

# A User's Guide to the GLUT for Markov Chain Monte Carlo Graphical Interface Version 1.0

Eric C. Anderson\*

April 8, 2002

## Abstract

The GLUT for Markov chain Monte Carlo (Gf(MC)<sup>2</sup>) library provides a system for visualizing the variables involved in Markov chain Monte Carlo. It is designed to provide a relatively simple user interface that allows the management of multiple windows which provide different views of the variables involved in the simulation. It is written in C/C++. The window management system uses calls to Mark Kilgaard's OpenGL Utility Toolkit (GLUT). GLUT is almost platform independent, so the features of Gf(MC)<sup>2</sup> should work in almost the same way across different platforms (*e.g.*, Macintosh, Windows, Linux, *etc.*).

This document describes the features available to the *user* of an application that draws upon the Gf(MC)<sup>2</sup> library. The table of contents provides a good overview of those features. A separate document will be prepared for *developers* interested in incorporating the Gf(MC)<sup>2</sup> library into their own software.

## Contents

<b>1</b>	<b>Several important reminders. Read these now!</b>	<b>3</b>
<b>2</b>	<b>System Requirements</b>	<b>3</b>
2.1	Macintosh . . . . .	3
2.2	Microsoft Windows . . . . .	4
2.3	Linux . . . . .	4
2.4	Unix . . . . .	4
<b>3</b>	<b>Introduction to the Gf(MC)<sup>2</sup> System</b>	<b>4</b>
3.1	Input Capabilities and Notation . . . . .	5
<b>4</b>	<b>Controlling the Simulation</b>	<b>6</b>
4.1	Starting/Stopping the Simulation . . . . .	6
4.2	Single Sweep Mode . . . . .	6
4.3	Initializing the Simulation from New Starting Conditions . . . . .	7
4.3.1	Starting from the same random seeds . . . . .	7
4.4	Resetting the Averages Being Accumulated . . . . .	7
4.5	Quitting the Program . . . . .	7

---

\*Department of Integrative Biology, University of California, Berkeley, [eriq@u.washington.edu](mailto:eriq@u.washington.edu)

<b>5</b>	<b>Window Management</b>	<b>7</b>
5.1	The “Info” window . . . . .	8
5.2	Opening Windows . . . . .	8
5.3	The Current Window . . . . .	8
5.4	Closing Windows . . . . .	9
5.5	Moving and Resizing Windows . . . . .	9
5.5.1	Automatic Full-Size Window . . . . .	9
5.6	Bringing Windows to the Front . . . . .	9
5.7	Views . . . . .	9
5.7.1	Using Predefined Views . . . . .	10
5.7.2	Saving and Using User-Defined Views . . . . .	10
5.7.3	Smashing and Stretching Views . . . . .	10
<b>6</b>	<b>Controlling the Viewable Area</b>	<b>11</b>
6.1	“Edgeward” Expansion and Contraction . . . . .	11
6.2	Moving the Viewing Area . . . . .	11
6.3	Drag-Box Zoom . . . . .	11
6.4	Auto-fitting the Viewable Area . . . . .	11
6.4.1	Fit to Default Viewing Area . . . . .	12
6.4.2	Fit to Max/Min Viewing Area . . . . .	12
<b>7</b>	<b>Legends</b>	<b>12</b>
7.1	Showing/Hiding the Legend . . . . .	12
7.2	Moving the Legend Around . . . . .	12
7.3	Changing the Legend Text Size . . . . .	12
<b>8</b>	<b>Axes</b>	<b>13</b>
<b>9</b>	<b>Color Schemes</b>	<b>13</b>
9.1	Selecting a Pre-Defined Color Scheme . . . . .	13
9.2	Designing Your Own Color Scheme . . . . .	13
<b>10</b>	<b>Selected Items and Displayed Items</b>	<b>14</b>
<b>11</b>	<b>Controlling Column Numbers</b>	<b>15</b>
<b>12</b>	<b>Gf(MC)<sup>2</sup> Error and Warning Messages</b>	<b>15</b>
<b>13</b>	<b>Wish List</b>	<b>15</b>
<b>14</b>	<b>Software Agreement</b>	<b>15</b>

# 1 Several important reminders. Read these now!

Before we start describing the system, these are some very important notices to take to heart:

1. Most operating systems have a facility for closing a window. For example by clicking in a box at the upper left of the window, *etc.* Using such a method for closing windows opened by Gf(MC)<sup>2</sup> may lead to unpredictable results. It is recommended that you close windows using the menu and keyboard commands available to do so through the Gf(MC)<sup>2</sup> interface.
2. The GLUT interface allows for mouse input from right, left, and center mouse buttons. Macintosh systems have only a single mouse button, but can emulate multiple mouse buttons by combining the option key with a mouse click (Center Mouse Button) or the ctrl key with a mouse click (Right Mouse Button). At the same time, the current GLUT library for PC's seems unable to attach pop-up menus to all three of the mouse buttons available. For this reason, some customization may be necessary.

## 2 System Requirements

GLUT is a windowing system that has been designed to be portable across most major operating systems. Some configuration of the system may still be necessary. Information about the requirements for running OpenGL (necessary for rendering the graphics in Gf(MC)<sup>2</sup>) on different platforms is found at

<http://www.opengl.org/users/downloads/index.html>

Libraries necessary for GLUT are in different locations, as indicated below for each platform. The information page for GLUT is at:

<http://www.opengl.org/developers/documentation/glut.html>

### 2.1 Macintosh

As you can read on the OpenGL website:

OpenGL ships with OS 9 and OS X. You can also optain the latest software version on the Apple OpenGL web site. You also need an OpenGL hardware accelerator driver. All new PowerMac G4, iMac, iBook and PowerBook computers ship with built in hardware acceleration and include the correct hardware driver. If you buy a different or additional hardware board, you can obtain the driver from each board manufacturer's web site.

So if you have OS 9 or higher you shouldn't have to do anything other than run the program. Otherwise, or if there is a problem, it is probably because the system does not have the necessary GLUT and/or OpenGL libraries. These may be found at

<http://docs.info.apple.com/article.html?artnum=120000>

As far as I can tell from the website this stuff will be compatible with OS 8.1 or greater.

## 2.2 Microsoft Windows

To run the GLUT portion of Gf(MC)<sup>2</sup> you have to be able to link the the dynamic library glut32.dll. Go to the website

<http://www.opengl.org/developers/documentation/glut/index.html#windows>

and follow the directions. Basically you have to download a file, unzip it, and put glut32.dll in your system folder.

For the OpenGL part, everything should be set to go, but you may need a hardware driver to speed things up. As reported on the OpenGL website:

OpenGL v1.1 software runtime is included as part of operating system for WinXP, Windows 2000, Windows 98, Windows 95 (OSR2) and Windows NT. So you only need to download this if you think your copy is somehow missing . The OpenGL v1.1 libraries are also available as the self-extracting archive file from the Microsoft Site via HTTP or FTP.

OpenGL v1.2 & 1.3 are included with the drivers for your OpenGL video cards. So you only need to make sure you have the latest OpenGL driver for your video card. If you do not have the latest driver with OpenGL 1.2 or 1.3 support, either use GLSetup (for Win95/98 only) or contact the video card manufacturer directly and ask them for an OpenGL 1.2/1.3 driver for your card and OS.

You do need an OpenGL hardware driver for your particular 3D hardware accelerator board.. For consumer-level boards under WIndows 95 or Windows 98 run Glsetup to automatically download and install the latest & greatest video driver. Glsetup is a program written specifically to analyze your hardware and install working OpenGL drivers for that hardware. Currently (Jan 2001), Glsetup supports cards using the following chipsets:

- 3Dfx Voodoo, Voodoo2, Voodoo Rush, Banshee, Voodoo3, Voodoo3 3500TV, Voodoo5
- 3Dlabs Permedia 2 and Permedia 3
- ATI Rage 128, Rage 128 Pro, Rage Fury MAXX, Rage Pro, Radeon
- Intel i740, i810 and i815
- Matrox G200, G400 and G450
- NVidia Riva 128/128ZX and Riva TNT/TNT2/GeForce1&2/Quadro1&2
- Rendition Verite 2200
- S3 Savage3D, Savage4, and Savage2000

For cards with chipsets not listed, for WinNT, Win2000 or WinXP or for professional workstation graphic cards you can obtain OpenGL drivers directly from each board manufacturer's web site.

## 2.3 Linux

Will deal with this later. In the meantime, if you are using Linux, you probably already know what you need to do.

## 2.4 Unix

Will deal with this later.

## 3 Introduction to the Gf(MC)<sup>2</sup> System

In this document, I try to describe the user-accessible features of Gf(MC)<sup>2</sup> as tersely and quickly as possible since I am really a statistical geneticist and not a software developer and I need to get back to other work ASAP. My apologies for the unfinished and rough nature of this document. Throughout I will refer to *users* and *developers*. Developers are those who have written MCMC

simulations (or other types of programs to which they wish to add the sort of graphical interface provided by Gf(MC)<sup>2</sup>). They write code and would probably be appalled at the way I have written the Gf(MC)<sup>2</sup> library. Users, on the other hand, are those who will be using the applications written by developers. This document is for the users. It is intended to be distributed by developers to the users of their applications that rely on Gf(MC)<sup>2</sup>.

### 3.1 Input Capabilities and Notation

GLUT provides a number of ways of providing input to the computer. The Gf(MC)<sup>2</sup> interface relies heavily on keyboard and mouse input, and on GLUT’s facility for cascading pop-up menus. Following is some notation that I will use to describe these things.

**Standard Keyboard Input:** Standard ASCII input is represented by the boxed, case-specific character. For example, pressing the “f” key is denoted in this document as `f`, while pressing the “F” key and holding the shift key together is represented by `F`. Likewise, keyboard input involving all other ASCII characters is the same, *i.e.*, `a`, `!`, `T`, `%`, *etc.*

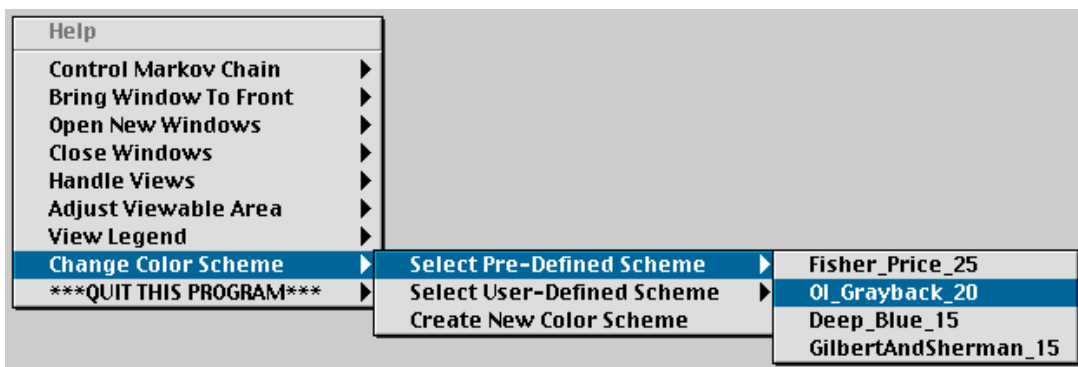
**Special Keyboard Input:** Most computers have special keys like the arrow and function keys. Input using those keys will be denoted, by, for example, `←`, `↑`, `→`, or `↓`, and `F1`, . . . , `F12`.

**Mouse Clicks:** There are three different mouse clicks possible: the left, center, and right mouse buttons denoted by `LtMouse`, `CnMouse`, and `RtMouse`. Mac users should read the important note in Section 1.

**Modifier Keys:** The three modifier keys available when doing input are the shift, control, and alt, keys denoted by `shift`, `ctrl`, and `alt`, respectively. Note that on Macintosh systems, `alt` is called the “option” key.

**Putting It Together:** The modifier keys and the mouse clicks and keyboard input can be combined in the obvious way, denoted here by the plus symbol: `shift` + `LtClick` and `alt` + `→`, *etc.*

**Cascading Menu Notation:** One of the nicest features of GLUT is the cascading pop-up menus. These are invoked by clicking on a particular mouse button (when the mouse cursor is inside a currently open window) and then navigating through the menu. For example, the following picture illustrates the selection of “Change Color Scheme” followed by the selection of “Select Pre-Defined Scheme,” and finally selecting “Ol\_Grayback\_20.”



This represents a selection from the “Main Menu,” and will be denoted in this document as **Main Menu**→*Change Color Scheme*→*Select Pre-Defined Scheme*→*Ol\_Grayback\_20*. The

Main Menu is accessed by doing `RtMouse`. Sometimes specialized menus are attached to other mouse click buttons. In such a case, the sequence of selection will be denoted, for example, `RtMouse`→*First*→*Second*→*Third*. To access a menu associated with a window, the mouse pointer must be inside that window!

Note that if there is a keyboard shortcut for a particular menu item, I have tried to include that shortcut in the menu item name (see for example, the next section on Controlling the Simulation).

## 4 Controlling the Simulation

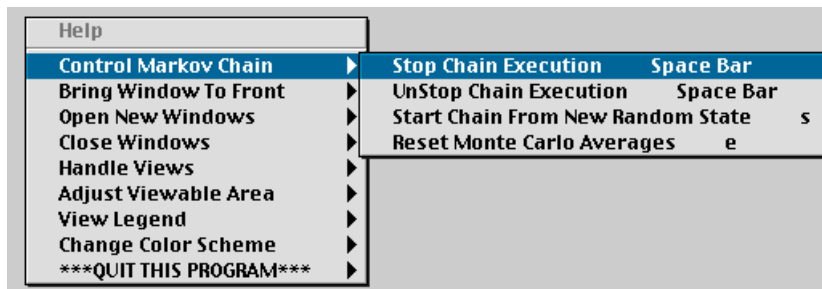
Gf(MC)<sup>2</sup> provides several simple means for starting and stopping the MCMC simulation, for re-initializing the starting values, and for resetting the Monte Carlo averages being accumulated as the chain runs.

### 4.1 Starting/Stopping the Simulation

This action causes the chain (or the main loop in the developer’s program) to suspend execution. Stopping and starting the chain is controlled by choosing in the main menu

**Main Menu**→*Control Markov Chain*→*Stop Chain Execution*  
or  
**Main Menu**→*Control Markov Chain*→*UnStop Chain Execution*

Or, as suggested in the pop-up menu:



hitting `Space Bar` is a keyboard shortcut for toggling between executing the simulation and momentarily suspending simulation execution.

### 4.2 Single Sweep Mode

Sometimes it is desirable to be able to watch the simulation change one step at a time. To do this, use the `o` key (that is a lower-case “o” as in orangutan). Think of it as “`o`ne step.” When you hit `o`, continuous simulation of the chain will stop and you may proceed one step at a time by hitting the `o` key. What is meant by “one step” in the simulation depends on how many variable updates the developer has built into a single sweep. In many cases it will mean all the variables in the simulation get updated (or updates are proposed to them) once. To restart the chain from Single Sweep Mode, just hit the `Space Bar`.

### 4.3 Initializing the Simulation from New Starting Conditions

The exact details of what these actions do depend on how the developer has decided to initialize the variables in his program for MCMC simulation. In programs for Bayesian computation I have written, I try to initialize variables by realizing them from their prior distributions. This gives “overdispersed” starting values. Watching a chain unfold from many different starting points is a good way to get a feel for how rapidly the chain mixes, and how prone it is to getting caught in certain parts of the space. This is one of the primary applications for which I wrote Gf(MC)<sup>2</sup>. Initializing the variables in the simulation can be done by choosing

**Main Menu**→*Control Markov Chain*→*Start Chain From New Random State*

or by hitting **[s]**.

Chain initialization should also invoke the resetting of Monte Carlo averages described below, however, this is left to the discretion of the developer.

#### 4.3.1 Starting from the same random seeds

Finally, if the developer has provided the right code to enable this feature, hitting **[alt] + [s]** will reinitialize the chain to random starting values *using the same random number seeds as when the chain was last re-initialized*. This is useful if you want to watch the Markov chain unfold in exactly the same way as it did last time. There is currently no menu option for this.

### 4.4 Resetting the Averages Being Accumulated

Monte Carlo is an exercise in approximating expected values by the sample means (averages) of simulated random variables. Quite often, in Markov chain Monte Carlo, upon initialization from random starting points, the Markov chain should be run for some time so that the probability of its being found in any particular state is close to the “stationary” or “limiting” probability of being in that state. This “warm-up” period for Markov chains that are not initialized by a random draw from their limit distribution is often called “burn-in,” and including values visited by the chain during this time in your Monte Carlo averages may distort them. For this reason, Gf(MC)<sup>2</sup> provides the menu option:

**Main Menu**→*Control Markov Chain*→*Reset Monte Carlo Averages*

which is equivalent to the keystroke command **[e]**. Once again, the details of this are developer-specific, but the command should reset the Monte Carlo averages that have been accumulated over the length of the Monte Carlo run. This has the effect of declaring all sweeps of the chain until the present one to be “discarded” as burn-in.

### 4.5 Quitting the Program

You can exit the program by choosing the menu option

**Main Menu**→*Control Markov Chain*→*\*\*\*QUIT THIS PROGRAM\*\*\**→*Yes. I Really Want To Quit*

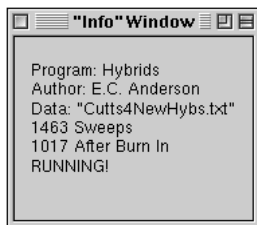
## 5 Window Management

The visual heart of Gf(MC)<sup>2</sup> lies in its management of multiple windows for viewing data. These windows, for the most part, behave just like the windows in your normal operating system except they will lack scroll bars, and perhaps a few other familiar features. Section 6 explains about

zooming in and out of windows and changing the area that is viewed within the window, while this section explains the basics of opening, closing, moving, and resizing windows. It also describes the use of “Views” which are collections of windows that have been opened and resized to give a particular view of the data and variables. Rather than having to tediously open and resize all the windows the next time you want to open them (*i.e.*, after quitting the program) you can save the current window settings, name it as a new “View” and then retrieve that view any time in the future.

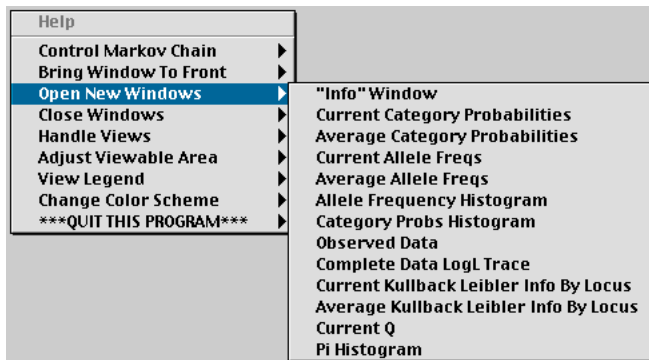
### 5.1 The “Info” window

There is one window that must always remain open. This is the “Info” window. Its contents are up to the developer, but at a minimum it should list the name of the program, and it should list the number of sweeps since the chain was last initialized and the number of sweeps since “burn-in” was finished (*i.e.*, the number of sweeps since the last issuing of the command to reset all the Monte Carlo averages), and whether the simulation is currently executing, or is stopped. The info window opens automatically when you launch the application. Though you may resize it and re-position it as desired, you will not be able to close it from within the set of keyboard and menu commands available in Gf(MC)<sup>2</sup>. An example Console window from one of my programs is:



### 5.2 Opening Windows

Gf(MC)<sup>2</sup> keeps track of the available types of windows that the developer has defined for the program. These may be selected by name under the **Main Menu**→*Open New Windows* menu selection. That is about all there is to it. There are no keyboard shortcuts for opening windows. The picture below shows an example of the sort of windows that might be available.



### 5.3 The Current Window

When multiple windows are open in Gf(MC)<sup>2</sup>, one of those windows will be in front and will have the look of a typical “active” window in your operating system. This window is called the “current” window. You can make a window the current window by clicking within its perimeter. Keyboard input and mouse input usually changes the parameters associated with the current window. Most



of the commands described in this document are universal (*i.e.*, the commands can be issued in and “understood” by any open window”) but there are some exceptions (and maybe more depending on the developer.)

## 5.4 Closing Windows

When multiple windows are open, then the current one can be closed by choosing

**Main Menu**→*Close Windows*→*Close The Current Window*

or by hitting `w`.

To close all of the open windows except for the Info window, you can choose

**Main Menu**→*Close Windows*→*Close All Non-“Info” Windows*

or hit `x`.

## 5.5 Moving and Resizing Windows

Windows may be moved and resized according to the rules for your operating system (for example, clicking on “handles” in the lower-right corner of the window and dragging should resize the window, while clicking and dragging on the window header should move it. *etc.*..

### 5.5.1 Automatic Full-Size Window

If the GLUT system has been able to determine your screen dimensions from the operating system (or if it has not, but you have provided  $\text{Gf}(\text{MC})^2$  with those dimensions when it prompted you in the beginning of the program), then you can make the current window expand to fill the entire screen by hitting the backslash key, `\`.  $\text{Gf}(\text{MC})^2$  will store the current position and size of the window, so that the next time you issue a `\` to that window, it will return it to its original position and size before you expanded it to fill the whole screen.

## 5.6 Bringing Windows to the Front

$\text{Gf}(\text{MC})^2$  keeps a list of the windows that you have opened, so that if one of them is obscured by another window, you may bring it to the front by choosing

**Main Menu**→*Bring Window To Front*→*“Name of Window”*

On some systems, this operation may not turn the window now in front into the “current window,” and it may be necessary to click on the window’s border to turn it into the current window.

## 5.7 Views

$\text{Gf}(\text{MC})^2$  is able to record which windows are open and with which settings, so that the next time you run the program you can quickly and easily open all those same windows. The collection of windows, their settings, positions, and sizes are referred to as a “View.” Most developers will include with their program a text file that encodes some pre-defined views for the user, and users may also record their own views.

Note that when views are recorded, information about the current screen size is also recorded. This way, if you then change your monitor size (for example if you have to change the monitor resolution to give a talk using a screen projector) then so long as GLUT can sense the change in

screen resolution, it will scale the view accordingly, so that all the windows will fit into the new screen space. *Note! You have to restart your Gf(MC)<sup>2</sup>-based application, after changing the screen resolution, for these changes to take effect.*

### 5.7.1 Using Predefined Views

The views that the developer has defined may be invoked by the menu call

**Main Menu**→*Handle Views*→*Restore Predefined View*→“Name of View”

And, there is also a keyboard shortcut. The first 9 predefined views may be restored to the screen by pressing the appropriate numbered key, 1 through 9. If the key pressed is greater than the number of views, nothing happens.

The developer will have written the program to read the predefined views from a file with a name like `ProgramName_PreDefdViews.txt`. If such a file is absent from the directory (“folder” in Mac-speak) in which the program resides, Gf(MC)<sup>2</sup> issues a warning to the file `GFMCMC.Errors.txt` and no predefined views will be available.

### 5.7.2 Saving and Using User-Defined Views

After configuring a number of open windows, the user may save the current view by selecting

**Main Menu**→*Handle Views*→*Save the Current View*

When this is selected, a new window will pop up on top of the current window, prompting the user for a name for the view. A descriptive name should be typed in followed by a carriage return. The new view is then saved, and will be available for use by choosing

**Main Menu**→*Handle Views*→*Restore User-Defined View*→“Name of View”

or by hitting alt + 1 through alt + 9 for the first nine user-defined views.

The user-defined views are also stored in a text file named by the developer—probably named something like `ProgramName_UserDefdViews.txt`—and which will be written in the directory in which you are executing the program. You can start with a clean slate by discarding that file. If you are feeling more adventurous, you can go in and edit that file and remove individual views. It’s pretty self-explanatory—erase the description of the view, and then decrease by one the “NumberOfViews” listed at the top of the file.

### 5.7.3 Smashing and Stretching Views

Sometimes, it is nice to be able to shrink or stretch all the open windows together, so that their relationship to one another remains the same, but their size changes. Someday I may more fully implement this. But for now we just have two keystroke commands: ; (a semi-colon) shrinks the whole view by a factor of 2, and : (a colon) stretches the whole view by a factor of 2.

If you shrink the view too many times, the proportions of some windows become unstable (due to rounding error, I believe) and it won’t be possible to expand it back up to look like it did before you smashed it. The scaling may also not be perfect because different operating systems have different-sized borders around their windows.

WARNING: Repeatedly shrinking the view until all windows are tiny (or stretching the view until all windows are huge and mostly off the screen) may lead to program failure. Just don’t do it.

## 6 Controlling the Viewable Area

Since GLUT windows lack the scroll bars familiar to many, I have hacked together some different ways of changing the viewable area on the screen. They aren't terribly intuitive, but after you get used to them you can buzz around pretty well with them.

Almost all development of Gf(MC)<sup>2</sup> so far has been toward viewing 2-dimensional images. This is because they are faster to render and simpler to deal with. Later versions of Gf(MC)<sup>2</sup> will probably include facilities for rotating 3-D images and such. But for now, we are merely interested in viewing things in a plane.

Note that some of the facilities here may be over-ridden by developers.

### 6.1 “Edgeward” Expansion and Contraction

The arrow keys can be used to make the viewable area of the window expand and contract. For example, `ctrl` + `→` makes the right edge of the viewing area (not the window itself, but the amount of the Cartesian plane that is visible in the window) expand to the right by a factor of .1 of the current viewable width. For example, if you were able to view the area of the unit square with corners at (0,0), (0,1), (1,1), and (1,0), then `ctrl` + `→` would change the two right hand corners of the viewable area to (1.1,1) and (1.1,0), respectively.

Issuing the keystroke `shift` + `→` would shrink the right hand edge back in so the edge of the viewable area would again be defined by the points at (1,0) and (1,1).

The same manipulation works with the three remaining arrow keys. Take some time to play around with it. As I said before, it is not very intuitive, but eventually you get used to it.

### 6.2 Moving the Viewing Area

Hitting one of the arrow keys without any modifiers (*i.e.*, without `shift`, `ctrl`, or `alt`), causes the viewing area to be translated as if you were looking at the viewed plane through a portal that was being moved in the direction of the arrow. For example, if you were looking at the unit square described above, then hitting `←` would cause the viewable area to shift by a factor of 0.01 of the current width of the viewable area, so that you would then be able to view a square with corners at (-0.01,0), (-0.01,1), (0.99,1), and (0.99,0). The key sequence `alt` + `←` does the same thing, but shifts the view by a factor of 0.25 of the current viewable width or height (depending on if you are shifting horizontally or vertically).

### 6.3 Drag-Box Zoom

To zoom into a specific rectangular region of the viewable area you can do `shift` + `LtMouse` and then drag the mouse pointer while continuing to hold down the `shift` key and the `LtMouse` button. A box will form on the screen. Drag the box out to the desired dimensions and release the mouse button and `shift` key. At this point, hitting the lowercase `z` (that is a “z” for zoom!) will cause the scene to zoom in to the area of the box.

If you don't hit `z`, the box will remain there innocuously until you do another mouse click of any sort.

This feature may be disabled in some windows.

### 6.4 Auto-fitting the Viewable Area

There are two different ways of “auto-fitting” the viewing area. One is to return the viewing area to that which was set when the window was created. This method is always available, but not always

useful (because the initial settings of the window may have been prescribed by a pre-defined view that was set up for a different set of data, *etc.*) The second method involves auto-fitting the view to encompass the extrema in the plotted figure. This method requires that the developer enable it, and this is not always the case.

#### 6.4.1 Fit to Default Viewing Area

The key command **V** (uppercase) causes the viewable area in the window to return to the values present at the time the window was created. In many instances, this will allow viewing of most if not all of the contents of the window, however, in some cases it will not. This is a useful keystroke for returning to the original viewing area after doing a drag-box zoom. It is also available as the menu item:

**Main Menu**→*Handle Views*→*Adjust Viewable Area*→*Set Viewable Area To Default*

#### 6.4.2 Fit to Max/Min Viewing Area

If the developer has added some code in the routine that draws the images that allows Gf(MC)<sup>2</sup> to “sense” the boundaries of the plotted image, then hitting **v** (lowercase) will cause the viewing area to adjust to fit all the plotted points in the viewable area, along with a developer-specified amount of “padding” on the edges. This is also available as the menu option

**Main Menu**→*Handle Views*→*Adjust Viewable Area*→*Auto Size Window To Fit Content*

## 7 Legends

There is a rather clunky facility for Legends which describe the meaning of the different colors in an image. The developer has to supply an array of strings and the number of color key labels for the legend feature to be available, so it may not always be available.

### 7.1 Showing/Hiding the Legend

To make a legend appear in the current window (if a legend is available) you choose

**Main Menu**→*View Legend*→*Hide or Show Legend*

or you can hit the **L** key.

### 7.2 Moving the Legend Around

Legends may appear on the top, the right, or the bottom of the image. To change the position of the legend you can hit the appropriate uppercase key **T**, **R**, or **B**, or you can choose the desired option from the **Main Menu**→*View Legend*→ submenu.

### 7.3 Changing the Legend Text Size

Perhaps the klugie-est thing about the Legend features (well, apart from the fact that I really should put the legend in a subfigure...but that is another story) is that the text in them is rendered as a bitmap font that only has three sizes. The consequence of this is that sometimes the text is going to run over from one item to the next. This can be taken care of by increasing the window size (the bitmap fonts do not scale up with window size) or by changing the bitmap text size, or both. There are three bitmap font sizes available for the legend text. You can cycle through them by

hitting the `b` key. This will cause the font size to increase to its maximum size, and then then next time you hit `b`, the size will return to its smallest.

## 8 Axes

The axes in  $Gf(MC)^2$  have a number of settings that allow differences in the number of major ticks and minor ticks, where the axes cross each other, the extent of the axes, whether the axes are fixed in space or “float” within the boundaries of the viewable area, whether the axes are viewable or not, and all that sort of stuff. These things are currently under the control of the developer. At some point I may make these user-changeable, but for the most part there are usually sound reasons for having axes set up as the developer wanted them to be.

The only kind of axes currently available are for 2-D figures.

## 9 Color Schemes

A color scheme in  $Gf(MC)^2$  is composed of the definitions for the colors of the:

1. background
2. on-screen text
3. legend background
4. legend border
5. axes
6. series of colors used to distinguish between different elements in a figure (for example, if you plot three different lines on a graph, they can be drawn in three different colors drawn from the Series colors in the color scheme).

### 9.1 Selecting a Pre-Defined Color Scheme

There are four color schemes that come predefined: `Fisher_Price_25` (a black background with lots of primary colors early in the series and 25 colors total in the series), `Ol_Grayback_20` (light gray background with 20 colors in the series), `Deep_Blue_15` (dark blue background with 15 colors in the series) and `GilbertAndSherman_15` (an olive green background with 15 colors in the series.) To switch the current window to any of those color schemes simply choose them from the

**Main Menu**→*Change Color Scheme*→*Select Pre-Defined Scheme*→

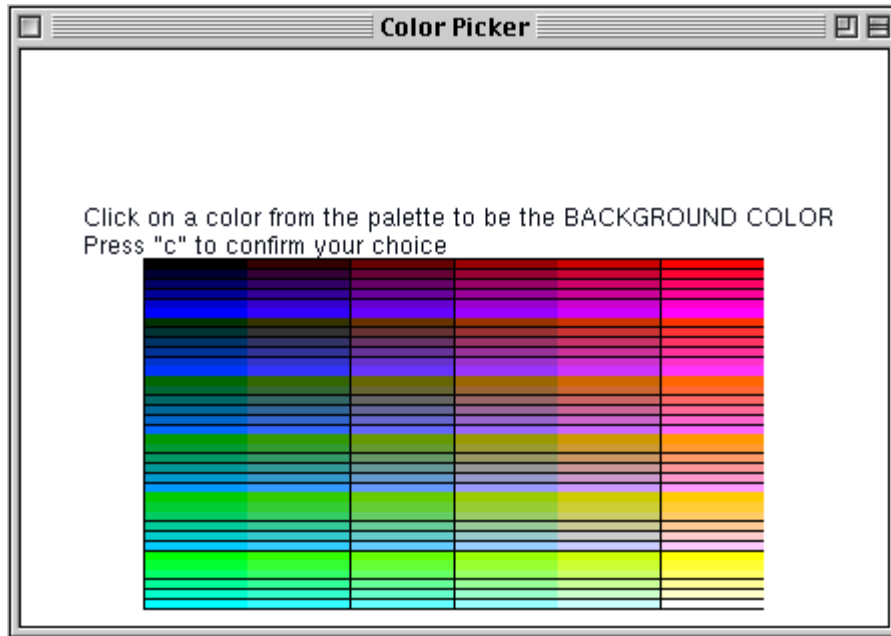
submenu. See the figure on Page 5 for an example.

### 9.2 Designing Your Own Color Scheme

There is a special “Color Picker Window” for designing your own color schemes. This window is opened by choosing

**Main Menu**→*Change Color Scheme*→*Create New Color Scheme*

The window is pretty self-explanatory, but it may not work with Windows machines. Still have to figure that out. The window looks like:



By clicking on the colors in the palette, you can change whatever part of the color scheme you are currently involved in selecting. The first color to choose is the background color. Once you have found a satisfactory choice, press **[c]** to confirm that choice. You will then be prompted to choose the text color. If at any time you would like to go back and change a former choice, cycle through the choices using **[b]** (b for back). When you are satisfied that you have defined a suitable color scheme and are ready to save it, hit **[x]**. A new window will spring up, prompting you to enter a name for the color scheme. Type a name and hit return. Your color scheme will then be saved in a text file with a name like `ProgramName_UDColorSchemes.txt` in the directory where the Gf(MC)<sup>2</sup>-based program you are using resides.

While within the color picker window, it is also possible to load a pre-existing color scheme into it so that you may modify a pre-existing color scheme. Do this by changing the color scheme for the window in the usual way. Then, when you are prompted for new color choices, the default will be the color associated with the color scheme that you just loaded. When you load a Pre-Defined Color Scheme into the color picker window, you modify that color scheme in the program memory, so it will change the appearance of all windows that currently have that color scheme. However, when you restart the program, the original Pre-Defined Color Scheme will be restored. This is not so if you modify an existing User Defined Color Scheme from within the color picker window. In that case, you will overwrite the original User Defined scheme. Some day I hope to change this.

If you find you have a profusion of User Defined schemes, you can delete the file `UserDefdColorSchemes.txt` from the directory where the Gf(MC)<sup>2</sup>-based program you are using resides, or you can edit that file. The first number in the file tells how many color schemes there are in the file (and this will have to be changed if you delete some of the existing schemes) and the rest of the file contains the definitions of the color schemes in human-readable text format. It should be pretty self-explanatory.

## 10 Selected Items and Displayed Items

Some windows will display only a particular part of the data corresponding to the “Displayed Item.” To increase the index of the Displayed Item use **[+]** and to decrease the index of the displayed item use **[-]** (the “minus sign” key. The developer should have seen to it that if you try to increase or

decrease the index of the displayed item beyond the available boundaries, then hitting either  $\boxed{+}$  or  $\boxed{-}$  should have no effect (or, perhaps, will cycle you back to the lowest value of the index).

Items in some windows may be selected by clicking on them. Sometimes you have to click on the mouse and wiggle the mouse cursor over the item to be selected in order to get the program to actually register your choice.

## 11 Controlling Column Numbers

In some windows, data are displayed in rows, and if there are too many rows, it is difficult to see each one. In such cases, the data can be split into rows filling two or more columns. The number of columns may be increased by hitting  $\boxed{M}$  (capital “M” for More) and decreased by hitting  $\boxed{F}$  (capital “F” for Fewer).

## 12 Gf(MC)<sup>2</sup> Error and Warning Messages

A file `GfMCMC_Errors.txt` might appear in the directory from which the Gf(MC)<sup>2</sup>-based application is running. This is just a little place where it barks out warnings and errors. Currently, I think it just spews warnings if Gf(MC)<sup>2</sup> can't open certain files that it thinks it might be looking for. In the future it will probably provide more useful information.

## 13 Wish List

There are many features I wish to eventually add to this. But that will happen in a later version.

## 14 Software Agreement

Copyright ©. The Regents of the University of California (Regents). All Rights Reserved.

Permission to use, copy, modify, and distribute this software and its documentation for educational, research, and not-for-profit purposes, without fee and without a signed licensing agreement, is hereby granted, provided that the above copyright notice, this paragraph and the following two paragraphs appear in all copies, modifications, and distributions. Contact The Office of Technology Licensing, UC Berkeley, 2150 Shattuck Avenue, Suite 510, Berkeley, CA 94720-1620, (510) 643-7201, for commercial licensing opportunities. Created by Eric C. Anderson, Department of Integrative Biology, University of California, Berkeley.

IN NO EVENT SHALL REGENTS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF REGENTS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

REGENTS SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE AND ACCOMPANYING DOCUMENTATION, IF ANY, PROVIDED HEREUNDER IS PROVIDED “AS IS”. REGENTS HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

Also, if you use this software while giving a talk or presentation, please cite it and note that it is available by contacting Eric C. Anderson: [eriq@u.washington.edu](mailto:eriq@u.washington.edu). Eventually I will have a web page from whence it may be downloaded without having to talk to me first!