Software for the control of experiments for investigation of low-vision reading from video displays

Robert A. Morris[1], Dean Yager[2], Kathy L. Aquilante[2]

Abstract. Rigorous object-oriented design and programming produced a robust, extensible, long-lived system for the control of real-time investigations of reading from video displays.

1 Introduction

This paper describes the architecture, engineering, use and evolution of a specialized system of laboratory control software for research in reading from video displays by subjects who are patients in the low-vision clinic at the SUNY College of Optometry. The National Eye Institute (NEI), a branch of the National Institutes of Health (NIH), funded the research, including much of the software development (RO1-11617 to DY and K23-00366 to KA).

Estimates indicate that more than 3 million Americans suffer from low-vision, which is defined as any chronic visual deficit that impairs everyday function and is not correctable by ordinary eyeglasses or contact lenses. Some investigators argue that these numbers under-estimate the prevalence of visual impairment and recommend a more inclusive definition of visual impairment bringing the number of visually impaired Americans up to 14 million (National Advisory Eye Council 1998). Based on fragmentary data that is believed to be an underestimate, the World Health Organization estimates the number of cases of blindness, defined as visual acuity of 20/400 or worse in the better seeing eye, to be greater than 15 million people world-wide. These figures do not include persons with visual impairment, and the number of blind and visually impaired is expected to increase due to increases in population and life expectancy (WHO Study Group (1973) 1973).

A wide variety of strategies and devices have been developed to aid low-vision readers, ranging from large print to computer controlled systems that magnify, alter presentation format, or use speech to improve accessibility of print (Goodrich and Bailey 2000) and there is a large scientific literature leading to their development and evaluating their efficacy. In our case, the team led by the authors has special interest in the typographic factors that might affect low-vision readers, and, as experimental controls, those with normal vision, as they use computer-based reading. Readability from a video display terminal (VDT) has been studied in both the reading research community and various segments of the Human Computer Interface (HCI) community (For a review, see (Muter 1996)). One recurring problem with these studies—especially as to typographic issues—is that the reading research community has sometimes suffered from a lack of typographic sophistication, and the HCI community is sometimes lacking in expertise about the subtleties of gathering data on reading performance. In the HCI community this has often led to claims based on out-of-date research and to confounding user preferences with user performance. In the reading community, it has often led to comparing typographic conditions that are varying in many more ways than the researchers imagined. Our interdisciplinary approach teams two vision scientists (Yager and Aquilante) who are specialists in low-vision reading, with a computer scientist (Morris), who has a long involvement in both digital typography, and in engineering complex software systems. We describe elsewhere the scientific results of our studies ((Aquilante, Yager, Morris and Khmelnitsky 2001); (Aquilante, Yager and Morris 2000); (Aquilante, Yager and Morris 1999) (Aquilante, Yager and Morris 1999) (Aquilante, Yager and Morris 1998)). Here we discuss only the kinds of problems we work on, the kind of data we gather, and the kind of data analyses we do, in order to leave the reader an understanding of the requirements of the software architecture.

Our software methodologies are now quite common: principally they comprise careful attention to the software life cycle and object-oriented design and programming (Jacobson 1992). The novelty arises in the

[1] Department of Computer Science, UMASS-Boston
[2] SUNY College of Optometry

object-oriented modeling of the control and analysis of experiments using human subjects. We believe that the ease with which we routinely added most enhancements desired by the vision scientists, as well as the difficulty we had with a few such requests, can inform others who might need to build systems for the control of experiments with human subjects.

2 Low-vision readers and RSVP.

The single biggest advantage that can be offered to most low-vision readers in any medium is to make the type bigger ((Dickinson 1998), (Rubin and Turano 1994), (Fine and Peli 1998), (Whittaker and Lovie-Kitchin 1993),). This is because, for a variety of reasons, almost all low-vision results in reduced visual acuity ((Leat, Legge and Bullimore 1999), (Dickinson 1998)). Besides acuity loss, low-vision readers also may suffer reduced contrast sensitivity ((Leat and Woodhouse 1993), (Rubin and Legge 1989), (Whittaker and Lovie-Kitchin 1993),) inability to control the eye movements necessary for traditional reading ((Bullimore and Bailey 1995), (Timberlake, Mainster, Peli, Augliere, Essock and Arend 1986), (Timberlake, Peli, Essock and Augliere 1987), (Whittaker, Cummings and Swieson 1991), (McMahon, Hansen and Viana 1991)). These and other factors have been much studied, but there is only a small literature carefully addressing them under paradigms that directly addresses the acuity issue ((Aquilante, Yager and Morris 1998),(Fine and Peli 1998), (Legge, Rubin, Pelli and Schleske 1985; Rubin and Turano 1994))(Whittaker and Lovie-Kitchin 1993). Two such paradigms that allow very large characters are scrolled text and Rapid Serial Visual Presentation (RSVP). In RSVP reading, words are flashed rapidly in the center of the screen, one after another. In normal reading of alphabetic languages such as English, as a substantial amount of the time spent in normal readings devoted to the eye-motion enterprise.. Typical reading by normal-vision adult readers consists of a fixation of about 250 ms on each word (depending slightly on the word length and/or its familiarity to the reader), followed by a jump, called a saccade, to the next word. This takes about 30-50 ms. (Rayner, Inhoff, Morrison, Slowiaczek and Bertera 1981). These numbers lead to reading rates of 240-300 words/minute, which is in fact typical for adult readers of English. However, a large fraction of the fixation time is devoted to the task of planning the next saccade. Indeed, the visual information for reading is available within the initial 50 msec of a fixation.(Rayner, Inhoff, Morrison, Slowiaczek and Bertera 1981). There is substantial controversy about the precise role of the saccade planning and how much of it can be done in parallel with the rest of the normal reading task ((Kennedy 2000), (Reichle, Pollatsek, Fisher and Rayner 1998), (Starr and Rayner 2001)), but it is clear that eliminating eye motion has the potential to increase reading speed and many studies find that normal-vision readers can increase their reading speed by a factor of 2-3 using RSVP. For low-vision readers the appeal of RSVP reading is clear: The words can be made much bigger—as big as will fit on the screen—and the subject need not move her eyes. This directly addresses two of the aforementioned impediments to reading by low-vision readers. Some patients in our studies, e.g. one with 20/400 vision[3], had no effective visual aid to reading except RSVP. Indeed, by use of video projection, we can make words as large as the projection surface and distance allows, and some of our experiments are carried out on a rear-projection screen with words approximately 5 inches (146 mm) high. Most of the research in our project has focused on RSVP reading. For this reason, we call our system UMB-SUNYOPT RSVP, or simply RSVP.

3 Engineering Requirements.

For software to support investigation of exactly what factors might be most useful to low-vision readers, a great many experimental controls must be supported. Many are obvious, such as typographic factors including the choice of typeface, character and word spacing, contrast, size and timing. Others come from standard scientific requirements to understand what results are really due to low-vision and what to normal reading, as well as more subtle requirements to reduce confounding of variables. In addition to their scientific importance, these factors must also be considered if the software is potentially to function not only to support data gathering and analysis, but also be useful as an instrument for clinical evaluation of vision deficits in order to formulate assistive strategies, and also to support the building of an actual reading aid. Both of these are long-term goals of our project.

Initially we identified two important classes of scientific controls that impose engineering requirements:

[3] A person with 20/400 vision must be at 20 feet to see what a normally-sighted person can see at 400 feet

Data must be gatherable from normal-vision readers, and reading paradigms other than RSVP must be easily chosen by the experimenters—particularly paradigms that are reported elsewhere in the literature, including those corresponding to more traditional forms of reading. Support for experiments with normal-vision readers might seem no different from support for low-vision subjects. However, normal-vision readers often read appreciably faster than low-vision readers ((Legge, Rubin, Pelli and Schleske 1985), (Rubin and Turano 1994, Aquilante, 2001 #348)). Some reports in the literature have claimed to measure RSVP reading rates of 1400-1800 wpm (23-30 words/sec) ((Rubin and Turano 1992), (Latham and Whitaker 1996)). Displays of a single word at this rate comprises only a few frame-times per word on standard video hardware and imposes performance requirements on the software that would not occur in presenting, say, a 10 word sentence for 500 ms.

As with most software systems, the designer must expect that functional requirements evolve with use. In our case, the anticipated evolution included migration to various computing platforms and O/S revisions, and the addition of experimental variables unspecified at the time of design. Another requirement that emerged early in iterations of our design documents was explicitly to model the roles of the experimenter as well as those of the subject. We will discuss these and other aspects of our architecture in the next sections.

4 Design.

Our design and analysis followed the Use Case methodology (Jacobson 1992). There are three principal actors: a subject, an experimenter, and the system. We will use upper case to refer to software objects modeling human actors and other facets of the enterprise.

A (human) experimenter decides the parameters of a reading experiment, which is modeled by an object called a *Run*. A Run consists of several *Trials*, each of which presents on the screen a sentence in ways decided upon by the experimenter and recorded in an initialization file. The subject reads the sentence aloud and the experimenter enters on the keyboard the number of errors made. If the subject has made more than the number of allowable mistakes indicated in the initialization file, then the software increases the length of time the next trial will take—i.e., decreases the reading rate. Otherwise that rate is increased. This so-called *staircase procedure* oscillates around a rate that is taken as the reading rate. The procedure stops when a predetermined number of oscillations have occurred. In our experiments this is taken to be four. The reading rate is taken to be the geometric mean of the turning points of the staircase. This common psychometric procedure is used to establish a maximum likelihood estimator of the true reading rate exhibited by the subject during the run. From run to run the experimenter may vary the conditions according to the variable being studied, e.g. font, character size, spacing, etc.

If we describe the system with the Model-View-Controller architecture (Borland 1994), we identify several software artifacts, of which those in the Model and Controller have the most explicit object representation. The View is quite simple and has no classes associated with it: there is a startup screen that allows the experimenter to initiate the experiment, including specifying the initialization file if not the default. There are (optionally) two screens supported, one from which the subject reads a sentence during the experiment and one on which the sentence is presented statically to the experimenter as an aid to scoring the reading (Otherwise, she has the sentences on paper). This description of the View is somewhat oversimplified. In fact, display is mediated by an API of our devising, the purpose of which is to support both modularity and portability. We discuss this at length later. We mention here in passing that a few of the facilities we implemented are available in the Standard Template Library (STL) (Silicon Graphics 2001) but when we began the project we were not satisfied that STL would be uniformly and correctly implemented on the platforms we targeted.

The Model consists of three classes, each with a single instance. These are the *Subject* and the *Experimenter* objects, and a *Staircase* object that holds the parameters of the staircase and accumulates data for analysis at the end of a run. Once a run is initiated the software is largely in control, and the methods of the Experimenter class play little role except to record the error count, and, optionally, to initiate the next trial if the experimental conditions do not require that this be automatic. Additionally the Experimenter may terminate the run before its normal end, because human subject protocols require that the subject be permitted to leave the experiment at any time during its course.

The Subject and Experimenter classes are in fact specialized from a pure virtual class called *User* whose principal methods represent various signatures of a method called *ShowTrialText*, which is devoted to the actual display. For the Experimenter, they consist of very little more than output to the standard I/O stream connected to the Experimenter's terminal if there is one. The Experimenter also has methods for input from the standard input stream in order to record the error count. For the Subject, the display methods have extreme performance and timing requirements required to insure validity of the data. This point is discussed below.

Several classes participate in the Controller. A *Setup* object is constructed from the initialization file when a run starts. It holds all the experimental parameters. A Run class, whose single constructor takes the Setup object as a parameter, controls the experiment in conjunction with the Staircase. The participation of the Staircase in both Model and View roles is one point of fragility in our implementation. The Staircase object has to be accessible to a number of different methods of the Run and its subservient Trial objects, and because the Staircase requires side-effects (namely the accumulation of data), enhancement engineering has to take special care at several points to maintain the contract of the Staircase. Although the code and methods were generally well documented, this particular fragility was not, and has required the first author—who is the sole system architect, original implementer, and supervisor of all of the other people who have ever modified the software—to be the principal implementer of modifications to the Staircase which have been required by enhancements to the data analysis.

The main roles of the Run object are to provide for the construction of a *Trial* object, call its Execute method, arrange for the recording of the timing and result of the Trial (i.e. the error count), negotiate with the Staircase object whether another Trial is warranted, and finally to set the timing parameters to be passed to the Trial for its next execution. As an implementation strategy, a single Trial object is reused by the Run, and in support of this, Trial objects do no dynamic memory management. In C++, careless dynamic memory management is the largest source of programming errors and memory leakage[4]. The Trial and its Execute method therefore directly know only how long its text presentations should take, what text it should present, and by reference, where the Subject and Experimenter objects may be found. This latter is because it is actually those objects which know how to display text on their respective screens under conditions that the Trial tells them. However, this isolation of the Trial from the rest of the system, while architecturally appealing, proved impractical at the implementation stage—representing inadequate modeling at the design stage that we were prepared to code around in a way that introduced the second major fragility of the system. Namely, we found that the Run object was in too much control of resources needed by a Trial, both for reading and updating. Consequently, we changed the definition of a Trial so that it always knows the Run that initiated it. Thus, deep in code initiated by the Trial, it is possible for carelessly coded methods to have side effects on the Run. Avoiding this or controlling its consequences requires extreme programming discipline, including consideration of the impact on the Run all the way up the call chain to the point where the method Run.Execute() has invoked Trial.Execute(). In general, C++ offers mainly the rather coarse and dangerous "friend" facility for allowing classes to access each others private methods, and we were forced to declare the Trial and Run classes to be mutual friends. Despite this, the careful separation of data gathering and analysis from control of the presentation of text, has allowed us successfully to make nearly two dozen functional enhancements in the course of five years of ongoing use of the software that has resulted in 10 studies and five refereed papers and conference presentations. Most of these enhancements have been quite easy, requiring only a few programmer hours or days to implement. In a subsequent section we will discuss those that were difficult.

5 Abstraction of typographic and display primitives.

The RSVP machine-independent C++ code has about 7600 lines in 25classes. For these classes to use machine dependent code, there is a small collection of primitives for manipulating and displaying text and processing I/O events. In order to port to a new architecture, we have only to implement these primitives. For display, the situation is much akin to the design of the Abstract Windowing Toolkit of Java 1 in which

[4] This is such a common problem that a Google web search on ' "Memory leak" C++ ' produced nearly 30,000 links, and quite a few open source and proprietary software packages have been developed to detect such leaks.

a "peer" is implemented in the particular architecture[5], and we will use that terminology below. The first implementation was for the Macintosh under MacOS version 7, using the QuickDraw GX graphics API and the VideoToolkit of Denis Pelli (Pelli 2000), and we have since implemented under Microsoft Windows using the Windows SDK and recently using OpenGL. The latter should run under the new MacintoshOS/X, but has not been tried. These implementations required from 1300 to 2000 lines of system dependent code.

There are several critical requirements for the peer. Foremost is the ability to support painting—and more particularly erasing—text for RSVP display in a single frame time, synchronized to the video vertical retrace. This synchronization guarantees that when an RSVP word is on the screen, sometimes for as little as 13.33 ms (one frame time on a 75 Hz display), the only pixels visible are those of the given word. Without this synchronization and erasure between words, the display code might deposit a word in the frame buffer before the previous word is erased and, as the pixels are read from the frame buffer to the screen by the sweeping beam, there would be pixels from both words on the screen for part of the display. This strategy limits our presentation rate to one word every two frames, which is 37.5words per second on a 75 Hz display. This corresponds to 2250 words per minute (wpm), which is 2-3 times faster than anyone can read. Our need for this strategy is clearly not a requirement to have presentations this fast. Rather it is to have fine temporal resolution in increasing and decreasing reading speed without having to consider whether the erasure time is a consequential fraction of the display time. For example a reading rate of 500 wpm, i.e. 120 ms per word, is quite feasible with RSVP for normal readers and this is only 9 frame times on a 75 Hz display. To assign a 500 wpm reading rate, the staircase procedure requires us to determine that the subject cannot reliably read an 8 frame-time display[6]. Other requirements for the typographic peer include the ability to scale and display outline fonts in standard formats, to provide control of color and intensity of characters and background, to support scrolled text, and to support two monitors independently programmable (one for presentation to the subject and one for the experimenter to see what sentence the subject is reading). Other requirements for the host support include reliable clock access with millisecond resolution and ease of accepting and ignoring mouse and keyboard events. All three current implementations provide these.

Two enhancements arising several years after initial delivery imposed additional requirements on the host interface. In investigations now ongoing, we control eye tracking hardware to discover whether subjects in fact keep their eyes fixed when they read RSVP presentations (low-vision subjects don't; normal-vision subjects do (Rubin and Turano 1994); Normal subjects reading RSVP text slowly make saccades(Aquilante, Yager and Morris 2000)). This requires relatively simple, albeit real-time, control of an external device. The Macintosh allows this easily, but Windows does not.

A second new functionality, still under development, requires various transformations of imaged text. The first is mirror image presentation. This will be used in experiments driving a Scanning Laser Opthalmoscope (SLO). An SLO images directly on the retina with a low-power laser and also captures a reflected image of the retina itself, from the light of a second low-power laser Many low-vision patients have physical retinal damage resulting in scotomas—holes in their visual field—whose location can be seen on the reflected retinal image. By comparing in software (or visually with captured video) the reflected retinal image with the projected text image, it is possible to know exactly where on the retina the subject is placing each word ((Timberlake, Peli, Essock and Augliere 1987), (Webb and Hughes 1980)). Simple geometric optics will convince the reader that the outside world is actually projected on the retina upside down and backwards from our learned sensation of it, and to use the SLO for our experiments we must adjust the projection for this.

In a contemplated set of experiments about reading from medicine bottles, we need to image text on a virtual cylinder that can be rotated during reading. Both this and the mirror image presentation have been implemented with OpenGL, which has standard matrix transformations for accomplishing these

[5] http://java.sun.com/products/jdk/1.0.2/api/Package-java.awt.peer.html

[6] : As long as we are prepared to ignore the psychophysical significance of the erasure time the distinction between counting erasure and not is actually irrelevant to our real purposes. The reason for this is that absolute reading rates are essentially never of interest. Rather, we want to compare reading rates under different conditions of presentation and typography

manipulations. At this writing, the mirror imaging has adequate performance, but rotating 3D text does not. We believe that we can accomplish this, and that the OpenGL implementation will become our standard on both Windows and Macintosh platforms.

The input side of the peer interface is quite simple, amounting only to a function *getkey()* to get a single character from the keyboard and several functions for dealing with mouse button events. Because of this simplicity, we easily added code to these functions that intercepts them in the based on a boolean variable signifying that system regression testing is in progress. This allows us during testing to take these events from a script file and thereby easily compare the results with execution of the same script on another platform or from a previous version, without need to maintain platform dependent native scripting arrangements for such tests.

6 Memory management

As remarked above, C++ is notorious for leaking memory because memory management is left to the application programmer. Indeed, had Java been mature when we started the project in 1996, its built-in garbage collection alone would have motivated us to use it instead of C++. Instead, we adopted a rigorous discipline in our class implementation that not only prevents memory leaks, but rewarded us handsomely in a simple enhancement that should have been foreseen at design time, but wasn't. The discipline is that for any class that does dynamic memory management we always provide destructors to release memory, deep copy methods, a copy constructor, and a method to override the assignment operator (these latter necessarily go hand in hand in C++). This allows cavalier assignments of complex objects. Although in general this entails a lot of copying and freeing memory, it has no impact on us because such events are rare, and most of our performance requirements are deep in the typographic peer. This issue too would have been moot in Java, where assignment is only by reference.

An experiment in a single condition (comprising a single Run) is concluded when the staircase procedure terminates as described earlier. This is typically a few minutes for normal-vision subjects, and rarely more than 10 minutes for low-vision subjects. Indeed, if the experimenter has a good sense of the likely reading rate for a given experiment, she can cause initiation near this rate and the staircase might terminate after only a few Trials. As we gained experience using the software it became obvious that rather than restarting once per experiment (i.e. per Run), it would be better to have a list of the varying experimental conditions required for a particular investigation (e.g. for several Runs at different type sizes when size is the controlled variable). In order to accomplish this it was merely necessary to control the Run by an object from a new class we called a *Session*. A Session executes a loop mediated by a list of Setup objects (initialized from external files) passed as arguments to the constructor for a Run object that is destroyed, along with the Setup object, after the Run's Execute method completes. In reality, our code review prior to doing this revealed that in fact we had not implemented destructors for the Run and Setup classes as our coding discipline required, though to do so was straightforward. In the single Run-per-execution implementation this was unnoticed and inconsequential. (However, we have seen versions of both Windows and MacOS in which the OS did not always recover application memory after termination, and a long series of executions could require a reboot due to memory exhaustion.)

7 Adding functionality

The most frequent enhancement request has been the addition of experimental conditions that can be controlled by the experimenter. In our design, this entails providing the experimenter a way to put a key and value into an external file, reflecting that key-value pair in the Setup object, and modifying the business logic of a Run to account for the new condition. Since the Run has access to the Setup object, no communication has to be modified as a consequence of the new facility. Because this pattern recurred in several other (less rapidly evolving) facilities, we implemented a map function of our own (actually an associative list with both forward and reverse lookup) using the somewhat controversial C++ template facility. Such maps are available in the now stable STL. Our associative list is simple (we were content with linear search through the associative lists because all were small and searched only once, at the construction of each Run), but we would probably use STL now. A recent enhancement records on the file system what sentences a subject has seen and never presents the same sentence twice, no matter how many experiments a subject participates in. For this it was indeed convenient to use the STL hash facility, since

our sentence library runs to the thousands and is growing.

A typical class of enhancements whose ease accrues from the strict object orientation was control of the output format of the data gathered. Because data is accumulated in the Staircase object, format is controlled in a single output method of the Staircase class, which method also has access to the experimenter's format requirements expressed in the initialization file. Indeed, though we presently support only two specific data recording formats (a human readable one with extensive labels and a tab-separated one for import into spreadsheets and data analysis software), a change to only a single line of code in the output method and the addition of one in the Staircase initialization file reader, could implement, for example, passing a C printf-style format string from the initialization all the way to the output. All the intermediate business logic would remain ignorant of this change.

To date, the only enhancements that have presented any consequential difficulties have been those requiring modifications to the peer to support continuously scrolling text under carefully timed conditions. The main reason for this is that in all our current architectures, the native API for scrolling text is largely devoted to processing messages from user interface components such as scroll bars on windows. In our case, we needed to provide the experimenter to control independently several scrolling parameters for comparison of their impact on reading. These included both the scrolling speed and its temporal resolution, i.e. how many pixels text moves on each motion as well as how long it is paused at the location. Together these determine the scroll velocity, but the reader's perception is different according to how smooth the scroll is. In addition, scrolled display must be possible both vertical and horizontal, and must also be viable with all transformations we might make of the text. OpenGL presents simpler interfaces for doing this, but in some cases the text must be converted to OpenGL "textures" or rendered using special fonts (Kilgard 1997). We expect that this will ultimately prove the most satisfactory solution, especially as increasing numbers of video accelerators are providing fast hardware support for OpenGL, which will probably make all our performance requirements met without sophisticated video programming on our part. Meanwhile, our scroll implementations have depended on rendering text off-screen and moving it as clipped images through the scroll region.

Finally, we note that although we have not done so, our software can support the investigation of other time-based psychophysical and cognitive investigations with little or no modification. This is because among the objects we can display are arbitrary images provided in the Portable Graymap Metafile format (PGM). Other formats could be supported with only programming in the peer, as described earlier.

8 Futures

Informal evidence from our low-vision subjects suggests that our software would make a useful low-vision reading aid in its RSVP display mode. We have already implemented a number of controls by which the subject can control the reading rate and re-read text that went by too fast. We have also implemented RSVP timing based on word length so that shorter words need not take as much time, and subjects report a preference for this and indeed read on average 33% faster than without this adjustment ((Aquilante, Yager, Morris and Khmelnitsky 2001)). This expression of satisfaction with control of presentation speed suggests that we try other speed control UI, such as joysticks and voice control.

There is also some reason to believe that prescribing reading lenses based on letter charts under-prescribes the optimal magnification for reading ((Lovie-Kitchin and Whittaker 1999)). Because our software is easily installed on a laptop computer, we are contemplating designing a diagnostic aid based on it.

Colored text is usually without impact on normal readers unless it has luminance contrast that is too low (Legge, Parish, Luebker and Wurm 1990). Colored text is often used on the web and has been widely studied (Muter 1996). However, very little seems to be known about the interaction between color and text size and there is some reason to expect such interaction because large and small objects are processed in different parts of the human visual system, and the subsystem responsible for the large objects is color blind. In addition, some controversial studies suggest that reading is mediated by this subsystem (Garzia 1995). Our present implementation supports the display of colored text, but should we pursue studies of such text it may become convenient to provide subject-specific color and luminance calibration, discussion of which is beyond the scope of this paper.

9 References

Aquilante, K., D. Yager and R. A. Morris (1998). How big must letters be for patients with central field loss to read RSVP text at their maximum rate? Vision Science and its Applications, Optical Society of America.

Aquilante, K., D. Yager and R. A. Morris (1999). Are low vision patients with central visual field loss more sensitive to changes in luminance when reading than are age-matched normals? Vision Science and Its Applications. Washington, Optical Society of America. **1:** 25-28.

Aquilante, K., D. Yager and R. A. Morris (1999). Does RSVP reading take practice to reach maximum reading rates? Vision Science and Its Applications.

Aquilante, K., D. Yager and R. A. Morris (2000). Can visual acuity predict the size of text that low vision readers need to read at maximum rates? Vision rehabilitation. Assessment, intervention and outcomes. C. Stuen, A. Arditi, A. Horowitzet al. Lisse, Swets &Zeitlinger**:** 288-292.

Aquilante, K., D. Yager, R. A. Morris and F. Khmelnitsky (2001). "Low vision patients with age-related maculopathy read RSVP faster when word duration varies according to word length." Optometry and vision science : official publication of the American Academy of Optometry **78**: 290-296.

Borland (1994). "C++ Design - Implementing a Model-View-Controller design in C++." Borland C++ Developer's Journal **1**(9).

Bullimore, M. A. and I. L. Bailey (1995). "Reading and eye movements in age-related maculopathy." Optom Vis Sci **72**(2): 125-38.

Dickinson, C. (1998). Low Vision. Principles and Practice. Oxford, Butterworth Heinemann.

Fine, E. M. and E. Peli (1998). "Benefits of rapid serial visual presentation (RSVP) over scrolled text vary with letter size." Optom Vis Sci **75**(3): 191-6.

Garzia, R. P., Ed. (1995). Vision and Reading, Mosby-Year Book, Incorporated.

Goodrich, G. and I. Bailey (2000). A history of the field of vision rehabilitation from the perspective of low vision. The Lighthouse handbook on vision impairment and vision rehabilitation. B. Silverstone, M. Lang, B. Rosenthal and E. Faye. New York, Oxford University Press**:** 675-715.

Jacobson, I. (1992). Object-oriented software engineering : a use case driven approach. Reading, Mass., Addison-Wesley Pub.

Kennedy, A. (2000). Attention allocation in reading: Sequential or parallel? Reading as a perceptual process. A. Kennedy, R. Radach, D. Heller and J. Pynte. Amsterdam, Elsevier.

Kilgard, M. (1997). A Simple OpenGL-based API for Texture Mapped Text, Silicon Graphics, Inc.

Latham, K. and D. Whitaker (1996). "A comparison of word recognition and reading performance in foveal and peripheral vision." Vision Research **36**(17): 2665-74.

Leat, S. J., G. E. Legge and M. A. Bullimore (1999). "What is low vision? A re-evaluation of definitions." Optom Vis Sci **76**(4): 198-211.

Leat, S. J. and J. M. Woodhouse (1993). "Reading performance with low vision aids: relationship with contrast sensitivity." Ophthalmic Physiol Opt **13**(1): 9-16.

Legge, G. E., D. H. Parish, A. Luebker and L. H. Wurm (1990). "Psychophysics of reading. XI. Comparing color contrast and luminance contrast." J Opt Soc Am A **7**(10): 2002-10.

Legge, G. E., G. S. Rubin, D. G. Pelli and M. M. Schleske (1985). "Psychophysics of reading--II. Low vision." Vision Research **25**(2): 253-65.

Lovie-Kitchin, J. and S. G. Whittaker (1999). "Prescribing near magnification for low vision patients." Clin Exp Optom **82**: 214-224.

McMahon, T. T., M. Hansen and M. Viana (1991). "Fixation characteristics in macular disease. Relationship between saccadic frequency, sequencing, and reading rate." Invest Ophthalmol Vis Sci **32**(3): 567-74.

Muter, P. (1996). Interface Design and Optimization of Reading of Continuous Text. Cognitive aspects of electronic text processing. H. van Oostendorp and S. de Mul. Norwood, N.J, Ablex Publishing Corp.

National Advisory Eye Council (1998). Visual Impairment and its rehabilitation. Vision Research - A National Plan: 1999 -2003,, National Institutes of Health. **Publication No. 98-4120:** 117-130.

Pelli, D. (2000). VideoToolbox: C routines for visual psychophysics on Macs.

PGM (2001). Portable Graymap Metafile.

Rayner, K., A. W. Inhoff, R. E. Morrison, M. L. Slowiaczek and J. H. Bertera (1981). "Masking of foveal and parafoveal vision during eye fixations in reading." J Exp Psychol Hum Percept Perform **7**(1): 167-79.

Reichle, E. D., A. Pollatsek, D. L. Fisher and K. Rayner (1998). "Toward a model of eye movement control in reading." Psychol Rev **105**(1): 125-57.

Rubin, G. S. and G. E. Legge (1989). "Psychophysics of reading. VI--The role of contrast in low vision." Vision Research **29**(1): 79-91.

Rubin, G. S. and K. Turano (1992). "Reading without saccadic eye movements." Vision Research **32**(5): 895-902.

Rubin, G. S. and K. Turano (1994). "Low vision reading with sequential word presentation." Vision Research **34**(13): 1723-33.

Silicon Graphics, I. (2001). Standard Template Library Programmer's Guide, Silicone Graphics, Inc.,. **2001**.

Starr, M. S. and K. Rayner (2001). "Eye movements during reading: some current controversies." Trends Cogn Sci **5**(4): 156-163.

Timberlake, G. T., M. A. Mainster, E. Peli, R. A. Augliere, E. A. Essock and L. E. Arend (1986). "Reading with a macular scotoma. I. Retinal location of scotoma and fixation area." Invest Ophthalmol Vis Sci **27**(7): 1137-47.

Timberlake, G. T., E. Peli, E. A. Essock and R. A. Augliere (1987). "Reading with a macular scotoma. II. Retinal locus for scanning text." Invest Ophthalmol Vis Sci **28**(8): 1268-74.

Webb, R. H. and G. W. Hughes (1980). "Flying spot TV ophthalmoscope." Applied Optics **19**: 2992-2997.

Whittaker, S. G., R. W. Cummings and L. R. Swieson (1991). "Saccade control without a fovea." Vision Research **31**(12): 2209-18.

Whittaker, S. G. and J. Lovie-Kitchin (1993). "Visual requirements for reading." Optom Vis Sci **70**(1): 54-65.

WHO Study Group (1973) (1973). The Prevention of Blindness. Geneva, World Health Organization. **Report No. 518**.