in the future. The prefixes consist only of coordinates of the laser in consecutive frames. Structuring the stream with prefixes can be seen in Figure 4.2.

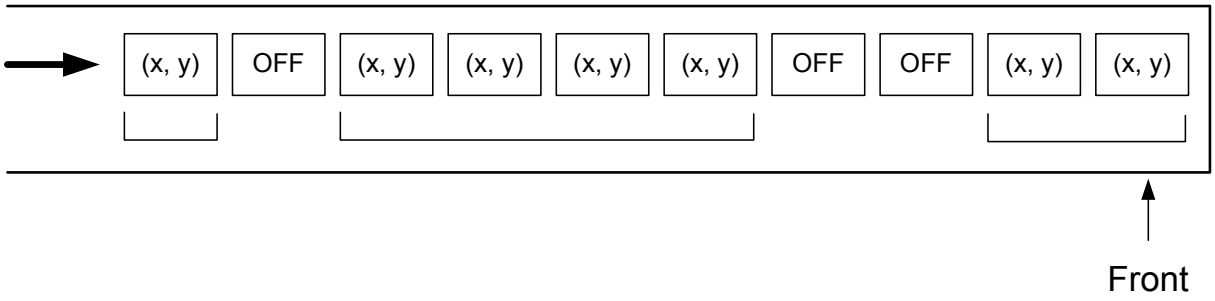

Figure 4.2: Structuring the stream with prefixes of coordinates

Figure 4.2 only shows a theoretical structuring of the stream of information, as if we could see in the future. If we are at the position in time indicated by the *Front* keyword, every arrival of a new packet of information can either increase the prefix with an extra coordinate or end the prefix. The structuring of the stream gives us a window in which an interaction event might happen. The window opens when the first coordinate arrives after the laser has been turned off and as the prefix is increased by the arrival of new packets, the window is increased.

A prefix of the stream contains more information than only the coordinates of the laser. We know the time between the packets of information, but we can derive more properties if we look at consecutive packets. If we assume that the prefix of the stream contains more than one packet, we can transform the prefix into a richer string of information, that contains derived properties about the position of the laser and the relation of a coordinate to the previous coordinate. The richer string has one less element than the prefix of the stream, in order to enable the calculation of the derived properties. The properties that can be derived when

looking at the current and previous coordinate of the laser and at the arrival time of the packets are as follows.

Derived properties:

- Current coordinate
- The distance between the current and the previous coordinate in the x and y direction (xdelta and ydelta)
- The angle between the current and the previous coordinate
- The speed of the current coordinate, using the arrival time

4.3.2 Generalized Gesture Detection Algorithm

The transformed prefix with derived properties gives us enough information to develop a generalized gesture detection algorithm. The algorithm has as input the richer string of information and needs to detect if an interaction event has happened in the string. Every time the length of the prefix increases, the algorithm needs to be run again.

The gestures we want to detect must be expressible in the derived properties of the prefix. Such a gesture that is present in the transformed prefix is visualized in Figure 4.3. As can be seen, we do not know the start and end of a possible gesture, but we do know how long the current transformed prefix is. Finding the gesture in the transformed prefix amounts to finding a substring that conforms to the correlations between the derived properties of consecutive packets as is defined belonging to the specific gesture. We define that only one gesture can be detected in the prefix, so once we have found a gesture, the algorithm can stop and return a result. This means that we only have to look for a single gesture in suffixes of the prefix of decreasing length. If the transformed prefix has length N, there are N suffixes of the prefix, so there is a maximum of N iterations needed to detect if a gesture is present.

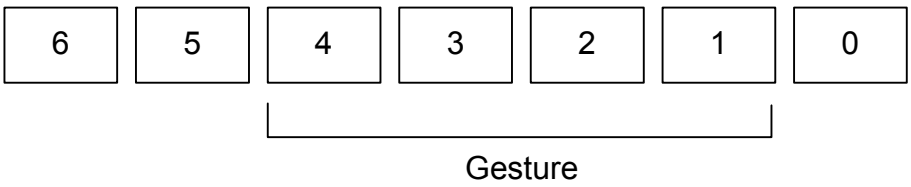

Figure 4.3: A gesture in the transformed prefix

Based on the discussion above, we can design an algorithm that can detect a gesture in a transformed prefix, if it is present.

Generalized Gesture Detection Algorithm:
1. Get the current transformed prefix
2. suffix = prefix
3. **while** gesture not found and there are suffixes left
4. Check if there is a prefix of the suffix that conforms to the properties of the gesture

5. Decrease the length of the suffix
6. **end**

The algorithm in step 4 of the generalized algorithm needs to be designed for each gesture we want to detect. The algorithm gets a suffix of the transformed prefix and must assume that the gesture starts at the beginning of the suffix. If the individual packets in the suffix conform to the properties of the gesture, but the gesture needs more packets, we have only detected the beginning of a gesture. It does not matter what we do with this information, because once new packets arrive and the transformed prefix gets bigger, the generalized algorithm is run again. If the packets conform to the properties of the entire gesture, we can return our results and end the generalized algorithm. The remaining information in the transformed prefix is assumed to be useless and a new prefix can be built from newly arriving packets of information.

Step 4 can be implemented by a special gesture library to allow complex gestures. It can also be implemented without a library to detect a specific gesture, but some flexibility and quality of the gesture detection might be lost. In the remaining of this chapter we present some manually designed algorithms to detect a couple of simple gestures, lines and dwelling.

4.3.3 Lines

We want to detect gestures that are in the form of a line. Horizontal lines, vertical lines and diagonal lines all belong to this category. In order to detect lines, we create a general line detection algorithm. The algorithm implements step 4 from the Generalized Gesture Detection Algorithm described in the previous section. The algorithm gets a string of packets that contains the derived properties as are described before.

In the algorithm below, the different linetypes can be characterized using two parameters. We define an x- and a y-direction. For instance, if we want to define a horizontal line that runs from left to right, we set the x-direction to 1 and the y-direction to 0. Similarly, if we want to define a diagonal that runs from right to left and from top to bottom, we set the x-direction to -1 and the y-direction to -1.

Generalized Line Detection Algorithm:
1. **while** gesture not found and packets left
2. Check if xdelta and ydelta of the packet conform to the x- and y-direction of linetype
3. If a minimum amount of packets conform and the length of the line is bigger than a minimum, the gesture is found
4. **end**

4.3.4 Dwelling

As described in [1], dwelling of the laser means to hold the laser in a single spot for a period of time to induce an interaction event. Dwelling can be seen as a sort of gesture. The gesture does not contain a lot of movement, but it does span multiple frames of the camera. Again, we implement step 4 from the general gesture algorithm.

Dwell Detection Algorithm:
1. **while** gesture not found and packets left
2. Check if the distance to the previous point is small enough
3. if a minimum amount of packets conform, the gesture is found
4. **end**

Chapter 5

Design

5.1 Introduction

One of the goals of our project is to design and implement our ideas into a working application. This application allows us to evaluate the solutions and see if the system we propose meets our expectations. To create our application, we use a form of rapid prototyping. We iteratively design and implement new features in the application to rapidly create working results.

In the design stage, we determine the high level requirements and create an architecture. The structure of the architecture determines how flexible the application is in terms of feature expansion and maintainability.

5.2 Requirements

We already discussed the goals and solutions in Chapter 2. They will be a basis for the requirements of the application. The requirements determine for a great part the structure of the program. Below we define the requirements of the application.

High level requirements:

- Define a presentation format with support for text, images and transitions
- Create a graphical user interface that enables displaying the presentation and allows for interaction using a camera and a laser
- Interface with a camera and retrieve images
- Process images from the camera and detect the position of the screen and the laser
- Enable highlighting of elements of a presentation using laser interaction
- Enable navigation controls using laser interaction
- Define an overview screen of slides that is navigatable

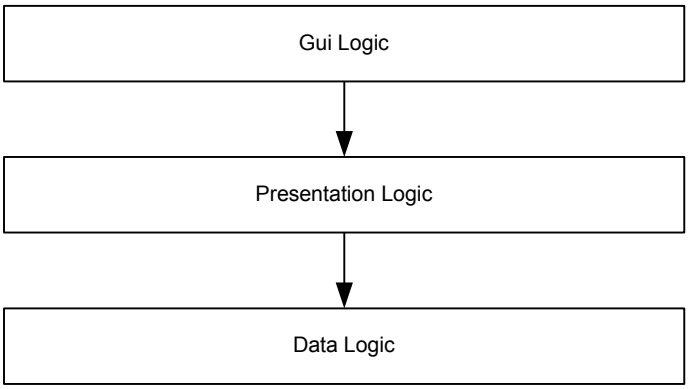

Figure 5.1: Layers of the architecture

5.3 Architecture

Now that we know the high level requirements, we can define structure in our application. The requirements guide us in defining three architectural layers. The layers divide the application in three units of logic with specific access rules. The rules state that a layer may only access the operations of the layer directly below itself. This implies that a layer is only dependent on a maximum of one other layer and that a layer is coupled to a maximum of two layers. Because of the low amount of coupling between layers, the architecture can be easily modified and maintained, while individual layers can be expanded with functionality without much change in the architecture.

We define the following layers: Data Logic, Presentation Logic and Gui Logic. We arrange them as can be seen in Figure 5.1. The lowest layer, Data Logic, does not depend on any other layer directly. Its purpose is to manage the data of a presentation in the context of the filesystem. The layer in the middle, Presentation Logic, uses functionality from Data Logic and manages the presentation from a higher level of abstraction. It is responsible for presenting the presentation on the screen. The highest level, Gui Logic, manages parts of the user interface and handles interactions between the user and the system.

We have seen a high-level overview of the architecture of the system. What follows is a more in depth look at the individual layers of the architecture.

5.3.1 Data Logic

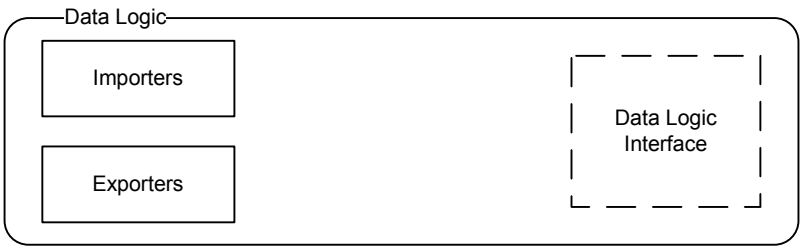

Figure 5.2: Data Logic layer

The function of the Data Logic layer is to handle all of the low-level operations to load and save presentations. It only looks at a presentation as a string of characters with some structure, therefore it does not need to know the exact meaning of the information that is encoded in the data. The meaning of the data is used in the layers higher up in the architecture.

As can be seen in Figure 5.2, the layer consists of two components of functionality. There are importers and exporters. An importer reads an external presentation that has some data format and converts it to an internal presentation format. An exporter starts with the internal presentation format, but can write the presentation to some other format. Any existing presentation format can be supported, as long as the appropriate importer or exporter is defined. We need to define the internal presentation format that is used by the importers and exporters.

The design of the internal presentation format determines how complex the presentation can be in terms of features. In the requirements we have decided to create a relatively simple presentation format. The format we define has the following features:

- Paragraphs of text with a specific color and optional bullets on a slide
- Images on a slide
- Transitions between slides

These basic features allow for presentations to be created that are appealing to an audience.

5.3.2 Presentation Logic

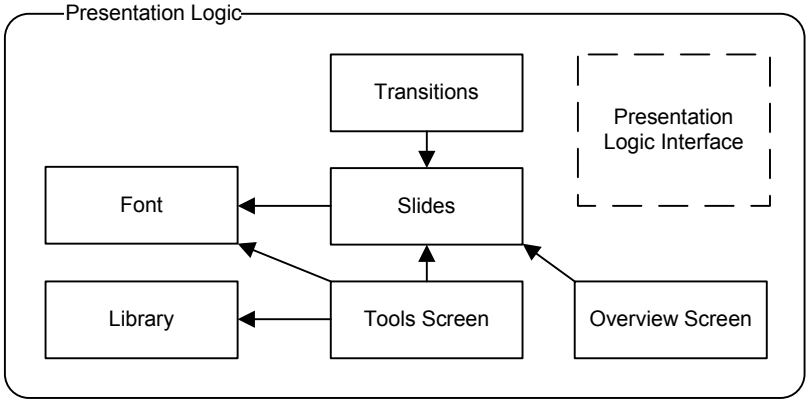

Figure 5.3: Presentation Logic layer

The Presentation Logic layer is responsible for presenting the presentation on the screen. It uses the presentation data from the Data Logic layer and looks at the data from a higher point of view. Meaning is extracted from the data to form the actual presentation that will be shown on the screen. The internal representation of the presentation in this layer can be seen in Figure 5.4. It shows the presentation format with some abstraction, without defining attributes.

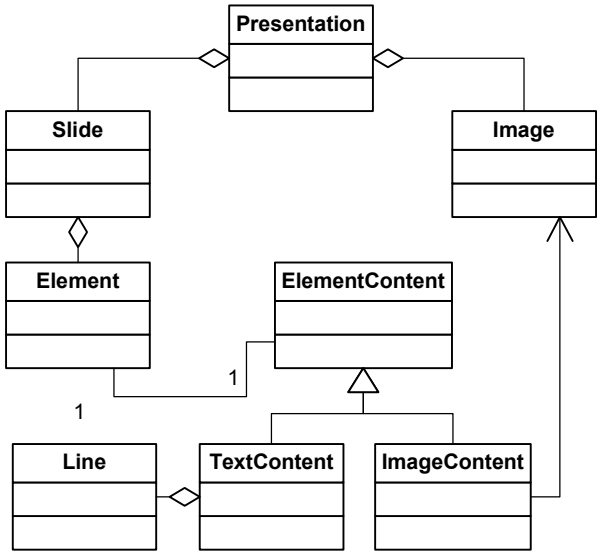

Figure 5.4: High level presentation format

The diagram shows that a presentation can consist of slides and images. A slide can contain elements, while the elements have a content. This content can be textual in the form of lines of text or graphical in the form of an image. The requirements of the presentation format can be expressed in the high level presentation format from Figure 5.4 by defining attributes in the appropriate classes.

The Presentation Logic layer is not only responsible for presenting the presentation, it has more functionality. The components of the Presentation Logic layer can be seen in Figure 5.3. The functionality of the components can be summarized as follows:

- Display the presentation
- Manage slides
- Manage transitions
- Manage internal graphics in the library
- Display and manage the overview screen and the tools screen

The overview screen is described in section 2.4. The tools screen makes it possible to set an interaction mode, like enabling the interaction mode in which the user can highlight parts of a slide.

5.3.3 Gui Logic

The final layer, the Gui Logic layer, handles interactions and displays elements of the graphical user interface. Interactions are always between the user and the application, but the way in which this interaction happens differs. There are standard input devices like a keyboard and

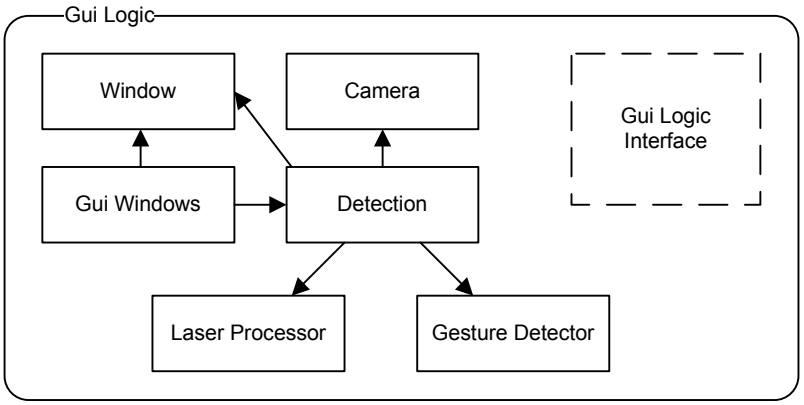

Figure 5.5: Gui Logic layer

a mouse, but we also want to support the alternative input method described before that uses a camera and a laser pointer.

Figure 5.5 shows the components of this layer. Interactions are made possible by interfacing with a camera to get the frames. The frames are processed in order to find the projection screen and the laser point. Once the position of the laser is detected in a couple of frames, gestures are detected in the sequence of detected laser points.

The usual window interactions are made possible by displaying windows that contain interactive widgets, like buttons or a list of items. These so called gui windows are used to load presentations, setup the presentation and display the presentation.

5.4 User Interaction Design

Before the application can display a presentation, we need to complete a few steps. The camera needs to be calibrated, the presentation needs to be loaded and further calibrations might be needed. We only want to enable the interaction using a laser when an actual presentation is displayed. The steps before reaching the actual presentation can be handled using traditional interactions. Once the presenter is presenting, the application should be fully controllable by the laser.

During a presentation we do not want to cloud the user interface with navigational controls. A slide has the size of the entire screen and we do not want to distract the audience by displaying elements of the user interface. When we are in a special menu however, it is acceptable to display visual interaction controls.

We use gestures as are described before to keep the user interface clean and allow the user to interact with the application. In special menus, an analog to the push-button is used that can be activated by dwelling, a form of clicking with the laser, as is described in [1].

5.4.1 Navigation and Menus

We define the gestures as are described in figure 5.6. All of these gestures can be applied when a slide is displayed in full screen. Gestures in the form of diagonal lines allow us to navigate the presentation and give us the option of displaying a transition between slides. A horizontal line in a specific location opens up a menu or the overview screen.

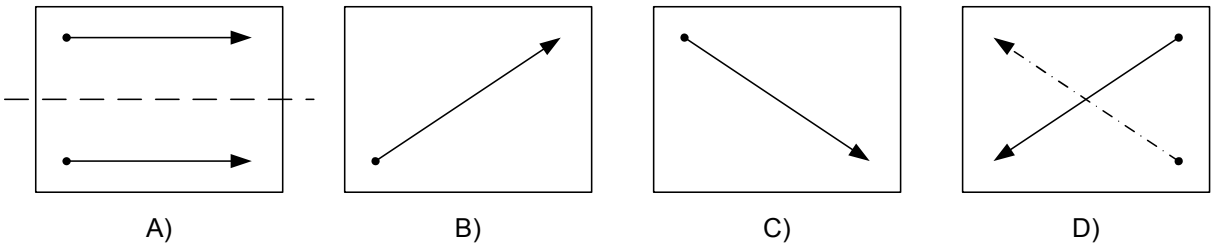

Figure 5.6: Gestures: A) Open options menu and overview screen, B) Next slide, with possible transition, C) Next slide, without transition, D) Previous slide

In a similar way, we can define interactions for menus. A horizontal line can be used to go to a menu, but we can only use it to close a menu. Alternatively, we can add push-buttons that can be clicked using a dwell action that might close the menu.

A menu can be seen as a set of widgets on the screen with a specific location, so using screen-wide gestures to interact with the widgets is not a good idea. We suggest to interact with widgets using the raw position of the laser or a dwell action.

5.4.2 Additional features

Highlighting and sketching are two additional features that are enabled by the camera and laser interface. We interact directly with the projector screen and this allows us to point to a specific element of the presentation and let our software highlight it. We can also use the information to sketch on the screen to let the presenter add notes or emphasize a specific element.

Enabling highlighting and sketching is straightforward, as we do not need to detect any gesture. We only need to know the position of the laser relative to the presentation and, in the case of highlighting, find the element of the presentation at the position of the laser.

Many more features can be designed using the laser interface, such as zooming in on a slide or using the laser as a document lens. We have chosen to support highlighting and sketching, but as is clear, direct screen interaction from a distance enables rich ways of interaction.

Chapter 6

Implementation

6.1 Introduction

We have implemented the design of our system in a working application. It contains all of the core features as are suggested in the design and enables a user to present a presentation to a real audience. The application works as a proof-of-concept and allows us to evaluate our propositions.

6.2 Camera

The camera we use in our setup is a Philips ToUcam PRO II. It is a webcam with support for resolutions up to 640 by 480 pixels and can deliver frames at 60 frames per second at certain resolutions. As is described in the environment, we set the camera to use 320 by 240 pixels at 30 frames per second to make the environment compatible with more low-end webcams. The settings of the camera can be changed manually in a property-sheet.

The smoothness of our method of interaction greatly depends on the combination of the resolution of the camera and the frames per second. A higher resolution gives us more detailed frames which allows us to find the screen and the laser more easily. A higher framerate allows the detection of gestures to become more reliable, because there is more data available in the same amount of time. Increasing the sampling rate allows us to detect smaller amounts of motion of the laser, thus making it easier to differentiate and classify the different gestures.

A system that uses high-resolutions cameras with a high fps is described in [8]. They use multiple cameras with a resolution of 1600 by 1200 pixels that have a framerate between 90 and 120 frames per second. With this setup, they have created a system that is very responsive and accurate. It should be obvious that our low-end setup cannot reach the same responsiveness and accuracy, but it might give satisfactory results.

6.3 Libraries and Platform

The application we have developed is implemented in the language C++ and is compiled on the Microsoft Windows platform. It does not really depend on the operating system, as the implementation uses mostly cross-platform libraries that makes the source code platform independent. Compilation on different platforms like Mac OS and Linux would only need minor adjustments to the source.

We use the following libraries:

- SDL
- OpenGL
- FTGL
- OpenCV
- videoInput
- Boost
- TinyXml

SDL creates a window that is used by our application. Graphics are drawn on the window by the graphics library OpenGL. FTGL is a library that uses the FreeType2 font library to draw pieces of text in an OpenGL context. We use OpenCV to apply our image processing algorithms and use it on platforms other than Microsoft Windows to interface with the camera. On Windows, we use the videoInput library to interface with the camera, because it allows the camera to use a higher framerate than when only using OpenCV. The videoInput library is platform dependent, but can be omitted on non-Windows platforms.

The presentations that our application supports are in the XML format. To read these files, we use the Boost filesystem library and we use TinyXml to parse the XML structure.

Chapter 7

Evaluation

7.1 Introduction

We have evaluated the concepts in our system using a hallway usability test as discussed by Nielsen [9]. Such a test consists of a handful of participants that are asked to follow a set of instructions and fill out a questionnaire. Only a handful of participants are needed to find most of the problems in the system.

The setup of the usability test as well as the results are discussed below.

7.2 Evaluation Setup

Four people that work or study at the university were asked to follow a set of instructions after being briefly instructed on how to use our system. The instructions cover all of the concepts of the application and users had to fill out a questionnaire afterwards to evaluate their experience with the system.

The following types of instructions were given:

- Use the interactive pointer to highlight parts of a slide and sketch on a slide
- Navigate through the presentation using gestures
- Interact with the overview screen

The environment was setup as is explained in Figure 1.1. The application was coupled to a projector in a darkened room using a webcam and a laser pointer. A real presentation was displayed that consisted of a number of different slides.

The usability test we performed delivered answers to our questionnaire and some additional discussion. The results of the test are interpreted and summarized below.

7.3 Results

7.3.1 Interactive Pointer

Changing from the passive laser pointer and the regular pointing stick to an interactive laser pointer is seen as very useful by all participants. The interactive pointer can highlight parts of the presentation using visual changes on the screen. Such a change on the screen is much clearer to the audience than a small laser dot or a pointing stick. The fact that you can highlight something on the screen at random also means that you do not have to prepare the slides of your presentation to add certain highlights in a specific order.

A difference between the pointing stick, the regular laser and the interactive laser is that the interactive laser can be used to sketch on the slide. Users of the system see this as very useful and an improvement over the regular pointing devices, because sketching adds more insight than highlighting alone. Just as with highlighting, sketching does not need modifications to the presentation itself. This adds flexibility to the system.

Making the pointer interactive and allowing a presenter to highlight and sketch is the highest rated feature of the presentation system. Participants think that this added interactivity of the presentation has the potential to improve the quality of the presentation. Furthermore, using a laser pointer for these kinds of interaction seems to be a natural choice.

7.3.2 Navigation

The presentation can be navigated using screen-wide gestures. All of the gestures are some type of line, like a horizontal line or a slash. The gestures need to be performed between certain speed limits for the detection algorithm to pick it up. So, in order to perform a gesture, a couple of limitations need to be kept in mind.

Even though there are limitations, navigating the presentation using gestures was surprisingly easy for most people. After only a brief instruction, all participants were able to use the gestures just learned consistently to navigate without much error. Going to the next slide, previous slide and opening menus was easy to accomplish. There is no need for more complex gestures, because the gestures that are present seem to be sufficiently effective to control the application.

For some, the gestures seem a bit too large and slow. They are screen-wide and performing such a gestures takes about two seconds, which causes a less responsive feel. Some want an instant response using a fast gesture. Using gestures as a means of navigation cannot compete with using a single push of a button for navigation that does deliver an instant response, therefore a remote control is more desirable than a gesture interface for navigation.

7.3.3 Overview Screen

The overview screen displays the current slide, as well as some of the previous slides and the next slides in a single image. The screen can be navigated by going to one of the previous or next slides using a scrolling paradigm. The screen is animated accordingly to see where the slides are going during navigation.

The overview screen works well, compared to the regular way of finding slides using fullscreen navigation through the slides. Especially the layout of the overview screen, containing previous and next slides, helps finding the slide one is searching for. It is believed that the overview screen is also beneficial for the audience of a presentation when asking questions, because they can see the context of the current slide.

The mechanics of the overview screen as well as the visual representation of the overview screen seem to be appealing and intuitive to the users of the system. The overview screen is preferred over the regular way of finding slides by navigating through the set of slides.

Chapter 8

Conclusions

We have created a system that allows a presenter to give a presentation, using only a laser pointer to interact with the presentation software. The interactive laser pointer allows for new ways of interaction when compared to the traditional interaction methods using a keyboard and a mouse. The traditional methods are not really fitting for a presenter, as a presenter is not sitting in a comfortable chair at a desk, but standing in front of an audience. Our system solves this problem by allowing the presenter to interact with the presentation software directly, instead of walking to the computer and issuing commands there.

Besides allowing a presenter to control a presentation from a great distance from the computer, the new ways of interaction enable or enhance additional features of the presentation software. Direct screen interaction with the laser pointer is much more natural than using a mouse to target something on the screen, because the mouse works on a representation of the actual screen. Allowing the highlighting of parts of the presentation really adds to the quality of the presentation, because it is visually more clear what a presenter is pointing to in comparison to a standard laser pointer or a pointing stick. Sketching on the slide to explain something or to add a note adds flexibility to the presentation and has the potential to improve the quality of the presentation.

Relatively easy gestures can be detected reliably in the consecutive positions of the laser pointer on the screen. The enabling of gestures overloads the function of the laser and allows for a richer form of interaction when compared to only using a single position of the laser, as is used in highlighting and sketching. Gestures can be used to navigate through the presentation and open or close menus for example. The setup of the environment gives us a camera with a relatively low resolution and a low framerate. We only have 30 frames per second and we do not want a highlight or sketch action to induce false detections of gestures, therefore to reduce false positives, the number of points needed per gesture is increased. Needing more points per gesture limits the flexibility and the responsiveness of our gestures. More complex and distinct gestures need even more sampling points to be able to be classified reliably. Gestures in general need to be detected and classified reliably with a low rate of false positives, which means that there should be enough sampling points available before we can reliably classify such a gesture. Needing a high number of sampling points while knowing that we only have 30 samples per second implies that the duration of our gestures must be relatively long. Although the limitations of the hardware and the overloading of the laser make it harder to

create a responsive user interface, our system can be used to navigate through a presentation using gestures in a reliable way. The responsiveness might be too low for some people who want an instant response and do not want to perform a gesture for two seconds.

The overview screen we have created solves the problem of finding slides without having to exit the presentation and without possibly confusing the audience by displaying fullscreen intermediate slides that have no relevance in the desired context. The overview screen is integrated in the presentation and is visually represented differently than a regular slide. The presenter as well as the audience can see what the context is of the current slide and the presenter can use this information to find a previous or next slide without having to know the exact position of a slide in the set of slides. The visual representation of the overview screen in combination with the mechanics that allow for navigating the screen works well and is appealing to the audience.

Our system implements the solutions we have proposed and designed using a low-end setup that uses readily available commodity hardware. Although this hardware does limit our solutions in terms of responsiveness and flexibility of some of the interactions, in general our system is satisfactory and potentially improves the quality of a presentation. Especially highlighting and sketching are highly rated, while the overview screen is very effective and appealing when compared to the traditional way of finding slides.

8.1 Future Work

We have designed a way to detect gestures in the stream of frames from a camera. The method works well, considering the amount of sampling points we get from the camera. The actual algorithms to detect a specific gesture are not optimal and can be improved upon. Such an algorithm aims to detect a gesture perfectly without any false positives and false negatives. The occurrence of false positives in a presentation is generally considered more disruptive than the occurrence of false negatives and hence need to be avoided stronger, possibly at the cost of getting more false negatives. Therefore our algorithms need a relatively high number of sampling points to classify a gesture reliably. The gesture detection algorithm can be improved upon so that the number of sampling points per gesture can be reduced, which increases the responsiveness of the interactions. Finding the minimal amount of sampling points needed to detect a gesture reliably is challenging.

Our evaluation of the system is rather informal by using a hallway usability test. This gives us an indication of the usability of the system and problems that the users encounter can be spotted directly. A comparison between the traditional system and the new system can be made mathematically if we use quantitative methods. Detection rates, error rates and the timing of the system can be measured and calculated. Quantitative methods will allow to draw better and more decisive conclusions in comparison to the more informal hallway usability test.

The design of our system and the prototype application serve only as a proof-of-concept, therefore they only show a limited amount of features. The laser interface enables a more direct interaction with the system when compared to the traditional input devices. The feature set that is enabled with this new way of interaction can be explored further in the context of presentation software. One might think of zooming in on slides using the laser.

Another feature is to add more dynamic and real-time interactions to the presentation, which will enable the repositioning of elements of the presentation during a presentation. All of these features remove the need to statically alter the presentation beforehand and enable a more dynamic way of giving a presentation without the time consuming preparations.

Bibliography

[1] Jr. Dan R. Olsen and Travis Nielsen. Laser pointer interaction. In *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 17–22, New York, NY, USA, 2001. ACM.

[2] Dominic Laberge, Jean-François Lapointe, and Emil M. Petriu. An auto-calibrated laser-pointing interface for large screen displays. In *DS-RT '03: Proceedings of the Seventh IEEE International Symposium on Distributed Simulation and Real-Time Applications*, page 190, Washington, DC, USA, 2003. IEEE Computer Society.

[3] Kelvin Cheng and Kevin Pulo. Direct interaction with large-scale display systems using infrared laser tracking devices. In *APVis '03: Proceedings of the Asia-Pacific symposium on Information visualisation*, pages 67–74, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[4] Microsoft Powerpoint
http://office.microsoft.com/powerpoint.

[5] Jock D. Mackinlay, George G. Robertson, and Stuart K. Card. The perspective wall: detail and context smoothly integrated. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 173–176, New York, NY, USA, 1991. ACM.

[6] J Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, 1986.

[7] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, 1972.

[8] Werner A. König, Joachim Böttger, Nikolaus Völzow, and Harald Reiterer. Laserpointer-interaction between art and science. In *IUI '08: Proceedings of the 13th international conference on Intelligent user interfaces*, pages 423–424, New York, NY, USA, 2008. ACM.

[9] Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann, 1994.

Appendix A

User Notes

This document describes the steps necessary to make the application work. It is not an in-depth manual but it helps in setting up the environment and executing the application.

A.1 System Requirements

The application is know to function on a single core Pentium computer with 1.4 Ghz on Microsoft Windows XP. No advanced graphics card is necessary. Your graphics card should be OpenGL 1.2 compatible.

A.2 Setup

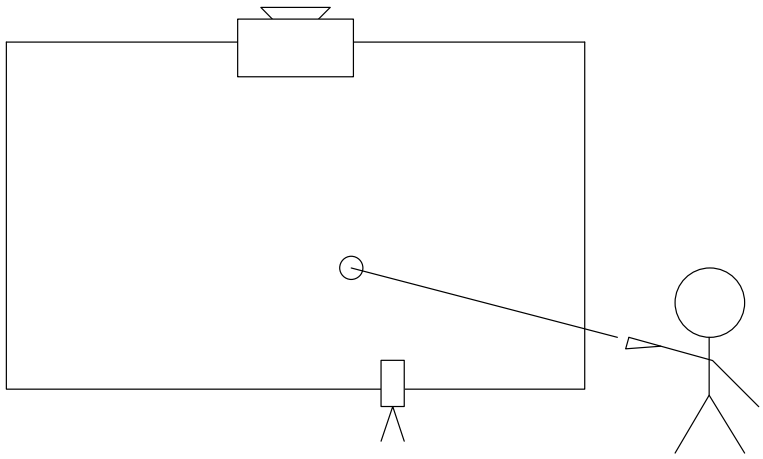

Figure A.1: Setup of the projector (top), the screen, the camera (bottom) and the presenter with a laser pointer

The application works best in a darkened room using an LCD projector. Some DLP projectors cannot be used, because they interfere with the images from the camera. A webcam is needed

that supports a resolution of 320 by 240 with 30 frames per second. The camera should have manually changeable shutter speed and gain settings. Finally, a laser pointer is needed.

The webcam and the projector should be arranged as can be seen in Figure A.1. The camera should be pointed to the projector-screen. Both the projector and the camera should be connected to a computer.

A.3 Application

After the projector and the camera are connected to the computer, the application can be started. Once the application is executed, the settings of the camera can be adjusted in the property-sheet that pops up.

Set the gain of the camera to a low settings, something like 5% should be enough. The shutter speed should be adjusted to a settings like $\frac{1}{50}$ sec. If the room is very light, the shutter speed should probably be set to a faster settings, like $\frac{1}{100}$ sec. The framerate of the camera should be set to 30 frames per second.

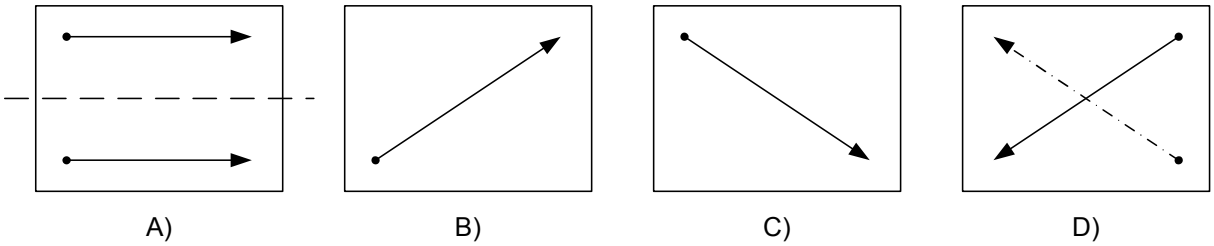

Figure A.2: Gestures: A) Open options menu and overview screen, B) Next slide, with possible transition, C) Next slide, without transition, D) Previous slide

Once a presentation is imported and the screen is detected, the presentation is displayed. The application can be set in fullscreen mode by pressing $F1$. Inside a presentation, the gestures from Figure A.2 can be applied with the laser pointer to interact with the application.

Some actions such as closing a menu can be performed using a *dwell* gesture. This type of gesture is performed when the laser is held at a specific location for a period of time.

Appendix B

Developer Notes

B.1 Libraries

Information about the libraries that are used in the application can be found at the following locations:

- SDL (http://libsdl.org)
- OpenGL (http://opengl.org)
- FTGL (http://sourceforge.net/projects/ftgl)
- OpenCV (http://sourceforge.net/projects/opencvlibrary)
- videoInput (http://muonics.net/school/spring05/videoInput)
- Boost (http://www.boost.org)
- TinyXml (http://sourceforge.net/projects/tinyxml)

It is believed that the FTGL library, that is used to display text on the screen in an OpenGL context, can cause some instability in the application. When the application is switched from windowed to fullscreen mode, SDL loses the OpenGL context and therefore all of the textures need to be regenerated. The FTGL library also needs to be re-initialized. In a specific situation, when the user switches the application from the overview screen to the normal presentation and the application has been switched to fullscreen mode, the application might become unstable. Using a different library than the FTGL library might solve this problem. Removing the need to re-initialize the textures can also be a solution, but then SDL needs to be modified, or a different library should be used instead of SDL.

B.2 Presentation File Format

The presentation file format that is supported is an XML format. Below is a description of this format. Angular brackets indicate parameters that can be filled in as desired.

Everything is contained inside a *presentation* block.

```
<presentation id="FLUiD/1.0/RDBR/2008" width="[int]" height="[int]" pagenumbers="[true|false]">
...
</presentation>
```

The presentation block can contain a number of *image* blocks.

```
<image filename="[string]" />
```

The presentation block can contain a number of *slide* blocks.

```
<slide r="[int]" g="[int]" b="[int]" transition="[none|flip|cube]">
...
</slide>
```

The slide block can contain several *element* blocks.

```
<element x="[int]" y="[int]" z="[int]" width="[int]" height="[int]" type="[text|image]">
...
</element>
```

Depending on the type of the *element*, the *element* block can contain several lines of text or an image.

Element with type of *text*.

```
<line fontsize="[int]" bullet="[true|false]" r="[int]" g="[int]" b="[int]"
        align="[left|center|right]">[string]</line>
```

Element with type of *image*.

```
<imageref index="[int(0-based)]" />
```