THE SMART CHAIR PROJECT: REDUCING CODING FOR THE SMART CHAIR PROGRAMMERS

NSF Summer Undergraduate Fellowship in Sensor Technologies
Catherine Lachance (Computer Science Engineering) – University of Pennsylvania
Advisor: Dr. Jim Ostrowski

ABSTRACT

The Smart Chair is a wheelchair that allows a user to navigate and communicate solely through an onboard computer. The research described in this paper dealt mainly with organizing the program's interface and facilitating the creation of the pages displayed on the Smart Chair. These innovations, including writing auxiliary functions and combining similar variables into larger data structures, make the wheelchair more user-friendly and make its code easier to read. Work was also started this summer to approach page display in a new way, so that code needed to generate a page was created dynamically at runtime, and thus one function would suffice to generate all pages, instead of one function per page. Furthermore, by creating a graphical user interface that generates the code needed to display each page, this research cut the amount of coding needed to run the program by more than forty-two percent, and thus minimized the total time needed to program the Smart Chair. Because of this newfound convenience for the programmer, the Smart Chair program can be expanded more easily, with more options for the user.

1. INTRODUCTION

The Smart Wheelchair is an autonomous wheelchair controlled by an onboard computer, projector, camera, motors, and sensors. By using several types of input devices, the Smart Chair user interacts with the computer to move about, speak, and access the Internet. Figure 1 shows the Smart Chair and its basic components.

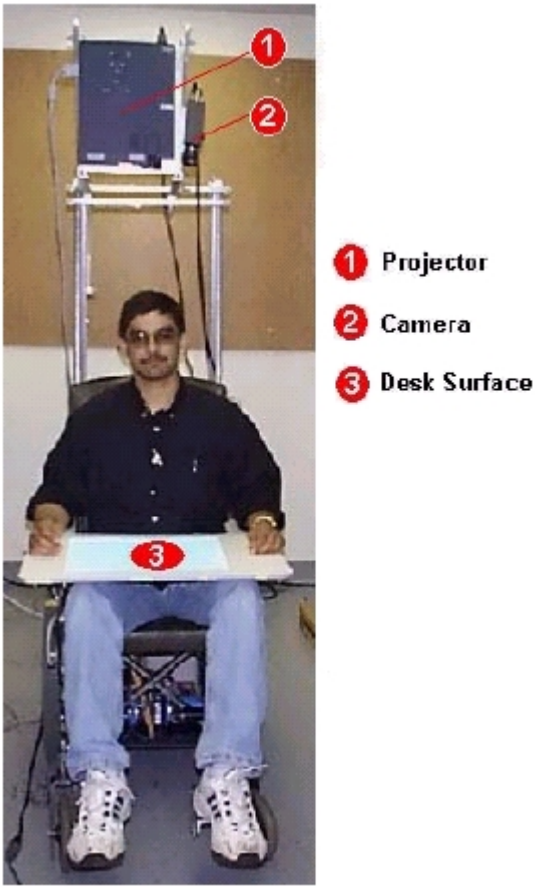


Figure 1: The Smart Chair in GRASP Lab at the University of Pennsylvania with a user [1, p.2].

The Smart Chair uses the projector to display computer-generated images onto the lap tray of the wheelchair. These images consist of icons (solid rectangles or bitmaps) and text, much like a web page. Figure 2 shows the opening page of the Smart Chair program. This page acts as a home page and thus is accessible from other pages.

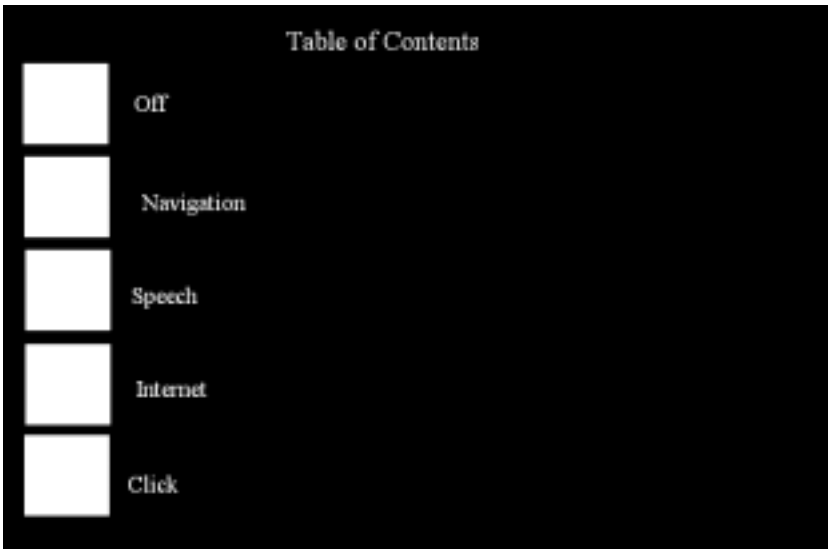


Figure 2: Image of the Smart Chair interface.

The user can then communicate with the computer by covering icons with his hands, using a finger mouse, or using a wireless mouse, depending on the page being displayed. The camera, which is located next to the projector and run by the frame grabber, constantly records the actual lap tray image [2]. Thus, when the user puts his hands over certain places on the lap tray, the camera will note a change in its view and return this information to the computer.

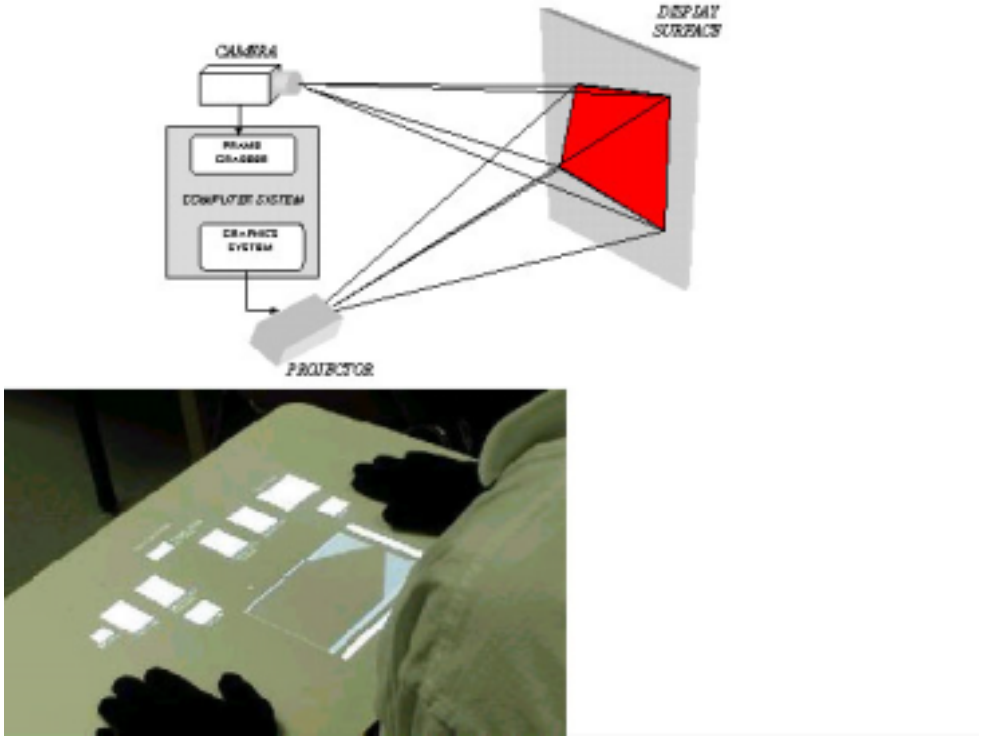


Figure 3: Image of Frame Grabber, projector, camera, and lap tray. [2]

The program then determines which icons the user has covered by comparing the coordinates of the changed image to those of each existing icon. Once the selected icon has been determined, the program instructs the wheelchair to perform the appropriate action.

The most important aspect of a wheelchair is its mobility. Thus, past innovations to the Smart Chair have dealt mainly with navigation. Because people who use wheelchairs might lack fine motor skills, the Smart Chair employs several options to allow users to move. There are two types of manual navigation, plus point-and-click navigation.

With manual navigation, it is assumed that the user can get from one point to another simply by indicating the exact movements the wheelchair should make. The user can manually control the Smart Chair in two ways. First, the user can call up a page that expected the hand to be used as the input device - covering up icons for directions, such as left, right, and straight. Second, the user can move the chair about by using a stylus in a navigation box. This box is a white square whose middle is considered the origin of a 2-dimensional plane; the positive y-axis denotes the forward direction and the negative y-axis denotes the backwards direction. If the stylus is positioned in the first quadrant, the wheelchair will veer to the right and forward; if the stylus is in the second quadrant the wheelchair will move to the left and forward, and so on. Furthermore, the farther away from the origin the stylus is located, the faster the wheelchair will move.

The other type of navigation currently programmed into the Smart Chair is geared toward users with less hand-eye coordination than those using manual navigation. In this mode, a 320-degree view of the room, encompassing the user's peripheral and front vision, is displayed on the lap tray. The omnidirectional camera located directly above the seat of the wheelchair captures this picture. The Smart Chair program flattens the 360-degree image so that it is easier for the user to interpret. With this view in front of the user, he can specify on the lap tray where he would like to go in the room simply by sliding his finger on the x- and y-axes of the perimeter of the lap tray picture and then clicking on the confirm icon. The Smart Chair program finds the coordinates of the points the user has specified on the lap tray, finds the position in the room these points refer to, and moves there.

In addition to moving around, the Smart Chair uses the Microsoft Speech software design kit to speak for the user [1, p.4]. By accessing the speech page, the user can prompt the program to project certain phrases through the onboard speakers. Karla Conn created this part of the chair, along with the finger mouse used primarily with this page, through the Sunfest program last summer. A finger mouse attaches to the user's index finger and provides an alternative from hand-coverage in receiving input. This mouse works by sending messages to a receiver that is attached to the wheelchair. The finger mouse and speech page are shown in Figures 4 and 5.

Figure 4:
Picture of finger mouse
[1, p. 5].

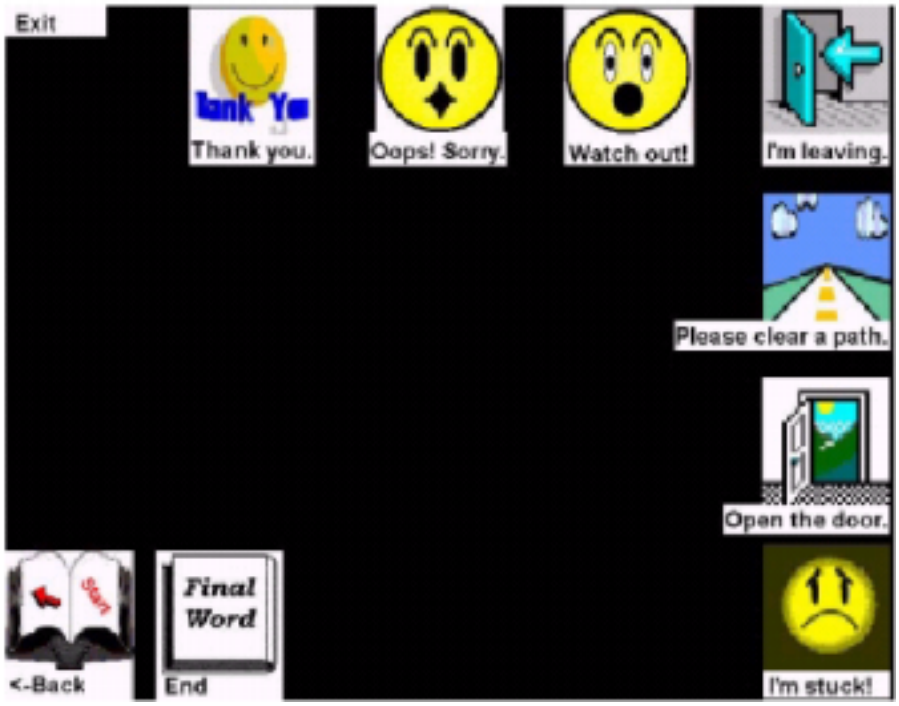


Figure 5:
The speech
page [1, p. 13].

Since it attaches to the user's finger, the finger mouse is more convenient than a regular wireless mouse, which could fall off the lap tray. Therefore, this finger mouse could be used for other pages too, perhaps taking the place of a wireless mouse. Currently, however, hand-covering serves as the primary form of input, and thus most pages do not support mouse-type input. Another feature that may be added to the Smart Chair in the future would allow different input modes to be available at runtime so that the user can switch between the types of input easiest for him to use.

2. PRELIMINARY STREAMLINING

At the beginning of the summer, the main file of the Smart Chair program consisted of over 2800 lines of code, which was somewhat cumbersome for programmers to read. First steps were to write supplementary functions and to combine similar data variables into arrays. Both steps eliminated a great deal of redundant code.

2.1 Auxiliary Functions

Besides cutting over 1200 lines from this file, adding auxiliary functions proved to be an effective way of becoming familiar with the Smart Chair code. It was clear that in every method that generated a page, the same lines were used repeatedly. These statements created the icon object, making the program aware of each icon's existence. These lines also received information about each icon's coordinates in OpenGL and then transformed them into coordinates that the projector could interpret. For these lines, two auxiliary functions, `create_icon()` and `add_button()`, were created. `create_icon()`takes in as arguments a pointer to the CPointsArray button that needs to be initialized and an array of that icon's OpenGL coordinates. This method then initializes the button object by calling `add_button()`, which entails setting the button's newly transformed coordinates. Although including both of these seven-line methods added fourteen additional lines to the main code, now a single line (calling `create_icon()`) replaces eight lines that originally created each button. Furthermore, having more consistent code makes the program as a whole easier to read.

The main Smart Chair program also benefited from a supplemental method in the function that displayed and ran the speech page, `sub2_IdleFunc()`. As described above, not only does this page display bitmaps and outlined text, but it also cause the Smart Chair speaker to project certain phrases. This function initialized variables for every button on the speech page and uttered selected phrases. However, as an alternative to having the same lines of code written for each icon, a new function was written, `speech_aux()`, which cut the size of the main program by 150 lines.

The last major area where an auxiliary function proved beneficial was in the `output()` functions. Originally, three functions all displayed text onto the screen. The methods differed only in the type of font shown, so the code was altered to have only one `output()` function that required an integer input referring to the type of font to be displayed. This trimmed the main file by 38 lines.

2.2 Combining Similar Variables

The header file that originally initialized all of the global variables in the main Smart Chair file consisted of many variables that took on the same function with very similar names. Many variable names differed only by their last character, such as `ogl_back_toc1` and `ogl_back_toc2`. To eliminate this redundancy, arrays were created that combined all similar variables. For example, the four variables named

`ogl_back_toc1`, `ogl_back_toc2, ogl_back_toc3, and ogl_back_toc4` were combined into an array named `ogl_back_toc` and each given the respective position denoted by their last character (minus one, since the Smart Chair program is written in Visual C++ and thus array indices start at zero). Incorporating similar variables into an array slightly reduced the size of the header file housing all of these variables. Furthermore, the code using these variables became easier to read and more consistent; when calling auxiliary functions that needed the same information from different icons, the array index could be passed, instead of the name of the whole variable.

3. PAGE CREATION

Since technology and computers go hand in hand, either every programmer has to be well versed in many disciplines, or inventors must be able to program well. For example, the Smart Chair draws on several areas of expertise, such as mechanical engineering and computer science. Getting the wheelchair to move in certain ways requires sufficient knowledge of linear algebra and physics. Thus, the mechanical engineers working on the Smart Chair must also have good programming skills.

To avoid having to program too much, it was suggested at the beginning of the summer that a graphical user interface (GUI) that helped create and design Smart Chair pages would be quite useful. Over the course of the summer, this GUI project transformed from writing a simple specialized paint program into an intricate method for generating code into two different page display approaches.

The paint program facilitates the creation of Smart Chair pages by allowing programmers to design each page simply by drawing on the screen. This program then takes the coordinates and other shape information and generates the code needed to display that image during the actual execution of the Smart Chair program.

Figures 6 and 7 show two views of the paint program.

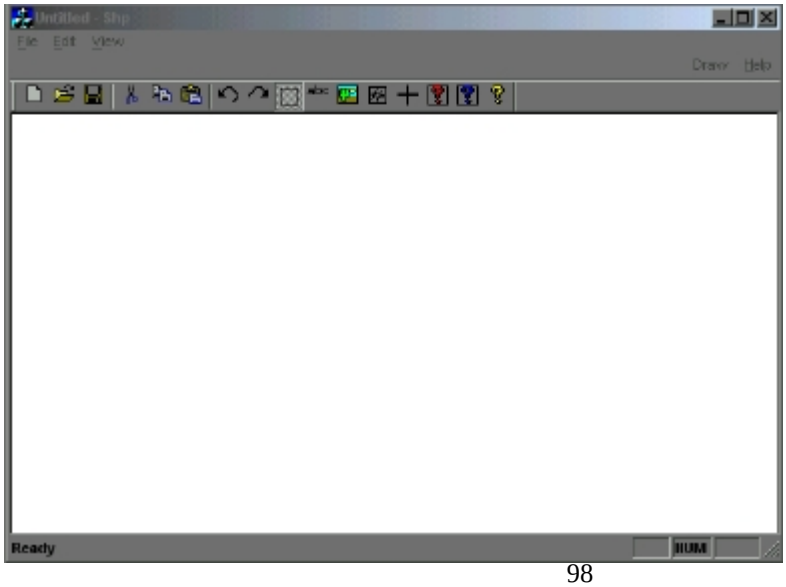


Figure 6:
The Smart Chair paint program.

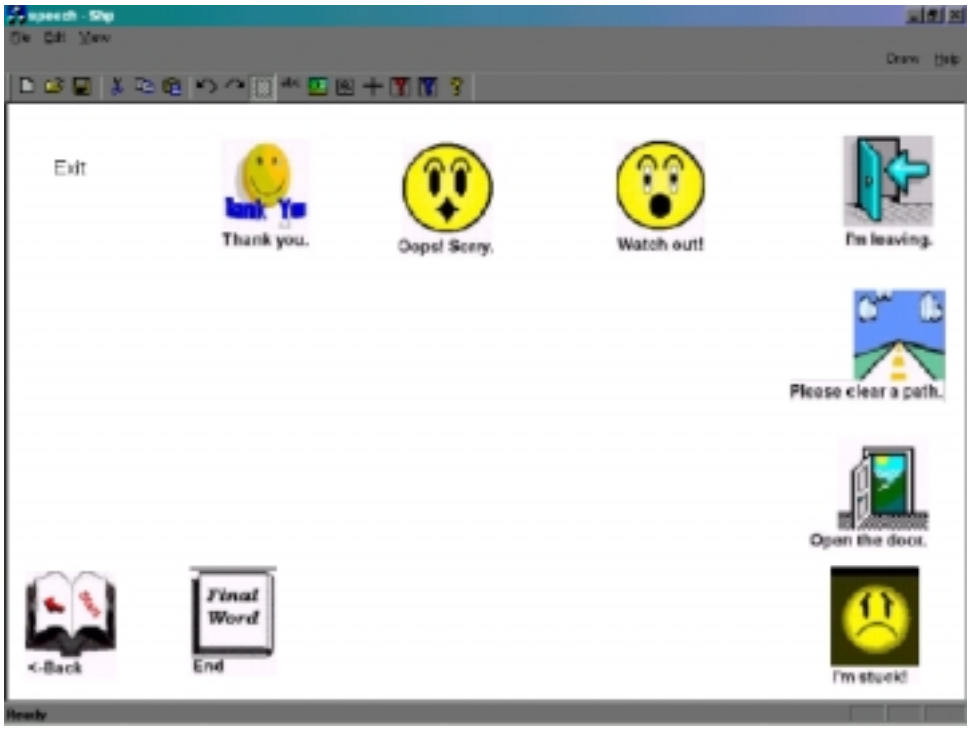


Figure 7:
The Smart Chair paint program with a representation of the original speech page.

As these figures illustrate, icons and text can be added onto a page using this paint program. Pages can be saved, opened, and edited at any time. Some of the paint program's features specifically designed for the Smart Chair are:

1. Drawing rectangles of any size
2. Adding text in different fonts
3. Placing bitmaps onto the page
4. Copying and pasting rectangles
5. Undo and redo options
6. Deleting any shapes
7. Saving and opening any pages created by the Smart Chair paint program
8. Adding generic buttons*
9. Saving a rectangle into the button library*
10. Getting information about the current page or selected icon*

* See Section 4 below.

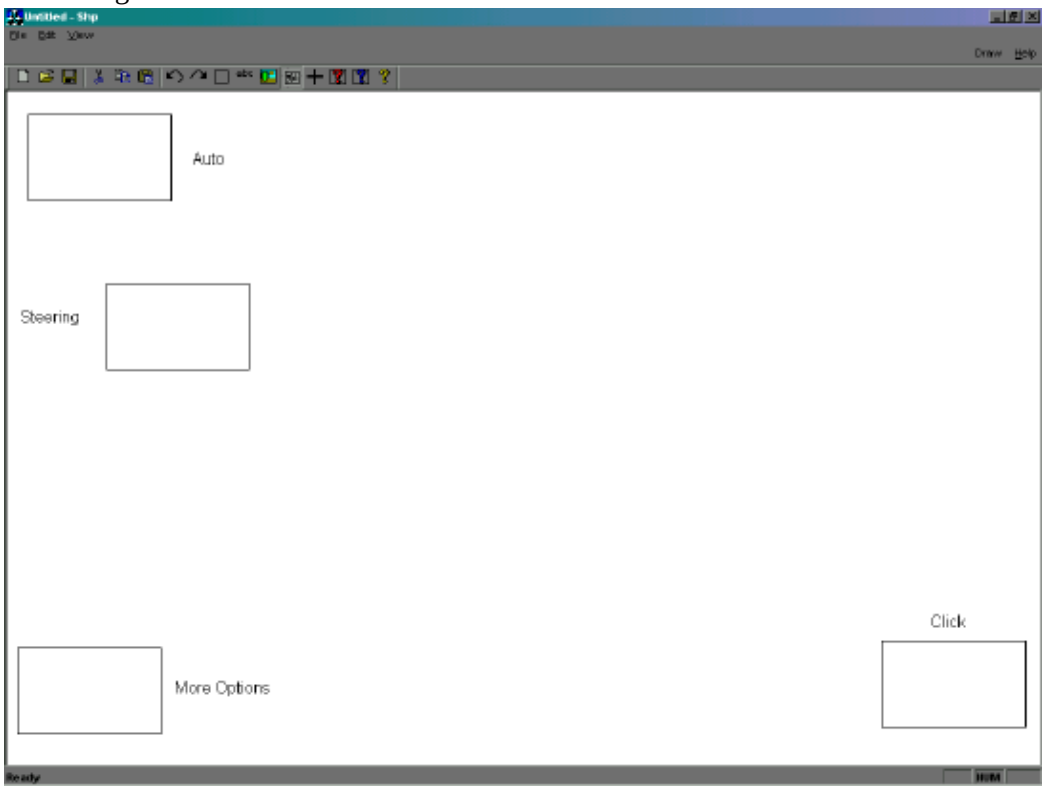
As the above list shows, this paint program acts much like the Microsoft Paint program. The likeness is not coincidental; since the Smart Chair program is written in Microsoft Visual C++, it is safe to assume that Smart Chair programmers are familiar with commands needed in any editing program, such as copy, paste, undo, redo, etc. Even for a programmer not well acquainted with these directions, the paint program is designed to be as user-friendly as possible.

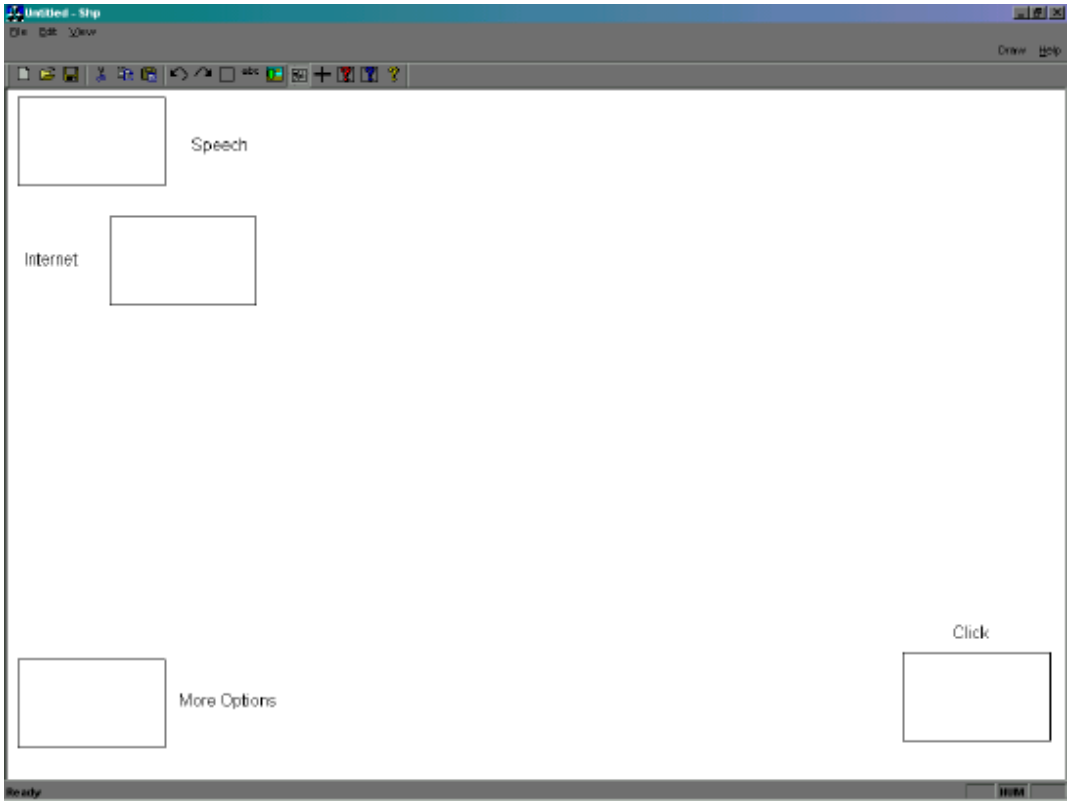
4. ICON LIBRARY

Another project started this summer and currently under way is the designing of a library of icons that take on the same functionality in different pages. Every good web page has a convenient collection of links that bring its viewers to other popular destinations in its set of pages. For example, when a web-based email program is used, some options remain constant to the user no matter what page he is currently viewing. If the user is looking at his inbox, then there should be a set of links somewhere on the page that can be used to get to his other mail folders, to log off, and to compose messages. If he were to move to another mail folder, these same links should appear on the new page in the same spot so that they are easily identifiable and accessible to the user. Since the Smart Chair pages can be thought of as web pages that do not access the Internet, this same type of consistency should be expected from the Smart Chair program, especially since one of the main goals of the Smart Chair is to make handicapped lives as easy as possible.

Some functions that should appear on each Smart Chair page are buttons that switch to different modes of navigation, a button that returns to the main table of contents (or home page), a button that displays more of the main options, and a button that confirms whether the user has chosen an existing option. Figure 8 is a snapshot of what these icons will look like, as drawn on the specialized paint program.

Figure 8: The first two screens of the new sidebar menu.





The paint program incorporates this concept of having a consistent menu on each page by supporting an icon library. This library allows icons that are used in multiple pages to be saved by the user. When the user creates a new page and wants to add an icon that was used in another page, the user just specifies the number of that button, and the icon will be added along with its caption.

To use the icon library, icons first need to be created and then selected for inclusion into the set of generic buttons. This is done by drawing an icon and creating text using the paint program, and then clicking on the “Add” button from the toolbar and the specific icon. The program prompts the user to enter a description of what this icon does and also to select the text to be associated with this icon. The button is then assigned a unique number for identification purposes.

To utilize a generic icon from the library, the user selects the “Draw generic icon” option from the toolbar and clicks on the drawing area. A dialog box appears, asking for the button number of the icon to be added. In case the user does not know the number of the icon they want, he can ask to see the icon list, which displays the list of existing buttons in the icon library and what each of them do. Once the user specifies the icon number, it is drawn onto the page.

At some points, the page designer might want to use an icon from the library but in a spot different from its usual place. For this reason users are required to click on the drawing area before they can add generic buttons - when the user is prompted for the button number, he will also be asked whether or not the icon should go in its normal spot. If the user says no, then the generic icon will be centered on the point in the drawing area that the user clicked.

In addition to using previously drawn icons, users can create their own buttons and specify the number themselves. Not only are the button numbers used to identify icons in the library, but they are also operation modes used by the main Smart Chair program. Thus, if a page designer wants an icon that goes to the speech page but wants to use a different-size icon than the one in the icon library, he can draw a rectangle using the paint program, and then assign it a name and number. The user can look at the icon list to find out the speech icon's operation mode, and then type in this same number for his newly made icon.

5. A NEW APPROACH TO PAGE DISPLAY

At the beginning of the summer the Smart Chair code used OpenGL and generated the code to display each page statically. Currently, changes are being made to the program so that each page will be created dynamically during runtime. Because specific code designated to each page display need not be explicitly generated, one function that can generically display any page suffices, rather than one function per each page displayed by the program. Thus, this new approach to displaying and representing pages shown by the Smart Chair will drastically reduce the size of the main file. Besides making the file more manageable to read, it also means that, instead of making sense of many different functions that all display various pages, only this new generic function must be understood in order to comprehend how the Smart Chair creates and displays each page.

5.1 New Data Structures

The Smart Chair paint program creates and attaches information onto a header file used by the new Smart Chair code. This header file contains data about each Smart Chair page. First, for every image on a page, including bitmaps and text, their modes of operation (or button numbers) are saved in a two-dimensional array, `page`. Then, the coordinates of each image and the bitmap and text strings information are saved in two 3-dimensional arrays, `Coord` and `displayInfo`, respectively. In order to keep the number of elements for each page consistent, so that the operation modes, coordinates, and other information correspond to the appropriate image, some missing data is replaced by a sentinel value. This value is assigned to the text button numbers in `page`, since text should have no functionality. Also, since normal icons do not need a bitmap path name or a string associated with them (like text), the `displayInfo` holds the icons' places by inserting the string, "-99," as their information.

6. CONCLUSIONS

The Smart Chair is still in its early stages, but the innovations made this summer will have a great impact on wheelchair users. The easier it is to program the Smart Chair, the easier it will be to add more features. Making the Smart Chair code easier to follow has. This entailed writing auxiliary programs to eliminate redundant code, using graphical user interfaces whenever possible to generate code, and reducing the overall amount of code needed. Furthermore, with the new approach to page display, work is currently under way to drastically reduce the size of the Smart Chair code even more.

7. ACKNOWLEDGMENTS

I would like to thank the Smart Chair group, in particular Rahul Rao, Sang Lee, Sarangi Patel, and my advisor, Dr. Jim Ostrowski. I also would like to thank Zafir Anjum for posting his code [see 3] explaining the insertion of bitmaps into an application at runtime. Last, I am very grateful to the National Science Foundation for sponsoring the research done on the Smart Chair this summer with the NSF-REU grant and to the Microsoft Corporation for their financial contributions.

8. REFERENCES

1. Karla Conn. “Finger Mouse and Text-to-Speech Application as Additions to the Smart Wheelchair.” [University of Pennsylvania, Electrical Engineering Department, Sunfest 2001 article] 2001. Available from: http://pender.ee.upenn.edu/~sunfest/pastProjects/Papers01/KConn.pdf Accessed: 1 August 2002.

2. “The Smart Wheelchair - An Overview” [University of Pennsylvania, GRASP laboratory project description] 2001. Available from: http://www.cis.upenn.edu/smartchair Accessed: 1 August 2002.

3. Zafir Anjum. “Drawing a Bitmap from a BMP File.” [online article from Code Guru, Bitmap & Palette] 1998. Available from: http://www.codeguru.com/bitmap/draw_bmp.shtml Accessed: 17 July 2002.

APPENDIX A: POSSIBLE ERRORS AND THEIR SOLUTIONS

Although the Smart Chair paint program is straightforward and self-explanatory, a user may receive some error messages. Described below are some of these messages and the reasons for them.

A.1 Saving Error: An error occurred while trying to insert the new changes into the Smart Chair code, so the save was terminated.

Upon inserting the changed page information of a previously saved page, the paint program could not find a specific string needed for a particular insertion location. To fix this, one must make sure that no additional code was added in specific areas of the files being altered. This error should not normally occur since the Smart Chair code must include the exact places that cannot be changed in order for the page to be displayed correctly. For a complete list of these areas, refer to Appendix B.

A.2 Invalid Button Number: Warning! The number you picked for this icon is not valid. You may change it now or leave it alone.

When new icons and bitmaps are added to a page their button number is checked to see if it corresponds to any existing buttons. If the user did not specify a mode of operation for the icon, he will be warned that the generated code will not be complete. This is not a problem since the user might prefer to assign the icon a number later. This message will also appear if the icon was assigned a number that did not correspond to any numbers in the icon library. The message merely warns the user that he will have to assign functionality to the icons later.

A.3 Library Save Unsuccessful: You did not click on any text, so your icon was not saved into the button library.

Each icon in the icon library needs to have text associated with it. Thus, if the user tries to save an icon without text, he will receive this message and the icon will not be saved in the icon library. If the user wishes to add an icon without text to the library, he can create a block of text that will not appear (using spaces or other blank characters) and click on that.

A.4 Icon Not Found: Sorry! The button you want to add does not exist!

This message appears when the “Generic Icon” option is used and the icon number given does not specify an icon included in the icon list.

A.5 Click Warning: A "Click" icon was not made. Do you still want to save this page?

Each page using hand-covering input should include a confirming icon so that the Smart Chair code will run properly. In determining whether a user has covered a certain icon on the lap tray of the wheelchair, the same code is used repeatedly (substituting in the name of the respective icon):

```
if ((OffPercent > 0.50) && ( ClickPercent > 0.50 ))
{
      MODE_OF_OPERATION = 1;
      break;
}
```

Thus, each page must have a click icon, and this warning will result if one does not exist.

A.6 Too Many Icons: 9,999 icons have already been added to the button library! If you want to add more buttons, insert another zero into the beginning of the "buttons" file, and then change the safety clause in ShpDoc.cpp.

The number of icons saved in the button library is read by a character array of size four. Thus, only four digits can be read, and therefore the icon list cannot exceed 9,999 buttons. Since the Smart Chair currently has only four pages with no more than ten icons on each page, this problem should not occur.

A.7 Too Many Pages: 9,999 pages have already been created! If you want to add more pages, insert another zero into the beginning of the "page1" file, and then change the safety clause in ShpDoc.cpp.

The number of pages used by the Smart Chair program is read by a character array of size four. Thus, only four digits can be read, and no more than 9,999 pages can exist. Since the Smart Chair currently has only four pages, this problem should not occur.

A.8 Too Many Shapes: This page has 100 icons and text objects! Please either delete some shapes or open a new document.

In the early stages of the paint program, attempts were made to change the data structure holding the shape information from an array with a fixed size of 100 to a linked list that could dynamically change size. Unfortunately, because of time constraints, this goal was never realized. Therefore, a maximum of 100 shapes (including icons, text, and bitmaps) may be added to one page.

A.9 Invalid Name: Please pick a new name. The name you entered is either invalid or already belongs to another icon.

In order for the code to be generated and compiled correctly in the main Smart Chair program, the shapes created with the paint program must have unique names. When an icon is added with the same name as another icon on the page, this error message will appear and will remain until a new valid name has been specified for the icon.

APPENDIX B: INSERTION KEY

```
1-> /*codefile*/void subsecond_index_IdleFunc()
{
      long Address;
      bool choice_made = false;
      MODE_OF_OPERATION = 4;
      MappTimer(M_TIMER_RESET, M_NULL);
      MdigChannel(fg.idDig, M_CH0);
      MdigGrab(fg.idDig, fg.idImage);
      Address = MbufInquire(fg.idImage, M_HOST_ADDRESS, M_NULL);
      ImageAddress = (unsigned char*) Address;

      double BUTTON4Percent;
      double ClickPercent;
2->   create_icon(ogl_BUTTON4, temp_BUTTON4, BUTTON4Button);

3->   while (!choice_made)
      {
            MdigGrab(fg.idDig, fg.idImage);
            BUTTON4Percent = Histogram(BUTTON4Button).Percent;
            ClickPercent = Histogram(second_index_ClickButton).Percent;

4->         if ((BUTTON4Percent > 0.50) && ( ClickPercent > 0.50 ))
            {
                  MODE_OF_OPERATION = 4;
                  break;
            }
            else if ((StatusPercent > 0.50) && ( ClickPercent > 0.50 ))
            {
                  MODE_OF_OPERATION = /*put number here*/;
                  break;
            }
            else if ((BUTTON5Percent > 0.50) && ( ClickPercent > 0.50 ))
            {
                  MODE_OF_OPERATION = 5;
                  break;
            }
            else if ((BUTTON6Percent > 0.50) && ( ClickPercent > 0.50 ))
            {
                  MODE_OF_OPERATION = 6;
                  break;
            }
5->         if (MODE_OF_OPERATION != 4)
                  choice_made = true;
      }

      if (choice_made)
            return;
}

6->void draw_second_index()
{
      glClearColor(0.0,0.0,0.0,0.0);
      glClear(GL_COLOR_BUFFER_BIT);
      glColor3f(1.0,1.0,1.0);
```

```
      glRectf(BUTTON4[0][0], BUTTON4[0][1],BUTTON4[1][0],
BUTTON4[1][1]);
      output(1, -0.673810, -0.877565, "More Options");
      glRectf(Status[0][0], Status[0][1],Status[1][0], Status[1][1]);
      glRectf(BUTTON5[0][0], BUTTON5[0][1],BUTTON5[1][0],
BUTTON5[1][1]);
      output(1, -0.650230, 0.805120, "Speech");
      glRectf(BUTTON6[0][0], BUTTON6[0][1],BUTTON6[1][0],
BUTTON6[1][1]);
      output(1, -0.964630, 0.459170, "Internet");
      glRectf(second_index_Click[0][0], second_index_Click[0][1],
            second_index_Click[1][0], second_index_Click[1][1]);
      output(1, 0.789930, -0.673140, "Click");
7->   glFlush();
      return;
}
```

The following strings (the second, third, and fourth arguments to findPosition) must be present in the given order in the file, codefile, for the function findPosition (in ShpDoc.cpp) to work correctly:

1) findPosition(position, "void ", nameOfFile, "IdleFunc()", "", REGULAR, info, bufLength, buf);
2) findPosition(position, "\tdouble", "\n", "// A c", "\tcre", AFTER_OPT, info, bufLength, buf); (The fifth argument will be searched for as well as the fourth since only one of these strings should be in the code.)
3) findPosition(position, "\tcreate_i", "Button);", "\n\twhile", "", REGULAR, info, bufLength, buf);
4) findPosition(position, "Percent = ", " ", "\n\t\tif", "", REGULAR, info, bufLength, buf);
5) findPosition(position, "\n\t\tif", "}", "\n\t\tif (MODE", "", REGULAR, info, bufLength, buf);
6) findPosition(position, "void draw_", nameOfFile, "{", "", REGULAR, info, bufLength, buf);
7) findPosition(position, "\tglRectf", "\n", "\tglFlush", "\toutput2", BEFORE_OPT, info, bufLength, buf); (Both the second and the fifth string will be searched for as the first string to be found.)

```
/* header.h: This header file contains information about all icons and
pages used by the Smart Chair program.*/
#define NUM_PAGE 1
8->#define MAX_BUTTON 10
int page[NUM_PAGE][MAX_BUTTON] =
11,12->     {{1, -99, 2}};
9->
int Coord[NUM_PAGE][MAX_BUTTON][2][2] =
11,13->     {{{{863, 628}, {1006, 542}}, {{911, 532}, {946, 514}}, {{21,
108}, {163, 22}}}};
10->
CString displayInfo[NUM_PAGE][MAX_BUTTON][2] =
14,15->     {{{"-99"}, {"Click", "1"}, {"-99"}}};
```

The following strings (the second, third, and fourth arguments to findPosition) must be present in the given order in the file, header.h, for the function findPosition (in Shp\ShpDoc.cpp) to work correctly:

8) findPosition(position, "*/", "\n", "#define MAX_BUTTON", "", REGULAR, info, headerLength, headerInfo);
9) findPosition(position, ";", "\n", "\nint Coord", "", REGULAR, info, headerLength, headerInfo);
10) findPosition(position, "", "", "\nCString displayInfo", "", REGULAR, info, headerLength, headerInfo);
11) findPosition(position, "=", "{", "{", "", REGULAR, info, headerLength, headerInfo);
12) findPosition(position, "},", "{", "{", "", REGULAR, info, headerLength, headerInfo);
13) findPosition(position, "}}},", "{", "{", "", REGULAR, info, headerLength, headerInfo);
14) findPosition(position, "=", "\t{", "{", "", REGULAR, info, headerLength, headerInfo);
15) findPosition(position, "}},", "{", "{", "", REGULAR, info, headerLength, headerInfo);