1 INTRODUCTION

The Eyegaze Edge Analysis Systemprovides a platform to develop and run eyetracking applications programs. This manual discusses the software programming interface for the Eyegaze Edge.

As an eyetracking instrument, the fundamental purpose of the Eyegaze Edge is to monitor a person's eye and continually measures the x-y coordinate of his gazepoint on a computer monitor. The eye is monitored by a video -camera, and the processing of the eye image is performed in software on a Personal Computer. The basic Eyegaze eyetracking instrument includes the camera, the PC, the frame-grabber hardware, and the Eyegaze image processing software (import library and dynamic link library).

For eyetracking application programs to access the data from the eyetracking instrument, the Eyegaze Edge Analysis Systemincludes an application program interface (API) called EgWin. EgWin interfaces Eyegaze to Windows NT/2000 programs written in C/C++. The Eyegaze Edge Analysis Systemincludes the Microsoft Visual C++ compiler, Version 6.0.

EgWin provides real-time access to the eyetracking data from the Eyegaze Edge. EgWin consists of four basic functions: EgInit(), EgCalibrate(), EgGetData() and EgExit(). The Eyegaze image processing software runs autonomously as a separate program thread, triggered by interrupts at 60 Hz (50 Hz CCIR) from the frame-grabber card. Eyegaze data is made available to the application program through an internal ring-buffer of structures, and the application controls Eyegaze through a structure called stEgControl. Eyegaze data is obtained by calling EgGetData().

Eyegaze application programs may be run directly on the Eyegaze computer, called the "single computer configuration", or on a separate computer, called the "double computer configuration," where the Eyegaze Edge acts as a peripheral device and communicates with the client computer via an Ethernet or serial communications link. The program's EgWin calls are essentially the same for both the single and double computer configurations - an argument in the EgInit() call tells EgWin which hardware configuration is being used to communicate with the application program.

The Eyegaze Edge Analysis Systemalso includes source code for several example eyetracking programs, such as the Trace program, which illustrates gazepoint capture, storage, and subsequent replay. To extract user fixations from the raw sequence of user gazepoints measured by Eyegaze, source and object code is provided for a function called DetectFixation().

Section 2 lists and summarizes all the Eyetracking software delivered with the Eyegaze Analysis System.

Section 3 discusses the single and double computer configurations that can be used to implement Eyegaze applications programs.

Section 4 describes the details of the EgWin programming interface; Section 5 provides the EgWin function reference guide, and Section 6 discusses compiling, linking and running Eyegaze Application programs.

Section 7 discusses the example Eyegaze Application programs provided with the Eyegaze Analysis System, and Section 8 provides a function reference guide for the fixation detection functions.

Sections 9 and 10 discuss the Eyegaze Edge sampling rate and the instrument's intrinsic gazepoint measurement delay.

2 EYEGAZE DIRECTORY CONTENTS

All of the Eyegaze-specific software on the Eyegaze Edge resides in the C:\EYEGAZE directory and subdirectories. The EYEGAZE directory contains the following files:

2.1 C Source Code and Executable Files

The following C source code files illustrate use of the Eyegaze software. The code may be used as is, modified to meet custom needs, or used as references for preparing other Eyegaze application programs.

GazeDemo.c GazeDemo.exe	GazeDemo is a very simple Eyegaze program that moves the mouse cursor around the screen as the user looks around. The purpose of the program is to illustrate the C code required to set up the Eyegaze software and track the user's gazepoint. The mouse pointer is used as a convenient marker to move around the screen but serves no other useful purpose. (For more information, see Section 7.1.)
Trace.c Trace.rc Trace.exe	Trace is a more complex example of an Eyegaze application. First, during a data collection phase, this program presents a display on the computer monitor screen and collects the gazepoint and pupil diameter data as the subject moves his eyes around. The gaze trace is also stored to a file. Second, during a replay phase, it superimposes the gazepoint trace on the original display. The gazepoint trace may be displayed statically or replayed dynamically. Eye position and pupil diameter versus time plots are also displayed. Trace presents a text or graphics screen for the user to scan. (For more information, see Section 7.3.)
EgServer.c EgServer.rc EgServer.exe	The EgServer program allows the Eyegaze Edge to act as a peripheral device and communicate with a client computer (in the Double Computer Configuration - See Section 3.2). Via either an Ethernet or serial communications link, EgServer receives and executes commands from the client computer. It performs the calibration procedure for a person looking at the client computer's monitor and transmits gazepoint data to the client in real time. (For more information on the EgServer program, see Sections 3.3 and 3.4)
EgClientDemo.c EgClientDemo.rc EgClientDemo.exe	The EgClientDemo program, which operates in a computer that is using the Eyegaze Edge as a peripheral device, demonstrates client computer interaction with the Eyegaze Edge. The code is intended to be a starting point for developers to create their own Eyegaze application programs on client computers or workstations. (For information on the Double Computer Configuration, see Section 3.2. For more information on the EgClientDemo program, see Section 7.2.)

Calibrate.exe	Calibrate performs the Eyegaze calibration procedure and writes the calibration coefficients into the PRES_CAL.DAT file. (See "Calibration Options" in Section 4.17 for a discussion of alternative ways to call Eyegaze calibration. An example of the use of this Calibrate.exe program is given in GazeDemo.c.)
fixfunc.c	The fixfunc functions extract fixation and saccade data from raw gazetracking data. See Section 8. These functions are included in Lctigaze.dll.
winsupt.c winser.c	Eyegaze application programs are written with generic function calls. The generic functions have the prefix lct_, such as lct_line() and lct_rectangle(). The conversion from the generic lct_ functions to the Win32 API is performed by "intermediate function layers". The conversion between the lct_ generic calls and the Win32 API is performed by the intermediate graphics layers WINSUPT.C and WINSER.C.

2.2 Eyegaze Header Files

EgWin.h	EgWin.h provides function prototypes for all the primary eyetracking functions and defines externally-available variables and constants used by these functions. This file should be included in all eye tracking application programs.
fixfunc.h	fixfunc.h provides the function prototypes and constant definitions for the fixation detection functions.
Igutil.h	Igutil.h contains function prototypes for the Eyegaze Utility functions and defines several constants used by the utility functions. Because the Igutil functions access Eyegaze data, they are useful only with Eyegaze programs.
lctcolor.h	Lctcolor.h defines mnemonic names for the 16 basic numeric color values used in the graphics functions.
lctfont.h	Lctfont.h defines the font type numbers used in Eyegaze application programs.
lctkeys.h	Lctkeys.h defines constants for many of the non-alphanumeric keyboard keys such as SPACE, ENTER, ESCAPE, LINE_FEED, F1, F2, SHIFT_F1, ALT_F1, HOME and PGUP, RIGHT_ARROW, etc.
lctsupt.h	Lctsupt.h contains function prototypes for functions that mapped general-purpose function calls to third-party software packages.
lctypdef.h	Lctypdef.h contains abbreviated definitions for many of C's data types. Lctypdef.h is included in EgWin.h so need not be explicitly included in eyetracking application programs.

2.3 Library Files

lctigaze.lib lctigaze.dll	Lctigaze.lib/dll contains all the primary eyetracking functions and support functions. The functions within the dll require several runtime library functions that are provided with the Microsoft C compiler. Eyegaze application programs running in the Single Computer configuration (see Section 3.1) link with lctigaze.lib and require lctigaze.dll at runtime.
EgClient.lib EgClient.dll	Eyegaze application programs running in the Double Computer configuration (see Section 3.2) link with EgClient.lib and require EgClient.dll at runtime.

2.4 Other Support Files

calibrat.lab	Screen labels used by calibration
EyeImage.state	State information for on-screen eye image display
pres_cal.dat	Calibration results
*.bmp *.dim	Bitmaps for the Trace program - The *.bmp files are normal Windows bitmaps. The trace program creates versions of these bitmaps for its own use and names them *.dim. If the custom version of a bitmap is not present, the Trace program will create it when it needs it.
*.txt	Text files for the Trace program
*.dsw	Visual C "workspace" file.
*.dsp	Gazedemo, EgClientDemo, EgServer, and Trace project files

3 SINGLE VS DOUBLE COMPUTER CONFIGURATIONS

The Eyegaze Edge can be used two ways. In the Single Computer configuration, an eyetracking application program may be written to run directly on the Eyegaze Edge computer. In the Double Computer configuration, the application program runs on another (client) computer, and the Eyegaze computer acts as a peripheral eyetracking instrument, becoming a "black box" that transmits gazepoint data to the client via an Ethernet or serial communications link.

The single and double computer configurations for the Eyegaze Edge are illustrated in Figure 1.

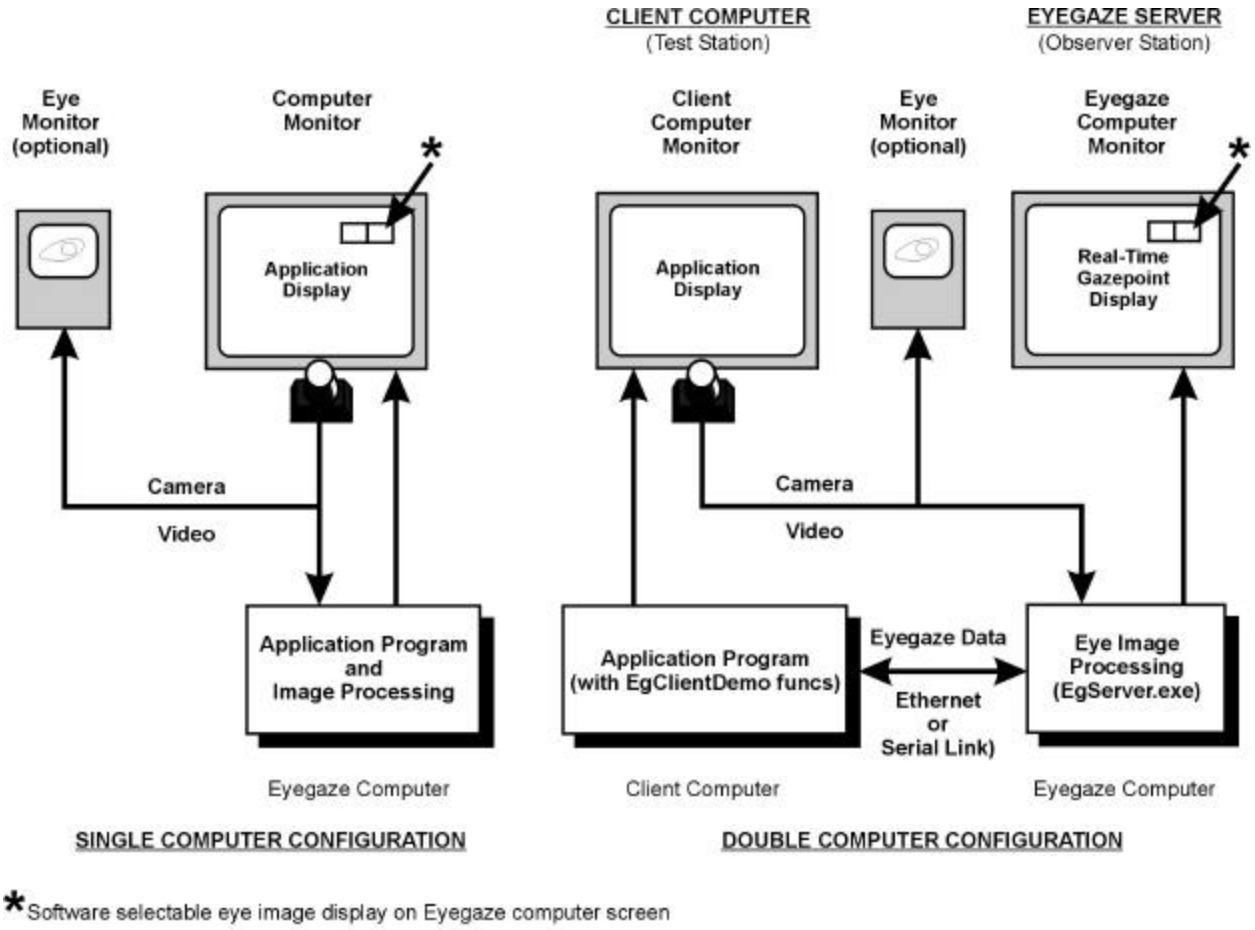


Figure 1: Eyegaze Single and Double Computer Configurations

Note: The application program's Eyegaze code for the Single and Double computer configurations are virtually identical (except for the specifications of iCommType and pszCommName in the stEgControl structure as discussed in Section 4.7), but the linking procedures are different (see Section 6).

3.1 Single Computer Configuration

If the Eyegaze application program is written to run directly on the Eyegaze Edge computer:

a) the Eyegaze camera is mounted below the Eyegaze computer's monitor,

b) the Eyegaze computer performs the applications functions and drives the application display on its monitor, and

c) the Eyegaze computer also performs the Eyegaze image processing functions required to track the test subject's gazepoint.

The single computer configuration is preferable if:

- it is desired to minimize the amount of equipment needed to implement the project.

3.2 Double Computer Configuration

If the Eyegaze application program is written to run on a second (client) computer:

a) the client computer performs the application functions and drives the applications display on its monitor,

b) the Eyegaze camera is mounted below the client computer's monitor,

c) the Eyegaze computer (running a program called EgServer) performs the Eyegaze image processing functions and transfers the measured gazepoint data to the client computer via either an Ethernet or a serial communications link in real time, and

d) the Eyegaze computer's monitor displays the subject's relative gazepoint in real time, allowing a test observer to view the gazepoint activity on-line during a test.

The double computer configuration is preferable if:

a) the application code consumes a large amount of CPU time (i.e. if there is not enough CPU time to execute both the application and Eyegaze image processing code in real time),

b) the application program is not compatible with Windows NT/2000, or

c) an observer station is desired for viewing a test subject's eyegaze activity online.

3.3 EgServer: Eyegaze Edge Program to Serve a Client Computer

When the Eyegaze Edge is used as a peripheral device in the Double Computer Configuration, it must run the EgServer program to communicate with the client computer. EgServer is designed to communicate with a client program such as EgClientDemo (see Section 7.2), running on the client computer. With the Eyegaze camera mounted under the client computer monitor, the Eyegaze Edge measures subjects' gazepoints on the client computer's monitor. In response to commands from the client computer, the EgServer program:

a) calibrates the Eyegaze Edge for a person looking at the client computer's monitor, and

b) transmits gazepoint data to the client computer in real time.

EgServer also presents on-line eye-image and gazepoint-tracking displays for experiment observers.

EgServer must be running on the Eyegaze computer before the client program is started.

3.4 Eyegaze Observer Displays in EgServer Program

To allow a test observer to monitor a subject's gazepoint activity on-line during gazepoint tracking sessions, EgServer provides real-time Eyegaze displays during calibration and testing sessions. Since the Eyegaze computer's monitor is different from the client computer monitor, these displays may be made available to experimental observers without distracting the test subject.

EgServer presents camera images of the eye in the upper right corner of its display window. As discussed more thoroughly in Sections 3.4 and 3.5 of the User's Guide, the Eye Image Displays include a full camera image, a magnified display of the pupil and corneal reflection, gazepoint tracking indicators, and a focus offset indicator. The Eye Image Displays are useful for determining that the camera remains properly pointed at and focused on the subject's eye.

During calibration, the EgServer window mimics the calibration display on the client computer, displaying the sequence of calibration points as the user progresses though the calibration procedure.

At all times other than calibration, EgServer displays a cross representing the subject's measured gazepoint on the client monitor, whether or not the gazepoint data are being sent to the client.

The window within the EgServer display represents the full screen of the client monitor. If the subject's gaze is within the client computer screen, the cross is plotted in yellow. If the gazepoint is off the client screen, the color of the cross is converted to green, and is plotted at the edge of the window closest to the predicted gazepoint. When the gaze is not tracked, the cross is plotted in red at the last tracked location.

Although the C source code for EgServer is provided, no modifications to the program are required for typical double-computer eyetracking applications.

4 EGWIN PROGRAMMING INTERFACE

Eyegaze runs on Windows NT and Windows 2000, and may be run from C or C++ programs. The Eyegaze functions run as a thread within your program.

4.1 Overview of Basic Operation

The programming interface is encapsulated in the "EgWin interface", which consists of a small number of Eyegaze application programming interfaces (APIs), C data structures, and defined constants. The complete API is defined in the **EgWin.h** header file. The object code for the Eyegaze functions is contained in the Eyegaze dynamic link library called **lctigaze.dll**.

The core of the EgWin interface consists of four principal functions:

EgInit()	creates and starts the Eyegaze thread. It also allocates the system resources associated with the frame grabber.
EgCalibrate()	performs the Eyegaze calibration procedure.
EgGetData()	synchronizes the application with the Eyegaze Camera field rate, and it tells the application program where to access the most recent eyetracking data.
EgExit()	shuts down Eyegaze, i.e. terminates the EgWin thread. It also releases the system resources associated with the frame grabber.

The Eyegaze application program must define a control data structures:

stEgControl	contains control parameters through which the application program controls the Eyegaze image processing functions.

Prior to calling any EgWin functions, the program initializes several control parameters in the stEgControl structure. The first Eyegaze call that the program makes is to EgInit().

After calling EgInit(), the application typically calls EgCalibrate(), which calibrates the Eyegaze Edge to the current subject.

To start collecting eyetracking data, the program sets stEgControl.bTrackingActive to TRUE. The Eyegaze image processing thread is interrupt-driven and when bTrackingActive is TRUE it processes each camera field image and places the gazepoint data into an internal ring buffer. The application calls the EgGetData() function to retrieve the next available gazepoint from the ring buffer. Typically, the program uses a tight loop to access and process the eyetracking data as it is generated. If no new unprocessed gazepoint data is available, the EgGetData() function waits for Eyegaze to produce the next gazepoint data sample.

Before the Eyegaze application program terminates, it calls EgExit() to terminate the Eyegaze functions.

The Eyegaze API is essentially identical for the Single and Double Computer configurations of the Eyegaze Edge. The application program informs the Eyegaze software about the hardware configuration by setting an appropriate value for stEgControl.iCommType.

4.2 Eyegaze Pseudo Code Example

The following pseudo code demonstrates typical function usage in an Eyegaze application program:

```
{
   allocate stEgControl structure
   setup stEgControl structure
   EgInit(stEgControl);           // create and start the Eyegaze thread
   if (new user)
   {
      EgCalibrate(&stEgControl, hwnd, EG_CALIBRATE_NONDISABILITY_APP);
                                  // have person perform the calibration procedure
   }

   for (loop)
   {
      EgGetData(&stEgControl);    // wait for and retrieve gazepoint from next camera
field
      use Eyegaze data from stEgControl.pstEgData
   }

   EgExit(&stEgControl);          // terminate the Eyegaze thread
}
```

For details on the use of the EgWin functions, see "EgWin Function Reference", Section 5.

4.3 Data Logging Functions

In addition to the four principal functions outlined above, there are several data logging functions that are used to write Eyegaze data to a disk file for later reference and more detailed analysis. The files are recorded on the Eyegaze Edge computer.

EgLogFileOpen()	opens a data log file -- the file name is specified as a function argument.
EgLogAppendText()	inserts one or more comment lines into the data log file.
EgLogStart()	begins logging eyetracking data to file -- one data line per sample.
EgLogStop()	stops logging eyetracking data to file.
EgLogWriteColumnHeader()	write column headers to the data log file.
EgLogMark()	increments a numerical event marker recorded in the log file.
EgLogFileClose()	closes the data log file.

These functions open/close a named ASCII text file, append text to the file (for marking system or application events), start/stop recording Eyegaze data, and increment the event marker integer in the file.

4.4 Eyegaze Control Data

The stEgControl structure contains:

1) control variables to Eyegaze -- these variables are set by the application program to setup and control the Eyegaze image processing software,

2) indicators from Eyegaze:
   a) ring buffer index of the most recent Eyegaze data sample,
   b) a ring buffer overflow indicator,
   c) the camera sampling rate,
   d) Eyegaze monitor scale factors, and
   e) address of the video buffer containing the most recent camera image.

The _stEgControl structure template is defined as follows:

```
struct _stEgControl
{                                   /* CONTROL INPUTS FROM APPLICATION:       */

   struct _stEgData *pstEgData;     /* pointer to the Eyegaze data structure  */
                                    /*   where EgGetData() places the next    */
                                    /*   Eyegaze data point.                  */
                                    /*   This memory is allocated by the      */
                                    /*   EgInit() function.  The pointer to   */
                                    /*   data structure is returned to the    */
                                    /*   application.                         */
   int    iNDataSetsInRingBuffer;   /* number of gazepoint samples in the     */
                                    /*   Eyegaze ring buffer                  */
                                    /*   The application must set the ring-   */
                                    /*   buffer length in the stEgControl     */
                                    /*   structure before calling EgInit()    */
   BOOL   bTrackingActive;          /* flag controls whether eyetracking is   */
                                    /*   presently on (TRUE = 1) or off       */
                                    /*   (FALSE = 0). If the flag is on when  */
                                    /*   a new camera field finishes, the     */
                                    /*   Eyegaze thread processes the image   */
                                    /*   and puts the results in the data     */
                                    /*   ring buffer; if the flag is off,     */
                                    /*   the camera field is not processed.   */
                                    /*   The application may turn this        */
                                    /*   tracking flag on or off at any       */
                                    /*   time.                                */
                                    /* NOTE for non-Windows users: This BOOL  */
                                    /*    is a 4-byte integer                 */
   int    iScreenWidthPix;          /* Pixel dimensions of the full           */
   int    iScreenHeightPix;         /*   computer screen                      */
   BOOL   bEgCameraDisplayActive;   /* flag controls whether or not the       */
                                    /*   full 640x480 image from the Eyegaze  */
                                    /*   camera is displayed in a separate    */
                                    /*   window on the VGA.                   */
                                    /*   The application must set this flag   */
                                    /*   prior to calling EgInit() and may    */
                                    /*   turn the display flag on and off     */
                                    /*   at any time.                         */
   int    iEyeImagesScreenPos;      /* Screen position of the eye images.     */
                                    /*   0 = upper left, 1 = upper right      */
   int    iCommType;                /* Communication type:       Comp Config: */
                                    /*   EG_COMM_TYPE_LOCAL,        Single    */
                                    /*   EG_COMM_TYPE_SOCKET, or    Double    */
                                    /*   EG_COMM_TYPE_SERIAL.       Double    */
   char   *pszCommName;             /* Pointer to serial port name or IP      */
                                    /*   address of server machine.           */
                                    /*   used only in Double Computer Config  */
   int    iVisionSelect;            /* Reserved – set to 0                    */

                                    /* OUTPUTS TO APPLICATION:                */

   int    iNPointsAvailable;        /* number of gazepoint data samples       */
                                    /*   presently available for the          */
                                    /*   application to retrieve from the     */
                                    /*   Eyegaze internal ring buffer         */
   int    iNBufferOverflow;         /* number of irretrievably missed gaze-   */
                                    /*   point data samples, i.e. the number  */
                                    /*   of valid data points at the tail     */
                                    /*   of the ring buffer that the          */
                                    /*   application program has not          */
                                    /*   retrieved but that Eyegaze overwrote */
                                    /*   since the application last called    */
                                    /*   EgGetData()                          */
   int    iSamplePerSec;            /* Eyegaze image processing rate -        */
                                    /*   depends on the camera field rate:    */
```

```
                                /*   RS_170              60 Hz          */
                                /*   CCIR                50 Hz          */
   float  fHorzPixPerMm;        /* Eyegaze monitor scale factors          */
   float  fVertPixPerMm;        /*   (pixel / millimeter)                 */
   void   *pvEgVideoBufferAddress;/* address of the video buffer containing */
                                /*   the most recently processed camera   */
                                /*   image field                          */

                                /* INTERNAL EYEGAZE VARIABLE              */

   void   *hEyegaze;            /* Eyegaze handle -- used internally by   */
                                /*   Eyegaze to keep track of which       */
                                /*   vision subsystem is in use.          */
                                /*   (not used by application)            */
};
```

For example code to set up the Eyegaze control variables, see the EgInit() function reference, Section 0

4.5 Eyegaze Output Data

The stEgData structure contains the results of the Eyegaze image processing from a given field of video camera data. The structure contains:

a flag indicating whether valid gazepoint data was measured in the camera field image,
the user's gazepoint on the computer screen,
the pupil radius,
the location of the eyeball in space,
the focus condition of the camera on the eye,
the time the camera image was taken,
the time and count of the last prior event timing mark sent by the client application.

The _stEgData structure template is defined as follows:

```
struct _stEgData
{
   BOOL   bGazeVectorFound;     /* flag indicating whether the image      */
                                /*   processing software found a valid    */
                                /*   glint pupil vector in the camera     */
                                /*   image field                          */
                                /*   (TRUE = 1, FALSE = 0)                */
                                /* NOTE for non-Windows users: This BOOL  */
                                /*    is a 4-byte integer                 */
   int    iIGaze;               /* integer coordinates of the user        */
   int    iJGaze;               /*   gazepoint - with respect to the full */
                                /*   computer screen (pixels)             */
                                /*   0,0 origin at upper left corner      */
                                /*   iIGaze positive rightward            */
                                /*   iJGaze positive downward             */
   float  fPupilRadiusMm;       /* actual pupil radius (mm)               */
   float  fXEyeballOffsetMm;    /* offset of the eyeball center from      */
   float  fYEyeballOffsetMm;    /*  the camera axis (mm)                  */
                                /*  Notes on polarity:                    */
                                /*  x positive: head moves to user's right */
                                /*  y positive: head moves up             */
   float  fFocusRangeImageTime; /* distance along the camera pointing     */
                                /*   axis from the camera sensor plane    */
                                /*   to the camera focus plane, at the    */
                                /*   time the camera captured the image   */
                                /*   (mm)                                 */
   float  fFocusRangeOffsetMm;  /* range offset between the camera focus  */
                                /*   plane and the corneal surface of the */
                                /*   eye - at image time (mm)             */
                                /*   A positive offset means the eye is   */
                                /*   beyond the lens' focus range.        */
   float  fLensExtOffsetMm;     /* distance that the lens extension would */
                                /*   have to be changed to bring the eye  */
                                /*   into clear focus (millimeters)       */
                                /*   (at image time)                      */
   ULONG  ulCameraFieldCount;   /* number of camera fields, i.e. 60ths of */
                                /*   a second, that have occurred since   */
```

```
                                /*   the starting reference time (midnight */
                                /*   January 1, this year)                 */
   double dGazeTimeSec;         /* Application Time that the gazepoint was */
                                /*   actually valid, i.e. original         */
                                /*   image-capture time, not gazepoint-    */
                                /*   calculation time                      */
                                /*   (seconds since call to EgInit())      */
   double dAppMarkTimeSec;      /* Application Time of the last recorded   */
                                /*   client event mark that occurred prior */
                                /*   to the image-capture time             */
                                /*   (seconds since call to EgInit())      */
   int    iAppMarkCount;        /* application's event-mark count          */
                                /*   corresponding to dAppMarkTimeSec      */
   double dReprotTimeSec;       /* application time that the gazepoint is  */
                                /*   is reported, i.e. the time that this  */
                                /*   data sample is released by EgGetData()*/
                                /*   or written to disk via EgLogStart()   */
};
```

NOTES:

1) bGazeVectorFound is typically false when the user blinks, moves his eye out of the camera field of view, or looks outside the gaze tracking cone (see Section 1.11 of the User's Guide).

2) As discussed in Section 4.6, "Gazepoint-Coordinate Reference Frame," the gaze coordinates iIGaze and iJGaze, are given within the full screen coordinates, not the client window coordinates.

3) Due to the latency period associated with capturing, digitizing, and processing the eye image (see Section 10), the processed gazepoint data represents the state of the eye approximately two camera field intervals ago (approximately 30 ms before the data is made available to the application). The term "at image time" is used to emphasize that the measurement data represents the state of the eye at the time the camera originally captured the image, i.e. the time the camera collected the photons from the eye, not the time that the gazepoint data becomes available.

4.6 Gazepoint-Coordinate Reference Frames

At is lowest level, i.e. at the EgWin program interface level, the Eyegaze Edge computes the user's gazepoint in terms of *full-screen* coordinates, not in terms of the *client-window* coordinates. The origin of the EgWin gazepoint coordinate frame (iIGaze = 0 and iJGaze = 0) is the upper left corner of the full-screen display.

Since EgWin reports gazepoint coordinates within the full-screen frame of reference, ***Eyegaze application programs are responsible for converting the gazepoint from full-screen to client-window coordinates.*** The examples in the EgGetData() function reference, Section 5.4, show typical conversions from full-screen to window coordinates. The variables iWindowHorzOffset and iWindowVertOffset represent the offset of the bitmap display area from the upper left corner of the screen. Note: these offsets vary depending on the screen resolution, the widths and heights of the sizing border, and the height of window title bar. Windows may also be moved during program operation, so it is critical for the application to remain aware of the window locations. The graphics support code in Appendix I illustrates how the application program can intercept window-repositioning messages from Windows to keep the window location data current.

For example, the Eyegaze **Trace** program, a typical Eyegaze application program, performs the full-screen to window transformation and records gazepoints within the *client-window* coordinate frame, not in terms of the full-screen display. Thus in trace.dat, a gazepoint of 0, 0 means that the user is looking at the upper left corner of the *displayed image*, not the upper left corner of the full screen.

4.7 Eyegaze Run Time Operation – Interrupt Driven Thread

During run-time, the Eyegaze software operates as a high-priority interrupt-driven thread that automatically processes the gazepoint data at the camera field rate. As each video field arrives from the frame grabber, Eyegaze performs the following operations:

1. The frame grabber generates an interrupt.
2. the frame-grabber's software drivers call back to the Eyegaze image processing software.
3. The Eyegaze image processing software processes the camera image and places the gazepoint data sample in the next element of a ring storage buffer (see Section 4.11).
4. The image processing software terminates until the next interrupt.
5. Eyegaze sets a semaphore which allows the EgGetData() function to return to your application with the new gazepoint data sample.

Because the image processing functions are interrupt driven and operate within their own thread, *the application program is not responsible for scheduling the Eyegaze image processing operations*. The image processing occurs automatically at a 60 Hz rate. The application has only to request the next data sample from the Eyegaze history buffer, which it does through the EgGetData() function.

The Eyegaze image processing typically consumes only a small percentage of the CPU time each video field interval. It never consumes more than half the 16.7 millisecond interval.

The Eyegaze image processing functions are given a high scheduling priority so they keep pace with the camera images generated in real time.

4.8 Retrieving Eyegaze Data

To retrieve a sample of Eyegaze data, the application program calls the **EgGetData()** function. At its simplest level of operation, the EgGetdata() function performs the following operations:

1. It waits until a new gazepoint data sample is available from Eyegaze.
2. It copies the gazepoint data sample into the stEgControl.pstEgData structure.
3. It returns to the application.

Note: EgGetData() does not perform any image processing; it only waits for new data (if appropriate), copies a gazepoint data sample, and returns. Recall that the image processing is done in the separate Eyegaze thread.

Typically, applications use a loop to collect and process a series of eyetracking data points:

```
for (loop)                    // data collection/processing loop
{
   EgGetData();               // wait for and retrieve the next gazepoint data sample
   - application code -       // process the gazepoint sample
}
```

Such a loop allows continuous and real-time processing of the gazepoint data as it becomes available. The call to EgGetData() waits for the next gazepoint data sample to arrive from the Eyegaze software and copies the sample into the application's copy of the seEgData structure. The application then processes the gazepoint data sample as it wishes.

Since the EgGetData() function waits for the next gazepoint data sample to become available, the function effectively synchronizes the data collection/processing loop to the 60 Hz sampling rate of the Eyetracker camera.

4.9 EgGetData() Waits by Blocking

The EgGetData() function uses a block on a semaphore to implement the time delay between the call to EgGetData() the availability of the next Eyegaze sample. Thus processor time is not wasted during the wait period.

4.10 Coordinating Client Application Events with Eyegaze Sample Times

There is an approximately 25 millisecond latent delay between the time when a gazepoint is actually valid (i.e. when the camera takes the picture of the eye) and the time that Eyegaze completes the computation of the gazepoint. Thus the gazepoint coordinate returned by EgGetData() represents a gazepoint that actually occurred approximately 25 ms earlier. (As explained in more detail in Section 10, "Gazepoint Measurement Delay," the delay results from the combination of the camera shutter period, image digitization, and image processing.)

To account for this Eyegaze measurement latency when recording real-time events, Eyegaze maintains an internal clock that represents the client's "**application time**," and keeps records of:

a) The application times when the Eyegaze camera originally captured eye images, and
b) The application times when application **"event marks"** were originally received from the client.

The client sends real-time event marks to Eyegaze by calling EgLogMark (see Section 5.6). The application may call EgLogMark() at any time, and Eyegaze maintains a record of when it originally received the event marks.

When Eyegaze finishes computing a gazepoint, it reports the following timing data along with the gazepoint coordinates:

1. **dGazeTimeSec** -- the Application Time that the gazepoint was *actually valid*. (dGazeTimeSec represents the original image-capture time, *not* the time that the gazepoint calculation was completed),
2. **dAppMarkTimeSec** -- the Application Time of the *last* recorded client event that occurred *prior* to the corresponding image-capture time, and
3. **iAppMarkCount** -- the event-mark count corresponding to the last event mark time dAppMarkTimeSec.
4. **dReportTimeSec** -- the Application Time that Eyegaze reported the gazepoint

The above variables are included in the stEgData data structure.

The table below illustrates a sequence of gazepoint samples reported by Eyegaze.

samp indx	Eye Found	Gazepoint X	Gazepoint Y	Pupil Diam	Eyeball-Position X	Eyeball-Position Y	Eyeball-Position Z	Focus Range	Mark Cnt	Last Mark Time	Gaze Comp Time
	(t/f)	(pix)	(pix)	(mm)	(mm)	(mm)	(mm)	(mm)	(-)	(sec)	(sec)
14	1	212	120	3.85	8.1	-2.7	2.7	698.0	0	00000.0000	00000.2298
00000.2598											
15	1	204	138	3.80	8.4	-2.7	3.2	698.0	0	00000.0000	00000.2555
00000.2855											
16	1	45	249	3.94	11.1	-2.1	3.2	698.0	1	00000.2604	00000.2722
00000.3022											
17	0	0	0	0.00	0.0	0.0	0.0	698.0	1	00000.2604	00000.2887
00000.3187											
18	1	42	245	3.95	11.0	-2.1	2.8	698.0	1	00000.2604	00000.3055
00000.3350											

As can be seen in the above example, Eyegaze reports the mark and gaze times in the time sequence in which they originally occurred.

Notes:

1. Application Times (dGazeTimeSec and dAppMarkTimeSec) are expressed in seconds and are computed from the Eyegaze Edge's CPU Time Stamp Counter (TSC).

2. While the Eyegaze computer clock has good *resolution* for interpolating short times between events, it is not highly *accurate* with respect to other clocks, so long time intervals should *not* be considered precise.

3. Application Time 0.0 is arbitrarily defined as the time that Eyegaze was initialized by EgInit().

4. The index of the first event timing mark received from the client via EgLogMark() is designated mark-count 1, not mark-count 0. The '0^{th}' mark time corresponds EgInit() time 0.0.

5. dAppMarkTimeSec does ***not*** account for the communications transit time it takes event-mark messages to get from the client to the Eyegaze server.

6. When a gazepoint sample contains a new mark count, it means that Eyegaze received an event-mark message during the Eyegaze sample period immediately preceding the image-capture time. As illustrated in the above example, the first event mark arrived between the times that the camera captured the images from Eyegaze samples 15 and 16.

7. If Eyegaze receives multiple event-mark messages within a single Eyegaze sample period, iAppMarkCount increments by the total number of marks received within that period, and only the time of the *last* mark received during that period is recorded in dAppMarkTimeSec.

8. If the application a) has Eyegaze writing gazepoint data to a disk file via EgLogStart(), and b) is also retrieving gazepoint data via EgGetData(), values of the *report* times for a given gazepoint are different for the logged and transmitted data. In data written to disk, the report time is the time that the disk-write command is executed; and in data sent to the client, the report time is the time that EgGetData() releases the data to the client.

9. dReportTimeSec does ***not*** account for the communications transit time it takes Eyegaze data to get from the server to the client.

4.11 Eyegaze Ring Buffer

Internally, the Eyegaze software maintains a ring-storage buffer of past gazepoint data samples. The key purpose of the buffer is to accommodate cases when the application program is temporarily unable to keep up with the real-time camera-sampling/image-processing rate, but where it is important that the application not to lose eyetracking data. For example, if the CPU at some point takes 100 ms between successive calls to EgGetData(), Eyegaze would produced100/16.67 = 6 gazepoint data samples in the interim. A ring buffer of 6 samples would allow the application to recover, albeit late, up to 6 past samples.

The application specifies the number of gazepoint samples in the ring buffer by setting the variable stEgControl.**iNDataSetsInRingBuffer** prior to calling the EgInit() function. The length of the ring buffer should be sized to accommodate the maximum time that the application might delay between successive calls to EgGetData().

4.12 Recovering Past Gazepoint Samples

For purposes of this discussion, the term "*unprocessed data samples*" refers to gazepoint data samples that Eyegaze has inserted into the ring buffer but that the application has not yet processed. When there is more than one unprocessed data sample, the application's data processing loop has "gotten behind" the Eyegaze data collection.

To allow the application to recover accumulated but unprocessed data points and to "catch up" to real time, the EgGetData() function has the following features (in addition to those discussed in Section 4.8 above):

1. If there are *any* unprocessed points in the ring buffer, EgGetData() *does not wait* for the next data sample from Eyegaze. It returns immediately, allowing the application's data-collection loop to catch up with the most recent data as quickly as the CPU will allow.

2. EgGetData() transfers the gazepoint data samples in original time order. When the application calls EgGetData(), the function copies the oldest, unprocessed data sample to the stEgControl.pstEgData structure.

The application may keep track of where its processing stands with respect to real-time by checking the following status variables:

1. stEgControl.**iNPointsAvailable** indicates the number of unprocessed data points currently available to the application in the ring buffer. If the number is 0, the data processing loop is fully caught up to real time. If the number is 1, one new data point is available, but the loop is less than one sample interval behind. If the number is greater than 1, the loop is behind real time.

2. stEgControl.**iNBufferOverflow** indicates whether the ring buffer has overflowed and how many gazepoint samples have been irretrievably lost since the last call to EgGetData(). If the ring buffer overflows, Eyegaze begins overwriting the oldest unprocessed samples. Though these overwritten points are irretrievably lost, Eyegaze keeps track of and reports the number of lost points. (Note: the application should check for buffer overflow *immediately prior* to calling EgGetData(), because EgGetData() resets the overflow count.)

NOTE: Long ring buffers may be used if it is desired to retrieve all the gazepoint data at the end of a data collection session. The memory requirements are approximately 10 Megabytes per hour. (48 bytes/Eyegaze-sample x 216K samples/hour ~= 10.3 MB/hr of data collection).

4.13 Collecting Eyegaze Data With Other *A*synchronous Data

In some eyetracking applications it is desired to collect data in real time from multiple, asynchronous data streams. To collect asynchronous data in proper time order, the data-collection loop typically cycles through all the input streams at high speed polling for new data as it becomes available.

In the case of Eyegaze, the EgGetData() function "waits" if new gazepoint data is not yet available (see Section 4.8) The waiting allows the application's data collection loop to synchronize its operation to the Eyegaze sampling rate. When collecting data from multiple asynchronous data streams, however, it is generally desired to avoid this waiting so that the program is free to poll for and collect data from the other input streams. To avoid waiting, the application should check that **iNPointsAvailable** is greater than 0 before calling EgGetData().

The following loop demonstrates the collection of data from multiple asynchronous data streams. Note that EgGetData() is called only when new gazepoint data is available, so the function always returns without delay.

```
/* High-speed data polling/collection loop:                              */
   for (EVER)
   {
/*    Get and process gazepoint data if available.                       */
/*    As long as unprocessed gazepoint data is currently available,      */
      while (stEgControl.iNPointsAvailable > 0)
      {
/*       Get the next gazepoint.                                         */
         EgGetData(&stEgControl);
/*       Process the eyegaze data.                                       */
      }

/*    Get and process other data if  available.                          */
      -------
   }
```

4.14 Eyegaze Communication Types

As discussed in Section 3, there are two fundamentally different ways to use Eyegaze with your application:

1) Single Computer Configuration: You can write your program to run locally on the Eyegaze computer system (the system with the frame grabber hardware and where the image processing software runs).

For most applications, the Single Computer Configuration is the simplest and most compact way to create an Eyegaze application. Your application program and the Eyegaze image processing software share (and compete with each other for) computer processor and memory resources. To operate in this mode, you set:

```
      stEgControl.iCommType = EG_COMM_TYPE_LOCAL;        // Eyegaze Single Computer
Configuration
```

before calling EgInit().

2) Double Computer Configuration: You can write your program to run as a client application on a second computer. It communicates with the Eyegaze Edge that acts as a server application (see Sections 3.3 and 3.4).

Running a two-PC configuration with an Eyegaze server and client Eyegaze application requires a form of communication between the two computers. Eyegaze supports communications over Ethernet or over an RS-232 serial connection. In this mode you set either:

```
      stEgControl.iCommType  =  EG_COMM_TYPE_SOCKET;        //  Eyegaze  Double  Computer
Configuration
                                                          // Ethernet communications link

      stEgControl.iCommType  =  EG_COMM_TYPE_SERIAL;        //  Eyegaze  Double  Computer
Configuration
                                                          // serial communications link
```

before calling EgInit().

Using Ethernet rather than RS-232 is both more robust and the higher-bandwidth way to communicate between the two computers. You can connect the machines back-to-back using an Ethernet crossover cable or via a standard networking hub. 10 Mbps and 100 Mbps networks work equally well in this application. When using Ethernet to communicate between your client application and the Eyegaze server application, TCP/IP is used. When using an Ethernet connection, the application must also specify the Eyegaze server IP address:

```
      stEgControl.pszCommName = "127.0.0.1";          // Specify server IP address
```

Using an RS-232 serial connection is very simple and does not require the use of networking cards in the two computers. Most computers already have RS-232 serial ports however they are often misconfigured or already in use for a computer mouse or modem. Still, this can be an effective way to communicate between the two computers. When using a serial connection, the application must also specify the Eyegaze com port:

```
      stEgControl.pszCommName = "COM1";               // Specify com port
```

Connecting using an RS-232 serial connection requires the use of a null modem cable. We have found that it is useful to put an RS-232 tester on the line that can display the current status of important RS-232 signals. This can be particularly helpful when trying to diagnose communications problems when using serial communications.

Note: All other EgWin function calls remain the same, independent of iCommType. Other than the settings for stEgControl.iCommType and stEgControl.pszCommName, no other changes to the code are required to use the single-PC approach or one of the two 2-PC approaches.

4.15 Starting/Stopping Eyegaze Image Processing

Once initialized, Eyegaze begins eyetracking operations (i.e. it begins processing camera images and filling in the Eyegaze data ring buffer) when the application program sets **stEgControl.bTrackingActive** to TRUE. Similarly, eyetracking operations cease when the application program resets the flag to FALSE.

```
      stEgControl.bTrackingActive = TRUE;          // start Eyegaze image processing
      stEgControl.bTrackingActive = FALSE;         // stop Eyegaze image processing
```

4.16 Displaying Eye Images

The EgWin API provides two functions to allow an application to display images of the users eye on the screen: EgEyeImageInit() and EgEyeImageDisplay().

The Eyegaze software can display the full 640x480 pixel camera image of the eye on the VGA monitor screen. The application program turns the camera image display on and off by changing the **stEgControl.bEgCameraDisplayActive** flag.

```
stEgControl.bEgCameraDisplayActive = TRUE;   // start large camera image display
stEgControl.bEgCameraDisplayActive = FALSE;  // stop large camera image display
```

4.17 Graphics Support for Calibration

With the exception of EgCalibrate(), none of the EgWin functions perform any graphics operations. Since the calibration procedure writes to the screen, however, it requires graphics support.

Several of the LC Technologies Eyegaze demonstration programs (i.e. EgServer, EgClientDemo and Trace) use a fairly generic LCT graphics approach to support their program displays, and the EgCalibrate() function utilizes this graphics support. If your application program uses this same LCT graphics support code, it may simply call the EgCalibrate() function when a calibration procedure is desired. A brief discussion of the LCT graphics approach is given in Appendix I.

If your program employs an alternative graphics approach, however, the EgCalibrate() function will typically not work. To permit Eyegaze application programs to use alternative graphics approaches, the program may spawn an external calibration program called **Calibrate.exe**, which sets up its own graphics, independent of the parent program's graphics.

NOTE: The Calibrate.exe program requires the use of the frame grabber resources, so if the application program has allocated the frame grabber via the EgInit() call, it must release the frame grabber, via a call to EgExit(), prior to spawning Calibrate.exe. After returning from calibration, the application must recreate the Eyegaze thread with a call to EgInit().

Code snippet for spawning Calibration program:

```
/* if Eyegaze is running,                                          */
   if (...)
   {
/*    Release frame grabber resources for calibration program.    */
      EgExit(&stEgControl);
   }

/* Spawn the Eyegaze calibration program.                          */
   spawnl(P_WAIT, "calibrate.exe", "CALIBRATE.EXE","runtime", NULL);

/* Start Eyegaze.                                                  */
   EgInit(&stEgControl);
```

5 EGWIN FUNCTION REFERENCE

5.1 Summary

Because there are a small number of EgWin functions, they are listed here in logical order of use rather than in alphabetical order:

The four basic eyetracking functions are:

EgInit()	initialize Eyegaze server	Section 5.2
EgCalibrate()	calibrate a user/subject	Section 5.3
EgGetData()	retrieve the next gazepoint data sample	Section 5.4
EgExit()	close the Eyegaze server	Section 5.5

For purposes of tracking application event times with respect to the gazepoint sample times, the EgLogMark() function allows an application to send an event timing mark to the Eyegaze Edge. (Also see Section 4.10, "Coordinating Client Application Events with Eyegaze Sample Times.")

EgLogMark()	send an application event mark to Eyegaze	Section 5.6

For eyetracking applications that do not require gazepoint input during run time, EgWin includes the following functions to handle the recording of gazepoint data to a log file on the Eyegaze computer. The logging functions also record application event marks signaled by EgLogMark().

EgLogFileOpen()	open a log file for recording gazetrace data	Section 5.7
EgLogAppendText()	write some text to the log file	Section 5.8
EgLogWriteColumnHeader()	write column headers to the log file	Section 5.9
EgLogStart ()	start writing Eyegaze data to the log file	Section 5.10
EgLogStop ()	stop writing Eyegaze data to the log file	Section 5.11
EgLogFileClose()	close the log file	Section 5.12

5.2 EgInit()

Description: Initialize the Eyegaze Edge image processing software.

```
#include <EgWin.h>

int EgInit(struct _stEgControl *pstEgControl);
```

stEgControl Control and status variables used to setup and control the Eyegaze image processing software. (See Section 4.4.)

Remarks: EgInit() creates and starts the Eyegaze thread. It also allocates the system resources associated with the frame grabber, and starts various subthreads which are transparent to the application.

In initializing the image processing software, EgInit() reads the calibration data from the existing pres_cal.dat file, which was generated the last time the calibration procedure was performed. Thus Eyegaze always uses the most recent calibration data.

EgInit() must be called before any other Eyegaze function. Prior to calling EgInit(), the application program must:

a) define an Eyegaze control structure, of type _stEgControl (see Section 4.4), and

b) set the values of several of the Eyegaze control parameters.

These operations are illustrated in the example code on the following pages.

If EgInit() is called when another instance of Eyegaze is already running, the second application terminates.

Return Value: EgInit() returns 0 in the Single Computer Configuration, i.e. when stEgControl.iCommType is set to EG_COMM_TYPE_LOCAL. In the Double Computer Configuration, i.e. when iCommType is set to EG_COMM_TYPE_SOCKET or EG_COMM_TYPE_SERIAL, the return value is 0 or positive on successful connection and negative on failure.

Example code for EgInit(): see next page

Example code for EgInit():

```
      #define EG_BUFFER_LEN 60         /* The constant EG_BUFFER_LEN sets the     */
                                       /*   number of past samples stored in      */
                                       /*   its gazepoint data ring buffer.       */
                                       /*   Assuming an Eyegaze sample rate of    */
                                       /*   60 Hz, the value 60 means that a      */
                                        /*   maximum of one                        */
                                       /*   second's worth of past Eyegaze data   */
                                       /*   is available in the buffer.           */
                                       /*   The application can get up to 60      */
                                       /*   samples behind the Eyegaze image      */
                                       /*   processing without losing eyetracking */
                                       /*   data.                                 */
      static struct _stEgControl stEgControl;
                                       /* The eyetracking application must define */
                                       /*   (and fill in) this Eyegaze control    */
                                       /*   structure                             */
                                       /*   (See structure template in EgWin.h)   */
      /* Set the input control constants in stEgControl required for starting     */
      /* the Eyegaze thread.                                                      */
         stEgControl.iNDataSetsInRingBuffer = EG_BUFFER_LEN;
                                       /* Tell Eyegaze the length of the Eyegaze  */
                                       /*   data ring buffer                      */
         stEgControl.bTrackingActive = FALSE;
                                       /* Tell Eyegaze not to begin image         */
                                       /*   processing yet (so no past gazepoint  */
                                       /*   data samples will have accumulated    */
                                       /*   in the ring buffer when the tracking  */
                                       /*   loop begins).                         */
         stEgControl.iScreenWidthPix  = iScreenWidthPix;
         stEgControl.iScreenHeightPix = iScreenHeightPix;
                                       /* Tell the image processing software what */
                                       /*   the physical screen dimensions are    */
                                       /*   in pixels.                            */
         stEgControl.bEgCameraDisplayActive = FALSE;
                                       /* Tell Eyegaze not to display the full    */
                                       /*   640x480 camera image in a separate    */
                                       /*   window.                               */
         stEgControl.iEyeImagesScreenPos = 1;
                                       /* Tell Eyegaze that the location for the  */
                                       /*   eye image display is the upper right  */
                                       /*   corner                                */
                                       /*   1 -- upper right corner               */
                                       /*   2 -- upper left corner                */
         stEgControl.iVisionSelect=0;  /* Set this reserved variable to 0         */

      /* The communications type may be set to one of three values.  Please see   */
      /* the documentation regarding the different values for communication type. */
      //  stEgControl.iCommType  =  EG_COMM_TYPE_LOCAL;     //  Eyegaze  Single  Computer
Configuration
      //  stEgControl.iCommType  =  EG_COMM_TYPE_SERIAL;    //  Eyegaze  Double  Computer
Configuration
         stEgControl.iCommType  =  EG_COMM_TYPE_SOCKET;    //  Eyegaze  Double  Computer
Configuration

      /* If the comm type is socket or serial, set one of the following:          */
      // stEgControl.pszCommName = "COM1";        // Eyegaze comm port
      //                                          //  for EG_COMM_TYPE_SERIAL
         stEgControl.pszCommName = "127.0.0.1";   // Eyegaze server IP address
                                                  //  for EG_COMM_TYPE_SOCKET

      /* Create the Eyegaze image processing thread                               */
         EgInit(&stEgControl);
```

5.3 EgCalibrate()

Description: Calibrate a user on the Eyegaze Edge

#include <EgWin.h>

void EgCalibrate(struct _stEgControl *pstEgControl, HWND hwnd, int iCalAppType);

stEgControl	Control and status variables used to setup and control the Eyegaze image processing software. (See Section 4.4.)
hwnd	window handle for the application program window
iCalAppType	EG_CALIBRATE_DISABILITY_APP 0 EG_CALIBRATE_NONDISABILITY_APP 1

Remarks: EgCalibrate() is used to calibrate a user/subject on the Eyegaze Edge. The function has one argument, the application program's window handle.

EgCalibrate() calibrates within the same window that the application program has open, thus is most accurate within that region of the full screen. In other regions of the screen, the gazepoint calculation is an extrapolation outside the calibration region and may have reduced accuracy. If the application window is moved subsequent to performing a calibration, gazepoint measurements outside the original calibration region are extrapolated.

If the user completes the calibration procedure before exiting, EgCalibrate():
a) updates the calibration data used by the Eyegaze Edge's gazepoint prediction algorithms, and
b) generates a file called **pres_cal.dat** that contains these most recent calibration results.

Unless you are writing applications for people with disabilities, the **iCalType** argument should always be set to EG_CALIBRATE_NONDISABILITY_APP.

Graphics Support: If the application program implements the Eyegaze calibration procedure by calling the EgCalibrate() function rather than by spawning the Calibrate.exe program, the *application program must use the LCT graphics support code* discussed in Appendix I. If the program uses other graphics support code, it must spawn the Calibrate.exe program rather than call the EgCalibrate() function. See "Graphics Support for Calibration", Section 4.17.

Section 4.0 of the "User's Guide" describes the Eyegaze calibration procedure in more detail.

Return Value: None.

Example:

```
/* Run the Eyegaze calibration procedure. */
   EgCalibrate(&stEgControl, hwnd, EG_CALIBRATE_NONDISABILITY_APP);
```

5.4 EgGetData()

Description: This function retrieves the next available gazepoint data sample and places it in the stEgControl.pstEgData structure. If a new gazepoint sample is not yet available, the function waits until Eyegaze collects the next sample before returning with that sample.

```
#include <EgWin.h>

int EgGetData(struct _stEgControl *pstEgControl);
```

stEgControl — Control and status variables used to setup and control the Eyegaze image processing software. (See Section 4.4.)

Remarks: The Eyegaze software operates as an independent, interrupt-driven thread that places gazepoint data samples into an internal ring storage buffer (see Section 4.7 for a discussion of the Eyegaze thread operation). Each successive call to EgGetData() transfers the most recent gazepoint data sample from the ring buffer to the stEgControl.pstEgData structure. (See Section 4.11 for a discussion of the Eyegaze ring buffer.) EgGetData() does not do any image processing – it simply copies a gazepoint data sample from the Eyegaze ring buffer to the stEgControl.pstEgData structure.

EgGetData() always transfers gazepoint samples in original time order, i.e. first-in-first-out. When multiple unprocessed samples are available, EgGetData() copies the oldest of the unprocessed samples to the stEgControl.pstEgData structure.

If *no* gazepoint data is currently available in the gazepoint ring buffer when EgGetData() is called (i.e. if the application is fully caught up with the gazepoint data collection), EgGetData() "*waits*" until the next gazepoint sample is available from the Eyegaze thread. The application program may use this waiting feature to synchronize its data collection and processing operations with the camera's video field rate.

The "waiting" operation is implemented in the Eyegaze software by blocking on a semaphore, so processor time is not wasted during the wait period. See Section 4.9

If *any* unprocessed gazepoint data is currently available in the ring buffer when EgGetData() is called (i.e. if the application is not fully caught up with the gazepoint data collection), EgGetData() *returns immediately* (with the next gazepoint sample loaded in the application's copy of seEgData). The immediate return is intended to give the application time to catch up with the real time gazepoint data steam.

Gazepoint data samples are skipped and irretrievably lost, if the ring butter overflows (see Section 4.12). The application may check if and how many data points have been lost by checking the value of stEgControl.iNBufferOverflow, i.e. the number of unprocessed gazepoint samples that the Eyegaze thread has overwritten since the application last called EgGetData(). The application must perform the overflow test immediately prior to calling EgGetData(), because EgGetData() resets the overflow count.

Return Value: In dual camera systems, the index of the vision-system that returned the most recent data:
0 – Right camera
1 – Left camera

Examples: See the following for 'simple' and 'complex' examples of EgGetData() used to *synchronize* the application's operation with the Eyegaze sampling rate.

See Section 4.13 for an example of collecting Eyegaze data while also collecting other data streams that are *asynchronous* with the Eyegaze camera.

Example code for EgGetData() -- *SIMPLE* Eyegaze processing loop – where the application operation is *synchronized* to the Eyegaze sampling rate:

The following simple Eyegaze processing loop may be used in cases where it is acceptable to lose eyetracking data if the loop gets behind the image processing. This simple code often suffices for a basic, minimum Eyegaze application that does not require that every measured gazepoint be processed -- such as an application that needs only the most recent gazepoint sample.

```
#define EG_BUFFER_LEN   1          // With a buffer length of 1, the Eyegaze
                                   //   ring buffer contains only the most
                                   //   recent eye image data so EgGetData()
                                   //   only returns the most recent sample.

int  iWindowHorzOffset;            /* offsets of the application's client   */
int  iWindowVertOffset;            /*   window within the full screen       */
                                   /* NOTE: if the application window is    */
                                   /*   moved during the program operation, */
                                   /*   these offsets must be modified      */
                                   /*   accordingly. (See Appendix I)       */

int  iXGazeWindowPix;              /* gazepoint coordinates within the      */
int  iYGazeWindowPix;              /*   application window                  */

   stEgControl.iNDataSetsInRingBuffer = EG_BUFFER_LEN;

/* Loop for ever until the program is terminated.                           */
   for (EVER)
   {
/*    Wait for Eyegaze to generate the next gazepoint sample, and           */
/*    retrieve it.                                                          */
      EgGetData(&stEgControl);

/*    Process the next Eyegaze data sample - contained in pstEgData.        */
/*    Convert the gazepoint from full-screen to client window coordinates.  */
      iXGazeWindowPix = stEgControl.pstEgData->iIGaze - iWindowHorzOffset;
      iYGazeWindowPix = stEgControl.pstEgData->iJGaze - iWindowVertOffset;
   }
```

Example code for EgGetData() -- *COMPLEX* Eyegaze processing loop – where the application operation is *synchronized* to the Eyegaze sampling rate:

The following, more complex code should be used in applications where it is required to process all eyetracking data:

```
#define EG_BUFFER_LEN   120        // A ring buffer length of 120 allows the
                                   //   application program to get up to 2
                                   //   seconds behind the real-time eye
                                   //   image processing without losing data.

int  iWindowHorzOffset;            /* offsets of the application's client   */
int  iWindowVertOffset;            /*   window within the full screen       */
                                   /* NOTE: if the application window is    */
                                   /*   moved during the program operation, */
                                   /*   these offsets must be modified      */
                                   /*   accordingly.  (See Appendix I)      */

int  iXGazeWindowPix;              /* gazepoint coordinates within the      */
int  iYGazeWindowPix;              /*   application window                  */

   stEgControl.iNDataSetsInRingBuffer = EG_BUFFER_LEN;

/* Loop for ever until the program is terminated.                           */
   for (EVER)
   {
/*    This code keeps the loop synchronized with the real-time Eyegaze      */
/*    image processing, but insures that all gazepoint data samples are     */
/*    processed, even if the loop gets up to two seconds behind the         */
/*    real-time Eyegaze image processing.                                   */

/*    If the ring buffer has overflowed,                                    */
      if (stEgControl.iNBufferOverflow > 0)
      {
/*       iNBubberOverflow gazepoint samples have been irretrievably lost.   */
/*       The application program acts on this lost data if necessary.       */
         (appropriate application code)
      }

/*    Get the next gazepoint sample.  If an unprocessed Eyegaze data sample */
/*    is still available, EgGetData() returns immediately, allowing the     */
/*    application to catch up with the Eyegaze image processing.  If the    */
/*    next unprocessed sample has not yet arrived, EgGetData blocks until   */
/*    data is available and then returns.  This call effectively puts the   */
/*    application to sleep until new Eyegaze data is available to be        */
/*    processed.                                                            */
      EgGetData(&stEgControl);

/*    Process the next gazepoint data sample – contained in pstEgData.      */
/*    Convert the gazepoint from full-screen to client window coordinates.  */
      iXGazeWindowPix = stEgControl.pstEgData->iIGaze - iWindowHorzOffset;
      iYGazeWindowPix = stEgControl.pstEgData->iJGaze - iWindowVertOffset;
   }
```

5.5 EgExit()

Description: Shut down Eyegaze operation, terminate eyetracking thread, and release the system resources associated with frame grabber.

```
#include <EgWin.h>

int EgExit(struct _stEgControl *pstEgControl);
```

stEgControl — Control and status variables used to setup and control the Eyegaze image processing software. (See Section 4.4.)

Remarks: If the application program initiates the Eyegaze Calibration procedure by spawning the Calibrate.exe program rather than by calling the EgCalibrate() function, and if it is using the frame-grabber resources prior to the spawn, the program must call EgExit() to release the frame grabber for use by the Calibrate.exe program.

Return Value: 0

Example:
```
EgExit(&stEgControl);
```

5.6 EgLogMark()

Description: Send an application event timing mark to Eyegaze.

```
#include <EgWin.h>

unsigned int EgLogMark(struct _stEgControl *pstEgControl);
```

stEgControl — Control and status variables used to setup and control the Eyegaze image processing software. (See Section 4.4.)

Remarks: EgLogMark() allows the client application to record the times of application events within the Eyegaze data sample stream. See Section 4.10, "Coordinating Client Application Events with Eyegaze Sample Times."

EgLogMark() increments the Eyegaze server's internal 'application event mark count' and records the corresponding 'event mark time.'

'Event mark count' is initialized to zero by EgInit(). 'Event mark count' is incremented by one each time EgLogMark() is called.

Application event mark times and counts are recorded in the Eyegaze data structures returned by EgGetData() and in log files opened by EgLogFileOpen().

Return Value: The current application event mark count.

Example:
```
EgLogMark(&stEgControl);
```

5.7 EgLogFileOpen()

Description: Open a log file for recording gazetrace data on the Eyegaze Edge computer.

```
#include <EgWin.h>

int  EgLogFileOpen(struct _stEgControl *pstEgControl, char *pszFileName, char
                   *pszMode);
```

stEgControl — Control and status variables used to setup and control the Eyegaze image processing software. (See Section 4.4.)

pszFileName — The name of the file to open / create

pszMode — The file-open mode – maps to the mode argument of the fopen function. Use either 'a' to append to an existing file or 'w' to overwrite any existing file when the file is opened.

Remarks: EgLogFileOpen() opens a gazepoint data log file on the Eyegaze Edge computer. The file is used to record gazepoint history and other data produced by the Eyegaze image processing software.

This file logging function is intended primarily for applications that do not require the receipt of gazepoint input during run time, and where it is desired to have Eyegaze handle the eyetracking data collection process.

EgLogFileOpen() initializes 'event mark count' to zero. 'Event mark count' and 'event mark time' are internal Eyegaze variables used to synchronize events within the client application program to the gazepoint data collection. See Section ___. The application calls EgLogMark() (see Section 5.6) to send timing mark events to Eyegaze.

The arguments (after &stEgControl) are analogous to C's fopen() function.

Return Value: 0 for success, negative value for failure to open file.

Example:
```
EgLogFileOpen(&stEgControl, "Trace.dat", "a");
```

5.8 EgLogAppendText()

Description: Append text into the current log file.

#include <EgWin.h>

void EgLogAppendText(struct _stEgControl *pstEgControl, char *pszText);

stEgControl — Control and status variables used to setup and control the Eyegaze image processing software. (See Section 4.4.)

pszText — ASCII string containing text to append to the log file.

Remarks: EgLogAppendText() is used to append text to the current log file opened by EgLogFileOpen(). The text will be inserted into the data stream and additional log entries will appear after it as they are collected. This function can also be used to insert header information into the log file if called after EgLogFileOpen() but before EgLogStart().

Return Value: None.

Examples:

```
EgLogAppendText(&stEgControl, "This data file was created using live displays");
EgLogAppendText(&stEgControl, "Application event number 5 occurred");
```

5.9 EgLogWriteColumnHeader()

Description: Write title headers for the data columns in the log file.

#include <EgWin.h>

void EgLogWriteColumnHeader(struct _stEgControl *pstEgControl);

stEgControl — Control and status variables used to setup and control the Eyegaze image processing software. (See Section 4.4.)

Remarks: EgLogWriteColumnHeader() appends column headers to the log file opened by EgLogFileOpen(). As illustrated in the example below, the column headers occupy 3 lines in the log file.

Return Value: None.

Example:

```
EgLogWriteColumnHeader(&stEgControl);
```

Header lines created by EgLogWriteColumnHeader():

```
samp   Eye       Gazepoint     Pupil   Eyeball-Position   Focus   Mark    Last          Gaze
Comp
indx Found     X       Y     Diam      X      Y      Z    Range   Cnt   Mark Time      Time
Time
     (t/f)   (pix)   (pix)   (mm)    (mm)   (mm)   (mm)    (mm)   (-)    (sec)        (sec)
(sec)
```

5.10 EgLogStart()

Description: Start recording Eyegaze data to the currently-open log file.

```
#include <EgWin.h>

void EgLogStart(struct _stEgControl *pstEgControl);
```

stEgControl — Control and status variables used to setup and control the Eyegaze image processing software. (See Section 4.4.)

Remarks: EgLogStart() causes Eyegaze to start recording gazepoint data to the log file opened by EgLogFileOpen(). Eyegaze continues to write gazepoint data to the log file until EgLogStop() is called.

Headers for the columns can be written into the log file using EgLogWriteColumnHeader().

Return Value: None.

Example:
```
EgLogStart(&stEgControl);
```

DATA EXAMPLE:

```
samp   Eye       Gazepoint     Pupil   Eyeball-Position   Focus   Mark    Last          Gaze
Comp
indx Found     X       Y      Diam      X      Y      Z    Range   Cnt   Mark Time      Time
Time
     (t/f)   (pix)   (pix)   (mm)    (mm)   (mm)   (mm)    (mm)   (-)     (sec)        (sec)
(sec)
 14      1     212     120    3.85     8.1   -2.7    2.7    698.0   0    00000.0000   00000.2298
00000.2598
 15      1     204     138    3.80     8.4   -2.7    3.2    698.0   0    00000.0000   00000.2555
00000.2855
 16      1      45     249    3.94    11.1   -2.1    3.2    698.0   1    00000.2604   00000.2722
00000.3022
 17      0       0       0    0.00     0.0    0.0    0.0    698.0   1    00000.2604   00000.2887
00000.3187
 18      1      42     245    3.95    11.0   -2.1    2.8    698.0   1    00000.2604   00000.3055
00000.3350
```

5.11 EgLogStop()

Description: Stop recording Eyegaze data to the currently-open log file.

```
#include <EgWin.h>

void EgLogStop(struct _stEgControl *pstEgControl);
```

stEgControl — Control and status variables used to setup and control the Eyegaze image processing software. (See Section 4.4.)

Remarks: EgLogStop() causes Eyegaze to stop recording gazepoint data to the log file opened by EgLogFileOpen(). EgLogStop() does *not* close the log file. Logging to the file may be resumed by calling EgLogStart().

Return Value: None.

Example:
```
EgLogStop(&stEgControl);
```

5.12 EgLogFileClose()

Description: Close the Eyegaze log file.

#include <EgWin.h>

void EgLogFileClose(struct _stEgControl *pstEgControl);

stEgControl — Control and status variables used to setup and control the Eyegaze image processing software. (See Section 4.4.)

Remarks: EgLogFileClose() closes the log file opened by EgLogFileOpen().

If data is currently being written to the file (i.e. if EgLogStart() has been called but EgLogStop() has not), the data collection is stopped before the file is closed.

Return Value: There is no return value.

Example:
```
EgLogFileClose(&stEgControl);
```

In order to use the eye image functions, the application must define the following structure:

```
struct _stEyeImageInfo stEyeImageInfo;
```

At initialization time, the application must call EgEyeImageInit():

```
EgEyeImageInit(&stEyeImageInfo, 4);
```

Ordinarily an Eyegaze application will call the function to display the eye images immediately after the call to EgGetData():

```
EgEyeImageDisplay(0,
                  iHorizontalScreenPosition,
                  iVerticalScreenPosition
                  iWidth,
                  iHeight,
                  hdc);
```

5.13 EgEyeImageInit ()

Description: Initialize the Eyegaze Eye Image functions. This function must be called once before calling EgEyeImageDisplay().

#include <EgWin.h>

struct _stEyeImageInfo *EgEyeImageInit(struct _stEyeImageInfo *stEyeImageInfo, int iDivisor);

stEyeImageInfo Control and status variables used to setup and control the Eyegaze EyeImageDisplay function.

iDivisor This is the divisor used to shrink the eye image. Full field source images are 640 pixels across and 480 pixels high. Ordinarily a divisor of 4 is used to produce an eye image that is 160 pixels across and 120 pixels high. The closeup image will be sized the same as the full field image.

Remarks: This function allocates memory and sets up bitmaps for the eye image displays.

Return Value: A pointer to the stEyeImageInfo structure.

Example:
```
EgEyeImageInit(&stEyeImageInfo, 4);
```

5.14 `EgEyeImageDisplay` ()

Description: Display the eye images on the screen at the location specified at initialization time.

#include <EgWin.h>

void EgEyeImageDisplay(int iVis, int iX, int iY, int iWidth, int iHeight, HDC hdc);

iVis The vision system number, normally 0.

iX Horizontal screen position (pixels).

iY Vertical screen position (pixels).

iWidth Width of the target display (normally use stEyeImageInfo.iWidth

iHeight Height of the target display (normally use stEyeImageInfo.iHeight.

hdc The handle of the device context for displaying the image.

Remarks: If EgEyeImageInit has been called, the eye images are created automatically each time the gaze is tracked. They are not placed on the screen until EgEyeImageDisplay is called. Normally, this function is called immediately following a call to EgGetData().

Return Value: none.

Example:
```
EgEyeImageDisplay(0,
      iHorizontalScreenPosition,
      iVerticalScreenPosition
      iWidth,
      iHeight,
      hdc);
```

6 COMPILING, LINKING AND RUNNING EYEGAZE APPLICATION PROGRAMS

The Eyegaze Edge Analysis Systemuses Microsoft Visual C++ version 6 to compile applications. Usage of Visual C++ is discussed in the product documentation and other books on the subject, so only the most simple usage information is provided here.

Visual C++ uses a 'workspace' and projects within the workspace to define the programs that it generates. The Eyegaze Edge Analysis Systemships with a single workspace and four projects. The Eyegaze workspace is c:\eyegaze\eyegaze.dsw and includes four projects: GazeDemo, EgClientDemo, EgServer, and Trace. Each of these four projects is contained in the Eyegaze directory.

To get started, start Visual C++, open the Eyegaze workspace by selecting File, Open Workspace, then select c:\eyegaze\eyegaze.dsw. All four projects within that workspace will be visible in a panel on the left side of the display. The source code can be modified by double-clicking on a .c file and editing it in the built-in editor. When editing is complete, right-click on the project and select 'build'. A new executable will be created in either the Debug or Release subdirectory depending on the selection made under Active Configuration.

6.1 Single Computer Configuration

Eyetracking application programs that run directly on the Eyegaze computer must be linked with lctigaze.lib, and these programs require lctigaze.dll at runtime.

6.2 Double Computer Configuration

Client eyetracking applications which run on a second computer must be linked with EgClient.lib, and these client programs require EgClient.dll at runtime.

Note on program starting sequence: When running in the Double Computer configuration, it is necessary to start the EgServer program on the Eyegaze Edge before the client program attempts to connect to the Eyegaze Edge.

7 EXAMPLE EYEGAZE APPLICATION PROGRAMS

The use of the eyegaze functions in typical eyegaze application programs is demonstrated in the GazeDemo.c, EgClientDemo.c, and Trace.c programs summarized above. These programs contain all the source code necessary to access and execute the eyegaze functions, and they may be used as a templates for user-developed programs.

7.1 GazeDemo: Basic Eyetracking Program

The GazeDemo.c program demonstrates simple eyetracking operation. This program demonstrates the use of the Eyegaze functions in a simple example Windows program. After performing standard Windows setup functions, GazeDemo executes the Eyegaze Calibration procedure, which in this example is a separate program spawned from within GazeDemo. Upon completion of the calibration process, GazeDemo creates and initiates an eyetracking applications thread which continually tracks the user's gazepoint and moves the Windows mouse cursor to follow the user's gazepoint as he scans the desktop.

In its delivered form, GazeDemo.exe operates on the Eyegaze Edge in the Single Computer configuration. It may be configured to run as a client in the Double Computer configuration by editing GazeDemo.c to define NO_RUN_LOCALLY (rather than RUN_LOCALLY) at the beginning of the program and recompiling the program. When running in the Double Computer configuration, the program call must be followed by an argument with the IP address.

7.2 EgClientDemo: Client Computer Functions to Communicate with the Eyegaze Edge

EgClientDemo.c contains source code for all the functions required to communicate with the Eyegaze Edge. It contains functions to:

a) support the Eyegaze Edge in performing a calibration for a person looking at the client computer monitor, and

b) receive and process gazepoint data from the Eyegaze Edge in real time.

To demonstrate the operation of the Eyegaze communications functions, EgClientDemo is also a fully operational program. Programmers may use the EgClientDemo.c source code as a starting point for developing custom Eyegaze application programs, or they may transfer these functions into their existing applications programs.

7.3 Trace: Passive Eyetracking with Later Playback

The Trace.c program demonstrates the use of the Eyegaze Edge to track a person's gaze while he is performing a task and play back the trace of the gaze subsequent to the data collection. Trace runs only in the Single Computer configuration.

Trace first displays a scene on the computer monitor and tracks and stores the user's gazepoint for a given period of time while the user looks at the screen. The program then replays the user's gazepoint by superimposing the gazepoint trace on the original display. The purpose of this program is to demonstrate the real-time capture and storage of gazepoint data for later retrieval and analysis. The code may be used as a starting point for developers who wish to write their own eyegaze data collection and analysis programs.

In this example program, a brief menu displays the various data collection and analysis options. During the data collection phase, either a graphics scene or a paragraph of text may be displayed. As the user looks

at the screen, the Eyegaze Edge continually tracks the user's eye and stores the gazepoint data for up to 20 seconds. The data collection period can be terminated by pressing the Escape key on the keyboard.

During the analysis phase, the program re-displays the scene and plots the gazepoint data on the screen. The gazepoint plots include both a) a superposition of the gazepoint trace on the display scene and b) position-versus-time graphs of the gazepoint x and y components and of the pupil diameter. The entire trace may be displayed statically or the trace may be played back dynamically using a replay display which moves a "snake" along showing the last half-second's worth of gazepoint data. The snake traces may be played back at various speeds.

The Trace program produces an ascii data file, called trace.dat, that contains the gazepoint trace data. Figure 2 shows the format of the trace.dat file. The first 8 lines are header data, primarily for human reference. Then come the raw gazepoint data, with each line representing a raw gazepoint data sample. The end of the raw data is indicated by a blank line. The next 6 lines are header data for fixation list that follows. Then come the fixation data, with a line for each fixation.

```
                    Gazepoint Trace Data File, 18:13:56 09/16/1999
 Samp Type: bitmap
 Raw Gazepoint Data (60 Hz Sampling Rate):

Samp   Eye      Gazepoint      Pupil     Eyeball-Position        Focus  Fix
Indx   Found    X      Y       Diam      X      Y      Z         Range  Indx
       (t/f)   (pix)  (pix)   (mm)      (mm)   (mm)   (mm)      (mm)

  0      1      610    281     3.40      6.1    3.2    -1.6      698.0    0
  1      1      610    295     3.40      6.1    3.2    -2.5      698.0    0
  2      1      611    300     3.38      6.1    3.2    -2.7      698.0    0
  3      1      611    294     3.39      6.1    3.2    -2.7      698.0    0
  4      1      612    299     3.39      6.1    3.2    -3.4      698.0    0
  5      1      618    301     3.40      6.1    3.2    -2.8      698.0    0
  6      1      621    303     3.39      6.1    3.2    -3.5      698.0    0
  7      1      621    301     3.40      6.1    3.2    -2.8      698.0    0
  8      1      619    298     3.39      6.1    3.2    -3.1      698.0    0
  9      1      617    295     3.40      6.1    3.2    -2.6      698.0    0
 10      1      617    295     3.39      6.0    3.2    -2.7      698.0    0
 11      1      613    305     3.41      6.0    3.2    -3.4      698.0    0
 12      1      614    303     3.41      6.0    3.2    -2.7      698.0    0
 13      1      618    302     3.39      6.0    3.3    -3.6      698.0    0
 14      0      0      0       0.00      0.0    0.0    0.0       698.0   -1 }
 15      1      665    301     3.42      6.0    3.3    -3.6      698.0   -1 } gazepoints between fixations
 16      1      671    297     3.40      6.1    3.3    -3.0      698.0    1
 17      1      670    299     3.41      6.1    3.3    -3.2      698.0    1

                    (additional raw gazepoint samples)

794      1      763    396     2.85      6.5    2.4    -1.1      698.0   36
795      1      764    405     2.82      6.5    2.4    -2.3      698.0   36
796      1      751    441     2.82      6.5    2.4    -2.8      698.0   37
797      1      745    447     2.84      6.5    2.3    -2.5      698.0   37
798      1      743    445     2.84      6.5    2.3    -3.1      698.0   37
799      1      742    436     2.83      6.5    2.3    -2.6      698.0   37
800      1      741    438     2.87      6.5    2.3    -3.1      698.0   37
801      1      743    443     2.86      6.5    2.3    -2.6      698.0   37
802      1      743    437     2.85      6.4    2.3    -2.6      698.0   37
803      1      743    439     2.85      6.4    2.3    -2.5      698.0   37
804      1      742    445     2.86      6.4    2.3    -3.2      698.0   37
805      1      725    384     2.87      6.3    2.3    -2.8      698.0   37
806      1      692    270     2.90      6.3    2.4    -2.0      698.0   37
807      1      667    251     2.90      6.2    2.5    -2.3      698.0   37
808      1      662    253     2.89      6.2    2.4    -1.6      698.0   37
809      1      663    252     2.92      6.2    2.4    -1.8      698.0   37

Fixation Data: (60 Hz Sampling Rate)

Fix      Fixation      Size     Fix      Fix
Indx     X      Y      Dur      Dur      Start
         (pix)  (pix)  (cnt)    (cnt)    Samp

  0      615    298    0        14       0
  1      671    300    2        11       16
  2      604    329    1        33       28
  3      636    208    2        11       63
  4      612    162    1        16       75
  5      620    129    0        19       91
  6      679    253    2        15       112
  7      665    226    1        10       128
  8      672    277    1        16       139
  9      719    342    2        15       157
  .      .
  .      .             (additional fixations)
  .      .
 26      474    453    1        16       619
 27      500    417    1        13       636
 28      550    386    1        15       650
 29      526    261    1        15       666
 30      504    254    0        21       681
 31      570    299    1        16       703
 32      650    261    2        16       721
 33      685    234    1        6        738
 34      844    166    2        18       746
 35      879    193    1        17       765
 36      765    396    3        11       785
 37      744    441    0        14       796
```

TRACE.DAT: Gazepoint history file output from the Trace program

Figure 2: TRACE.DAT - Gazepoint history file output from the Trace program

8 FIXATION ANALYSIS FUNCTION REFERENCE

The fixation analysis functions convert a series of uniformly-sampled (raw) gazepoints into a series of variable-duration saccades and fixations.

C source file: fixfunc.c
Include file: fixfunc.h

The Fixation library consists of two functions. Typically, InitFixation() is called initially, and DetectFixation() is then called in a loop for each successive gazepoint sample:

```
InitFixation();
for (gazepoint sample sequence)
{
   EgGetData();      // collect the current gazepoint sample
   DetectFixation(); // returns fix state: MOVING, FIXATING, or FIXATION_COMPLETED
}
```

The final results of any given fixation are available when DetectFixaton() returns FIXATION_COMPLETED. As illustrated above, the Fixation functions may be used in real time to monitor the user's current fixation status.

8.1 InitFixation()

Description: Initialize the fixation functions.

```
#include <fixfunc.h>
void InitFixation(int iMinimumFixSamples)
```

iMinimumFixSamples	minimum number of gaze samples that can be considered a fixation Note: if the input value is less than 3, the function sets it to 3.

Remarks: This function clears any previous, present and new fixations, and it initializes DetectFixation()'s internal ring buffers of prior gazepoint data. InitFixation() should be called prior to a sequence of calls to DetectFixation().

8.2 DetectFixation()

Description: Detect fixations in a gazepoint stream.

```
int  DetectFixation ( int       bGazepointFound,
                      float   fXGaze,
                      float   fYGaze,
                      float   fGazeDeviationThreshold,
                      int     iMinimumFixSamples,
                      int    *pbGazepointFoundDelayed,
                      float *pfXGazeDelayed,
                      float *pfYGazeDelayed,
                      float *pfGazeDeviationDelayed,
                      float *pfXFixDelayed,
                      float *pfYFixDelayed,
                      int    *piSaccadeDurationDelayed,
                      int    *piFixDurationDelayed )
```

	INPUT PARAMETERS:
BYTE bGazepointFound	Flag indicating whether or not the image processing algorithm detected the eye and computed a valid gazepoint (TRUE/FALSE)
float fXGaze	present gazepoint
float fYGaze	(user specified units)
float fGazeDeviationThreshold	distance that a gazepoint may vary from the average fixation point and still be considered part of the fixation (user specified units)
int iMinimumFixSamples	minimum number of gaze samples that can be considered a fixation. Note: if the input value is less than 3, the function sets it to 3
	OUTPUT PARAMETERS:
	Delayed Gazepoint data with fixation annotations:
BYTE *pbGazepointFoundDelayed	sample gazepoint-found flag, iMinimumFixSamples ago
float *pfXGazeDelayed	sample gazepoint coordinates
float *pfYGazeDelayed	iMinimumFixSamples ago
float *pfGazeDeviationDelayed	deviation of the gaze from the present fixation, iMinimumFixSamples ago
	Fixation data - delayed:
float *pfXFixDelayed	fixation point as estimated
float *pfYFixDelayed	iMinimumFixSamples ago
int *piSaccadeDurationDelayed	duration of the saccade preceding the preset fixation (samples)
int *piFixDurationDelayed	duration of the present fixation (samples)

Return Value: Eye Motion State:

MOVING	0	The eye was in motion iMinimumFixSamples ago.
FIXATING	1	The eye was fixating iMinimumFixSamples ago.
FIXATION_COMPLETED	2	A completed fixation has just been detected. With respect to the sample that reports FIXATION_COMPLETED, the fixation started (iMinimumFixSamples + *piSaccadeDurationDelayed) ago and ended (iMinimumFixSamples) ago. Note, however, that because of the (approximately 2-field) measurement latency in the eyetracking measurement, the start and end times occurred (iMinimumFixSamples + *piSaccadeDurationDelayed + 2) and (iMinimumFixSamples + 2) ago with respect to real time now. See Section 10. Include FIXFUNC.H for function prototype and above constant definitions.

This function converts a series of uniformly-sampled (raw) gazepoints into a series of variable-duration saccades and fixations. Fixation analysis may be performed in real time or after the fact. To allow eye

fixation analysis during real-time eyegaze data collection, the function is designed to be called once per sample. When the eye is in motion, i.e. during saccades, the function returns 0 (MOVING). When the eye is still, i.e. during fixations, the function returns 1 (FIXATING). Upon the detected completion of a fixation, the function returns 2 (FIXATION_COMPLETED) and produces:

a) the time duration of the saccade between the last and present eye fixation (eyegaze samples)

b) the time duration of the present, just completed fixation (eyegaze samples)

c) the average x and y coordinates of the eye fixation (in user defined units of fXGaze and fYGaze)

Note: Although this function is intended to work in "real time", there is a delay of iMinimumFixSamples in the filter that detects the motion/fixation condition of the eye.

8.3 Principle of Operation

This function detects fixations by looking for sequences of gazepoint measurements that remain relatively constant. If a new gazepoint lies within a circular region around the running average of an on-going fixation, the fixation is extended to include the new gazepoint. (The radius of the acceptance circle is user specified by setting the value of the function argument fGazeDeviationThreshold.) To accommodate noisy eyegaze measurements, a gazepoint that exceeds the deviation threshold is included in an on-going fixation if the subsequent gazepoint returns to a position within the threshold. If a gazepoint is not found, during a blink for example, a fixation is extended if a) the next legitimate gazepoint measurement falls within the acceptance circle, and b) there are less than iMinimumFixSamples of successive missed gazepoints. Otherwise, the previous fixation is considered to end at the last good gazepoint measurement.

Typical parameter values are:

minimum fixation duration: 6 camera fields = 100 ms
radius of the fixation circle: 6.35 mm = 0.25 inch.

8.4 Units of Measure

The gaze position/direction may be expressed in any units (e.g. millimeters, pixels, or radians), but the filter threshold must be expressed in the same units.

8.5 Initializing the Function

Prior to analyzing a sequence of gazepoint data, the InitFixation() function should be called to clear any previous, present and new fixations and to initialize the ring buffers of prior gazepoint data.

8.6 Fixation Function Notes

For purposes of describing an ongoing sequence of fixations, fixations in this program are referred to as "previous", "present", and "new". The present fixation is the one that is going on right now, or, if a new fixation has just started, the present fixation is the one that just finished. The previous fixation is the one immediately preceding the present one, and a new fixation is the one immediately following the present one. Once the present fixation is declared to be completed, the present fixation becomes the previous one, the new fixation becomes the present one, and there is not yet a new fixation.

8.7 Example Code

See the Trace.c program for an example of how the Fixation functions are used.

9 EYEGAZE EDGE SAMPLING RATE

The Eyegaze software processes each camera field image separately. Thus the Eyegaze sampling rate equals the camera field rate, not the frame rate.

Typically, the cameras used in Eyegaze Edges operate at 60 Hz and use the U.S. RS-170 video format. Some cameras (made by Sanyo) may be synchronize their frame rate to the electrical power source, so the sampling rate for these cameras is 60 Hz in countries with 60 Hz power, and the sampling rate is 50 Hz in countries with 50 Hz power.

10 GAZEPOINT MEASUREMENT DELAY

For purposes of relating eye motions with events in the Eyegaze application program, it is important to realize that there is a finite delay, i.e. latency period, between the time that a subject's eye moves and the time that the Eyegaze Edge reports the results. The net delay is typically less than two sample intervals, 25 milliseconds on systems with 60 Hz cameras, 32 milliseconds on systems with 50 Hz cameras.

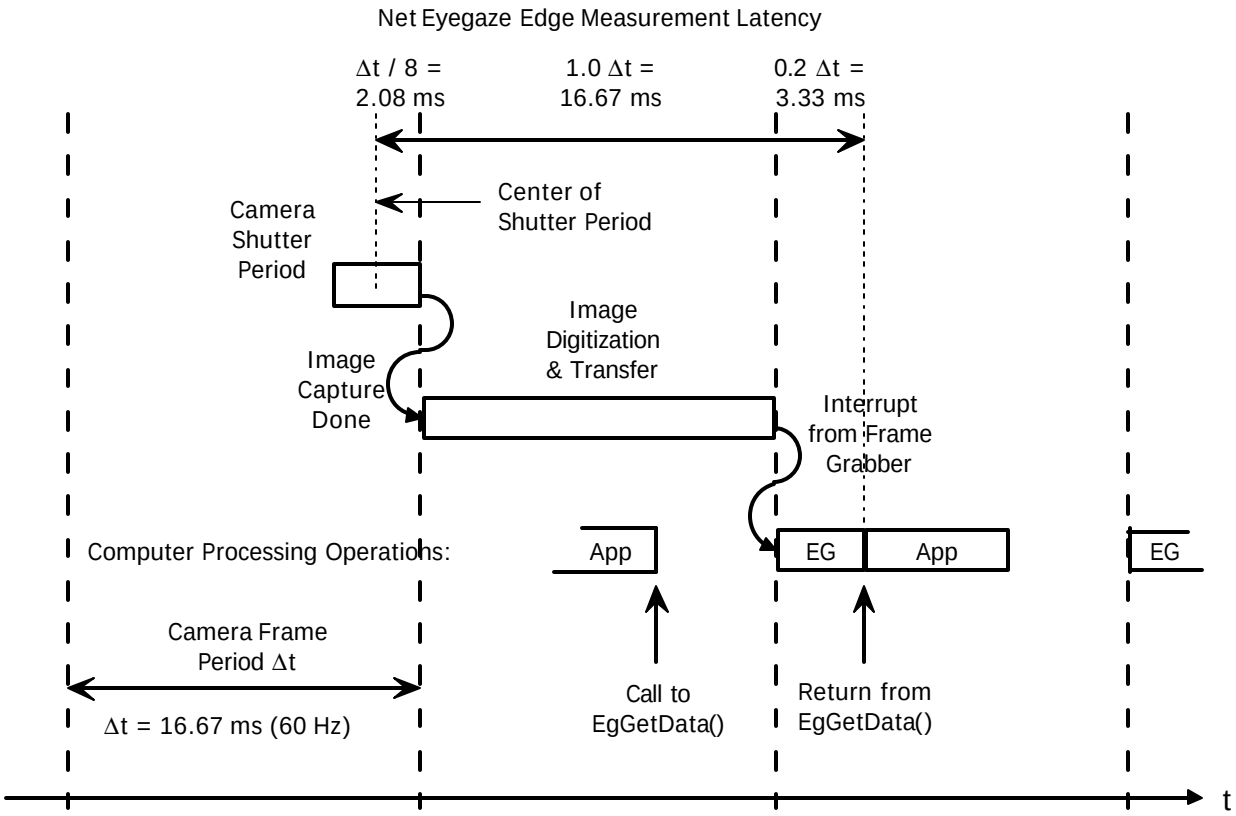
The net measurement delay results from the sequence of hardware and software operations involved in measuring the gazepoint:

a) the shutter period of the video camera (contributes 1/2 a sample interval),

b) the video transfer period from the camera to the frame-grabber (contributes a full sample interval), and

c) the Eyegaze image processing time (contributes approximately 1/10th a sample interval).

Figure 3 illustrates the timing of these operations. Figure 3 also illustrates the typical timing of Eyegaze and application code, as controlled by EgGetData() and the frame-grabber interrupt.



Δt = Camera frame Period = 16.67 ms (60 Hz)
Camera shutter period = Δt / 4 = 4.17 ms
Net Measurement Latency = 2.08 + 16.67 + 3.33 ms = 22.08 ms

EG = Eyegaze image processing
App = Application processes gazepoint sample

Figure 3: Gazepoint Measurement Operations

APPENDIX I: LCT GRAPHICS APPROACH

The following graphics support code is used in the LC-Technologies-supplied Eyegaze applications programs EgClientDemo.c, EgServer.c, and Trace.c. This code is sufficient to support the Eyegaze function EgCalibrate(). If your Eyegaze application program uses an alternative graphics approach, the calibration must be performed via a spawn to the Calibrate.exe program, rather than a call to the EgCalibrate() function. See "Graphics Support for Calibration", Section 4.17.

This code also illustrates how a program can intercept window-repositioning messages from the Windows operating system, to maintain current information about the application's window location. The window location data is important for converting gazepoint locations from full-screen to client window coordinates. (See Note 2 in "Eyegaze Output Data", Section 4.5.)

Under Windows, an application program has to be ready to redraw any portion of its display at any time. One simple way to accommodate this is to do all of your drawing in an off-screen bitmap and then bitblt from that bitmap to the display window whenever your display needs to be updated. This is the method that LC Technologies' programs use.

In the example programs (EgClientDemo, EgServer, and Trace), you will see quite a bit of code that exists to support this graphics approach. The code is sprinkled throughout the source file so it doesn't lend itself to being compartmentalized in C functions.

Function prototype for the lctSetWindowHandle function:

```
void lctSetWindowHandle(HWND hwnd, HDC memdc, HDC hdc, char *sz);
```

Variables used by the graphics support code:

```
int      iScreenWidthPix;          /* pixel dimensions of the full          */
int      iScreenHeightPix;         /*   computer screen                     */

RECT     stWindowRect;
int      iWindowWidthPix;
int      iWindowHeightPix;
int      iWindowHorzOffset;
int      iWindowVertOffset;
RECT     stEyegazeRect;
HDC      memdc;                 /* Stores the virtual device handle         */
HBITMAP  hbit;                  /* Stores the virtual bitmap                */
HBRUSH   hbrush;                /* Stores the brush handle                  */
HDC      hdc;


/*****************************************************************************/
```

In WinMain, set the two variables:

```
/* Obtain the current screen dimensions so Eyegaze knows the screen size.   */
   iScreenWidthPix  = GetSystemMetrics(SM_CXSCREEN);
   iScreenHeightPix = GetSystemMetrics(SM_CYSCREEN);
```

Maximize the window at creation time, or maximize before calling EgCalibrate:

```
/* Create the GazeDemo window.                                               */
   hwnd = CreateWindow (szAppName,            // window class name
                        szAppName,            // window caption
                        WS_OVERLAPPEDWINDOW,  // window style
                        0,                    // initial x position
                        0,                    // initial y position
```

```
                        iScreenWidthPix,      // initial x size
                        iScreenHeightPix,     // initial y size
                        NULL,                 // parent window handle
                        NULL,                 // window menu handle
                        hInstance,            // program instance handle
                        NULL);                // creation parameters
```

Inside WM_CREATE processing perform the following:

```
      case WM_CREATE:
/*       Determine the upper-left corner of the client area in screen coordinates.*/
         point.x = 0;
         point.y = 0;
         ClientToScreen(hwnd,&point);
         iWindowHorzOffset = point.x;
         iWindowVertOffset = point.y;

         GetClientRect(hwnd, &stWindowRect);
         iWindowWidthPix   = stWindowRect.right - stWindowRect.left+1;
         iWindowHeightPix  = stWindowRect.bottom - stWindowRect.top+1;

/*       Create the virtual window.                                          */
         hdc    = GetDC(hwnd);
         memdc  = CreateCompatibleDC(hdc);
         hbit   = CreateCompatibleBitmap(hdc, iScreenWidthPix, iScreenHeightPix);
         SelectObject(memdc, hbit);
         hbrush = GetStockObject(BLACK_BRUSH);
         SelectObject(memdc, hbrush);
         PatBlt(memdc, 0, 0, iScreenWidthPix, iScreenHeightPix, PATCOPY);

         lctSetWindowHandle(hwnd, memdc, hdc, szAppName);
```

At WM_PAINT time, bitblit the area of the Eyegaze bitmap to the screen:

```
      case WM_PAINT:
         BeginPaint(hwnd, &ps);
         BitBlt(hdc,
                ps.rcPaint.left, ps.rcPaint.top,
                ps.rcPaint.right  - ps.rcPaint.left,
                ps.rcPaint.bottom - ps.rcPaint.top,
                memdc,
                ps.rcPaint.left, ps.rcPaint.top,
                SRCCOPY);
         EndPaint(hwnd, &ps);
         break;
```

Catch WM_MOVE and WM_SIZE messages and adjust variables accordingly:

```
/*    If the user sizes or moves the window, obtain the new size and        */
/*    offsets from the screen origin.                                       */
      case WM_MOVE:
         point.x = 0;
         point.y = 0;
         ClientToScreen(hwnd,&point);
         iWindowHorzOffset = point.x;
         iWindowVertOffset = point.y;
       break;

      case WM_SIZE:
         iWindowWidthPix  = LOWORD(lParam);
         iWindowHeightPix = HIWORD(lParam);
         break;
```

At exit time, call EgExit():

```
      case WM_ENDSESSION:
      case WM_CLOSE:
/*       Call the EgExit function to shut down the vision subsystem.        */
         EgExit(&stEgControl);
```

```
PostMessage(hwnd, WM_DESTROY, (WORD)0, (LONG)0);
break;
```

LC TECHNOLOGIES EYEGAZE ANALYSIS SYSTEM
SECTION 4: PROGRAMMER'S MANUAL
TABLE OF CONTENTS

LIST OF FIGURES