MFDRAW
Specifications for revised version, October 2, 2010

George Reeke
Laboratory of Biological Modelling
The Rockefeller University

PURPOSE AND GENERAL DESCRIPTION

mfdraw is the component of the Laboratory of Biological Modelling-NSI graphics system that renders graphics images on an X windows display. It appears to the user as a sort of "video player" in the sense that it allows controlled motion through a series of related images, known as "frames". Images are supplied in the form of an NSI graphics metafile. Metafiles contain instructions to draw objects such as lines, rectangles, circles, and text, color and line thickness changes, and so forth. Graphics metafiles exist in ASCII and binary versions–mfdraw should accept either. The formats of these files and the software library that can be used to create them is described in a separate document "plotting.pdf".

mfdraw has two major modes of operation and within each of those are three submodes: It may operate as a stand-alone application that displays images from a stored NSI graphics metafile and allows the user to browse through that file as described in the "USER CONTROLS" section below. This is known as "offline mode" or "metafile mode". Alternatively, mfdraw can be invoked as a server via the inetd (or xinetd) mechanism when an application linked with the plot library attempts to open a socket connection to it. This mechanism permits drawing to occur while the application is running, perhaps on a different host than the one running the application. This is known as "online mode" or "socket mode".

In either major mode, the user is permitted to select between "step" submode, "batch" submode, and "movie" submode. In step submode, either online or offline, image display is entirely under control of the user. An attached application is permitted to compute only one image beyond the latest one displayed, allowing the user to interrupt the computation almost immediately when erroneous results are observed. (Forcing the application to wait for each image to be approved before continuing would introduce unacceptable delays in long simulations.)

In batch submode, the user specifies a minimum display time in seconds per image. Offline, images are displayed sequentially at this rate. Online, images are displayed as soon as they are computed, ignoring the minimum display time. When the end of the input file is reached, mfdraw terminates and any windows on the display are closed. Movie submode is the same as batch submode, with the exception that when the end of the input file is reached, mfdraw continues running until explicitly terminated by the user. The user can still navigate through the frames in memory. All other specifications below relating to movie submode also apply to batch submode.

The user may switch between step, movie, and batch submodes at any time.

APPEARANCE

mfdraw may open one or more graphics display windows under control of data in the input metafile. In particular, the size of each window is specified in the metafile, while the position on the display screen is generally under control of the window controller. (The user is allowed to indicate a preferred position, which the window controller may or may not observe.)

Each window always has the following components, whether or not an image is being displayed (see screen snapshot below), in top to bottom order:

(1) The title bar. This should display a centered title provided in the metafile, or, if no metafile has yet been read, the word “mfdraw”. (It may be necessary to truncate the title if it is too long to fit in the specified window width.) The title bar also contains the usual controls supplied by the window controller on every window. In gnome/metacity, these are a window menu button (top left) and minimize, maximize, and close buttons (top right).

(2) User controls bar. (All of these are described in more detail in the next section.) From left to right, this bar contains (a) the File menu pull down button; (b) Video navigation controls – rewind, fast backspace, single backspace, pause, forward space, fast forward, forward to end of input; (c) GoTo button; (d) Home or Fit button; (e) Invert background button; (f) Magnification controls. If the width specified in the metafile is too narrow to accommodate all of these controls, the width is made large enough to contain them. Each button has an associated “tooltip” that pops up when the user hovers the mouse over the button for one second without clicking.

(3) Image display area. This is an area the size specified in the metafile (or wider if necessary to match the user controls bar). It contains the drawing objects specified in each image.

USER CONTROLS

If mfdraw is started without an initial file specification, the display area is blank and the user is expected to select a file to display using the File menu. Otherwise, images from the file specified on the command line (sysin if running in socket mode) are displayed.

The File Menu

The File menu has the following buttons (shortcut keys underlined):

(1) Open file. This brings up a standard file selection menu for the window manager in use. The file specified by the user immediately becomes the input file.

(2) Settings. This brings up for editing a user preferences file. This file provides defaults for common options. Its contents are described later.

(3) About. Displays the current version number of mfdraw and a brief synopsis of keyboard shortcut keys that may be used during display. (These are described below.)

(4) Exit. Causes mfdraw to terminate immediately. If running in server mode, a message (described below) is sent to the attached application telling it to terminate.

Navigation Buttons

The video controls (“navigation buttons”) should have icons as indicated at the start of each line and should work as follows:

|<	Jump immediately to the earliest available frame (always frame 1 if viewing a metafile).
<<	Step backwards through frames continually until another button is clicked or the first available frame is reached. Speed as in Details below.
<	Step back one frame. With Ctrl key, back to previously viewed frame (see below).
||	Pause ongoing fast-forward or fast-reverse sequence. Continue when pressed again.
>	Step forward one frame. With Ctrl key, forward to next viewed frame (see below).
>>	Step forward through frames continually until any other button is clicked or the last available frame is reached. Speed as in Details below.
>|	Jump immediately to the latest available frame (always the last frame in the file if viewing a metafile.

These buttons should work as stated in metafile mode or any step submode. None of them causes entry into movie submode. Any of them, if clicked in socket movie submode, causes updating of the display with new images to stop while new frames continue to be acquired from the application until the in-frame memory limitation is reached. Display is then under user control as in step submode except image acquisition continues. Normal movie display continues if the “Jump to end” ('>|') button is clicked.

When in fast-forward, clicking the '>>' button should cause a small increment in speed (decrement in minimum viewing time) for each click, and clicking the '<<' button should cause a small decrement in speed (increment in minimum viewing time). Conversely, when in fast-backwards, clicking the '<<' button should cause a small increment in speed for each click, and clicking the '>>' button should cause a small decrement in speed. These do not change the minimum viewing time set in the preferences file. Alternatively, the minimum viewing time can be modified during a run by invoking the File->Settings menu.

When in fast-forward or fast-backwards (socket or offline), clicking any navigation button other than '>>' or '<<' causes exit from that mode and viewing of the single frame implied by the button that was clicked.

The GoTo button should have an entry widget where the user can enter a frame number, whether in metafile or socket mode. If clicked while in socket-movie mode, frame updating is handled as for the other navigation buttons as described above. The frame number should count from the first frame in the metafile or socket file, regardless of any frames that may have been discarded. If a frame is requested that is not available, a nice little beep would be sufficient error message.

Fit (aka Home) Button

The Fit button is used to adjust the magnification and positioning of the frame to just fit in the current window. It should also reset the scale origin to the original origin. A suitable icon for the Fit button is a square with a cross inside it.

Invert Background Button

Normally, all graphics objects are drawn on a black background. When this button is clicked, the background toggles between black and white. The purpose of this is to provide a white background when a display is to be captured for printing.

Magnification Controls

See the current version of mfdraw for an acceptable version of these controls. Other versions can be considered. There are three adjacent square buttons, corresponding to three levels of magnification, approximately 6 X, 1.5 X, and 1.05 X. When clicked with the Shift or Ctrl key held down, demagnification by the corresponding percentage occurs. To the right of the three buttons is a small text widget that displays the current magnification level. This is the cumulative amount by which the scale of the image has been multiplied relative to a scale of 1.0, considered as the size that just allows the frame to fit in the current window. This must take into account changes made with the arrow keys, but not window resizes.

At any given moment, there is a center in the image about which magnification occurs. By default, this is the center of the image. The user can move the magnification center to any other position by clicking on the image. The new center is remembered until another center is chosen or until the Fit button is clicked. [See Rules of Inheritance below.]

SHORTCUT KEYS

The following keys (upper or lower case) should be active at all times (except where otherwise noted) when the mfdraw application has keyboard focus:

A	Display the “About” information (program version number).
B	Toggle background between black and white.
F	Bring up file dialog (only active when no file has been selected).
H	Same as clicking “Fit” (aka “Home”) button.
M	Toggle between Movie and Step submodes.
Ctrl-Q or Alt-F4	Bring up the “Quit” or “Continue” popup.
Up-Down arrow keys	Precision zoom up or down.
Left-Right arrow keys	Step forward or back one frame.

POPUP MENUS

The main popup menu is invoked by clicking on the right hand mouse button with the curser inside the boundaries of the application window. Each of the options on the popup menu can be activated by clicking on it with the left mouse button. The following are the options to be presented to the user by the popup menu:

Mode of Operation: Choose "Step", "Movie", or "Batch" as described above. The initial mode when the application starts up is set by the application (possibly from a command-line option). The mode can be changed at any time.

Memory Limit: Allows the user to modify the maximum amount of memory to be allocated for frame drawing command storage.

Interrupt. Interrupts the execution of the application. The action taken when this button is clicked depends on the application. In CNS, this button terminates the current trial and causes Group III control cards to be read.

Quit. Terminates execution of the application and closes the graphics window and menu. A further popup gives the user a chance to confirm this action.

Continue. Closes the popup menu. The program continues as before.

When the application completes and mfdraw is in "Movie" mode, a popup menu should appear with the choices "Continue" or "Quit". "Quit" terminates the application and the graphics display. "Continue" causes the popup menu to disappear, possibly allowing the clean display to be photographed. To quit after making this choice, the user must either use the 'X' button in the title bar, the Ctrl-Q or Alt-F4 shortcut keys, or right-click in the graphics window to bring up the main menu again. Additional menus with "Yes"/"No" or "Continue"/"Quit" choices may appear in the event of certain errors.

USER PREFERENCE FILE

There should be a user-editable preferences file. This file will be called ".mfdrawrc". When mfdraw starts up in metafile mode with the file name on the command line, it will look for this file first in the directory where the current metafile being viewed was found, then in the directory specified by the MFDRAW_HOME environment variable, if present, then in the home directory of the current user. In other modes, the first of these is not applicable, and the second refers to the home directory of the user under whose userid the program is being run. In socket mode, MFDRAW_HOME can be supplied by xinetd. In the event that no preferences file is found, the program will use compile-time defaults for all preferences.

Items for inclusion in the preferences file (others may be added during program development) are: Preferred startup location of main window, background color, minimum frame viewing time for fast-forward and fast-reverse buttons, default X display, default index size, maximum memory to be devoted to storing frames.

I would like to be able to specify startup location by one of the following options or similar, or with explicit x,y coordinates: ULHC, URHC, LLHC, LRHC -- Upper left-hand corner, etc., Center, Center Left, Center Right, Center Top, Center Bottom. (This option ignored if not supported by the window system in use.)

ENVIRONMENT VARIABLES

The following environment variables are recognized if set on the host where the application is running:

MFDRAW_HOST	Name of host where mfdraw is to be run. The application usually provides an alternative method to specify this information.
MFDRAW_HOME	Name of directory on host where mfdraw is running where the .mfdrawrc preferences file may be found.

DETAILS

All coordinates in the metafile are specified in inches. It is understood that an exact mapping from inches to pixels may not be available, and in any event a window onscreen may be expanded or shrunk under user control, but mfdraw will make a best-effort attempt to make each window initially the specified size.

There must be a minimum display time for the fast forward and fast reverse buttons, rather than just drawing as fast as possible as mfdraw does when in socket movie mode. This time can be read from the preferences file and overridden while running by an item on the "File->Settings" menu. The minimum frame viewing time does not apply in socket movie mode.

The user must be able to move around among the frames currently in memory while mfdraw continues to request and store new frames from the running application. However, when the available frame memory is full, mfdraw will delete frames and acquire new frames only when the current frame being viewed is at or beyond halfway through the set of frames currently in memory. The user can move or resize the main viewing window, change background color, and zoom the image while navigating behind the latest frame, but not when new frames are being displayed as soon as received ("true movie mode"), as then confusion can arise as to which frame is intended to be changed. The user can use the Pause button to freeze the movie-mode display on the current frame in order to perform these functions.

Metafile viewing should behave as follows: On startup, the program should read the first frame and display it immediately. It should then continue reading (but not displaying) frames in the background until the number of frames specified by the in-memory limit is reached (or, of course, fewer if the metafile contains fewer than this number of frames). The navigation buttons then work within this set, except that, when located at the last in-memory frame, the '>' button results in reading one more frame (and discarding a frame (or frames) if necessary); the '>>' button steps to the last frame if not already located there, then continues reading new frames and deleting old frames until stopped by another button or by reaching the end of the metafile.

The rules of viewing parameter inheritance are as follows: Whenever the user views frames in number order, i.e. using '<', '<<', '>', or '>>' buttons, or in movie submode, each frame inherits the window size (unless the aspect ratio changes due to the frame size given in the file) and position, magnification, origin offset, and background color of the most recently viewed frame. If GoTo or '|<' is used to view a frame already viewed that is not the immediate previous or next frame in frame number order, then the parameters of that frame when it was most recently viewed will apply. If GoTo or '>|' is used to view a frame not already viewed (in a metafile), it should inherit from the highest- numbered frame in memory that is below the requested frame. If implemented as suggested below, the inherited values can also be stored with all of the unviewed frames that are skipped over.

When the frame size requested by a new frame command in the input metafile or socket is different from the value that would be inherited according to the above rules, the size in the header takes precedence.

The program should keep track of the order in which frames are viewed when the user applies any of the navigation controls. The size of this list should be a preference file item, with default 100 frames. Each time the '<' button is clicked with the CTRL key held down, the program goes to the next previous frame on this list, i.e. undoes a GoTo. Each time the '>' button is clicked with the CTRL key held down, the program goes to the next frame moving forward on this list.

There should be a command-line parameter that specifies the maximum memory to be used for storing frame commands. If using the mffm memory management routines (see below), there is really no reason this memory limit cannot be changed up or down (with some fixed minimum) during a run via a right-click menu item. There should be a default in the preferences file that will be used if the command-line parameter is not found at start up.

The following paragraphs refer to the interaction between frame movement and memory restriction:

Socket-still mode: We always want to request from the application just one more frame than the highest frame number ever viewed. The user can click Next and Previous and Goto ad lib anywhere within what is in memory, but a new frame is requested from the application only when the latest frame is viewed. When memory fills, lowest-numbered frames are thrown out until enough memory is freed up. This prevents creating gaps in the set of viewable frames. When the user tries to use the Back or Goto button to go back to a frame that has been removed, there should be an error beep.

Socket-movie mode. When in movie mode, either viewing current frames or navigating, mfdraw requests frames from the application until the memory quota is filled. When the currently viewed frame is more than half way through the cache of frames in memory, and a new frame is received, mfdraw (a) removes old frames if necessary to make enough memory to hold the new frame on the list of frames that are recorded in the index, (b) puts a pointer to the new frame in the index, (c) asks the application for another frame, and (d) (if in true movie mode) displays the new latest one.

Transition from still mode to movie mode: With the current frame positioned anywhere in the list of viewable frames, the user can get to movie mode via the 'M' key or the right-click menu.

The first thing that happens is that mfdraw requests a new frame from the application if one is not already pending. Then, if the currently displayed frame is not the most recent frame, mfdraw quickly steps through the frames in memory, as it does now, displaying each one for a brief time (using the preference file minimum-time item) until the displayed frame is the most recent frame. Then it changes the global switch to movie mode and continues as above. No other special action is needed. Note that if the user does not desire stepping through all the frames as described here, he/she need only press the '>|' button before entering movie mode.

Regarding navigation when reading from a metafile on disk: The new thing in this new implementation is that we want to be able to use arrow keys and Goto to access any frame in the metafile. (This would not apply when reading from a socket.) This clearly requires being able to reread frames that may have previously been discarded from memory.

IMPLEMENTATION

The current Linux version of mfdraw is implemented in C++ using the 'fltk' drawing package and the 'fluid' GUI designer. These packages have some shortcomings, particularly in the area of event handling. It is expected that use of Qt4 and its GUI designer component should be much more satisfactory.

It is anticipated that mfdraw should be implemented as a threaded application, where one thread handles the GUI and another is responsible for reading and interpreting the input file. This will allow image navigation to occur while reading ahead from the application in socket mode. GNR has written a set of metafile file management ('mffm') routines in C to handle metafile navigation and frame memory management. These routines are contained in the source file mffm.c and prototyped in the header file mffm.h. Detailed instructions for use are in the source file.

Most of the frame navigation, particularly with reference to the GoTo, can be implemented via an in-memory index that contains pointers to an area for each frame that contains its plotting commands and any other information needed for plotting, i.e. current scale, origin, and location. If the index is a simple array of structs, then information about frame x can be obtained quickly via index[x]->variable.

Regarding reading metafiles: When the current memory limit is full, delete frames in order from least recently viewed (see above) until the memory is below the limit. So, whether the user goes through starting at frame 1 and clicking Next over and over, or uses Goto and views frames in some crazy random order, what the code has to do is go to the index, see if that frame is in memory, if so, display it, if not, read it in and note in its index slot that it is now in memory and where it is.

One of the items in the index has to be the seek location of that frame in the metafile. If the user uses Goto to view frames out of order, it may be necessary in this situation to scan through the metafile from the last frame whose seek location is known to find the requested frame. The seek locations of the frames skipped over can still be recorded in the index so these can be found quickly when needed later. This overhead will be unavoidable (the other choice would be to read the entire metafile at startup), but will be relatively rare in practice.

Reading from a socket. We do not know in advance how many frames there will be. The program can allocate index space, but not frame command space, for some large number that is a compile-time constant, e.g. 10000. I suppose that if that number is ever exceeded, a realloc() would take care of it (this requires that there be no pointers in the structs that point to other items in the index--offsets must be used instead so they do not have to change when a realloc() occurs) or the program could keep the index as a circular list and when full throw away old index entries along with the frames they point to. (This situation requires further thought and discussion.)

Note that with mffm as currently written, the frame memory limit can be changed at any time, but not the size of the index. The next time a new frame is requested, it may result in throwing away a bunch of frames if the limit was lowered, or in throwing away nothing for a while if it was raised.

The natural structure for the viewing-order list is a circular buffer so that nothing except a pointer to the most- recently viewed frame ever needs to be moved. The only information that needs to be stored in this list is the frame number, which can be used to access frames via the master index. The list is updated when normal navigation is performed and is traversed, but not updated, when the CTRL arrow buttons are used. Clearly, there is some special startup code needed to handle backtracking when the list is not yet full. Old entries are simply written over when the list becomes full.

Some better error handling code is needed. In the current design, if the client program receives a "msgbutt" structure that does not conform to the documented ranges of its four one-character field (see /home/cdr/include/glu.h), it assumes that an error message has been written to stdout by some component of mfdraw. This was useful during testing when various unexpected errors occurred at unexpected times. This message is written to the client's stdout and an abexit 220 is generated. This protocol is satisfactory and can be kept. However, there are still many situations where no error message is generated and mfdraw simply exits, closing the socket to the client, leaving the user with no idea what went wrong. For example, this happens if a bad display is requested in the metafile header. Efforts should be made to capture as many errors as possible and return suitable messages to the client. Possibly the "msgbutt" structure can be modified to accept additional error codes.

INSTALLATION

mfdraw as distributed uses port 10090 to communicate with an online application. If this port is already in use on your system, change the variable MFDRAW_PORT in header file mfio.h and remake mfdraw. Then log in as superuser to carry out the following steps. Install mfdraw in any appropriate directory. Add the following line to the /etc/services file, changing "10090" to the new port number if necessary:

```
mfdraw          10090/tcp
```

If inetd is in use, add the following line to the /etc/inetd.conf file:

```
mfdraw  stream  tcp  nowait  <usr>  /<path>/mfdraw  mfdraw
```

replacing <usr> with the userid of the user under whose account mfdraw is to be run, and <path> with the full path to the mfdraw executable file. Then send signal 1 (-HUP) to the inetd process to make it read the revised inetd.conf file.

If xinetd is the startup program, instead of editing /etc/inetd.conf, add a suitable entry to the /etc/xinetd.d directory. The following is typical:

```
------Start xinetd mfdraw configuration file------
# description:  The server for NSI metafile graphics
# New service, 3/25/05, GNR

service mfdraw
{
        socket_type             = stream
        protocol                = tcp
        wait                    = no
        user                    = cdr
        server                  = /usr/local/bin/mfdraw
        server_args             = -display :0.0
        port                    = 10090
        log_on_success          += USERID
        log_on_failure          += USERID
        disable                 = no
}
------End xinetd mfdraw configuration file------
```

DIFFERENCES FROM CURRENT VERSION

Here are the main reasons a new version is needed:

(1) By switching from fltk to Qt4, and programming according to Qt4 guidelines, mfdraw should run on any operating system supported by Qt4, in particular, MS Windows.

(2) Ability to navigate freely anywhere in a stored metafile.

(3) Ability to open more than one window under control of metafile commands.

(4) Implement metafile commands not implemented in the current version, e.g. color indexing

and others.

(5) “Previous view” navigation as well as “Previous frame” navigation.

(6) Read binary as well as ASCII metafiles. (Probably more frames can be held in memory if saved as binary metafile commands rather than class structures used in present code.)

(7) Multithreaded approach will give a smoother user experience, especially on multicore hardware.

(8) Capacity limitation by memory amount rather than number of frames.

(9) Better error reporting.

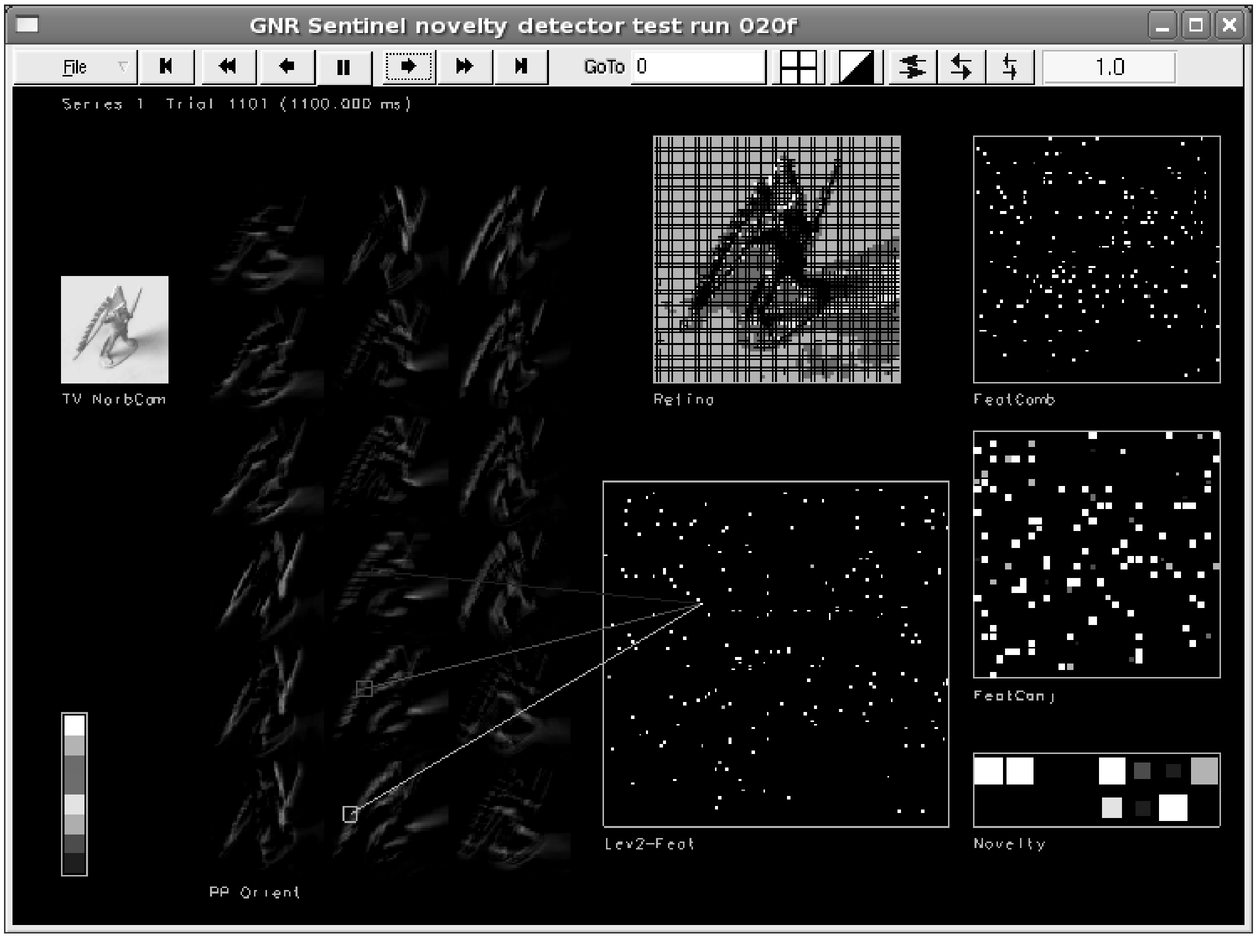
GNR Sentinel novelty detector test run 020f
File
GoTo 0
1.0
Series 1 Trial 1101 (1100.000 ms)
TV NorbCam
Retina
FeatComb
FeatConj
Lev2-Feat
Novelty
PP Orient

Figure 1. Sample mfdraw screen illustrating toolbar and frame with lines, rectangles, bitmap images, and lettering.