Electronic Research Archive of Blekinge Institute of Technology
<http://www.bth.se/fou/>

This is an author produced version of a conference paper. The paper has been peer-reviewed but may not include the final publisher proof-corrections or pagination of the proceedings.

Citation for the published Conference paper:

Title:

Author:

Conference Name:

Conference Year:

Conference Location:

Access to the published version may require subscription.

Published with permission from:

Gaze and Voice Controlled Drawing

Jan van der Kamp
Trinity College Dublin
Ireland
jan.hvdk@googlemail.com

Veronica Sundstedt
Blekinge Institute of Technology
Sweden
veronica.sundstedt@bth.se

ABSTRACT

Eye tracking is a process that allows an observers gaze to be determined in real time by measuring their eye movements. Recent work has examined the possibility of using gaze control as an alternative input modality in interactive applications. Alternative means of interaction are especially important for disabled users for whom traditional techniques, such as mouse and keyboard, may not be feasible. This paper proposes a novel combination of gaze and voice commands as a means of hands free interaction in a paint style program. A drawing application is implemented which is controllable by input from gaze and voice. Voice commands are used to activate drawing which allow gaze to be used only for positioning the cursor. In previous work gaze has also been used to activate drawing using dwell time. The drawing application is evaluated using subjective responses from participant user trials. The main result indicates that although gaze and voice offered less control than traditional input devices, the participants reported that it was more enjoyable.

Categories and Subject Descriptors

H.5.2 [User Interfaces]: Input devices and strategies

General Terms

Design, experimentation, human factors

Keywords

eye tracking, drawing, gaze based interaction

1. INTRODUCTION

Eye trackers work by measuring where an individual's gaze is focused on a computer monitor in real-time. This allows for certain applications to be controlled by the eyes which benefits disabled users for whom keyboard and mouse are not an input option. These include sufferers of cerebral palsy, motor neuron disease, multiple sclerosis, amputees and other physical paralysis. Since eye trackers give us this data, they

open up new opportunities for control. However, gaze based interaction is not without its issues.

One of the main problems involved in gaze based interfaces is that of the "Midas Touch". This is where everywhere one looks, another command is activated; the viewer cannot look anywhere without issuing a command [6]. This arises because our eyes are used to looking at objects rather than controlling or activating them [7]. When using gaze as input for a drawing program for example, this can lead to frustration as drawing can be activated without a user intending it. Previous work in gaze based drawing tools have used dwell time to activate drawing. Dwell time works by requiring the user to fixate their gaze at one point for a particular amount of time to confirm a selection. It was found, however, that this is not a perfect solution to the problem due to both the delay involved and the possibility of drawing still being activated without intent [4].

This paper proposes a novel approach of using voice commands to activate drawing. The intention is that removing confirmation of drawing from gaze data will lead to improved user experience and drawing possibilities. It should also be quicker to draw since users will not have to wait for a dwell to be picked up in order for drawing to be activated. A novel gaze and voice based drawing tool is implemented. The tool is evaluated with two groups of users: (1) users working with interactive entertainment technologies and (2) users who are not working with computer graphics. The main result indicates that although gaze and voice based drawing offers less control than traditional input devices it is perceived as the more enjoyable option. The remainder of the paper is organised as follows:

Section 2 summarises the relevant background information on eye tracking and eye movements. It also reviews the state of the art with regard to gaze and voice controlled entertainment applications. The design and implementation of the drawing tool are described in Section 3 and 4 respectively. Section 5 describes the experimental design of the user evaluation and Section 6 presents the obtained results. Finally, in Section 7 conclusions are drawn and future work is discussed.

2. BACKGROUND

Traditional input devices include mice, keyboards, and specific game controllers. Recent innovations in the video game industry include alternative input modalities to provide an

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

NGCA '11, May 26-27 2011, Karlskrona, Sweden

Copyright 2011 ACM 978-1-4503-0680-5/11/05 ...\$10.00

enhanced, more immersive user experience. Examples include motion sensing, gesture recognition, and sound input. Eye tracking has recently been explored as an input modality in games [5, 18]. Nowadays eye tracking technology has advanced and it is possible to obtain both cheaper, easier to use, faster, and more accurate eye tracking systems [2].

As eye trackers become less intrusive to the user, the technology could well be integrated into the next generation of interactive applications. It is important therefore to ascertain its viability as an input modality and explore how it can be used to enhance these applications. Alternative means of interaction are especially important for disabled users for whom traditional techniques, such as mouse and keyboard, may not be feasible.

2.1 Eye Movements and Eye Tracking

The information in the environment that reaches our eyes is much greater than our brain can process. Humans use selective visual attention to extract relevant information. Our highest visual acuity is in the foveal region. To reposition the image onto this area, the human visual system uses different types of eye movements. Between eye movements fixations occur, which often last for about 200-300 ms and are rarely shorter than 100 ms [16]. Approximately 90% of viewing time is spent on fixations [2]. During a fixation the image is held approximately still on the retina; the eyes are never completely still, but they always jitter using small movements called tremors or drifts [16].

Eye-tracking is a process that records eye movements allowing us to determine where an observer's gaze is fixed at a given time. The point being focused upon on a screen is called a gaze point or point-of-regard (POR). Eye-tracking techniques make it possible to capture the scan path of an observer. In this way we can gain insight into what the observer looked at, what they might have perceived, and what drew their attention [2]. Eye tracking can be used both for interactive and diagnostic purposes. In interactive systems the POR is used to interact with the application and it can be used as an alternative input device.

The most common technique used today for providing the POR is the video-based corneal reflection eye-tracker. Video-based eye trackers use simple cameras and image processing in order to provide the POR. It works by shining an infra-red light (which is invisible to the subject) toward the eyes and measuring the positional difference between the pupil centre and the corneal reflection, or Purkinje reflection. Since this relative difference stays the same with minor head movements but changes with eye rotation, it can be used to determine the POR on a planar surface [2].

The Tobii X120 eye tracker used in the project is a portable videobased eye tracker. It is portable and situated in front of the monitor. Its accuracy is reported as 0.5 degree and it has a sampling rate of 120Hz (It can also be run with 60Hz). Prior to recording the eye tracker needs to be fine-tuned to each user in a calibration process [13]. This is normally achieved by allowing the user to look at specific grid points. The calibration process can be incorporated within the interactive application so that the user is being calibrated before it starts.

2.2 Related Work

Early work in perceptually adaptive graphics mainly falls into gaze-contingent where parts of the virtual environment are modified based on the gaze of the observer [10]. Starker and Bolt [17] introduced one of the first systems with real-time eye tracking and intentionally constructed storytelling. When the user focused on objects for a certain duration, the system provided more information regarding the object using synthesized speech. In the last few years there have been an increasing amount of work done in the field of gaze controlled games [5],[18]. Although this work is still in its early stages and there are not many games that support eye tracking technology [5].

In [8] two open source games were adapted to use gaze, *Sacrifice* and *Half Life*. In *Sacrifice* using gaze control for aiming was compared to using mouse. Participants scored higher with gaze than mouse and it was also perceived as more fun. In [15] three different game genres were tested using gaze, *Quake 2*, *Neverwinter Nights*, and *Lunar Command*. Only *Lunar Command* was found to favour mouse control, where gaze had been used to aim at moving objects. One of the main results was that gaze as input can increase immersion.

In [19] a small third person adventure puzzle game was developed which used a combination of non intrusive eye tracking technology and voice recognition for novel game features. The game consists of one main 3rd person perspective adventure puzzle game and two first person sub-games, a catapult challenge and a staring competition, which use the eye tracker functionality in contrasting ways. In [12], a game using gaze and voice recognition was developed. The main concept of this was to escape from a maze while conducting common gaming tasks. When being controlled by gaze, a cross hair appeared where a user's gaze was fixed on the screen. By gazing towards the edge of the screen, buttons to change the orientation of the camera were activated.

While one user thought that using voice commands to move felt slow, gaze was found to be an easy method for aiming the cross hair, and overall gaze and voice was found to be the most immersive form of interaction as opposed to keyboard and mouse. There were some issues with voice recognition where some words had to be substituted in order to be recognized properly. The word "maze" had to be substituted for "map", and "select" was also found to be inconsistent as a word to choose menu options. For a more extensive overview of gaze controlled games please see [5, 18].

2.3 Gaze Based Drawing

There are two significant gaze based drawing programs in existence at the moment, *EyeDraw* [4] and *Eye Art* [11]. Both programs have certain limitations that are addressed in the project. *EyeDraw* is somewhat limited in operation and version 2.0 has drawing options for lines, squares, circles and certain clipart pictures. In this application the icons for selecting tools or menus are displayed on screen at all times. Although they are large enough to select comfortably it puts a limit on the amount that can be on the screen at once. As a result this limits the scope of the application. Because the icons are along the side of the screen, sometimes the users were found to choose them by accident [4]. In order to choose a drawing action, users needed to dwell their gaze

on a point on the screen for 500 milliseconds, and for the same amount to confirm it, which led to frustration when trying to draw. This was due both to the inherent delay for each drawing command when using dwell time to activate drawing and also because drawing was sometimes activated by mistake if users gazed at a point for too long.

EyeArt was developed in response to EyeDraw, which was found to be missing some essential parts of a drawing program [11]. While it is a more substantial program the video on the EyeArt wiki page [3] still shows that users need to dwell their gaze for a fixed amount of time in order to confirm drawing making it a time consuming process. This application has, however, more drawing options, such as fill, erase, text and polygon. This means scope for more complicated drawings, but since the icons are still visible along the left hand side of the screen this requires them to be smaller. As a result, this introduces further frustration, since they are difficult to select with gaze. Both programs also had an issue with drawing accuracy. If a user wants to start a line from the endpoint of another line, the chances of hitting the point spot on with gaze are minimal.

This project aims to overcome these issues by using voice recognition along with gaze. This avoids the need to dwell gaze in order to confirm drawing, and allows menus to only become visible when certain commands are spoken. As far as the authors are aware there is no other drawing application available that uses both gaze and voice recognition to alleviate the problem with dwell time and selection.

3. DESIGN

This section aims to provide an overview of the overall design of the application. It begins by discussing both the hardware used and the tools used for development, before moving onto the application itself. A Tobii X120 portable eye tracker was used to gather information about eye movements from the participants. The application was developed from scratch in C++ using several SDKs. The Microsoft Speech 5.1 SDK was used to process voice commands. The Tobii SDK provides different levels of access to the eye tracking hardware, and the high-level TETComp API was used. This allowed access to a subjects POR and contained tools for calibration and its own GUI. Microsoft's DirectX API was used for the graphics.

3.1 Application Design

When users start the drawing mode the drawing tool is set to line. The menu for changing drawing tools is easily accessible by giving one voice command. This menu allows users to choose between six different tools: line, rectangle, ellipse, polyline, flood-fill, and curve. Separate menus can be accessed for changing the current colour and the line thickness. A fourth menu is used for saving and quitting.

When considering the actions necessary for drawing the shapes themselves, it was decided to keep them relatively similar to those in mainstream paint programs as much as possible. For example, saying "start" adds a line to the screen going from the point at which the user was gazing when saying "start", to the current gaze point. Saying "stop" stops updating the second point of the line effectively finishing the current line.

Rectangle works by using two points given with "start" and "stop". A rectangle containing horizontal and vertical lines is then drawn based on these two corners. Ellipse works by drawing an ellipse in the area described by an imaginary rectangle based on these two points. Polyline is an extension of the line command, so whenever a user says "start" while drawing a line, the current line stops, and a new line starts at this point. A table containing all possible voice commands is shown in Table 1.

Voice Command	Action
Start	Starts drawing a shape
Stop	Stops drawing a shape
Snap	Starts drawing a shape at nearest vertex
Leave	Stops drawing a shape at nearest vertex
Undo	Removes the most recent drawing action
Fix	Fixes the current line to being vertical/horizontal
Unfix	Allows line to be drawn at any angle with x axis
Open Tools	Opens the tools menu
Open Colours	Opens the colours menu
Open Thickness	Opens the line thickness menu
Open File	Opens the file menu
Select	Chooses a menu button
Back	Exits from current menu screen

Table 1: Voice Commands.

Curve is implemented by having the user specify four control points. A curve is then drawn which starts at the first point, passes through the next two, and ends on the fourth. It is not possible to modify the curve once it has been drawn, since it was felt that this would be difficult to achieve with gaze. The application also contains helper functions which allow snapping a shape to the nearest vertex and fixing lines to be horizontal or vertical.

There is also a separate mode for colouring in linedrawings with gaze which can be chosen when starting up the application. After choosing this mode, users can choose a linedrawing which would take up the whole screen. The drawing tool is then set to be flood-fill, and users are able to fill in the line drawing with different colours, and save the picture when finished. This feature allows users to complete nice looking pictures in a much shorter period of time then using the drawing tools, which helps in acclimatizing oneself to this new input method for the first time.

4. IMPLEMENTATION

This section discusses the implementation of the project and is split into sections discussing the voice recognition system, the gaze system, and the drawing system.

4.1 Voice Recognition

The Microsoft Speech SDK made it possible to implement a command and control system for voice recognition. This system allows an application to "listen" for specific words or phrases, rather than interpreting everything that is picked up by a microphone, which occurs in a dictation system.

Each word or sentence that needs to be recognized is inserted in an XML file and is associated with a code number, or Rule ID. This XML file is then compiled to a binary .cfg grammar file using the grammar compiler, a tool which is provided in the SDK. In the application itself, a recognition context is initialized and registered to send messages to the main window whenever a word in the list has been recognized. A switch statement is then run on the Rule ID of the word to determine which command to process

4.2 Gaze System

The gaze system consists of two COM objects from the TET-Comp API. The first, ITetClient, communicates with the eye tracking hardware and receives the user's POR. The other is ITetCalibProc and is used to calibrate the eye tracker. In order to interface with these COM objects, "event sinks" are defined which listen to events fired by the objects. A function titled `:OnGazeData` is defined in the ITetClient's sink which is called every time the eye tracker returns gaze data. The coordinates are given in the range of 0-1. This value is then scaled by the screen dimensions in order to provide screen coordinates and used to update the cursor position.

The human visual system has natural micro-saccadic eye movements which keep the image registered on the retina, but this translates to jittery movements of the cursor. To overcome this, a method of smoothing was investigated in [9] which uses a weighted average to smooth gaze data.

$$P_{fixation} = \frac{1P_0 + 2P_1 + \dots + nP_{n-1}}{1 + 2 + \dots + n} \quad (1)$$

This is shown in Equation 1 which is reprinted from [9], where P_i are gaze samples, with P_n referring to the most recent sample and P_0 the least recent. By keeping track of previous gaze points, the jitter is removed, and by giving higher weights to the more recent points, the cursor ends up at the current POR. The amount of gaze points which are kept track of determines the extent of smoothing. Taking too many provides lots of stability but introduces lag. Since fixations usually occurred when deciding where a shape should start or finish, high accuracy was needed and stability was of primary importance. When quick saccadic eye movements were occurring that covered large distances of the screen, stability could be compromised in favour of responsiveness. To determine which of these movements were being made, the velocity of the cursor was measured.

By measuring the distance in pixels between the most recent gaze point and a previous gaze point, the velocity of the cursor could be evaluated. After some testing it was decided to sample the position at 100 millisecond intervals. This was frequent enough to give an up to date value for the velocity. By measuring the distance in pixels between the most recent gaze point and the previously sampled gaze point, the velocity of the cursor in pixels per 100 milliseconds could be evaluated. After some more quick tests were completed by the author using various velocities as thresholds between fixations and saccades, a velocity of 100 pixels per 100 ms was chosen since it seemed to give the best response for correct identification of each. If the velocity was above 100, the movement was flagged as a saccade and the amount of gaze points to average was set immediately to 15. 15 gave very fast response but still resulted in a small amount of jitter.

Fixations were more difficult to account for. If the amount of gaze points to average was simply set to a larger amount straight away, there would be a lot of empty elements which made the cursor unstable for a short length of time. By simply incrementing this amount by one every 100 milliseconds, a smooth increase was achieved. 100 milliseconds was initially chosen since this could be done with minimal interruption in the same area of the program that samples the velocity and after initial testing was found to produce very satisfactory results. An upper limit of 50 was put in place and this provided a seamless way of providing both quick response and great stability automatically when necessary.

4.3 Drawing System

The drawing system for the paint application is implemented in 3D on a virtual canvas, with all shapes being given a Z coordinate of zero. The shapes are implemented by initializing their respective vertex buffers and sending the vertices to the GPU to be drawn. In order to convert screen coordinates to coordinates in 3D space, a ray is cast from the camera through the point on the projection window which corresponds to the cursor position. When the Z coordinate of this ray equals 0, it has reached the correct point on the virtual canvas.

Lines were the first tool to be implemented. When a user says "start" an instance of the line class is initialized with the users POR given as the first point of the line. If the current line thickness is set to 0.0f, the line is drawn with two vertices as a line list primitive. If it is greater than 0.0f, however, it is drawn as a triangle list with four vertices as shown in Figure 1 (Top).

Figure 1: Top Left, line being rendered in wire-frame mode. Top Right, line rendered with solid fillstate. Bottom Left, Catmull-Rom interpolation given 4 points. Right, modified version.

The four vertices are evaluated from the two end points by getting the vector which is perpendicular to the line itself, normalizing and scaling it by the line thickness, and then adding or subtracting this vector to the two end points. At this point of initialization, a temporary second point has been chosen. This is overwritten almost immediately with the current POR by locking the vertex buffer to allow new positions to be specified for the vertices based on this point.

Rectangles and ellipses work similarly to lines. With rectangles, instead of making a line between the two points specified with "start" and "stop", a rectangle with these two points as opposite corners is formed with horizontal and vertical

lines. It is drawn with four lines if the thickness is 0.0f or eight triangles if the thickness is greater. Ellipses are formed by evaluating the equation of an ellipse contained in an imaginary rectangle formed by these two points. Thick ellipses are drawn by displacing vertices either side of the curve and joining the points with a triangle list.

In order to construct a curve which passes through all the control points given by a user, Catmull-Rom interpolation was used. In contrast if a Bezier curve was used users would have to place control points some distance away from the path of the final curve which would cause frustration while drawing. The main drawback to using Catmull-Rom interpolation was that by providing four control points, the resulting curve would only pass from the second point to the third point which can be seen in Figure 1 (Bottom Left).

Since it was desired to have the user supply 4 points, and have the curve start on the first point, pass through the second and third point, and finish on the fourth point, it was necessary to find two other temporary points. These can be seen in Figure 1 (Bottom Right), where the first temporary point is calculated based on the angle α . The distance from point 1 to imaginary point 5 is the same as the distance from point 1 to point 2. A similar process is followed to get the new point 6. With these extra points it was possible to construct a curve which passed through all four points. As per the other shapes, it is drawn with lines if the thickness is 0.0f and triangles otherwise. Polyline is just a series of lines. It works by adding a new line to the system every time a user says “start” or “snap”.

In order to perform the flood-fill operation the screen is rendered to a texture to get pixel information into a buffer. A flood-fill algorithm is executed on this buffer before it being copied back to another texture for displaying on screen. Various different flood-fill algorithms were investigated, and a scanline recursive algorithm was found [1] which was robust and gave good performance when tested. Subsequent shapes are drawn in front of this texture in order to be seen.

Instead of having a stack of drawing operations to be performed every frame and popping the most recent off the top when an “undo” operation was needed, a method involving textures was used. The scene is rendered to a new texture every time a shape is drawn, so if the user draws three shapes, three textures would be kept in memory. The first would show only the first shape, the second would show the first two, and the third would show all three shapes. The application always displays the most recent texture to the screen, and undo can be performed by simply removing the most recent texture added. The amount of textures to keep track of was limited to twenty to avoid too much memory being taken up. If a stack system was used there was the possibility of a shape being drawn and rendered to the texture when a flood-fill is performed. A user could remove this shape from the stack, but it would still be present in the texture related to the flood-fill operation.

By adopting the method of smoothing described in Section 4.2, the cursor was made very stable but it still could not compete with the accuracy of using a mouse at the pixel level. This made it difficult to start or end a shape at a

specific vertex. It was also difficult to draw lines that were perfectly horizontal or vertical. In order to account for this two helper functions were implemented, “snap” and “fix”. By saying “snap” instead of “start” the application starts drawing a shape at the nearest vertex. It does this by maintaining a list of vertices which are added to when a line, rectangle, or curve is drawn. When the command is given, the program loops through these vertices checking to see if it is within a thresholded distance from the POR and if it is closer than the shortest distance found so far. By saying “leave”, a similar process is followed to end the current shape at the nearest vertex. “Fix” works only for lines and checks the angle that the current line is making with the X axis. It then forces the line to be either horizontal or vertical depending on this angle. Saying “unfix” reverts back.

The menu system works by checking the position of the cursor when a user says start. If it is over a particular button, the actions pertaining to that button are carried out. With the menu system in place it was straightforward to implement the colouring in mode. Users can select this mode on startup where they are taken to another menu screen containing buttons representing different pictures. They can select a picture that they would like to colour in, and a texture containing this picture is then shown on screen. The only drawing tool available is flood-fill and users can fill the line-drawing in with colour.

5. USER EVALUATION

In order to evaluate the drawing application a user study was run. Two different groups were recruited from volunteers to evaluate the application. The first group was made up of users working with developing interactive entertainment applications. The second group was recruited from outside the field of computer science and had no experience with computer programming. It was expected that group one would have substantially more experience with paint programs.

The main aims of the user evaluation was to assess the difficulty in using gaze and voice as input for a paint program (when compared to mouse and keyboard) and to assess whether the evaluation ratings of the two groups would differ. The gaze and voice recognition based drawing was compared with mouse and keyboard on the basis of participants prior experience with paint programs.

It was decided not to have participants test out the colouring in mode, partly due to the fact that it would have made the overall trial time too long. Also, since this mode uses just the flood-fill tool, participants experience with this tool in the free-drawing mode could give an impression of how well the colouring in mode might work. The evaluation took the form of asking the participants to experiment with the application and try out each drawing tool, followed by completing a drawing task within a certain time limit.

5.1 Participants and Setup

The participants were all volunteers. There were eleven people recruited for each group. One participant was excluded from each group due to issues with voice recognition (based on a foreign accent) and difficulty in maintaining the calibration for the other. In the end, results for ten participants from each group were collected. The age range for group one

and two was between 20-30 and 21-40 respectively. Group one had a 10:0 balance of males to females, with an average age of 26.1 and average amount of paint program experience of 3.5. Group two had an even balance of males to females, with an average age of 25.1 and average amount of paint program experience of 1.4. Participants were recruited on the basis of having normal vision in order to avoid running into similar issues with calibration.

The Tobii X120 eye-tracker was positioned below a widescreen monitor along with a USB microphone which was placed in front of the keyboard. The participants were asked to sit comfortable so that their eyes were reflected back at themselves in the front panel of the eyetracker (which ensured that they were sitting at the right height) and were told they could adjust the seat height if needed. The distance from their eyes to the eye tracker was measured using a measuring tape to ensure that this was in the range of 60-70cm.

5.2 Procedure and Stimulus

Participants were first given an information sheet which gave some details on the experiment and how it would be carried out. They were also given a consent form to sign. After signing the consent form, they filled out a questionnaire which collected data on their age, gender, and number of paint programs they had experience with. This page also asked if participants had any history of epilepsy. If a participant answered yes, they were to be excluded from the experiment immediately.

In order to keep each trial as similar as possible, it was decided to hand each participant an instruction leaflet to read after this point. This leaflet explained how to use the drawing tools and helper functions. The eye tracker was then calibrated. This was done after participants had read the instructions since it was desirable to conduct calibration immediately before starting drawing. Once calibration was completed participants were asked to start the free drawing mode and to test out each drawing tool at least once. They were told that they could ask questions at any time if there was something they did not understand.

Once the participant felt they were ready, the application was reset to a blank canvas, and they were given a picture of a house to draw. They were told it did not have to be exactly the same, but to draw it as best they could and that they had a time limit of ten minutes. When they were ready to start, a key was pressed on the keyboard which started a timer in the application. The length of time in seconds from this moment was kept track of and if it exceeded ten minutes, the application saved the picture to an image file and automatically closed down. The whole experiment took about 20 minutes per participant.

Once the application had terminated, participants were handed another questionnaire to complete. This questionnaire allowed each participant to rate the application and experience based on the following headings: Ease of navigating the menus, how much control participants felt they had, how fast they drew, precision of controls, enjoyment, and how natural the controls were. Each question asked participants to rank an aspect of either input method on a scale from 1 to 7, with 1 being most negative and 7 being most

positive. Participants were also asked to rate the ease of giving voice commands, though this could not directly be compared to mouse and keyboard.

6. RESULTS

The results look at the ratings obtained and also the comments from the participants. One participant from each group failed to complete the section of the questionnaire pertaining to mouse and keyboard. These participants were not taken into account when performing statistical tests. Since the mean amount of paint programs that participants in group one had experience with is 3.5 and the mean for group two is 1.4, group one was deemed to have more experience with paint programs overall.

6.1 Statistical Analysis

Each question on the questionnaire was analyzed by a two tailed Wilcoxon Matched-Pairs Signed-Ranks test [14] to ascertain whether there was a significant difference between both methods of input. The questionnaire also asked participants to rate the ease of giving commands with voice on a scale of 1 to 7. Since this question was specific to using gaze as input and did not apply to mouse and keyboard, statistics were not run on these results. They resulted in a mean of 6.1 for group one and a mean of 6 for group two.

6.2 Appraisal of Results

The rankings obtained for aspects of each input method were quite promising. The question relating to ease of use of the menus returned no significant difference between input methods. This is promising as it shows that participants felt that using the menu system in this application was close to being as easy as with a mouse or keyboard. It had been intended to have the menus as accessible as possible with large enough buttons for choosing with gaze. Perhaps it was felt to be more intuitive to look at a large icon with a picture on it than to use a mouse to select words on a menu, as is found in most programs.

The next two questions, "How much control was there?" and "How fast did you draw?" both returned a significant difference favouring mouse and keyboard, which indicates that participants felt that traditional programs using mouse and keyboard offer more control and faster drawing. This result was expected though, since gaze simply cannot compete with the sub-pixel accuracy of the mouse. The fourth question 'How much precision of the controls was there?' only returned a significant difference from group one, and favoured keyboard and mouse. It had been expected that this would have also been the result for the other group. It is thought that this is because group two had less experience with paint programs overall than group one, and therefore found less of a difference in precision between the two modes of input.

Both groups felt that using gaze and voice as methods of input was significantly more enjoyable than keyboard and mouse which was an interesting result. There was no significant difference in how natural each group found each input method. This was also a good result as it indicated that this application is on par with using keyboard and mouse even though this was the first time that each participant had used gaze to control a cursor. Overall the comments

from the participants were positive and all of them felt that it would be of benefit to disabled users.

The voice recognition worked well also, though several female participants had difficulty with their voices being recognized. One participant commented: “Found it hard to Stop and undo, but if it recognized my voice better, than it would be brilliant! thanks”. The overall participant response was very promising for the question of “Ease of giving voice commands” where there was a mean of 6.1 and 6 for groups one and two respectively. This is a high score and shows that the voice commands worked quite well.

It can be seen that using gaze and voice as input methods offers less control than keyboard and mouse (and also less precision with group one). This is expected due to the lower accuracy of gaze and most participants were able to complete the drawing task satisfactorily. Each participant’s experience of using gaze and voice consisted of roughly ten minutes where they tested out each drawing tool before the drawing task. Since this is such a short time to get used to such a different input method, it is natural that gaze and voice might score less than keyboard and mouse with speed and control. When considering the statistical results for each question, both groups are seen to have had a relatively similar level of difficulty with the program. This shows that group 1 who had more experience overall with paint programs were not at an advantage to group 2. Along with the fact that 30% of participants remarked that with practice this would become much easier (“Yes because with practice this type of input could be as user friendly as a keyboard and mouse”), this fits in with the idea that controlling a cursor on screen with gaze is a new skill which needs to be practiced if used regularly. A house drawn by a participant from each group is shown in Figure 2.

6.3 Participant Comments

In general the comments from participants were promising. Everybody replied that this application could benefit users who cannot use traditional forms of input. Some of the comments relating to this are:

- “The menus were easy to navigate with large icons making tool selection simple and while not as precise as typical tools it is certainly a viable alternative if the user is unable to utilize traditional tools”
- “Yes, because I can’t think of an application with such intuitive alternative input devices”
- “I think with a lot of practice, it could be really beneficial to anyone who cannot use a mouse or keyboard, (and it’s really fun)”
- “The combination of voice and eye control after getting used to it is very similar to mouse use. So for people not able to use a mouse it would be quite useful”
- “It could provide a much needed outlet for people with limited mobility”
- “Very enjoyable and very interesting way to use computers for people with physical disabilities”

Figure 2: Left, Group 1 participant. Right, Group 2 participant.

Several respondents felt frustrated with the precision offered with gaze, “The eye tracking was difficult to use to pick precise points on the screen, but was intuitive and immediate for menus”, “Commands were straightforward to use and remember, but lack of precision in tracking eyes became somewhat frustrating”, “As a tool though it is not precise enough to replace other peripherals like the mouse or tablet”. Some participants had suggestions for features that would make drawing with gaze easier; “Could be an idea to make cursor change colour to confirm that the menu option has been activated as I was not sure it had registered until my shape darted across the screen!” while another participant suggested “An aid for focusing, like a grid because it’s difficult to focus on white space”.

7. CONCLUSIONS AND FUTURE WORK

The main aim of this project was to create a paint program controllable by gaze and voice. A user evaluation was carried out to evaluate how successful such an application would be. It was found that while using gaze and voice offers less control, speed and precision than mouse and keyboard, it is more enjoyable with many users suggesting that with more practice it would get significantly easier. All participants felt it would benefit disabled users. The project intended to improve on previous work in this area by implementing a novel approach of using voice recognition along with gaze.

The voice recognition helped in several ways. By using it to activate drawing, users do not have to wait for a fixation to be picked up. This avoids the delay involved in using dwell time. Also the problem of accidentally activating drawing by fixating gaze at a point is removed. Using voice recognition also made it possible to have menus that were completely invisible when not in use and can be accessed without gaze. This removed the problem of having distracting icons along the side of the screen that were limited in size. These improvements were seen to be successful according to participants responses given to the “Ease of giving voice commands” and “Ease of use of menus” discussed in Section 6.

The voice recognition worked well. There were some issues with female users (the system was trained with a male voice) and one user who had a very different accent to most others. This is not seen as a major problem since it is possible for end users to train the voice recognition engine themselves. It had been decided not to do this for each participant due to the extra delay it would introduce for each trial. Drawing was also made easier with gaze by implementing both a smoothing algorithm and helper functions. The helper functions were not used by all participants, but it is thought

that with more time and practice, participants would learn how to use them to their advantage to increase the quality of pictures produced with a gaze application.

There are several possibilities for future work. Visual feedback is important and the image of the cursor could change depending on whether a shape was being drawn. This would take away ambiguity of the exact time that a command had been processed and dissuade users looking away from the desired second point of a shape as soon as they said “stop”. Some users had trouble concentrating on pure white space and suggested a series of optional grid points which would help with positioning shapes.

Another possible addition would be the ability to add line drawings to a picture that was being drawn in free-drawing mode, in order to mix both drawing modes. A settings menu could also be included to alter parameters in the application. This menu could also be responsible for changing how many points would be used for calibration since sometimes five can be enough for calibrating satisfactorily. Other features related to the drawing system could be set here such as the distance threshold for the snap function and what format the image file should have. Finally, in order to suit the majority of disabled users it would be beneficial to have a mode that only recognized a noise being spoken to activate drawing, since some might have speech impediments which would prevent them from using all the voice commands. A video of the application can be found here: <http://www.youtube.com/watch?v=PugwLwKRz6I>.

8. ACKNOWLEDGEMENTS

The authors would like to thank Acuity ETS limited for providing the loan of a Tobii X-120 eye-tracker and Jon Ward from Acuity for his support in its operation. We would also like to thank Paul Masterson for his help in any hardware issues that arose and all the participants that took part in the evaluation of this project.

9. REFERENCES

- [1] Codecadex. Implementing The Flood Fill Algorithm. http://www.codecadex.com/wiki/Implementing_the_flood_fill_algorithm#C, Accessed 11 September 2010.
- [2] A. T. Duchowski. *Eye Tracking Methodology, Theory and Practice*. Springer, second ed. edition, 2007.
- [3] EyeArt. Gaze-controlled drawing program. <http://www.cogain.org/wiki/EyeArt>, Accessed 18 January 2010.
- [4] A. J. Hornof and A. Cavender. Eyedraw: enabling children with severe motor impairments to draw with their eyes. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 161–170, New York, NY, USA, 2005. ACM.
- [5] P. Isokoski, M. Joos, O. Spakov, and B. Martin. Gaze controlled games. volume 8, pages 323–337. Springer Berlin / Heidelberg, 2009. 10.1007/s10209-009-0146-3.
- [6] R. J. K. Jacob. What you look at is what you get: eye movement-based interaction techniques. In *CHI '90: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 11–18, New York, NY, USA, 1990. ACM.
- [7] R. J. K. Jacob. Eye movement-based human-computer interaction techniques: Toward non-command interfaces. In *In Advances in Human-Computer Interaction*, pages 151–190. Ablex Publishing Co, 1993.
- [8] E. Jonsson. If looks could kill-an evaluation of eye tracking in computer games. In *Masters Thesis*. KTH Royal Institute of Technology, 2005.
- [9] M. Kumar, J. Klingner, R. Puranik, T. Winograd, and A. Paepcke. Improving the accuracy of gaze input for interaction. In *ETRA '08: Proceedings of the 2008 symposium on Eye tracking research; applications*, pages 65–68, New York, NY, USA, 2008. ACM.
- [10] D. Luebke, B. Hallen, D. Newfield, and B. Watson. Perceptually driven simplification using gaze-directed rendering. Technical report, Rendering Techniques 2001, Springer-Verlag (Proc. Eurographics Workshop on Rendering, 2000).
- [11] A. Meyer and M. Dittmar. Conception and development of an accessible application for producing images by gaze interaction, eyeart (eyeart documentation). http://www.cogain.org/w/images/d/da/EyeArt_Documentation.pdf.
- [12] J. O'Donovan, J. Ward, S. Hodgins, and V. Sundstedt. Rabbit run: Gaze and voice based game interaction. In *EGIRL '09 - The 9th Irish Eurographics Workshop*, Trinity College Dublin, Dublin, Ireland, 2009. EGIRL.
- [13] A. Poole and L. J. Ball. Eye tracking in human-computer interaction and usability research: Current status and future. In *Prospects, Chapter in C. Ghaoui (Ed.): Encyclopedia of Human-Computer Interaction*. Pennsylvania: Idea Group, Inc, 2005.
- [14] F. Sani and J. Todman. *Experimental Design and Statistics for Psychology, A first Course*. Blackwell Publishing, 2006.
- [15] J. D. Smith and T. C. N. Graham. Use of eye movements for video game control. In *In Proceedings of the 2006 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology (p. 20)*. ACM Press, 2006.
- [16] R. Snowden, P. Thompson, and T. Troscianko. *Basic Vision: an introduction to visual perception*. Oxford University Press, 2006.
- [17] I. Starker and R. A. Bolt. A gaze-responsive self disclosing display. In *CHI '90: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 3–10, New York, NY, USA, 1990. ACM.
- [18] V. Sundstedt. Gazing at games: using eye tracking to control virtual characters. In *SIGGRAPH '10: ACM SIGGRAPH 2010 Courses*, pages 1–160, New York, NY, USA, 2010. ACM.
- [19] T. Wilcox, M. Evans, C. Pearce, N. Pollard, and V. Sundstedt. Gaze and voice based game interaction: the revenge of the killer penguins. In *ACM SIGGRAPH 2008 posters*, pages 81:1–81:1, New York, NY, USA, 2008. ACM.