SensaWave

SENSAWAVE Manual

Table of Contents

1 Introduction

Welcome to the SENSAWAVE Computing Base

Thank you for your interest in the SENSAWAVE Computing Base, a powerful system for rapid prototyping of scientific and engineering applications, to which this manual serves both as an informal introduction and in-depth reference.

New users are encouraged to read Chapter 3 [quicktour], page 6, executing the examples in an interactive SENSAWAVE session to quickly gain an understanding of the SENSAWAVE fundamentals. Experienced users can find detailed information about the language syntax in the remainder of the manual.

We invite you to contact us with any questions or comments related to this manual or the SENSAWAVE program. Please visit our website for complete contact information: `http://www.sensawave.com/contact.html`

We hope that the SENSAWAVE Computing Base will prove useful to you.

The SensaWave Team, 2008.

1.2 SENSAWAVE features and benefits

The SENSAWAVE Computing Base is a powerful and flexible interactive programming environment tailored for scientific and technical computationally intensive applications.

SENSAWAVE bridges the gap between rapid prototyping and direct interfacing to high performance scientific libraries. Applications benefit both from an easy-to-use general purpose multi-threaded language with sophisticated debugging facilities and a thin interface to an extensive repository of pre-compiled scientific libraries.

1.2.1 Fast execution speed

SENSAWAVE can compile applications to portable C language, and native machine code is generated using an efficient C compiler on the host computer. This enables very high execution speeds to be obtained. Compiled and interpreted code can be mixed freely for maximum flexibility.

1.2.2 Scientific repository

SENSAWAVE provides a complete interface to a large repository of industry standard numerical and scientific code, including the ARPACK, BLAS, EISPACK, FFTPACK, LAPACK, LINPACK and SLATEC packages. A total of 33 such packages are provided, giving technical computing developers unprecedented choice of implementation, and full control behind the scenes.

1.2.3 Advanced visualization

SENSAWAVE provides a number of powerful systems optimized for creating high-quality visualizations in scientific computing applications. This includes a device independent graphing system, frame buffers operating on of 32 bit image data arrays, as well as both 2D and 3D engines providing graphics primitives in real-world coordinates. Direct OpenGL rendering is also supported, and input event callbacks allow complex interactive graphical applications to be developed.

1.2.4 Cross-platform development

SENSAWAVE provides maximum freedom to developers by supporting a wide range of operating systems and offering multi-platform licenses. Code runs unaltered on all supported platforms, and can be compiled to native binaries on each system.

1.2.5 Choice of programming language

SENSAWAVE applications can be written in two different programming styles: functional (prefix) or C/Java (infix). Experienced users will find that the functional style is the most terse and effective, but the C style alternative allows new users to start writing applications right away in a familiar syntax. The two programming styles can be mixed freely.

2 Installation

This chapter describes the software installation process. All files of the SENSAWAVE Computing Base resides in a single directory.

2.1 Installation from CDROM

When you purchased the Software you received a CDROM containing a licensed copy of the SENSAWAVE Computing Base. The CDROM contains the following files:

`sensawave-1.0-win32.exe`	Self-extracting installer for Windows platforms
`sensawave-1.0-macosx.dmg`	Disk image for MAC OS X (universal) installations
`sensawave-1.0-linux.tar.gz`	Linux (ix86) installation archive
`sensawave-1.0-openbsd.tar.gz`	OpenBSD (ix86) installation archive
`license.bin`	The SENSAWAVE license key

These files can be used to install SENSAWAVE Computing Base on the supported platforms. The version on the CDROM is the most current at the time of purchase. To install, simply select the installation file that matches your host system and follow the instructions below to extract the distribution directory. Once the installation directory has been created, the license key must be copied into it, in order to activate the program.

2.1.1 Installing on Windows

To install on Windows, open the CDROM and double click on the executable Windows installer. The installer will by default create a directory on the desktop containing the SENSAWAVE files. Once the installer is finished, drag a copy of `license.bin` from the CDROM into the new directory.

2.1.2 Installing on MAC OS X

To install on MAC OS X, simply double click on the `macosx` disk to mount it. Inside the image is a directory containing the SENSAWAVE files. Copy this to a location of your choice. Drop a copy of `license.bin` from the CDROM into the new directory.

2.1.3 Installing on Linux

To install on Linux, extract the SENSAWAVE `tar` archive with the following command

```
$ tar -zxf sensawave-1.0-linux.tar.gz
```

in the location where you want the SENSAWAVE directory to appear. Drop a copy of `license.bin` from the CDROM into the new directory.

2.1.4 Installing on OpenBSD

To install on OpenBSD, extract the SENSAWAVE `tar` archive with the following command

```
$ tar -zxf sensawave-1.0.2-openbsd.tar.gz
```

in the location where you want the SENSAWAVE directory to appear. Drop a copy of `license.bin` from the CDROM into the new directory.

2.2 Installing from the Internet

The files described in Section 2.1 [cdrom], page 3 can also be downloaded from the SensaWave website: `http://sensawave.com`. The exception is your license key, which can only be found on the distribution CDROM. This key must then be dropped into the distribution directory after the installation of the downloaded files. Your license key will work with all maintenance releases of the major release for which it was purchased. This allows you to benefit from the later revisions of that release.

2.3 Changing the system PATH environment

SENSAWAVE can be run from within its directory with no further actions. However, on systems with a command line interface, in particular Linux and OpenBSD, you will benefit from having the SENSAWAVE directory added to the `PATH` environment to allow the system to be started from anywhere on the host. This can be accomplished like this (in bash/korn shell):

```
$ PATH=$PATH:/path/to/sensawave
$ export PATH
```

where `/path/to/sensawave` is the fully qualified path of the SENSAWAVE directory. The above command can be added to the appropriate startup files to make the change permanent. Please refer to your host system for the information about how to do that.

The above is not normally needed on Windows, unless you plan to call the SENSAWAVE executable from the command prompt. In that case you can do the following:

1. Open the System Properties under the Control Panel.
2. On the Advanced tab click on Environment Variables, select the existing variable Path, and click Edit.
3. Add the SENSAWAVE directory to the path and click OK:

```
c:\your current path setting\;c:\path\to\sensawave
```

2.4 Host system dependencies

The only dependency that SENSAWAVE has on the host system is that the `gcc` compiler is needed when generating native binaries. Windows does not have this dependency, since the SENSAWAVE Windows distribution ships with its own gcc compiler suite for your convenience. On Mac OS X the `gcc` compiler is found in X code developer suite that can be obtained for free from Apple. Linux and OpenBSD distributions are likely to have the compiler installed by default. If not, it can be installed from a separate standard package.

Please note that `gcc` is only needed if you intend to compile SENSAWAVE programs to binaries. Normal operation does not require the compiler to be present.

2.5 Uninstalling SENSAWAVE

Uninstalling simply involves deleting the SENSAWAVE directory from your computer.

2.6 Further Questions

In case of any problems or questions, please contact SensaWave Technology Inc. by email to `support@sensawave.com`, or call us at 1-604-322-9029 (pacific standard time). Please have your order number at hand.

3 Quick Tour

3.1 What is SENSAWAVE?

SENSAWAVE is a rapid prototyping environment for scientific and engineering applications. It is capable of performing a wide range of advanced scientific, statistical and mathematical operations, and contains extensive visualization capabilities.

SENSAWAVE provides a simple and thin interface to the underlying computation/visualization libraries, getting in the developers way as little as possible. For example, rather than providing a restrictive higher level interface to the underlying scientific libraries, SENSAWAVE gives access to all functions in the libraries, including auxiliary and intermediate routines. This makes SENSAWAVE a unique environment for building new applications with a high technology content.

Unlike most other technical computing environments, SENSAWAVE is built on a full-fledged multi-threaded general-purpose language, and is capable of compiling code to native binaries through the use of an efficient C compiler. This enables applications to achieve execution speeds rivaling C. Compiled and interpreted code can be mixed freely, and external C code can easily be integrated through an efficient foreign function interface.

3.2 Running SENSAWAVE

On systems that support command line interfaces, SENSAWAVE can be started by typing `sensawave` at the command prompt. Systems with graphical user interfaces can activate SENSAWAVE by clicking on the SENSAWAVE program icon.

Once SENSAWAVE is started, you will see a message followed by an input prompt, like this:

```
 -- --     --                --
(_ |_ |\ |(_  /\ |  | /\\  /|_
__)|__| \|__)/--\|/\|/--\\/ |__
SENSAWAVE Computing Base ver. 1.0.2
Copyright (C) 2008 SensaWave Technology Inc. All rights reserved.
See user manual for full copyright statement and terms of use.
[/home/doe/sensawave-1.0.2-win32]
Single User License #1234567890 ACME Inc.
Use (exit) to terminate, (demo) for graphical demonstration.
>
```

The SENSAWAVE input prompt accepts commands, expressions, debugging commands etc. Each time you press enter the SENSAWAVE engine will process your command, and print a result.

```
> (display "hello world\n")
"hello world"
```

3.3 Basic command line syntax

SENSAWAVE is built on top of Gambit-C, a powerful implementation of the functional language Scheme. By default, the command prompt follows the Scheme prefix syntax. Expressions are usually of the following format:

```
(proc [arg1 [arg2 ...]])
```

This calls the function `proc` with the provided arguments. For example:

```
> (min 3 1 5 2 4)
1
> (string-append "A" "B" "C")
"ABC"
> (> 2 1)
#t
> (string=? "apples" "oranges")
#f
> (* 1 2 3 (+ 1 2 3))
36
```

Variables are bound using the `define` command, and the bindings can be changed with `set!`:

```
> (define x 1.23)
> x
1.23
> (set! x "test")
> x
"test"
> (define (myfunction x) (+ x 10.))
> (myfunction 2.3)
12.3
```

3.4 Numerical operations

SENSAWAVE features a powerful numerical subsystem that supports complex, rational and exact numbers:

```
> (complex? 1+1.0i)
#t
> (sin (* 1.2+2.4i 0+1.6i))
2.240067903791034-2.5558335585424428i
> (+ 1/214 1/128)
171/13696
> (rationalize M_PI 1e-6)
355/113
> (rational? 1/2)
#t
> (exact? 1)
#t
> (exact? 1.0)
#f
```

The fundamental data structure in SENSAWAVE is the list:

```
> (define a (list 1 3 2 4))
> (list-ref a 0)
1
```

```
> (map square a)
(1 9 4 16)
> (apply max a)
4
> (sort a <)
(1 2 3 4)
```

As a simple example, ley us graph `sin(x)` in the range -2Pi<=x<=2Pi. We first store the values from -2*Pi to 2*Pi in a list with 100 points:

```
> (define x (: (- M_2PI) M_2PI 100))
```

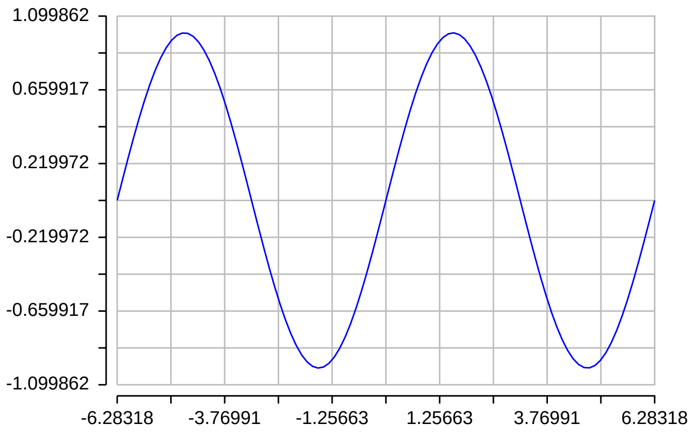
The colon function creates a list of 100 numbers evenly distributed from -2Pi to 2Pi. We then create a new list that applies `sin` element by element to `x`:

```
> (define y (map sin x))
```

To plot these points we do:

```
> (ezgraph x y)
```

This will create a simple graph using `x` and `y` as coordinates:



3.5 Vectors, matrices and the Scientific Repository

SENSAWAVE also supports a number of different homogenous vector types that are often more efficient for numerical operations. For example, we can create a complex single precision vector like this:

```
> (define C (c32vector 5 7.8 M_PI 12 5+1.0i))
```

This vector has five elements. We can also create vectors from lists such as those generated by the colon operator:

```
> (define B (list->c32vector (: 1 5 5)))
```

Let us calculate the scalar product of these two vectors using the BLAS function `cdotu`. BLAS (Basic Linear Algebra Subprograms) is a library routines that provide standard building blocks for performing vector and matrix operations. SENSAWAVE provides a loadable precompiled BLAS library:

```
> (module-require 'BLAS)
> (BLAS_cdotu 5 C 1 B 1)
103.0247802734375+5.i
```

We can confirm this result by hand:

```
> (+ (* 1. 5.) (* 7.8 2.) (* M_PI 3.) (* 12. 4.) (* 5+1.0i 5.))
103.02477796076938+5.i
```

There is a slight discrepancy between the two results. This is due to the difference in precision. SENSAWAVE uses double precision numbers, while `cdotu` is a single precision BLAS function. Let us repeat the calculation using the double precision BLAS equivalent, `zdotu`:

```
> (define C (c64vector 5 7.8 M_PI 12 5+1.0i))
> (define B (list->c64vector (: 1 5 5)))
> (BLAS_zdotu 5 C 1 B 1)
103.02477796076938+5.i
```

This result is now identical to the internal SENSAWAVE calculation, as expected.

The SENSAWAVE Computing Base also supports simple homogenous matrix structures internally, as a simple extension of the corresponding vectors. A matrix is often conveniently built from a list of lists. For example, here is a 3x3 double precision matrix M, and a 1x3 matrix X:

```
> (define M (listlist->f64matrix '((3. 1. 3.)
                                    (1. 5. 9.)
                                    (2. 6. 5.))))
> (define A (listlist->f64matrix '((-1.)(3.)(-3.))))
```

The quotes in the above command instruct SENSAWAVE to treat the quoted lists and data, instead of attempting to evaluate them as expressions.

Let us now use these to solve the equation `M*X=A` using the double precision LAPACK function `dgesv`. LAPACK, the Linear Algebra PACKage, is a library for numerical computing that provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems etc. We can load it just like we did with BLAS:

```
> (module-require 'LAPACK)
> (define P (make-s32vector 3))
> (LAPACK_dgesv 3 1 M 3 P A 3 0)
> (f64matrix->listlist A)
((-1.) (-1.) (1.))
```

`P` is a three element vector of signed 32bit integers that is needed for LAPACK to complete the calculation. The value returned in A is the matrix X solving M*X=A, as can readily be confirmed.

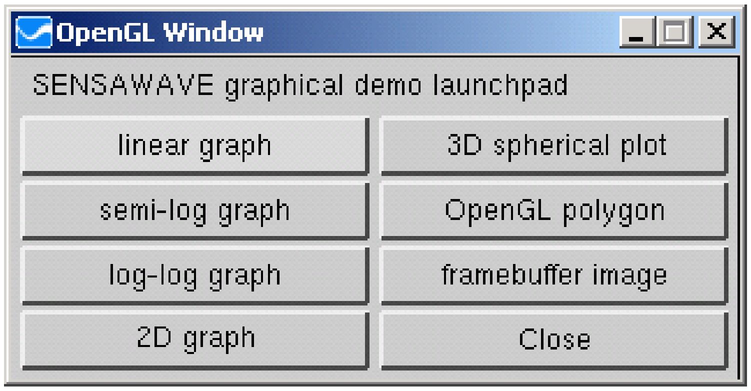
These simple examples illustrate how simple the interface to the fortran repository is. All functions of the entire fortran repository can be accessed in this way, giving the developer the ability to fine-tune numerical calculations and even utilize auxiliary functions in the repository.

3.6 Graphics in SENSAWAVE

The SENSAWAVE Computing Base provides a number of powerful visualization mechanisms. We will use the `demo` procedure to illustrate some of the capabilities of the system:

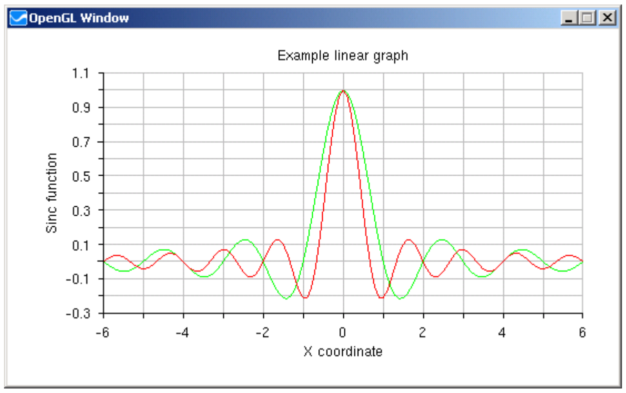
```
> (demo)
Please wait, building demonstration data...
#t
```

A new window will now appear on the screen. This is the graphical demonstration launchpad:



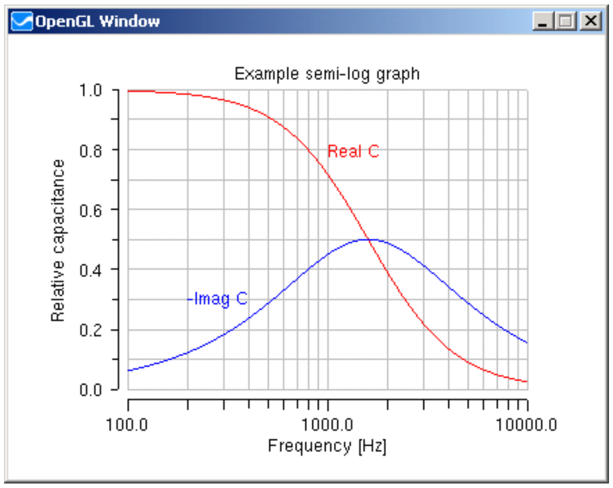
The launchpad contains a number of buttons that will launch additional windows, and is itself a small demonstration of the SENSAWAVE OpenGL interface. This graphical interface provides a simple way to create powerful interactive programs. Each window is running in an independent thread using the very efficient multi-threading capabilities of the system.

Let's click on the top left button labeled `linear graph`. This will launch an example of a linear graph built with the device independent graphing subsystem in SENSAWAVE:

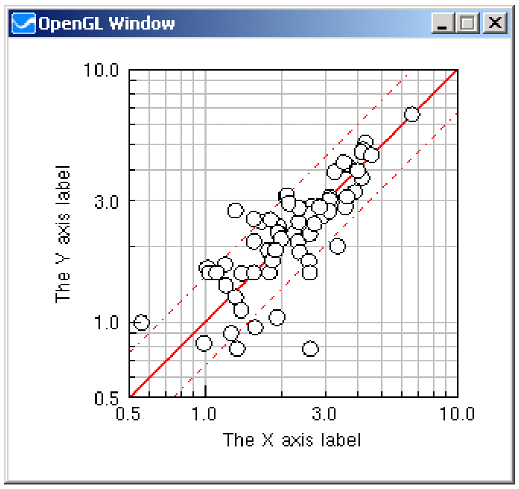


SENSAWAVE graphs can be rendered to a multitude of devices, including PDF, EPS, PNG, PPM, framebuffers or direct OpenGL (the above), and will appear identical on all output devices. This provides a powerful framework for developing multi-purpose scientific visualizations.

Clicking on the `semi-log graph` button of the launchpad opens another window, this time containing an example of a semi-logarithmic graph:

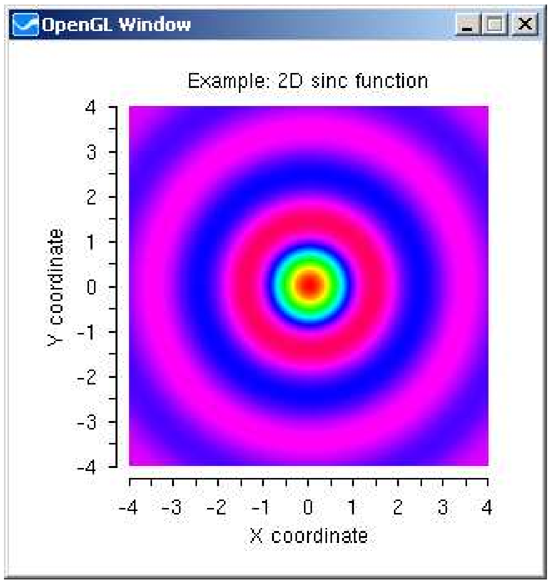


This graph uses labels to identify the two curves. Another illustration of the graphing subsystem opens when clicking on the `log-log` button of the launchpad. This is a double logarithmic graph with data point markers:

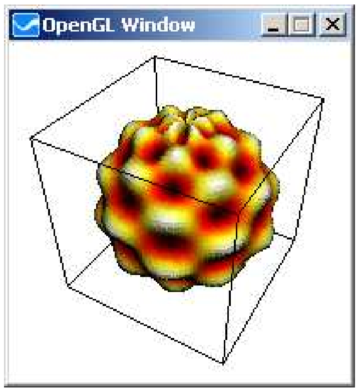


Graphs are highly customizable and can easily be combined to build larger graphical data presentations.

SENSAWAVE also contains a powerful framework for generating raster images using device independent framebuffers with full alpha channels. Framebuffers support both two-dimensional and three-dimensional operations, making it easy to create complex images. Framebuffers can be combined with the graphing subsystem, for example to provide two-dimensional graphs. The `2D graph` launchpad button illustrates such a combination of framebuffer and graph:



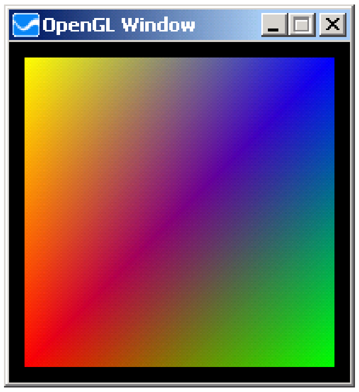
An example of a pure framebuffer rendering opens when clicking the `3D plot` lauchpad button:



This is a spherical plot illustrating the use of color gradients and shading with 3D framebuffer drawing primitives.

In addition to the device independent graphing and framebuffer facilities, SENSAWAVE also provides direct access to the OpenGL API, making it simple to create OpenGL graphics.

The `OpenGL polygon` launchpad button opens a very simple direct OpenGL rendering of a shaded polygon:



The last `framebuffer image` button on the graphical demo launchpad illustrates loading of images from file:

This image is loaded from a PNG file into a framebuffer, and rendered to the screen with a single command:

```
> (png->glwindow (path-expand "~~/examples/balloons.png"))
```

3.7 Reading and writing data files

Most technical computing applications require that numerical data be saved to and read from permanent media such as hard disks. SENSAWAVE provides a simple mechanism for loading and saving data as comma separated values (CSV). Such files can also be handled by most spreadsheet programs such as Excel.

Let us save an example data set to file:

```
> (define data '((1 2 3)(4 5 6)(7 8 9)))
> (datafile-save "example.csv" data)
```

The resulting file contains three lines of comma separated values. We can reload the data like this:

```
> (datafile-load "example.csv")
((1 2 3) (4 5 6) (7 8 9))
```

If the data is not going to be used for import into other programs, it can be more efficient to save it in native format. For example:

```
> (object->file data "example.dat")
> (file->object "example.dat")
((1 2 3) (4 5 6) (7 8 9))
```

In addition to these simple interfaces, SENSAWAVE also supports general purpose file I/O that can be used to generate and/or parse more complicated data file formats.

3.8 Writing SENSAWAVE programs

So far we have been typing commands at the SENSAWAVE prompt. For real applications it is much faster to write program files that SENSAWAVE can execute. Two different types of program files are supported. `*.scm` files are in the same functional style that we have worked with in this chapter, and `*.six` are in a C/Java style.

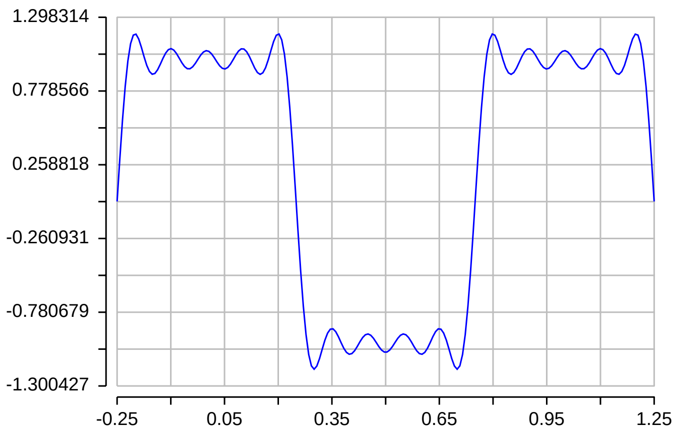
3.8.1 Example: Fourier approximation to a square wave

A program to plot a Fourier approximation to a square wave might look like this in functional style:

```
;; examples/nthharmonic.scm
;; program to plot fourier series to n-th harmonic

;; element-wise addition of two lists
(define (list+ l1 l2)
  (let loop ((L1 l1)(L2 l2)(r '()))
   (if (= (length L1) 0) r
     (loop (cdr L1) (cdr L2) (append r (list
       (+ (car L1) (car L2))))))))

(define (nthharmonic n)
  (let* ((w0 (* 2. M_PI))
         (t (: -.25 1.25 200))
         (v (let loop ((i 3.)(factor -1.)
                  (res (map (lambda (x) (cos (* w0 x))) t)))
            (if (>= i n) (map (lambda (x) (/ (* 4 x) M_PI)) res)
              (loop (+ i 2.) (* factor -1.)
          (list+ res (map (lambda (x)
            (* factor (cos (* i w0 x)) (/ 1 i))) t)))))))
  (ezgraph t v)))
```

The program contains two functions, the first of which is a simple helper routine that adds the elements of two lists element by element. The second function calculates the nth harmonic and opens a graph of the result:

```
> (load "~~/examples/nthharmonic.scm")
> (nthharmonic 11)
```

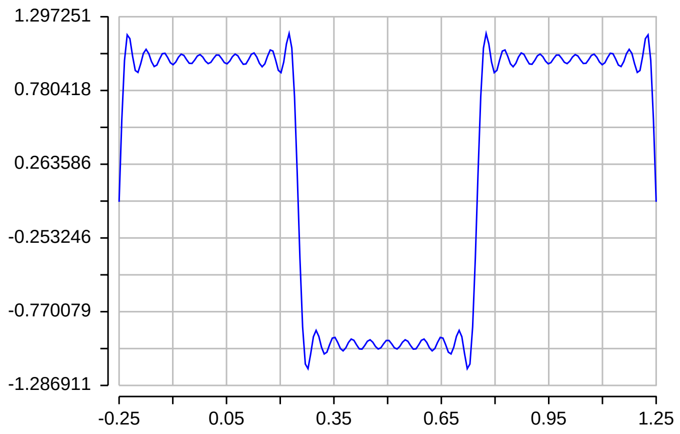


The following accomplishes the exact same thing using the C/Java programming style:

```
// examples/nthharmonic.six
// program to plot fourier series to n-th harmonic

// element by element addition of two lists
obj add_lists(obj l1, obj l2)
{
  obj l3 = [];
  for (int i=0;i<length(l1);i++) {
   l3 = append(l3,list(list_ref(l1,i)+list_ref(l2,i)));
  }
  return l3;
}

obj nthharmonic(int n)
{
  obj w0 = 2. * M_PI;
  obj t = \:(-.25,1.25,200);
  obj v = map( obj (obj x) { return cos(w0*x); },t);
  obj factor = 1;
  for (int i=3; i<n; i+=2) {
    factor *= -1;
    v = add_lists(v, map( obj (obj x)
           { return factor*cos(w0*i*x)/i; }, t));
  }
  v = map( obj (obj x) { return 4*x/M_PI; },v);
  ezgraph(t,v);
}
```

This program can be loaded in exactly the same way as the previous one:

```
> (load "~~/examples/nthharmonic.six")
> (nthharmonic 21)
```



3.8.2 Example: linear regression

This an example of linear regression in standard functional style:

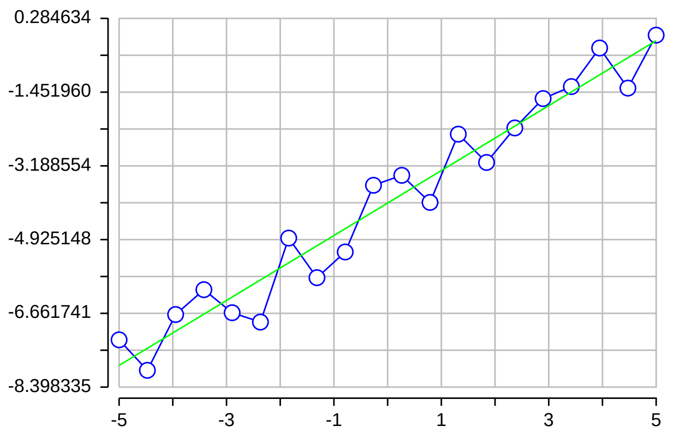
```
;; examples/linear-regression.scm
;; simple linear regression example

;; test data with random noise
(define x (: -5 5 20))
(define y (map (lambda (x)
  (+ (* 0.8 x) -4 (* 2. (- (random-real) 0.5)))) x))

;; perform linear regression
(define result (linear-regression x y))
(display (format
  "Linear regression: y = ~4F * x + ~4F\n"
  (car result) (cadr result)))

;; fitted data
(define x2 (: -5 5 100))
(define y2 (map (lambda (x)
  (+ (* (car result) x) (cadr result))) x2))

;; draw the result
(ezgraph x y x2 y2)
```

And here is the same example in C/Java style:

```
// examples/linear-regression.six
// simple linear regression example
// test data with random noise
obj x = \:(-5, 5, 20);
obj y = map( obj ( obj x) {
  return 0.8*x-4 + 2*random_real()-1; }, x);

// perform linear regression
obj result = linear_regression(x,y);

display( format(
  "Linear regression: y = ~4F * x + ~4F\n",
  car(result), cadr(result)));

// fitted data
obj x2 = \:( -5, 5, 100);
obj y2 = map( obj (obj x) {
 return car(result)*x + cadr(result); }, x2);

// draw the result
ezgraph( x, y, x2, y2);
```

The programs generate the same output:

```
> (load "~~/examples/linear-regression.six")
Linear regression: y = 0.81 * x + -3.9
```



3.9 Compiling SENSAWAVE programs

A unique feature of SENSAWAVE is the ability to compile programs like the one shown in the previous section. In this section we will use this capability on a small benchmark program:

```
;; examples/fib.scm
;; simple fibonacci benchmark
```

```
(define (fib n) (if (< n 2) 1 (+ (fib (- n 1)) (fib (- n 2)))))
(define n 30)
(display (format "Running (fib ~D)..\n" n))
(define start-time (time->seconds (current-time)))
(fib n)
(define end-time (time->seconds (current-time)))
(display (format "(fib ~D) took ~4F seconds to complete.\n"
   n (- end-time start-time)))
```

This program performs a recursive CPU intensive function call, and performs a simple timing:

```
> (load "~~/examples/fib")
Running (fib 30)..
(fib 30) took 2.97 seconds to complete.
"examples/fib.scm"
```

We can see that it took almost three seconds to complete the call. To improve this we now compile the program to binary form:

```
> (compile-file "~~/examples/fib.scm")
#t
```

Note that this call requires that the host system has a gcc compiler installed (see Section 2.4 [host dependencies], page 4).

We can now repeat the call:

```
> (load "~~/examples/fib")
Running (fib 30)..
(fib 30) took 0.12 seconds to complete.
"examples/fib.o1"
```

We notice a speed improvement of about twenty times due to the compilation (this will vary depending on the host system). Notice how the file loaded has the extension `.o1`. This is the binary that was generated by the compiler.

3.10 Further reading

We recommend that you browse the examples in the `examples` subdirectory, and use those as a starting point for exploring the capabilities of the software further.

It is beyond the scope if this manual to offer a complete general description of the Scheme programming language on which SENSAWAVE is built. There are several texts available on the Internet, doing so, including:

The formal Scheme language definition can be found in Revised (5) Scheme Report:
http://sensawave.com/manuals/scheme-r5rs.pdf

SENSAWAVE is built on top of the Gambit-C portable Scheme to C compiler. Gambit-C extends standard Scheme with a large number of additional features. For a complete description of extensions, compiler architecture and optimization, please refer to the Gambit-C manual:
http://sensawave.com/manuals/gambit-c.pdf

4 Console Editing

SENSAWAVE supports interactive development through a simple console interface. Commands can be entered at the console and executed one by one. The console supports command line completion, command history, result history and parentheses balancing.

4.1 Command line completion

The console features command line or tab completion capable of automatically filling in partially typed symbols. For example, typing `glB` followed by pressing the tab key will produce `glBegin`, an OpenGL procedure. Repeated tab key presses will cycle through all matches of the partially entered symbol. Typing `glB` and pressing the tab key twice will produce `glBitmap`, another matching OpenGL procedure, and so on.

4.2 Command history

The console keeps a history of previously typed commands. The history is accessed with the arrow keys. Pressing the up arrow will return the previous command, and the history can be navigated by multiple presses of up and down arrow. The history is maintained between sessions.

4.3 Result history

The console also keeps a short history of the last returned command results. These are accessed by typing one or more hash signs. A single hash sign expands to the last command result, two hash signs the next to last, and three the second to last.

```
> (+ 1 2)
3
> (+ 2 #)
5
> (+ # ##)
8
> (+ # ## ###)
16
```

4.4 Console editing commands

Several editing commands are available at the console. Here is a list of the most commonly used commands:

`<ctrl>a`	Moves the cursor to the beginning of the line.
`<ctrl>c`	Interrupts evaluation of a command.
`<ctrl>d`	Generates an end-of-file.
`<ctrl>e`	Moves the cursor to the end of the line.
`<ctrl>k`	Deletes the text from the cursor to the end of the line and places it on the clipboard.
`<ctrl>l`	Clears the console.
`<ctrl>y`	Pastes any text that is on the clipboard.

4.5 Console on-line help

The console offers a simple help mechanism through the `?` function:

```
> (? 'list-ref)
list-ref list k
Returns the kth element of a list.
> (? "framebuffer-interpbox")
framebuffer-interpbox framebuffer x1 y1 x2 y2 color1 color2
color3 color4 Draws a box, but interpolates color values
between values specified at the four corners. This is
commonly used to produce smoothing effects. The color values
are assigned as (x1,y1)=color1, (x1,y2)=color2,
(x2,y1)=color3, and (x2,y2)=color4.
> (? '?)
? functionname
Displays help information regarding the named function, if
available. functionname must be a symbol or a string.
```

4.6 Console debugging and further reading

The console provides a number of mechanisms to facilitate debugging of interpreted procedures, including single stepping, break-points and tracing. These capabilities are documented in the Gambit-C manual, which contains a complete description of the console interface:
http://www.sensawave.com/manuals/gambit-c.pdf

Some users will notice that the Gambit-C facility for nested read-eval-print loops has been disabled by default in SENSAWAVE. This functionality can be activated by placing the line

```
nested-repl = true
```

in the SENSAWAVE configuration file `setup.ini`.

5 Scientific Repository

A large number of widely recognized high-quality scientific software libraries were developed by scientists at universities and research centers in the past decades. Many are available in repositories such as Netlib, but often in a form that requires significant effort to use. SENSAWAVE makes a large number of these libraries directly accessible as loadable modules.

At the time of writing the SENSAWAVE repository totals 1,581,626 lines of code providing 8,301 distinct function calls in 33 libraries:

ALFPACK, ARPACK, BESPAK, BLAS, CDFLIB, CMLIB, COULOMB, DATAPAC, DFFTPACK, DIFF, EISPACK, ELLIPGRID, FFTPACK, FNLIB, LAPACK, LINPACK, MINPACK, NSWCLIB, ODEPACK, QUADPACK, RANLIB, REGRIDPACK, SEISPACK, SLATEC, SMINPACK, SODEPACK, SPECFUN, STARPAC, STOEPLITZ, TOEPLITZ, TRANSFINITE, VFFTPACK, VFNLIB.

Instead of imposing yet another limited higher level application programming interface to these functions, SENSAWAVE aims at providing a very thin interface that follows the calling conventions of the original code, and provides access to all functions in the repository, including auxiliary routines.

The repository functions must be called with homogenous vector/matrix arguments of the type that matches the function call (`f64` for double precision real, `f32` for single precision real, etc.). Immediates can also be used, and are then automatically converted to a vector of the correct homogenous type of length one.

Function callbacks must be provided as a pair of the function and its number of arguments, and the callback must then extract the calling arguments from a list of pointers.

To avoid namespace contamination, the function names are prefixed with the name of the library (in capitals) followed by an underscore.

The following sections provide a number of examples intended to illustrate the use of the repository interface.

5.1 BLAS examples

Basic Linear Algebra Subprograms (BLAS) is the de facto application programming interface standard for performing basic linear algebra operations such as vector and matrix multiplication.

SENSAWAVE contains a precompiled BLAS library. The following code demonstrates access to the BLAS matrix library:

```
;; examples/sci-blas.scm
;; simple demostration of BLAS calls

;; load the BLAS module
(module-require 'BLAS)

;; scalar product, single precision
(define X (f32vector 1. 2. 3.))
(define Y (f32vector 7. 7. 7.))
(define N (f32vector-length X))
```

```
(for-each display (list
  "---\nBLAS sdot: " (BLAS_sdot N X 1 Y 1)
  "\nExpecting: 42.\n---\n"))

;; matrix multiply, double precision
(define A (listlist->f64matrix
  '((0.11 0.12 0.13)(0.21 0.22 0.23))))
(define B (listlist->f64matrix
  '((1011. 1012.)(1021. 1022.)(1031. 1032.))))
(define C (make-f64matrix 2 2))
(BLAS_dgemm 78 78 (f64matrix-rows A)
   (f64matrix-columns B) (f64matrix-columns A) 1.0
   A (f64matrix-rows A) B (f64matrix-columns A)
   0.0 C (f64matrix-rows A))
(display "BLAS_dgemm: ")
(pp (f64matrix->listlist C))
(display "Expecting: ((367.76 368.12) (674.06 674.72))\n---\n")

;; matrix-vector multiply, single precision complex
(define M (listlist->c32matrix
  '((3.0 1.0 3.0)(1. 5. 9.)(2. 6. 5.))))
(define X (listlist->c32matrix
   '((-1.0i)(-1.0i)(+1.0i))))
(define Y (make-c32matrix 3 1))
(BLAS_cgemv "N" (c32matrix-rows M) (c32matrix-columns M)
    1.0 M (c32matrix-rows M) X 1 0.0 Y 1)
(display "BLAS_cgemv: ")
(pp (c32matrix->listlist Y))
(display "Expecting: ((0.-1.i) (0.+3.i) (0.-3.i))\n---\n")

;; matrix-vector multiply, single precision complex
(define M (listlist->c64matrix
   '((3.0 1.0 3.0)(1. 5. 9.)(2. 6. 5.))))
(define X (listlist->c64matrix '((-1.0i)(-1.0i)(+1.0i))))
(define Y (make-c64matrix 3 1))
(BLAS_zgemv #\N (c64matrix-rows M) (c64matrix-columns M)
            1.0 M (c64matrix-rows M) X 1 0.0 Y 1)
(display "BLAS_zgemv: ")
(pp (c64matrix->listlist Y))
(display "Expecting: ((0.-1.i) (0.+3.i) (0.-3.i))\n---\n")
```

Try loading this program:

```
> (load "~~/examples/sci-blas.scm")
---
BLAS sdot: 42.
Expecting: 42.
```

```
---
BLAS_dgemm: ((367.76 368.12) (674.06 674.72))
Expecting: ((367.76 368.12) (674.06 674.72))
---
BLAS_cgemv: ((0.-1.i) (0.+3.i) (0.-3.i))
Expecting: ((0.-1.i) (0.+3.i) (0.-3.i))
---
BLAS_zgemv: ((0.-1.i) (0.+3.i) (0.-3.i))
Expecting: ((0.-1.i) (0.+3.i) (0.-3.i))
---
```

The following code illustrates a simple interface to the general BLAS multiplier `gemm` in all four precisions:

```
;; examples/sci-blas2.scm
;; interfacing the [s,d,c,z]gemm BLAS multipliers

(module-require 'BLAS)

;; generic wrapper for the gemm multipliers
(define (blas:multiply A B ll->m m->ll mmake gemm)
  (if (and (listlist? A) (listlist? B))
     (let ((a_r (length A))
           (a_c (length (car A)))
           (b_r (length B))
           (b_c (length (car B))))
       (if (= a_c b_r)
         (let ((ma (ll->m A))
               (mb (ll->m B))
               (mc (mmake a_r b_c 0.)))
           (gemm "N" "N" a_r b_c
              a_c 1.0 ma a_r mb a_c 0.0 mc a_r)
           (m->ll mc))
       (error "inner matrix dimension mismatch")))
    (error "arguments must be lists of lists")))

;; multipliers for all precisions
(define (blas-multiplyf32 A B)
  (blas:multiply A B listlist->f32matrix f32matrix->listlist
     make-f32matrix BLAS_sgemm))
(define (blas-multiplyf64 A B)
  (blas:multiply A B listlist->f64matrix f64matrix->listlist
     make-f64matrix BLAS_dgemm))
(define (blas-multiplyc32 A B)
  (blas:multiply A B listlist->c32matrix c32matrix->listlist
     make-c32matrix BLAS_cgemm))
(define (blas-multiplyc64 A B)
  (blas:multiply A B listlist->c64matrix c64matrix->listlist
     make-c64matrix BLAS_zgemm))

> (load "~~/examples/sci-blas2.scm")
> (blas-multiplyf32 '((1. 2. 3.)(4. 5. 6.)) '((7. 8.)(9. 10.)(11. 12.)))
((58. 64.) (139. 154.))
> (blas-multiplyc32 '((1. 2. 3.)(4. 5. 6.)) '((7. 8.)(9. 10.)(11. 12.)))
((58.+0.i 64.+0.i) (139.+0.i 154.+0.i))
> (blas-multiplyf64 '((1. 2. 3.)(4. 5. 6.)) '((7. 8.)(9. 10.)(11. 12.)))
((58. 64.) (139. 154.))
> (blas-multiplyc64 '((1. 2. 3.)(4. 5. 6.)) '((7. 8.)(9. 10.)(11. 12.)))
((58.+0.i 64.+0.i) (139.+0.i 154.+0.i))
```

5.2 LAPACK examples

The Linear Algebra PACKage, LAPACK, is a software library providing routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, Householder transformation to implement QR decomposition on a matrix and singular value problems.

The following code demonstrates access to the precompiled LAPACK library distributed with SENSAWAVE:

```
;; examples/sci-lapack.scm
;; simple LAPACK demonstration

(module-require 'LAPACK)

;; gaussian elimination - double precision
(define M (listlist->f64matrix
  '((3. 1. 3.)(1. 5. 9.)(2. 6. 5.))))
(define X (listlist->f64matrix '((-1.)(3.)(-3.))))
(define P (make-s32vector (f64matrix-rows M) 0))
(LAPACK_dgesv (f64matrix-rows M) (f64matrix-columns X)
   M (f64matrix-rows M) P X (f64matrix-rows X) 0)
(display "---\nLAPACK_dgesv: ")
(pp (f64matrix->listlist X))
(display "Expecting: ((-1.)(-1.)(1.))\n---\n")

;; tridiagonal gaussian elimination - double precision
;; we solve the system
;;  / 2 -1 0 0 0 \    / x0 \   / 1 \
;; | -1 2 -1 0 0 |    | x1 |   | 2 |
;; | 0 -2 3 -1 0 | *  | x2 | = | 3 |
;; | 0 0 -1 3 -2 |    | x3 |   | 2 |
;;  \ 0 0 0 -1 1 /    \ x4 /   \ 1 /
(define l (f64vector -1. -2. -1. -1.))
(define d (f64vector 2. 2. 3. 3. 1.))
(define u (f64vector -1. -1. -1. -2.))
(define x (f64vector 1. 2. 3. 2. 1.))
```

```
(LAPACK_dgtsv 5 1 l d u x 5 0)
(display "LAPACK_dgtsv: ")
(pp (f64vector->list x))
(display "Expected: (6.5 12.0 15.5 19.5 20.5)\n---\n")

;; auxiliary division - double complex function
(define z (LAPACK_zladiv 10.0+0.1i 2.0-0.1i))
(display (format "LAPACK_zladiv: ~F +i ~F\n"
  (real-part z) (imag-part z)))
(display (format "Expecting: ~F +i ~F\n---\n"
  (real-part (/ 10.0+0.1i 2.0-0.1i))
  (imag-part (/ 10.0+0.1i 2.0-0.1i))))
```

Loading this program gives:

```
> (load "~~/examples/sci-lapack")
---
LAPACK_dgesv: ((-1.) (-1.0000000000000002) (1.))
Expecting: ((-1.)(-1.)(1.))
---
LAPACK_dgtsv: (6.5 12. 15.5 19.5 20.5)
Expected: (6.5 12.0 15.5 19.5 20.5)
---
LAPACK_zladiv: 4.985037406483791 +i 0.29925187032418954
Expecting: 4.985037406483791 +i 0.29925187032418954
---
```

The following code illustrates a simple interface to the general LAPACK solver `gesv` in all four precisions:

```
;; examples/sci-lapack2.scm
;; interfacing the LAPACK [s,d,c,z]gesv solvers

(module-require 'LAPACK)

;; generic wrapper for the gesv solvers
(define (lapack:solve A B ll->m m->ll gesv)
  (if (and (listlist? A) (listlist? B))
    (let ((n (length (car A)))
          (n2 (length A))
          (n3 (length B))
          (bnrsh (length (car B))))
      (if (= n n2 n3)
         (let ((ma (ll->m A))
               (mb (ll->m B))
               (mp (make-s32vector n 0)))
          (gesv n bnrsh ma n mp mb n 0)
          (m->ll mb))
         (error "illegal array dimensions")))
```

```
   (error "arguments must be lists of lists")))

;; the specific solvers for all precisions
(define (lapack-solvef32 A B)
  (lapack:solve A B listlist->f32matrix
     f32matrix->listlist LAPACK_sgesv))
(define (lapack-solvef64 A B)
  (lapack:solve A B listlist->f64matrix
     f64matrix->listlist LAPACK_dgesv))
(define (lapack-solvec32 A B)
  (lapack:solve A B listlist->c32matrix
     c32matrix->listlist LAPACK_cgesv))
(define (lapack-solvec64 A B)
  (lapack:solve A B listlist->c64matrix
     c64matrix->listlist LAPACK_zgesv))
```

The same can be accomplished in C/Java style:

```
// examples/sci-lapack2.six
// interfacing the LAPACK [s,d,c,z]gesv solvers

module_require("LAPACK");

// generic wrapper for the *gesv solvers
obj _lapack_solve(obj A, obj B, obj llm, obj mll, obj gesv)
{
  if (listlist_qm(A)&&listlist_qm(B)) {
    int n = length(car(A));
    int n2 = length(A);
    int n3 = length(B);
    int bnrsh = length(car(B));
    if (n==n2&&n==n3) {
      obj ma = llm(A);
      obj mb = llm(B);
      obj mp = make_s32vector(n,0);
      gesv(n, bnrsh, ma, n, mp, mb, n, 0);
      mll(mb);
    } else {
      error("illegal array dimensions");
    }
  } else {
   error("arguments must be lists of lists");
  }
}

// the specific solvers for all precisions
obj lapack_solvef32(obj A, obj B) {
  _lapack_solve(A, B, listlist_to_f32matrix,
     f32matrix_to_listlist, LAPACK_sgesv);
}
obj lapack_solvef64(obj A, obj B) {
  _lapack_solve(A, B, listlist_to_f64matrix,
     f64matrix_to_listlist, LAPACK_dgesv);
}
obj lapack_solvec32(obj A, obj B) {
  _lapack_solve(A, B, listlist_to_c32matrix,
     c32matrix_to_listlist, LAPACK_cgesv);
}
obj lapack_solvec64(obj A, obj B) {
  _lapack_solve(A, B, listlist_to_c64matrix,
     c64matrix_to_listlist, LAPACK_zgesv);
}
```

```
> (load "~~/examples/sci-lapack2.six")
> (lapack_solvef32 '((3. 1. 3.)(1. 5. 9.)(2. 6. 5.)) '((-1.)(3.)(-3.)))
((-1.) (-.9999999403953552) (1.))
> (lapack_solvef64 '((3. 1. 3.)(1. 5. 9.)(2. 6. 5.)) '((-1.)(3.)(-3.)))
((-1.) (-1.0000000000000002) (1.))
> (lapack_solvec32 '((3. 1. 3.)(1. 5. 9.)(2. 6. 5.)) '((-1.)(3.)(-3.)))
((-1.+0.i) (-.9999999403953552+0.i) (1.+0.i))
> (lapack_solvec64 '((3. 1. 3.)(1. 5. 9.)(2. 6. 5.)) '((-1.)(3.)(-3.)))
((-1.+0.i) (-1.0000000000000002+0.i) (1.+0.i))
```

5.3 SLATEC examples

SLATEC is a comprehensive scientific software library containing over 1400 general purpose mathematical and statistical routines.

This is a simple example of accessing this library from within SENSAWAVE:

```
;; examples/sci-slatec.scm
;; simple SLATEC examples

(module-require 'SLATEC)

;; cube root - single precision
(display (format "---\nSLATEC_cbrt: ~F\n" (SLATEC_cbrt 10.) ))
(display (format "Expecting: ~F\n---\n" (expt 10. (/ 1. 3.))))

;; cube root - double precision
(display (format "SLATEC_dcbrt: ~F\n" (SLATEC_dcbrt 10.) ))
(display (format "Expecting: ~F\n---\n" (expt 10. (/ 1. 3.))))

;; cube root - complex single precision
(display (format "SLATEC_ccbrt: ~F\n" (real-part (SLATEC_ccbrt 10.)) ))
(display (format "Expecting: ~F\n---\n" (expt 10. (/ 1. 3.))))
```

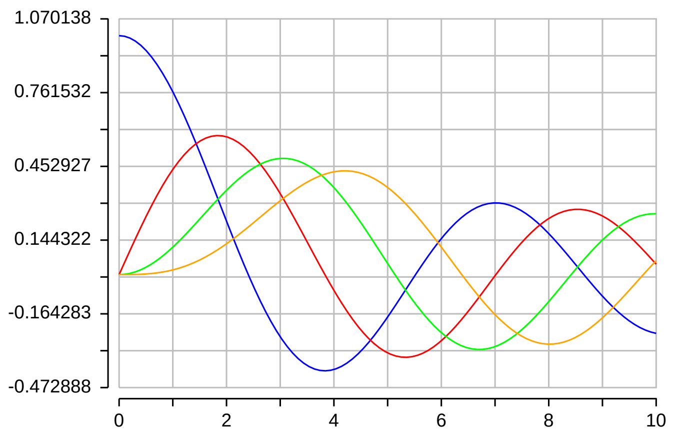
```
;; bessel function of first kind - double precision
(define (J n x)
  (let ((Y (f64vector 0.)))
    (SLATEC_dbesj x n 1 Y 0)
    (f64vector-ref Y 0)))

;; plot the bessel functions of order 0,1,2,3
(define x (: 0 10 100))
(define y1 (map (lambda (x) (make-rectangular
    (J 0 x) (J 1 x))) x))
(define y2 (map (lambda (x) (make-rectangular
    (J 2 x) (J 3 x))) x))
(ezgraph x y1 x y2)
```

Loading this code results in:

```
> (load "~~/examples/sci-slatec.scm")
SLATEC_cbrt: 2.1544346809387207
Expecting: 2.154434690031884
---
SLATEC_dcbrt: 2.154434690031884
Expecting: 2.154434690031884
---
SLATEC_ccbrt: 2.1544346809387207
Expecting: 2.154434690031884
---
```

Note that the precision of the single precision cube root results are slightly different than the reference SENSAWAVE calculation due to the limited resolution.

The code also draws the bessel function of the first kind of orders 0, 1, 2 and 3:



5.4 QUADPACK examples

QUADPACK is a standard library for the numerical computation of definite one-dimensional integrals.

The following code demonstrates using QUADPACK with the SENSAWAVE function callback mechanism:

```
;; examples/sci-quadpack.scm
;; simple demonstration of QUADPACK calls

(module-require 'QUADPACK)

;; our function callback - cos(100*sin(x))
(define (function xp)
  (let* ((xv (ptr->f64vector xp 1))
         (x  (f64vector-ref xv 0)))
    (cos (* 100. (sin x)))))

;; numerical integration from 0 to Pi
(let* ((result (f64vector 0.)) ;; this will hold our result
       (limit 1000)
       (lenw  (* 4 limit))
       (iwork (make-s32vector limit))
       (work  (make-f64vector lenw)))
  (QUADPACK_dqag (cons function 1) 0. M_PI
       0. 0.001 6 result 0. 0 0 limit lenw 0 iwork work)
  (for-each display (list "---\n"
     (format "QUADPACK_dqag: ~10F\n" (f64vector-ref result 0))
     "Expecting: 0.06278740\n" )))

;; our function to integrate - log(x)/(1+100*x*x)
(define (function2 xp)
  (let* ((xv (ptr->f64vector xp 1))
         (x  (f64vector-ref xv 0)))
    (/ (log x) (+ 1. (* 100. x x)))))

;; numerical integration from 0 to infinity
(let* ((result (f64vector 0.)) ;; this will hold our result
       (limit 1000)
       (lenw  (* 4 limit))
       (iwork (make-s32vector limit))
       (work  (make-f64vector lenw)))
  (QUADPACK_dqagi (cons function2 1) 0. 1 0. 0.001
       result 0. 0 0 limit lenw 0 iwork work)
  (for-each display (list "---\n"
    (format "QUADPACK_dqagi:  ~10F\n" (f64vector-ref result 0))
     "Expecting: -0.3616892\n"
     "---\n")))
```

Loading this code results in:

```
> (load "~~/examples/sci-quadpack")
---
QUADPACK_dqag: 0.06278740
Expecting: 0.06278740
---
QUADPACK_dqagi:  -0.3616892
Expecting: -0.3616892
---
```

5.5 NSWCLIB examples

The Naval Surface Warfare Center mathematical library, NSWCLIB, is an extensive library of high quality mathematical subroutines. The example below calls functions for finding a function zero in this library. It provides another demonstration of the callback mechanism that allows a precompiled library module to use interpreted SENSAWAVE functions:

```
;; examples/sci-nswc.scm
;; simple NSWCLIB example

(module-require 'NSWCLIB)

;; single precision function callback
(define (sfunction xp)
  (let* ((xv (ptr->f32vector xp 1))
         (x  (f32vector-ref xv 0)))
    (- x 0.5)))

;; double precision function callback
(define (dfunction xp)
  (let* ((xv (ptr->f64vector xp 1))
         (x  (f64vector-ref xv 0)))
    (- x 0.5)))

;; find a zero of the function
(define sresult (NSWCLIB_zeroin
  (cons sfunction 1) 0. 1. 1e-3 1e-3))
(define dresult (NSWCLIB_dzero
  (cons dfunction 1) 0. 1. 1e-3 1e-3))

(for-each display (list "---\n"
  (format "NSWCLIB_zeroin: ~F\n" sresult)
  (format "NSWCLIB_dzero: ~F\n" dresult)
  "Expected: " 0.5 "\n---\n"))
```

Loading this code results in:

```
> (load "~~/examples/sci-nswclib")
---
NSWCLIB_zeroin: 0.5
NSWCLIB_dzero: 0.5
Expected: .5
---
```

5.6 Other packages in the repository

The following is a non-exhaustive summary of the capabilities of some of the other libraries in the SENSAWAVE repository.

5.6.1 ARPACK

ARPACK is a library of routines for solving large scale eigenvalue problems, designed to compute a few eigenvalues and corresponding eigenvectors of a general matrix. It is based upon an algorithmic variant of the Arnoldi process called the Implicitly Restarted Arnoldi Method. ARPACK is capable of solving large scale symmetric, non-symmetric, and generalized eigen problems.

5.6.2 CDFLIB

CDFLIB contains routines to compute cumulative distribution functions, inverses, and parameters of the distribution for the statistical distributions Beta, Binomial, Chi-square, Noncentral, Chi-square, F, Noncentral F, Gamma, Negative Binomial, Normal, Poisson and Student's t.

5.6.3 FFTPACK and DFFTPACK

FFTPACK (single precision) and DFFTPACK (double precision) are libraries for fast Fourier transform of periodic and other symmetric sequences. They include complex, real, sine, cosine, and quarter-wave transforms.

5.6.4 EISPACK and SEISPACK

EISPACK (double precision) and SEISPACK (single precision) are libraries of routines that compute the eigenvalues and eigenvectors of nine classes of matrices: complex general, complex Hermitian, real general, real symmetric, real symmetric banded, real symmetric tridiagonal, special real tridiagonal, generalized real, and generalized real symmetric matrices. In addition, routines are included that use singular value decomposition to solve certain least-squares problems.

5.6.5 LINPACK

LINPACK is a library of routines that analyze and solve linear equations and linear least-squares problems. LINPACK solves linear systems whose matrices are general, banded, symmetric indefinite, symmetric positive definite, triangular, and tridiagonal square. In addition, the library computes the QR and singular value decompositions of rectangular matrices and applies them to least-squares problems.

5.6.6 MINPACK

MINPACK is a library of routines for solving nonlinear equations and nonlinear least squares problems. The algorithms proceed either from an analytic specification of the Jacobian matrix or directly from the problem functions. MINPACK includes facilities for systems of equations with a banded Jacobian matrix, for least squares problems with a large amount of data, and for checking the consistency of the Jacobian matrix with the functions.

5.6.7 ODEPACK and SODEPACK

ODEPACK (double precision) and SODEPACK (single precision) are collections of solvers for the initial value problem for ordinary differential equation systems. They provide nine solvers suitable for both stiff and non-stiff systems. Systems can be given in explicit or linearly implicit form.

5.6.8 RANLIB

RANLIB is a library of routines for random number generation. The bottom level routines provide 32 virtual random number generators. Using this base, routines are provided that return Beta random deviates, Chi-square random deviates, exponential random deviates, F random deviates, Gamma random deviates, Multivariate normal random deviates (mean and covariance matrix specified), noncentral chi-square random deviates, noncentral F random deviates, univariate normal random deviates, random permutations of an integer array, real uniform random deviates between specified limits, binomial random deviates, negative Binomial random deviates, multinomial random deviates, Poisson random deviates and integer uniform deviates between specified limits.

5.6.9 REGRIDPACK

REGRIDPACK is a library of routines for interpolating values between one-, two-, three-, and four-dimensional arrays defined on uniform or nonuniform orthogonal grids, an operation commonly referred to as "regridding." Linear or cubic interpolation can be selected independently in each dimension.

5.6.10 STARPAC

STARPAC is library of routines for statistical data analysis from the National Institute for Standards and Technology (NIST). STARPAC is an acronym for Standards Time Series and Regression Package, and contains computational routines for normal random number generation, univariate sampling, one-way analysis of variance, correlation analysis, linear least squares, nonlinear least squares, digital filtering, complex demodulation, correlation and spectrum analysis, and time series analysis.

5.6.11 TOEPLITZ and STOEPLITZ

TOEPLITZ (double precision) and STOEPLITZ (single precision) are libraries of routines for linear systems of Toeplitz or circulant form , and for orthogonal factorization of column-circulant matrices.

5.7 Further reading

Many of the libraries are best documented in the Fortran source code. For this reason the subdirectory `repository` of the SENSAWAVE directory contains full source code for the entire SENSAWAVE scientific repository.

6 Fonts and Colors

6.1 Fonts

SENSAWAVE provides a font system designed to generate identical output on all supported raster and vector rendering backends, including PDF, EPS, PNG, PPM and OpenGL. The supported font faces are `FONT_MONOSPACE`, `FONT_SANSSERIF`, `FONT_SERIF` and `FONT_SYMBOL`. Each font except the latter is available in four different styles: `FONT_NORMAL`, `FONT_BOLD`, `FONT_ITALIC`, `FONT_BOLDITALIC`. The supported font sizes are `FONT_8PT`, `FONT_10PT`, `FONT_12PT`, `FONT_14PT`, `FONT_18PT` and `FONT_24PT`.

A `font-new` call is used to create a font object, and `font?`, `font-name` and `font-size` can be used to query a font object:

```
> (define f (font-new FONT_14PT))
> (font? f)
#t
> (font-name f)
"Helvetica"
> (font-size f)
14
```

The name returned by `font-name` is the one used for rendering on PDF and EPS backends.

The `font-stringascent` `font-stringdescent` and `font-stringwidth` can be used to calculate string dimensions for a given font:

```
> (font-stringascent f "a")
8
> (font-stringdescent f "p")
-3
> (font-stringwidth f "Hello World")
75
```

Unlike normal scheme objects, a font object must be explicitly deleted to free its storage:

```
> (font-delete f)
```

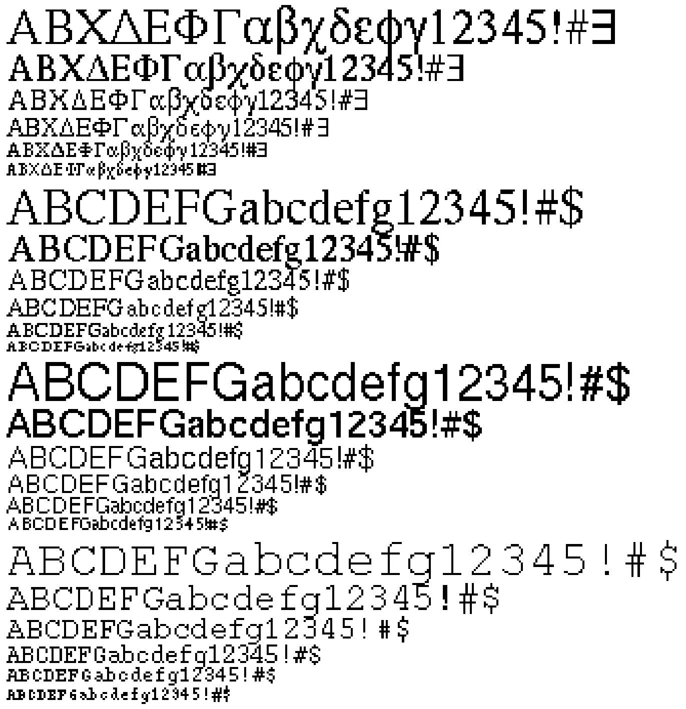
The following example renders some of the supported fonts in a framebuffer:

```
;; examples/fonts.scm
;; displays loadable fonts in a framebuffer window

;; routine to render a font
(define (show-font fbuf face points y)
  (let* ((label "ABCDEFGabcdefg12345!#$")
         (fnt  (font-new points face))
         (dy (- (font-stringascent fnt label)
                (font-stringdescent fnt label))))
    (framebuffer-text fbuf 20 y Black fnt label HORIZONTAL)
    (font-delete fnt)
    (+ y dy)))
```

```
(define f (framebuffer-new 375 385))
(framebuffer-clear f White)
(let ((faces (list FONT_MONOSPACE FONT_SANSSERIF
                   FONT_SERIF FONT_SYMBOL))
      (points (list FONT_8PT FONT_10PT FONT_12PT
                    FONT_14PT FONT_18PT FONT_24PT)))
 (let loop ((fs faces)(ps points)(y 20))
   (if (> (length fs) 0)
     (let* ((newpoints (if (= (length ps) 1) points (cdr ps)))
            (newfaces (if (= (length newpoints) 6) (cdr fs) fs))
            (newy (show-font f (car fs) (car ps) y))
            )
        (loop newfaces newpoints newy)))))

(framebuffer->glwindow f #t)
```

Loading this example:

```
> (load "~~/examples/fonts")
```



6.2 Colors

Colors are represented as 32bit RGBA values in SENSAWAVE, and a large number of colors are predefined, following the color naming of the X11 graphics system. `color-red`, `color-green`, `color-blue` and `color-alpha` can be used to extract the channels of a given color:

```
> Red
4278190335
> (color-red Red)
255
> (color-blue Red)
0
```

`color-rgb`, `color-rgba`, `color-rgbf` and `color-rgbaf` can be used to build custom colors:

```
> (color-rgb #xff #x00 #x00)
4278190335
> (color-rgbf 1.0 0.0 0.0)
4278190335
```

SENSAWAVE also supports color gradients, which is often useful for data visualization.

```
> (color-gradient GRADIENT_GRAY 0.)
4278190080
> (color-gradient GRADIENT_GRAY 1.)
4294967295
```

The supported gradients are `GRADIENT_GRAY`, `GRADIENT_RAINBOW`, `GRADIENT_THERMAL` and `GRADIENT_COPPER`.

7 Framebuffers

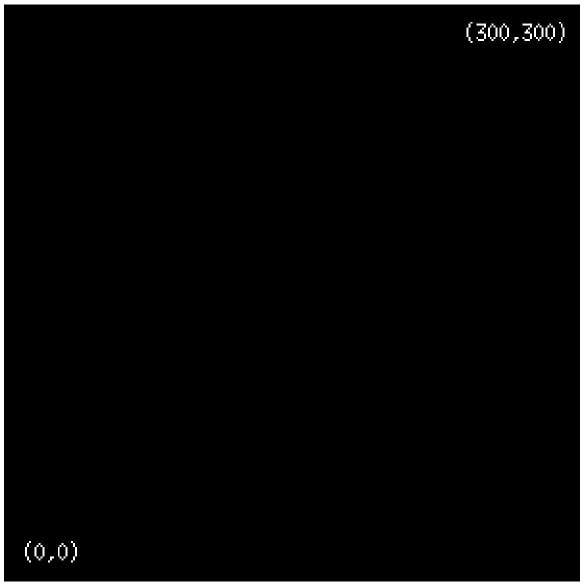
A SENSAWAVE framebuffer is a device independent raster device that can be used to create graphical images.

7.1 Framebuffer fundamentals

We willcreate a framebuffer with a 300x300 pixel array using the command:

```
> (define fb (framebuffer-new  300 300))
```

`fb` is now pointing to a new framebuffer. The framebuffer is organized with the point (0,0) corresponding to the lower left corner and (300,300) corresponding to the upper right corner as shown here:



We can do operations on the framebuffer. For example:

```
> (framebuffer-clear fb Black)
> (framebuffer-solidbox fb 50 50 250 250 Blue)
> (framebuffer-text fb 105 150 White (font-new)
     "HELLO WORLD" HORIZONTAL)
```

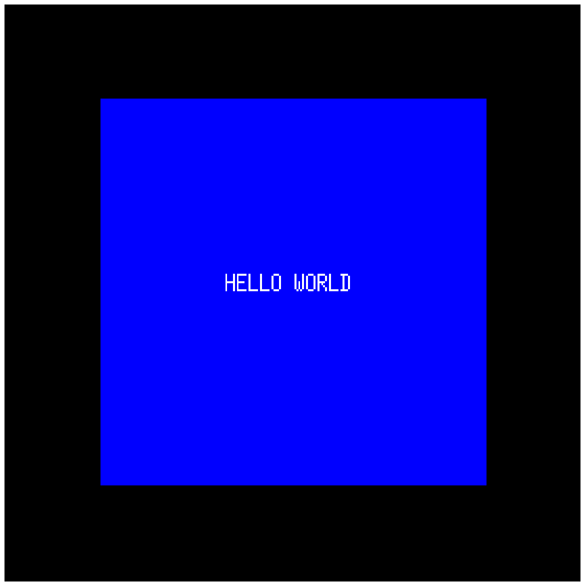
This clears the framebuffer to black, draws a solid blue rectangle, and finally writes a message in white color. The resulting image can now for example be written to a PNG image file with the command:

```
> (framebuffer-savePNG fb "helloworld.png")
```

We can also display the framebuffer on screen:

```
> (framebuffer->glwindow fb)
```

The resulting image looks like this:



The framebuffer framework supports a wide range of graphical operations listed in the reference section of this manual. Most applications however benefit by accessing the framebuffer through the 2D or 3D abstractions that are described by example in the two following sections.

7.2 Example: 2D Plotting Primitives

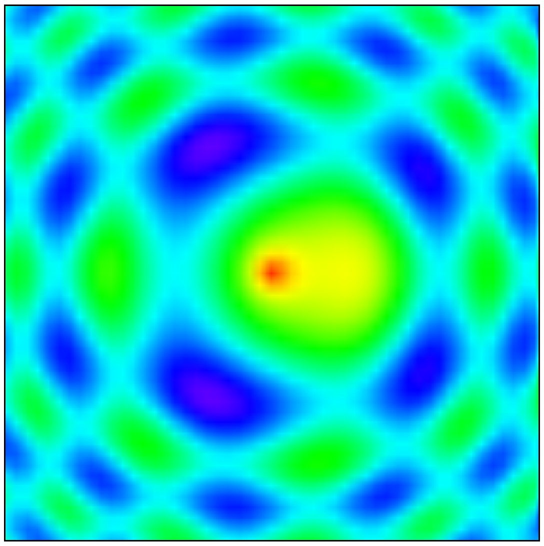
SENSAWAVE provides higher level functions for drawing 2D graphics primitives in a framebuffer. These functions provide scaling to real-world coordinates. The following example draws a 2D function map using color interpolated rectangles:

```
;; examples/plot2d.scm
;; simple example of 2D framebuffer primitives

;; function to render
(define func (lambda (x y)
  (let ((r (sqrt (+ (* x x) (* y y)))))
    (/ (+ (sin (* 0.3 r x)) (cos (* 0.3 r y))) (+ 1. r)))))

;; draw the framebuffer image
(define f (framebuffer-new 400 400))
(framebuffer-clear f White)
(define p2 (plot2d-new f -6.3 -6.3 6.3 6.3))
(plot2d-setview p2 15 15 385 385)
(framebuffer-noclip f)
(framebuffer-box f 14 14 385 385 Black)
(plot2d-start p2)
(plot2d-clear p2 Black)
```

```
(let ((dx 0.25)(dy 0.25))
  (let loop ((x -6.3)(y -6.3))
    (if (< x 6.3)
       (let* ((z1 (func x y))
              (z2 (func (+ x dx) y))
              (z3 (func (+ x dx) (+ y dy)))
              (z4 (func x (+ y dy)))
              (newy (if (> y 6.3) -6.3 (+ y dy)))
              (newx (if (= newy -6.3) (+ x dx) x)))
         (plot2d-interpbox p2 x y (+ x dx) (+ y dy)
                 (color-gradient GRADIENT_RAINBOW (/ (+ z1 1.0) 2.0))
                 (color-gradient GRADIENT_RAINBOW (/ (+ z4 1.0) 2.0))
                 (color-gradient GRADIENT_RAINBOW (/ (+ z2 1.0) 2.0))
                 (color-gradient GRADIENT_RAINBOW (/ (+ z3 1.0) 2.0)))
           (loop newx newy)))))

;; render the framebuffer to a window
(plot2d->glwindow p2 #t)
```

This program opens a window with the framebuffer graphics:

```
> (load "~~/examples/plot2d")
```



7.3 Example: 3D plotting primitives

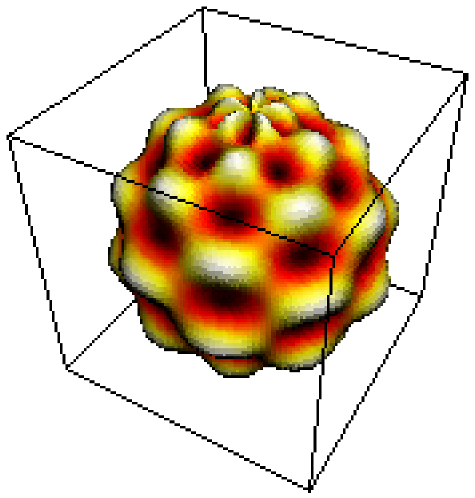
Framebuffer devices also support simplified 3D primitives. The following example builds a 3D spherical plot with shading and color interpolation:

```
;; examples/plot3d.scm
;; Examples of 3D primitives on framebuffers
;; function to render
(define (func u v)
  (+ 1. (* 0.1 (sin (* 5. v))
     (sin (* 10. u)))))

;; setup the rendering
(define f (framebuffer-new 200 200))
(framebuffer-clear f White)
(define p3 (plot3d-new f -1.5 -1.5 -1.5 1.5 1.5 1.5))
(plot3d-setview p3 0 0 199 199)
(plot3d-lookat p3 5.5)
(plot3d-autoperspective p3 40.)
(framebuffer-noclip f)
(plot3d-start p3)
(plot3d-clear p3 White)
(plot3d-rotu p3 50.)
(plot3d-rotz p3 60.)
(plot3d-up p3 4.)
(plot3d-right p3 2.)
(plot3d-lightat -0.5 -0.5 1.)

;; draw the axis frame
(plot3d-quad p3 -1.1 -1.1 -1.1
     1.1 -1.1 -1.1 1.1  1.1 -1.1 -1.1  1.1 -1.1 Black)
(plot3d-quad p3 -1.1 -1.1 1.1  1.1 -1.1 1.1 1.1
     1.1 1.1 -1.1  1.1 1.1 Black)
(plot3d-line p3 -1.1 -1.1 -1.1 -1.1 -1.1 1.1 Black)
(plot3d-line p3 1.1 -1.1 -1.1 1.1 -1.1 1.1 Black)
(plot3d-line p3 1.1 1.1 -1.1 1.1 1.1 1.1 Black)
(plot3d-line p3 -1.1 1.1 -1.1 -1.1 1.1 1.1 Black)

;; now draw the function
(let ((du 0.04)(dv 0.04))
  (let loop ((u 0.)(v 0.))
    (if (< u M_PI)
       (let* ((r1 (func u v))
              (x1 (* r1 (sin u) (cos v)))
              (y1 (* r1 (sin u) (sin v)))
              (z1 (* r1 (cos u)))
              (r2 (func (+ u du) v))
              (x2 (* r2 (sin (+ u du)) (cos v)))
              (y2 (* r2 (sin (+ u du)) (sin v)))
              (z2 (* r2 (cos (+ u du))))
              (r3 (func (+ u du) (+ v dv)))
              (x3 (* r3 (sin (+ u du)) (cos (+ v dv))))
              (y3 (* r3 (sin (+ u du)) (sin (+ v dv))))
              (z3 (* r3 (cos (+ u du))))
```

```
                 (r4 (func u (+ v dv)))
                 (x4 (* r4 (sin u) (cos (+ v dv))))
                 (y4 (* r4 (sin u) (sin (+ v dv))))
                 (z4 (* r4 (cos u)))
                 (c1 (color-gradient GRADIENT_THERMAL
                    (* (- r1 0.9) 5.0)))
                 (c2 (color-gradient GRADIENT_THERMAL
                    (* (- r2 0.9) 5.0)))
                 (c3 (color-gradient GRADIENT_THERMAL
                    (* (- r3 0.9) 5.0)))
                 (c4 (color-gradient GRADIENT_THERMAL
                    (* (- r4 0.9) 5.0)))
                 (newv (if (> v (* 2.0 M_PI)) 0. (+ v dv)))
                 (newu (if (= newv 0.) (+ u du) u)))
            (plot3d-shadedinterpquad p3 x1 y1 z1 c1 x2 y2 z2
                                     c2 x3 y3 z3 c3 x4 y4 z4 c4)
            (loop newu newv)))))

;; open the the graph in a window
(plot3d->glwindow p3 #t)
```

This program opens a window with the framebuffer graphics:

```
> (load "~~/examples/plot3d.scm")
```



8 Graphs

SENSAWAVE contains a system for creating high quality scientific graphics. It supports multiple output formats, including Encapsulated postscript (EPS) and PDF, PNG, PPM, as well as unscreen rendering either through framebuffers or direct OpenGL. The purpose of the library is to facilitate generation of complex graphs directly from SENSAWAVE. It is particular useful for producing many graphs of a similar format for a large number of data sets quickly and with minimum human interventions. For this type of situation, the standard approach of transferring data from the analysis program to spread sheet, and then to menu driven commercial graphics packages is not practical.

The graphing system can generate output suited for both on-screen display and final printing from the same graph definition. This provides a mechanism for fast interactive development of publication quality material, and also allows graphs to be build into interactive applications.

We will demonstrate the graphing system through four examples, a linear plot, semi-logarithmic plot, double-logarithmic plot and a 2D image map. The source code for these examples are included with the SENSAWAVE distribution.

8.1 Example 1: Linear graph

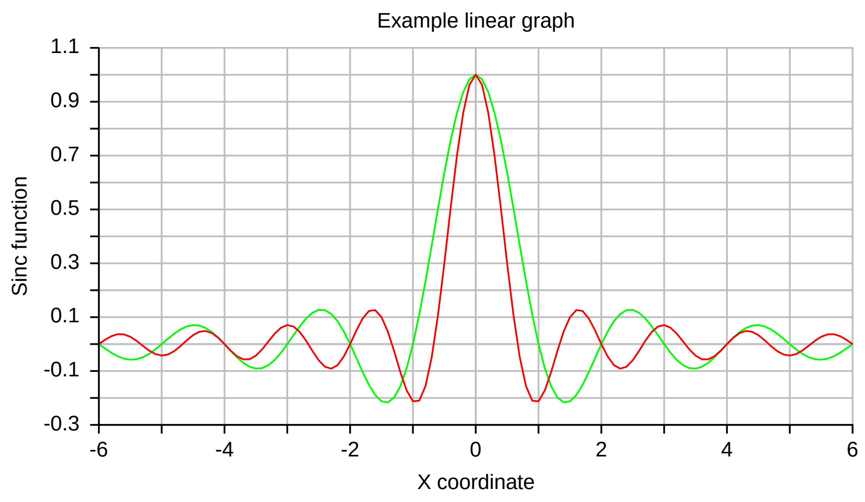
The following is an example of a linear graph:

```
;; examples/graph-linear.scm
;; example of a linear graph

;; create the graph
(define g (graph-new 550 320))

;; select the font
(graph-font g (font-new))

;; set the origin
(graph-aorigin g 1.2 0.9)

;; set the style of the log axis
(graph-linearstyle g 4 0 5 1 5)

;; initialize the x axis
;; 6 inches wide, from -6 to 6, offset 0 inch,
;; tick every 1., label every 2 tick
(graph-xlinear g 6.0 -6. 6. 0. 1. 2)

;; initialize the y axis
;; 3 inches high, from -0.3 to 1.1, offset 0. inch,
;; tick every .1, label every 2 tick
(graph-ylinear g 3.0 -0.3 1.1 0.0 0.1 2)

;; draw a domain mesh
```

```
(graph-color g Grey)
(graph-moveto g -6.0 0.) (graph-lineto g 6.0 0.)
(graph-moveto g 0. -0.3) (graph-lineto g 0 1.1)
(graph-stroke g)
(graph-mesh g)

;; draw the sinc function curves
(graph-color g Green)
(graph-moveto g -6. (sinc -6.))
(let loop ((x -6.))
    (if (< x 6.) (begin
      (graph-lineto g x (sinc x))
      (loop (+ x 0.1)))))
(graph-stroke g)
(graph-color g Red)
(graph-moveto g -6. (sinc -6.))
(let loop ((x -6.))
    (if (< x 6.) (begin
      (graph-lineto g x (sinc (* 1.5 x)))
      (loop (+ x 0.1)))))
(graph-stroke g)

;; draw the x and y axis
(graph-color g Black)
(graph-xaxis g)
(graph-yaxis g)

;; draw the axis labels
(graph-ylabel g "Sinc function")
(graph-xlabel g "X coordinate")

;; draw a caption in physical coordinates
(graph-setcoord g GRAPH_PHYS)
(graph-htextcenter g 3. 3.2 "Example linear graph")

;; uncomment to output the graph as EPS, PDF, PNG, PPM
;;(graph-output g GRAPH_EPS "graph-sinc.eps")
;;(graph-output g GRAPH_PDF "graph-sinc.pdf")
;;(graph-output g GRAPH_PNG "graph-sinc.png")
;;(graph-output g GRAPH_PPM "graph-sinc.ppm")

;; show the graph on screen
(graph-output g GRAPH_WIN)
```

This produces the following:

```
> (load "~~/examples/graph-linear")
```



8.2 Example 2: Semi-logarithmic graph

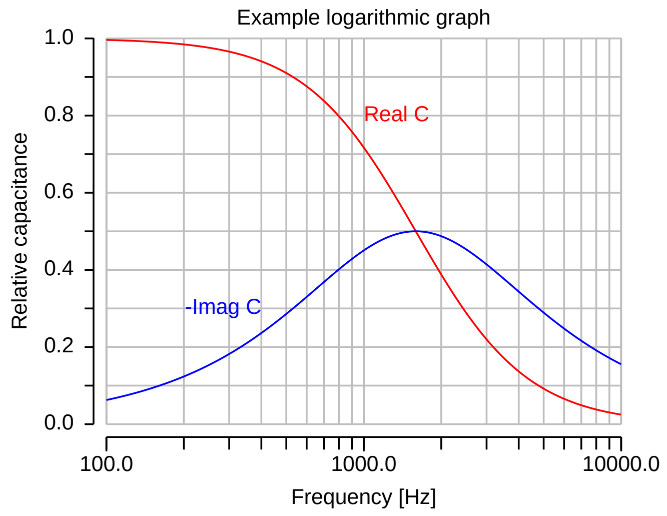
The following is an example of a semi-logarithmic graph:

```
;; examples/graph-semilog.scm
;; example of a semi-logarithmic graph

;; create the graph
(define g (graph-new 430 320))

;; set the origin
(graph-aorigin g 1.2 0.9)

;; select the font
(graph-font g (font-new))

;; set the style of the axis
(graph-linearstyle g 4 0 5 1 5)
(graph-logstyle g 2 1022 4 1 7 8)

;; initialize the x and y axis
;; 4 inches wide, from 100 to 10000 Hz, offset -0.1 inch
(graph-xlog g 4.0 100. 10000. -0.1)
;; 3 inches high, from 0 to 1, offset -0.1 inch,
(graph-ylinear g 3.0 0.0 1.0 -0.1 0.1 2)

;; draw a domain mesh
(graph-color g Grey)
(graph-mesh g)
```

```
;; render the capacitance spetrum
(let* ((R 1.0e6)(C 1e-10)
      (cnorm (lambda (f) (/ 1. (+ 1 (* +1.0i M_2PI f R C))))))
  (graph-color g Red)
  (graph-moveto g 100. (real-part (cnorm 100.)))
  (let loop ((f 100.))
    (if (< f 10000.) (begin
      (graph-lineto g f (real-part (cnorm f)))
      (loop (+ f 10.)))))
  (graph-stroke g)
  (graph-color g Blue)
  (graph-moveto g 100. (- (imag-part (cnorm 100.))))
  (let loop ((f 100.))
    (if (< f 10000.) (begin
      (graph-lineto g f (- (imag-part (cnorm f))))
      (loop (+ f 10.)))))
  (graph-stroke g)
)

;; draw the x and y axis
(graph-color g Black)
(graph-xaxis g)
(graph-yaxis g)

;; draw the axis labels
(graph-ylabel g "Relative capacitance")
(graph-xlabel g "Frequency [Hz]")

;; draw labels to identify real and imaginary curves
(graph-color g Red)
(graph-htextleft g 1000. 0.8 "Real C")
(graph-color g Blue)
(graph-htextright g 400 0.3 "-Imag C")

;; draw a title
(graph-color g Black)
(graph-htextcenter g 1000 1.05 "Example logarithmic graph")

;; show the graph on screen
(graph-output g GRAPH_WIN)
```

This produces the following graph:

```
> (load "~~/examples/graph-semilog")
```



8.3 Example 3: Double-logarithmic graph

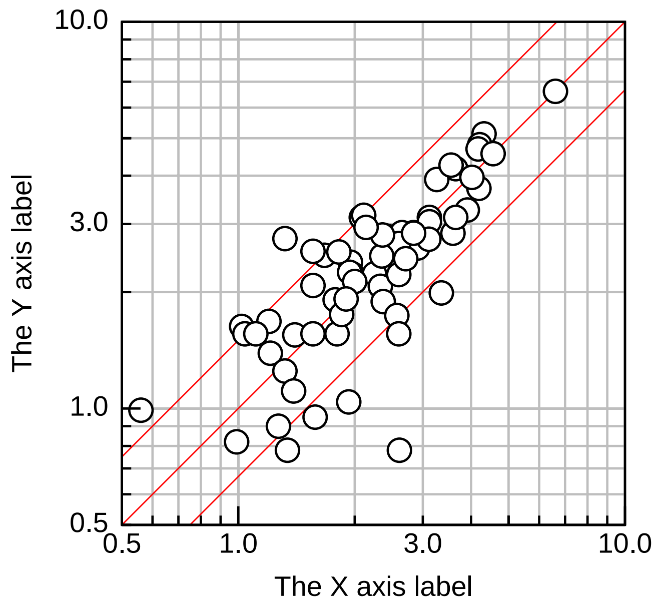
```
;; examples/graph-loglog.scm
;; Example of a double logarithmic graph
;; Adopted from an original Cgraph example

;; the data to plot
(define data '((1.95 2.39) (0.99 0.82) (1.21 1.39) (1.97 2.23)
  (2.25 2.23) (3.65 4.17) (2.84 2.85) (2.42 2.79) (1.34 0.78)
  (3.26 3.91) (1.67 2.49) (3.59 2.84) (2.65 2.85) (1.56 2.55)
  (1.58 0.95) (2.33 2.07) (1.02 1.63) (1.93 1.04) (1.32 2.75)
  (1.82 2.54) (1.2 1.68) (1.78 1.91) (1.4 1.55) (2.37 1.89)
  (3.91 3.26) (2.91 2.6) (1.32 1.25) (3.65 3.12) (4.19 3.71)
  (2.57 1.74) (3.35 1.99) (4.02 3.96) (4.32 5.13) (1.27 0.9)
  (2.5 2.59) (4.21 4.81) (3.12 3.12) (4.17 4.69) (2.6 2.67)
  (1.56 2.08) (2.6 2.22) (2.71 2.44) (1.8 1.56) (2.61 0.78)
  (1.39 1.11) (1.94 2.25) (1.85 1.75) (0.56 0.99) (1.04 1.56)
  (2.08 3.12) (2.11 3.16) (3.12 3.04) (2.6 1.56) (2.35 2.48)
  (3.11 2.74) (1.56 1.56) (2 2.13) (1.9 1.92) (4.56 4.56) (2.36 2.81)
  (2.84 2.84) (3.55 4.26) (1.11 1.56) (6.61 6.61) (2.14 2.94)))

;; create the graph
(define g (graph-new 330 290))

;; set the font
(graph-font g (font-new))

;; set the origin
```

```
(graph-aorigin g 1.1 0.75)

;; set the style of the log axis
(graph-logstyle g #xb #x7ff 4 3 4 8)

;; initialize the log x and y axis
(graph-xlog g 3.0 0.5 10. 0.0)
(graph-ylog g 3. 0.5 10. 0.0)

;; draw a domain mesh
(graph-color g Grey)
(graph-mesh g)

;; draw some lines
(graph-color g Red)
(graph-linewidth g 0.6)

;; a solid diagonal line
(graph-moveto g 0.5 0.5)
(graph-lineto g 10. 10.)
(graph-stroke g)

;; dashed lines
(graph-dash g 6 0.1364)
(define vLmin 0.5) (define vLmax 10.0)
(define vRmin 0.5) (define vRmax 10.0)
(define ratio 1.5)
(graph-moveto g (* vLmin ratio) vRmin)
(graph-lineto g vLmax (/ vRmax ratio))
(graph-moveto g vLmin (* vRmin ratio))
(graph-lineto g (/ vLmax ratio) vRmax)
(graph-stroke g)

(graph-dash g 0 1.0)
(graph-linewidth g 1.0)

;; plot markers here
(let loop ((d data))
  (if (> (length d) 0) (begin
    (graph-color g White)
    (graph-marker g (car (car d)) (cadr (car d)) GRAPH_SOLIDCIRCLE 10)
    (graph-color g Black)
    (graph-marker g (car (car d)) (cadr (car d)) GRAPH_OPENCIRCLE 10)
    (loop (cdr d)))))

;; draw an axis frame
(graph-frame g)
```

```
;; draw the x and y axis
(graph-xaxis g)
(graph-yaxis g)

;; draw the axis labels
(graph-xlabel g "The X axis label")
(graph-ylabel g "The Y axis label")

;; show it on screen
(graph-output g GRAPH_WIN)
```

This produces the following graph:

```
> (load "~~/examples/graph-loglog")
```



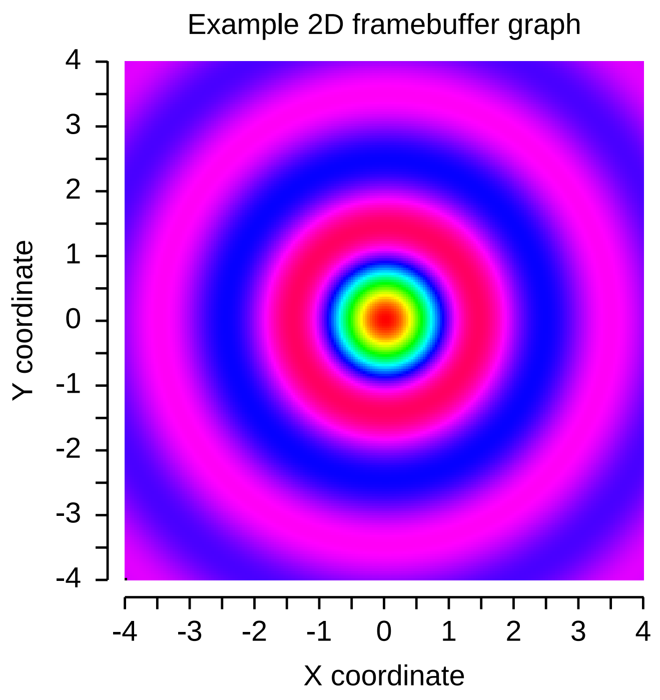
8.4 Example 4: 2D image graph

We can combine the graphing system with framebuffers like this:

```
;; examples/graph-image.scm
;; example of a 2D image graph

;; create a framebuffer with data
(define myfb (framebuffer-new 216 216))
(let loop ((x 0)(y 0))
  (if (< y 216)
    (let* ((newx (if (< x 215) (+ x 1) 0))
           (newy (if (= newx 0) (+ y 1) y))
           (rx (- (/ x 27.) 4.))
           (ry (- (/ y 27.) 4.)))
    (framebuffer-pixel myfb x y
      (color-gradient GRADIENT_RAINBOW
         (/ (+ (sinc (sqrt (+ (* rx rx) (* ry ry))))  0.3) 1.3)))
    (loop newx newy))))
(framebuffer-pixel myfb 0 0 Black)

(define g (graph-new 320 320))

;; set the origin
(graph-aorigin g 1.0 0.9)

;; select the font
(graph-font g (font-new))

;; set the style of the log axis
(graph-linearstyle g 4 0 5 1 5)

;; initialize the axis
;; 3 inches wide, from -4 to 4, offset 0 inch,
;; tick every .5, label every 2 tick
(graph-xlinear g 3.0 -4. 4. -0.1 .5 2)
(graph-ylinear g 3.0 -4. 4. -0.1 .5 2)

;; draw a domain mesh
(graph-framebuffer g -4. -4. myfb)

(graph-color g Black)

;; draw the x and y axis
(graph-xaxis g)
(graph-yaxis g)

;; draw the axis labels
(graph-ylabel g "Y coordinate")
(graph-xlabel g "X coordinate")

;; draw a caption in physical coordinates
(graph-setcoord g GRAPH_PHYS)
(graph-htextcenter g 1.5 3.2 "Example 2D framebuffer graph")

;; show the graph on screen
(graph-output g GRAPH_WIN)
```

This produces the following graph:

```
> (load "~~/examples/graph-image")
```

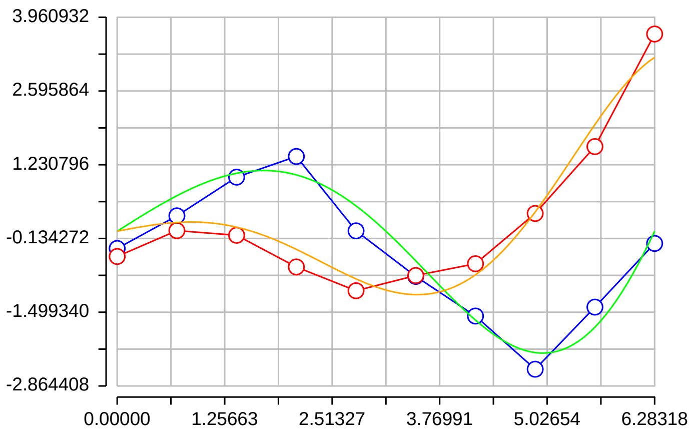


8.5 Quick data visualization

While the graphing system in SENSAWAVE is capable of generating very complex presentation quality graphs, it can be overkill when data needs to be quickly visualized. For this reason SENSAWAVE provides a function that automatically builds a simple graph from lists of data. For example:

```
> (define (f x) (sin (* 1.+0.3i x)))
> (define (r) (+ (random-real) (* +1.0i (random-real)) -0.5-0.5i))
> (define x (: 0 M_2PI 10))
> (define y (map (lambda (x) (+ (f x) (r))) x))
> (define x1 (: 0 M_2PI 100))
> (define y1 (map f x1))
> (ezgraph x y x1 y1)
```

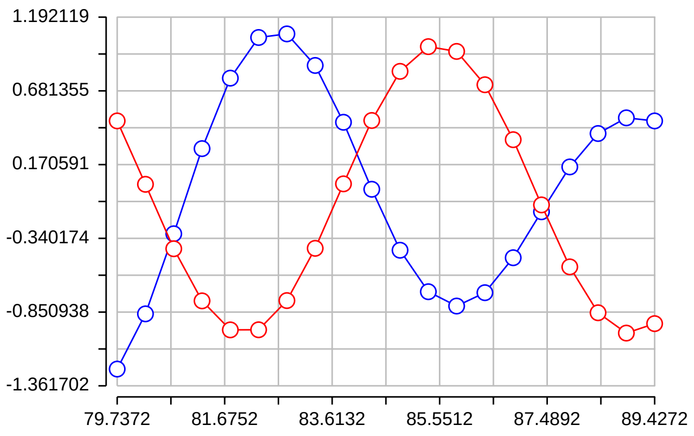
This builds two sets of data lists, a 10 element list `x` with a corresponding noisy representation `y` of the complex function `f`, and a higher resolution rendering in `x1` and `y1`. The result looks like this:



SENSAWAVE also supports a real time rendering construct intended mainly for direct display of measured data. For example:

```
> (rtgraph 20 (lambda (t) (+ (cos t) (exp (* +0.9i t)))) 0.5)
```

This opens a graph containing 20 data points that updates in real time by applying the function argument every 0.5 seconds. The function parameter is the elapsed time in seconds:



9 OpenGL

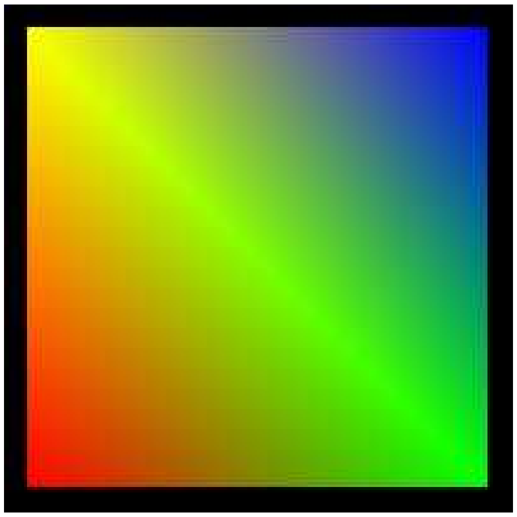
SENSAWAVE supports direct programming in OpenGL. The `glwindow-open` call can be used to open a window and specify a callback to handle the rendering and input events of the window.

This example below creates a simple rendering of an OpenGL shaded polygon:

```
;; examples/opengl.scm
;; simple opengl rendering of a smooth shaded polygon

;; the OpenGL window event handler
(define (opengl:eventhandler)
  (current-user-interrupt-handler (lambda () #t))
  (let loop ()
    (let ((msg (thread-receive)))
       (if (list? msg) (if (= (length msg) 4)
         (let ((w (car msg))  (t (cadr msg))
               (x (caddr msg))(y (cadddr msg)))
            (cond
              ((= t WINDOW_KEYPRESS)
                 (if (= x WINDOW_KEYESCAPE) (glwindow-close w)))
              ((= t WINDOW_CLOSE) (glwindow-close w))
              ((= t WINDOW_REDRAW)
                 (glcontext-grab!)     ;; grab the rendering context
                 (glcontext-window w)  ;; assign the rendering window
                 (glClearColor 0. 0. 0. 0.)  ;; setup OpenGL
                 (glMatrixMode GL_PROJECTION)
                 (glLoadIdentity)
                 (glOrtho -1.1 1.1 -1.1 1.1 -1. 1.)
                 (glMatrixMode GL_MODELVIEW)
                 (glLoadIdentity)
                 (glClear GL_COLOR_BUFFER_BIT)
                 (glBegin GL_QUADS)          ;; render OpenGL
                 (glColor Red)
                 (glVertex3f -1. -1. 0.)
                 (glColor Green)
                 (glVertex3f 1.0 -1.0 0.)
                 (glColor Blue)
                 (glVertex3f 1. 1. 0.)
                 (glColor Yellow)
                 (glVertex3f -1. 1. 0.)
                 (glEnd)
                 (glwindow-swapbuffers w)    ;; update window
                 (glcontext-release!)   ;; release the rendering context
               ))))))
    (loop)))
;; open the OpenGL window
(glwindow-open 200 200 opengl:eventhandler)
```

This program can be loaded to open the window:

```
    > (load "~~/examples/opengl.scm")
```



The window callback can be seen to be a separate thread that receives events through the mailbox interface. Please see Gambit-C documentation for more information about threads and their mailbox interface.

Each event has four parameters, the window id `w`, the event type `t` and two auxiliary parameters `x` and `y`.

The possible window event types are:

`WINDOW_BUTTON1DOWN`
`WINDOW_BUTTON1UP`
`WINDOW_BUTTON2DOWN`
`WINDOW_BUTTON2UP`
`WINDOW_BUTTON3DOWN`
`WINDOW_BUTTON3UP`
: These events are generated by the mouse buttons. The auxiliary parameters `x` and `y` contains the position of the mouse within the window at the time of the particular button event.

`WINDOW_CLOSE`
: This is generated when the user closes the window.

`WINDOW_KEYPRESS`
`WINDOW_KEYRELEASE`
: These events are generated by the keyboard. The `x` parameter contains the ascii code of the relevant key. Special values are `WINDOW_KEYENTER`, `WINDOW_KEYTAB`, `WINDOW_KEYBACKSPACE`, `WINDOW_KEYRIGHT`, `WINDOW_KEYLEFT`, `WINDOW_KEYUP`, `WINDOW_KEYDOWN` and `WINDOW_KEYESCAPE`.

`WINDOW_MOTION`
: This event is generated when the mouse is moved inside the window. The auxiliary parameters `x` and `y` contains the position of the mouse within the window.

`WINDOW_REDRAW`
: The event is generated when the window needs to be redrawn. This is where the OpenGL rendering code must be called. Prior to calling any OpenGL functions, the event handler must first call `glcontext-grab!` to acquire the OpenGL rendering context, and then assign it to the window with `glcontext-window`. After the rendering is complete the OpenGL rendering context must be released with a call to `glcontext-release!`.

This simple framework is sufficiently advanced to allow complex interactive multi-window graphics to be created.

9.1 Further reading

For more information about OpenGL programming, please refer to: `http://opengl.org`.

10 RS232 Interfacing

The RS232 serial interface provides a simple cross-platform framework for serial communication from within the SENSAWAVE Computing Base. It is designed to enable communication with multiple devices concurrently, and offers an easy way to interface a wide range of devices such as scientific instruments and network devices.

The RS232 modules must be loaded manually:

```
> (module-require 'rs232)
```

Let us assume that a serial device is connected to port COM1 at 9600 8N1 (9600 baud, 8 bit characters, no parity and 1 stop bit). Communication with this device can be started with the command:

```
> (define port (rs232-open "COM1" RS232_9600BAUD
     RS232_8BITS RS232_NOPARITY RS232_ONESTOPBIT))
```

A message can now be sent to the device using the communication handle:

```
> (rs232-writeline port "hello\n")
```

and responses can be received:

```
> (rs232-readline port)
"world!"
```

Once communication is complete, the interface is closed with the command:

```
(rs232-close port)
```

Serial port names are platform dependent. Please refer to your operating system documentation for the naming scheme. Attempting to open an unconnected serial interface may cause `rs232-open` to hang for some time.

This implementation assumes that the communication flow control is completely controlled by the external device.

11 Additional functionality

The SENSAWAVE Computing Base includes several general purpose libraries, provided in compiled form as a convenience for SENSAWAVE users.

11.1 pregexp

The `pregexp` module provides regular expressions modeled on perl and includes such powerful directives as numeric and nongreedy quantifiers, capturing and non-capturing clustering, POSIX character classes, selective case- and space-insensitivity, backreferences, alternation, backtrack pruning, positive and negative lookahead and lookbehind, in addition to the more basic directives familiar to all regexp users.

11.1.1 `pregexp` copyright statement

Web site: `http://www.ccs.neu.edu/home/dorai/pregexp/pregexp.html`

11.2 schelog

The `schelog` module provides Prolog-style logic programming. It contains the full repertoire of Prolog features, including meta-logical and second-order predicates.

11.2.1 `schelog` copyright

Web site: `http://www.ccs.neu.edu/home/dorai/schelog/schelog.html`

11.3 ssax-sxml

The `ssax-sxml` module is an extensive library to manipulate and parse XML data. `ssax-sxml` was written by Kirill Lisovsky, and is in the public domain.
Web site: `http://ssax.sourceforge.net`

11.4 Third party module disclaimer

SensaWave offers no support on these third party modules.

12 Copyright Information

This chapter provides third party copyrights and attributions related to the SENSAWAVE distribution. The copyright and terms for SENSAWAVE itself is provided in the Chapter 13 [license], page 63 chapter.

12.1 Gambit-C

The SENSAWAVE Computing Base relies on the powerful Gambit-C Scheme compiler, copyright 1994-2008 by Marc Feeley. SENSAWAVE uses an unmodified version of the compiler under the Apache version 2.0 license. The compiler is in the file `bin/gsc`, and the accompanying license in the file `bin/gsc.license`.

12.2 MinGW

The Windows version of the SENSAWAVE distribution includes a MinGW compiler suite to facilitate compilation of the C code generated by Gambit-C to native Windows binaries. The MinGW compiler suite is located in the `mingw` directory, and is distributed under the GNU General Public License, see `mingw/COPYING` for details.

12.3 PDF library

SENSAWAVE includes the PDF library written by Takeshi Kanno:

12.4 GLFW library

SENSAWAVE includes portions of the GLFW library written by Camilla Berglund:

12.5 GIFPlot library

SENSAWAVE includes portions of the GIFPlot library, licensed as follows:

12.6 The Portable Network Graphics library

SENSAWAVE includes the Portable Network Graphics library:

12.7 The zlib compression library

SENSAWAVE includes the zlib compression library:

12.8 Scientific repository packages

The SENSAWAVE distribution includes a large number of pre-compiled numerical libraries that make up the scientific repository. These libraries are available in the public domain or similar unrestrictive licenses in source code form at various sites on the Internet. The libraries are not part of the SENSAWAVE Computing Base proper and are merely provided in binary form as a convinience to SENSAWAVE users. Attribution related to the individual libraries can be found in the original source code, which is located in the `repository` directory. SensaWave makes no representation whatsoever in regards to these libraries. All of the libraries are provided on an 'as-is' basis, and SensaWave Technology Inc. have no obligation to provide maintenance, support, updates, enhancements or modifications. In no event shall SensaWave Technology Inc., its owners, contributors or distributors be liable to any party for direct, indirect, special, incidental or consequential damages arising out of the use of the libraries and their documentation, even if advised of the posibility of such damage. SensaWave Technology Inc. specifically disclaim any warranty including, but not limited to, implied warranties of merchantability and fitness for a particular purpose.

12.9 Trademarks

13 License Terms

This chapter provides information about the SENSAWAVE licensing terms. Usage and distribution of SENSAWAVE must be in compliance with the appropriate license.

In simple terms, SENSAWAVE is covered by three different licenses (full legal terms follow in the next sections):

Evaluation License
: Software downloaded from the SensaWave website is subject to the SENSAWAVE Evaluation License, and may only be used for the sole purpose of evaluating the SENSAWAVE Computing Base functionality for a limited time.

Single User License
: SENSAWAVE distributions purchased under a Single User License can only be used on a single computer at a single physical site at any given time. All supported platforms are covered by the license, but only one platform can be used at any given time. SENSAWAVE software keys purchased under a Single User License cannot be used to deploy applications.

Site and Deployment License
: SENSAWAVE distributions purchased under a Site and Deployment License can be used by any number of users on all supported platforms at a single physical site. In addition, SENSAWAVE software keys purchased under a Site and Deployment License can be redistributed with an application (including commercial) developed by the holder of the License, as long as the end use of the application is not in direct competition with the SENSAWAVE distribution itself, and the application does not offer any form of an interactive command interface. Such deployments must only include the subset of SENSAWAVE actually used.

Please note that the SENSAWAVE Computing Base cannot be redistributed under any of the license terms set forth below.

If you do not agree to the terms of the license agreement that applies to you, please do not install or use SENSAWAVE.

13.1 SENSAWAVE Evaluation License

The SENSAWAVE Computing Base ("the Software") is copyright (C) 2007, 2008 SensaWave Technology Inc. The Software is protected by the copyright laws of Canada. By copying, installing or using the Software, you are agreeing to this license and disclaimer.

In downloading, configuring or using the Software, you are not obtaining title to the Software or any copyrights. You may not sublicense, rent, lease, convey, distribute, copy, modify, translate, convert to another programming language, decompile, or disassemble the Software for any purpose. You may not redistribute the Software. You may not use the Software for any other purpose except evaluation of its functionality. You must remove the Software from your system after the expiry of the temporary evaluation period.

Conditioned on your honoring the terms of this agreement, SensaWave grants you a temporary non-exclusive, non-transferable license to use the Software, including the accompanying documentation for your lawful, non-infringing evaluation on a single physical computer at a single physical site in accordance with this Agreement.

THIS SOFTWARE IS PROVIDED BY SENSAWAVE TECHNOLOGY INC. “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL SENSAWAVE TECHNOLOGY INC. BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

13.2 SENSAWAVE Single User License

The SENSAWAVE Computing Base ("the Software") is copyright (C) 2007, 2008 SensaWave Technology Inc. The Software is protected by the copyright laws of Canada. In downloading, configuring or using the Software, you are not obtaining title to the Software or any copyrights. You may not sublicense, rent, lease, convey, distribute, copy, modify, translate, convert to another programming language, decompile, or disassemble the Software for any purpose. You may not redistribute the Software. By installing the Software, you are agreeing to this license and disclaimer.

Conditioned on your honoring the terms of this agreement, SensaWave grants you a non-exclusive, non-transferable license to use the present version of the Software, including the accompanying documentation for your lawful, non-infringing use on a single computer at a single physical site in accordance with this Agreement.

You may make up to two exact, unmodified copies of the Software solely for your own back-up or archival use.

THIS SOFTWARE IS PROVIDED BY SENSAWAVE TECHNOLOGY INC. “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL SENSAWAVE TECHNOLOGY INC. BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

13.3 SENSAWAVE Site and Deployment License

The SENSAWAVE Computing Base ("the Software") is copyright (C) 2007, 2008 SensaWave Technology Inc. The Software is protected by the copyright laws of Canada. In

Index