Winter 12

Headset Display Planetarium Prototype

Technical Documentation

Team Pancake: Lehi Alcantara, Michael Hughes. Kellie Kercher, Ian McGuire, Owen Riley

Brigham Young University School of Technology

Table of Contents

Software

Microsoft Visual Studio 2010 was the platform used to write and edit the program. MFPlayer Sample was the program that was used as a base for the video player. It can be downloaded here: http://msdn.microsoft.com/en-us/library/windows/desktop/bb970516(v=vs.85).aspx. MFPlayer Sample was written in Microsoft Visual Studio 2008 and will need to be converted to the 2010 format. The program is stored on a Dropbox shared folder, which allows for revisions and multiple people to view and edit the program. It is stored in a folder currently called 'current working.' To transfer the entire project onto another computer, just copy the 'current working' folder to the new location. Everything is self-contained in that project folder. The executable for each application can be found in the release folder of the different application folders. If space is an issue, the executables can be solely copied. However, before doing this the test application will need to be updated with the new paths of the MFPlayers. The test application is the parent program that opens both of the other applications.

Restrictions

This software, MFPlayers and the Opening Application cannot be successfully operated in a Virtual Machine.

Opening Application (Parent Program)

Restrictions

This application, named Test, consists of an interface that prompts the user for two video files. This application must be used to open the other MFPlayers. Each MFPlayer is expecting parameters, without them they will not run. A user can either open them through a shell or use the Test application.

Description

The first is the main educational video. The second is the ASL interpretation video. Following both video file selections, the user is asked to type in their first and last name only. Both names have to be included with one space in between or the MFPlayer will not be opened correctly. Once all the inputs are filled and the user confirms the entries, the application will use a shell command to open both MFPlayers with the coordinating video file paths. Each MFPlayer reads in the video path coming in through the call and opens the video immediately on pause. MFPlayer 1 also receives the user's name through the calling parameters. This will be used in the naming convention of the test subject's log file.

MFPlayer 1 (ASL Video and controls)

Restrictions

In order to start this program you must start the opening program first. If you go into Microsoft Visual Studio to edit the code, the debugging properties must include a real video path as a parameter.

Description

MFPlayer 1 is the video player that displays the ASL video on the augmented reality (AR) glasses, and the user controls on the main monitor/projector. The player is currently hard coded to send the video to the AR glasses (second monitor) off to the right of the main display, and is coded for a resolution of 800x600. This resolution can be changed, it is just hard coded in the application, but for optimal resolution on the current AR glasses it should be set to 800x600. The user controls are in a separate window that stays on the main display. The user controls can manipulate the following: play/pause, zoom, brightness, contrast, and position (up, down, left, and right).

MFPlayer 1 also includes the logging functions (log, OnStartTimer, OnStopTimer and logTimer) and the syncing function (writeState) for the two players. The logging functions track and record the user control settings every 2 seconds. This function writes to a CSV file that can be opened in Excel to view the data. The logging records the computer time, video time, brightness, contrast, zoom and the positions of the video. The 2-second writing variable can be changed to whatever time is found best to record. The syncing function writes a 'state.txt' file, simply writing 'play' or 'pause.' This happens anytime the user hits the play or pause button. The state.txt file is currently written to the same directory as the videos being played in MFPlayer 1.

MFPlayer 2 (Education video)

Restrictions

In order to start this program you must start the opening program first. If you go into Microsoft Visual Studio to edit the code, the debugging properties must include a real video path as a parameter.

Description

MFPlayer 2 is the video player that displays the educational video on the main display. The current player has the laptop's resolution hardcoded in, which is 1366 X 768. If another computer is used, this will need to be changed. All controls for MFPlayer 2 have been removed; the user will be controlling everything for the two videos from MFPlayer 1. MFPlayer 2 is programmed to have the video fit a dialog box that is the same size as the screen, immediately on the monitor. The user controls from MFPlayer 1 will be transparently overlaid on top of the MFPlayer 2 video. MFPlayer 2 includes the second half of the syncing function (syncState). This function is setup to read the 'state.txt' file every 100 milliseconds. It then has an if statement that checks to see if the file says 'play' or 'pause' and acts according to the instructions. For further understanding about the syncing state, see below 'Communication between MFPlayer 1 and MFPlayer 2.'

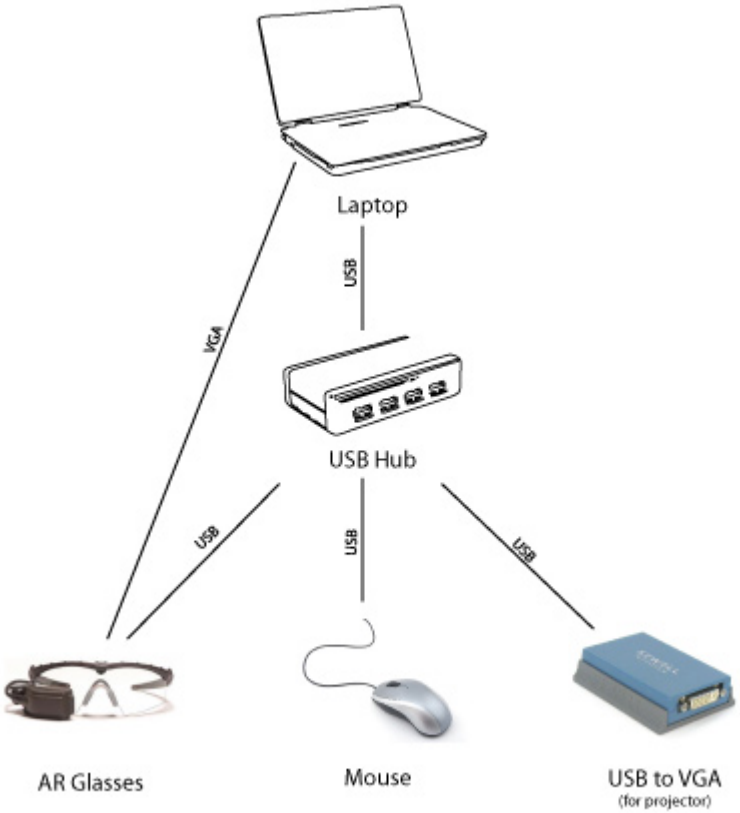
Headset Hookup

Hardware

Laptop (Dell Latitude E6220)
AR Glasses (Vuzix Tac-Eye)
USB Port Hub
USB to VGA adapter (if hooking up with projector as well)
Mouse

Explanation of Hookup

The Vuzix Tac-Eye glasses require two connections to the laptop, a VGA and a USB for power. The Dell Latitude E6220 only has 2 USB connections and thus a USB Port Hub is required. For user testing, if using the laptop's monitor and the AR glasses, no USB to VGA adapter is needed, just the USB Port Hub. If during user testing connection to a projector is required, use the USB to VGA adapter. Connect the USB Port Hub to the laptop and then connect the AR glasses' USB, the Mouse and the USB to VGA adapter to the USB Port Hub. Configure the display settings so that the projector and the laptop monitor are mirrored and the AR glasses act as a second display.



Custom/Modified Functions

Opening Application (Test)

Name: CTestDlg::OnBnClickedOk()
Location: TestDlg.cpp
Application: Test
Description: This function handles reading user input and then passes it on to the two MFPlayer scripts. It is required to set the path and path2 variables to the physical locations of the executables MFPlayer 2 and MFPlayer 1. First it gets the input of the first file browse. This is the main video file. Then it retries the input of the ASL video path and the user's name. The user has to return a first and last name. A shell request is called twice to open each MFPlayer with the coordinating video path in the parameters. In the case of MFPlayer 1, the user's name is also passed through the parameters in calling the script.

MFPlayer Settings

Name: MainDialog:OnInitDialog()
Location: MainDialog.cpp
Application: MFPlayer1
Description: The ability to make the control dialog invisible was added to this function

Name: MainDialog::OnPlayOpen()
Location: MainDialog.cpp
Application: MFPlayer1 & MFPlayer2
Description: The purpose of this function is to open the video path provided in the executable call. This script retrieves the video path provided in the parameters of the executable call.. It then converts it into a readable format to be opened into the video dialog.

Name: WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE /*hPrevInstance*/, LPWSTR lpstrCmdLine, int nCmdShow)
Location: winmain.cpp
Application: MFPlayer1 & MFPlayer2
Description: This method is used to initiate MFPlayer. We added in a command line parser. The video file's path and the log file's name are both passed through the command line to MFPlayer 1 and MFPlayer 2. szArgList is used to hold the parsed command line arguments. It is parsed by

CommandLineToArgvW(), which parses by white space. The Windows API allows us to convert each object in the szArgList array into a string by using T2A(szArgList[#]). In MFPlayer 1, we assume that the name of the test subject will be the last 2 arguments. Thus, a loop is used to append the path of the video file into one string. The last 2 arguments are combined into a separate variable that can be used later for the save location of the log file. This differs slightly in the MFPlayer 2 application. Instead, all variables are concatenated into a filename string since there is no name coming in.

Communication between MFPlayer1 and MFPlayer2

Name: MainDialog::writeState()
Location: MainDialog.cpp
Application: MFPlayer1
Description: writeState() - is called under case IDC_PLAY in the OnCommand Function. Opens the state file (Inside ARI Project folder). Checks the player state and writes "play" if the video is playing and writes "pause" if the video is paused

Name: OnStartTimer()
Location: MainDialog.cpp
Application: MFPlayer2
Description: Called by onPlayOrPause(). Starts checking the state.txt file every 100ms. It keeps calling CALLBACK syncState method, where mainDialogPointer in turn calls stateLog() for checking the state file.

Name: syncState(HWND hWnd, UINT nMsg,UINT nIDEvent, DWORD dwTime)
Location: MainDialog.cpp
Application: MFPlayer2
Description: Called by OnStartTimer(). Calls the stateLog() function.

Name: MainDialog::stateLog()
Location: MainDialog.cpp
Application: MFPlayer2
Description: Called by syncState(). This method essentially opens the state file and it acts according to what the state file is. If the state file says "play" then the method will call the OnPlay() method. If the state file says "pause" then the method will call the OnPause() method.

Name: MainDialog::OnPlay()
Location: MainDialog.cpp
Application: MFPlayer2

Description: Called by stateLog(). It checks the state (playing or paused) of MFPlayer2, if the video is paused the function Play() will be called to start playing the video. If the player is playing, no action will be taken.

Name: MainDialog::OnPause()
Location: MainDialog.cpp
Application: MFPlayer2
Description: Called by stateLog(). It checks the state (playing or paused) of MFPlayer2, if the video is playing the function Pause() will be called to pause the video. If the player is paused, no action will be taken.

Beginning of Logging

Name: MainDialog::MainDialog(string log) --- The constructor
Location: MainDialog.cpp
Application: MFPlayer1
Description: Constructs a new instance of MainDialog. When it is initiated, it will copy the name of the log file, which was passed through the method parameters. It then requests the location of the video via external calling. Finally, it loads the video to be ready to play.

Name: MainDialog::OnPlayOrPause()
Location: MainDialog.cpp
Application: MFPlayer1
Description: This method is called whenever the Play/Pause button is pressed. When the button is clicked to play, the method OnStartTimer() is called to begin the logging timer. When the button is clicked to pause, OnStopTimer() is called to kill the timer and stop logging.

Name: OnStartTimer()
Location: MainDialog.cpp
Application: MFPlayer1
Description: Begins a timer which calls LogTimer(). The timer is started with the SetTimer() function from the Windows API. It is set for 2000 miliseconds.

Name: OnStopTimer()
Location: MainDialog.cpp
Application: MFPlayer1
Description: Uses KillTimer() from the Windows API. The KillTimer() requires the variable that OnStartTimer() creates. Currently, this is named m_nTimer. KillTimer() will stop the timer which causes logging to be ended. It starts back up when OnStartTimer() creates a new instance of the timer.

Name: MainDialog::LogTimer(HWND hWnd, UINT nMsg,UINT nIDEvent, DWORD dwTime)
Location: MainDialog.cpp
Application: MFPlayer1
Description: Calls the log() method of the clone of MainDialog object. The privileges would not allow us to access all we needed. The clone is used to get all the necessary information. The log() method is where all the information and writing to a file is done.

Name: MainDialog::log()
Location: MainDialog.cpp
Application: MFPlayer1
Description: This method is used to get all the variables, open a file, write the variables, and then close the file.

Name: MainDialog::computerTimer()
Location: MainDialog.cpp
Application: MFPlayer1
Description: The current time in relation to the computer is requested. C++ has an object to retireve time, time_t. This is used and gets the localtime. It is then converted with a provided method into a string. The string variable, time, is returned in the proper format to log().

Name: MainDialog::OnTimer()
Location: MainDialog.cpp
Application: MFPlayer1
Description: This method helps to update the seekbar if it was used. However, to help our methods, a clone of MainDialog is made. The clone is updated every ½ second to make sure it is as current as possible without using too many resources. In addition, if it is already equal, it is not updated.

Notable Variables:
Name: string logFileName
Location: MainDialog.cpp
Application: MFPlayer1
Description: This variable is the path to where the program will store the log file. It begins with the folder where it will be stored. In the constructor (MainDialog), the name of the log file is appended onto the end. It is used in log() to open, write to, and close the file.

Name: MainDialog *clone
Location: MainDialog.cpp
Application: MFPlayer1
Description: The variable is a clone of MainDialog once MainDialog is running. This is so that we can have all the permissions and access all the information we need. It is updated every ½ seconds in OnTimer().

Brightness and Position Functions

Note: These functions are called in MainDialog.cpp under the functions "OnScroll" and "MoveX" , and are attached to the UI controls in the function "InitializeControls"

Name: PlayerVideo::SetBrightness(float fBrightness)
Location: Feature_Video.cpp
Application: MFPlayer1
Description: This function takes in a float value that represents the brightness and sets the ASL Video's brightness accordingly. It is called whenever the brightness slider is dragged. Values range from -100 to 100

Name: PlayerVideo::SetContrast(float fContrast)
Location: Feature_Video.cpp
Application: MFPlayer1
Description: This function is the same as SetBrightness, except for contrast. Values range from 0 to 2

Name: PlayerVideo::MoveUp()/MoveDown()/MoveRight()/MoveLeft()
Location: Feature_Video.cpp
Application: MFPlayer1
Description: These functions handle the position of the ASL video. When any arrow button is clicked on the UI, the corresponding function is called which moves the window in that direction by an amount defined by the variable "interval".

Name: PlayerVideo::SetVideoPosition()
Location: Feature_Video.cpp
Application: MFPlayer1
Description: This is a generic function that is called by all four of the position functions above. Each of these functions modifies the corresponding top left and bottom right corners of the display. This function then applies those new values and sets the actual position of the video.

Files we Created: log.text and state.txt

Name: Log File
Application: MFPlayer1
Description: The log file is being saved in the CSV format. The path is currently on the Desktop in the Logs folder. The path can be modified in the MainDialog.cpp in MFPlayer1. The program will write to this file every 2 seconds. It creates a new row for each time it records the variables. The variables are labeled per column. They include: Zoom, Brightness, Contrast,

Left Position, Top Position, Right Position, Bottom Position, Video Time, and Computer Time. The values for Zoom and Contrast are from 0 to 1. Brightness is from -1 to 1.

Name: State file
Application: MFPlayer1 and MFPlayer2
Description: The state file enables MFPlayer1 and MFPlayer2 to communicate with each other. When the Play/Pause button is pushed in MFPlayer1, "play" or "pause" will be written to state.txt. MFPlayer2 is constantly checking state.txt to see what is written.

Changing the Application

The paths from test.exe to MFPlayer1.exe and MFPlayer2.exe in the ARI folder are different from the paths in the source code. When updating the executables in the ARI folder the following paths need to be changed before the executable is created. Correct paths are commented out in the code.

Application: Test
Location: TestDlg.cpp
Method: CTestDlg::OnBnClickedOk()
Change:

```
TCHAR path[] =
TEXT("..\\..\\MFPlayer2\\MFPlayer\\Release\\MFPlayer.exe");
to
TCHAR path[] = TEXT("MFPlayer2.exe");
```

Application: MFPlayer1
Location: MainDialog.cpp
Method: MainDialog::writeState()
Change:

```
myFile.open ("C:\\Users\\Public\\Videos\\Sample
Videos\\state.txt");
to
myFile.open ("state.txt");
```

Location: Constants section at the beginning of the file
Change:

```
string logFileName = "C:\\Users\\ARI\\Desktop\\Logs\\";
to
```

string logFileName = "Logs\\";

Application: MFPlayer2
Location: MainDialog.cpp
Method: stateLog()
Change:

myFile.open ("C:\\Users\\Public\\Videos\\Sample Videos\\state.txt");

to

myFile.open ("state.txt");