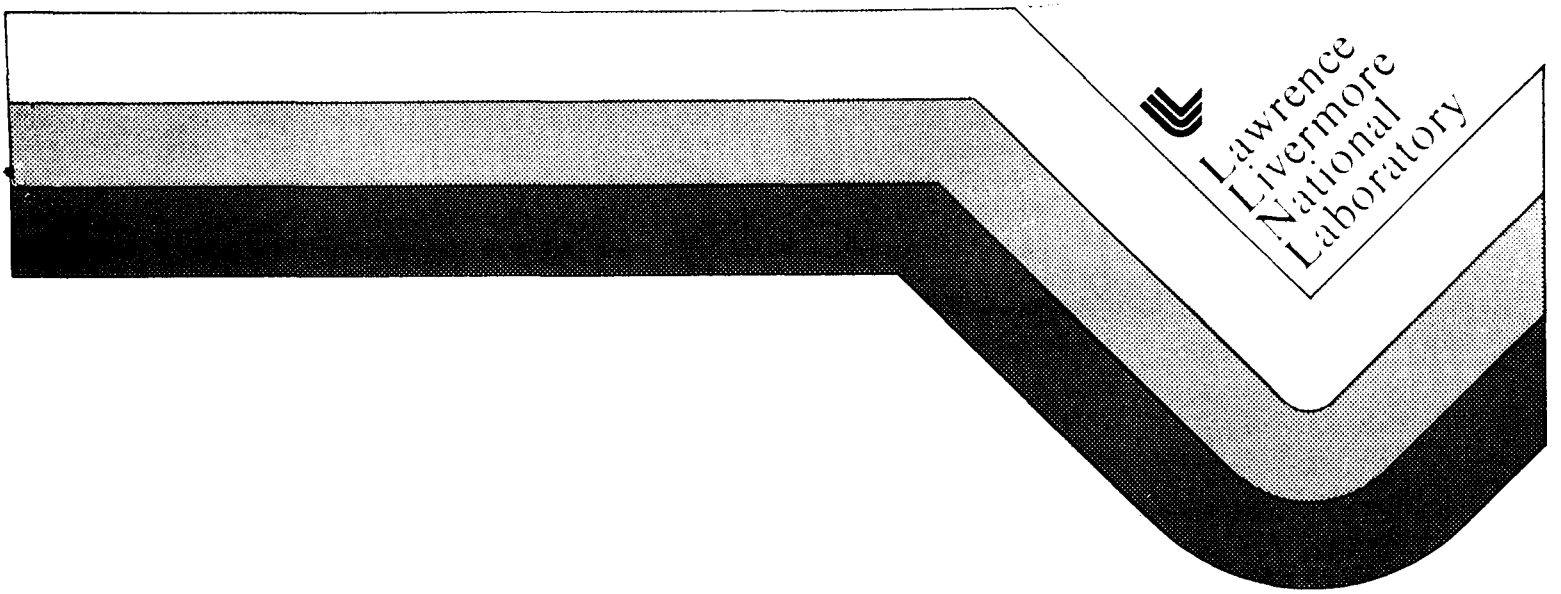
UCRL--21273

DE90 008525

PIXAR DATA VISUALIZATION TOOLS OVERVIEW

M. C. Miller

February 1990



Final Report

Mark Miller

1.1 Introduction

The goal of the work of the 1989 year was divided into three main areas.

- Provide assistance in the creation of visualizations used in the analysis of data from various experiments.
- Enhance and put the finishing touches into a visualiztion tool known as Slicetool.
- Provide new tools for users to become easily acquainted with and start using the Pixar visualization tools.

1.2 Programmatic

During the first few months of 1989, various experimentors required assistance in creating visualzations, particularly video tapes of film loops of volume renderings. In fact, the tremendous need for this work is what motivated the development of tools to automate this process which were the focus of the second half of 1989 and are described in the last section. In the meantime, while these tools were under development, many visualiztion tasks arose which required special attention to complete prior to certain deadlines. All of this work, called Programmtic work, was done in the first few months of 1989.

1.3 Slicetool Enhancements

Slicetool allows a user to interactivly analyze a volume of data by slicing the volume along arbitrary planes specified with a mouse and displaying the resulting data in normal view[1]. In its initial version, only a sweep based mode of interaction was avialble in which the slice

[1]Please see the final report of 1988 for a longer description of Slicetool

1.4.2 Get, Makevol and Associated Libraries

Get and Makevol are the two main interfaces from the outside world to the Pixar. Get allows a user to read an image stored in any common LLNL data format, such as View, Ras or Raw (binary dumped) into the framebuffer of the Pixar. ¿From there, it may be manipulated via any of the Pixar tools and software libraries. Makevol allows a user to create a data object on the Pixar known as the Pixar Volume. Since all the the volume visualization tools on the Pixar require a Pixar Volume as input, this one tool allows users to perform all their own visualizations completely unassisted. In the development of Get and Makevol, libraries to manipulate filenames and to run the Pixar via the UNIX shell from within a C program were also created.

Pixar Data Visualization Tools Overview*

Mark C. Miller

September 30, 1989 (modified December 19, 1989)

Abstract

This article explains how to perform basic visualization operations on the Pixar. The three main operations performed on the Pixar are interactive planometric rendering, video film loop production, volumetric rendering and some simple image processing. All these and other high-level visualization operations use a common data format for storing sets of images known as the Pixar Volume which is created using Makevol. A number of other low-level tools facilitating visualization operations are also introduced and summarized.

1 Introduction

Currently, the Pixar Image Computer[1] at the Lawrence Livermore National Laboratory (LLNL) is used mostly for interactive data analysis, video film loop production, image processing, and volumetric rendering. These tools, and with few exceptions all other data visualization tools on the Pixar, utilize a common data format for storing sequences of images known as the *Pixar Volume*. A Pixar Volume is created using the command **makevol**. From the user's standpoint, **makevol** is the most important Pixar command because it provides the interface from existing data formats, like Ras [Gran88], View [Bras88], or raw (binary dumped) data, to the Pixar. Once the data to be examined has been converted to a Pixar Volume using the **makevol** command, one can use the following tools to visualize it.

- Cubetool and Slicetool: These tools offer interactive data analysis capabilities based on displaying a slice of the data along a plane specified by the user with a mouse. Cubetool, [PixIID] works only with planes orthogonal to the primary axes while Slicetool works with arbitrary planes. Both run in X windows. Cubetool also runs in Suntools[2].
- Loop: This is used to display film loops, [PixIS], for videotape production. One of the Pixar's most important capabilities is that it has enough memory to store about 17 seconds

*Work performed under the auspices of the U. S. Department of Energy by the Lawrence Livermore National Laboratory under Contract Number W-7405-ENG-48.

[1] *Pixar*, *Pixar Image Computer*, and *Chap* are trademarks of Pixar.

[2] *Suntools* is a trademark of Sun Microsystems, Inc.

of 256 x 256 monochrome video images, and enough speed to replay them at 30 frames per second!

- Pixcolor: This tool allows the user to easily manipulate and interact with the color tables of the Pixar. It also runs in X windows. An alternative color command, pixlut, is more limited but does not require any window system.
- Volumetric Rendering, [PixIIV], can be performed efficiently on the Pixar only by users with some knowledge of its operation. While the Pixar is relatively fast for volumetric rendering, it is not user friendly and other friendlier and more general rendering algorithms, Vrender [Gran89] and Cell-Tracer [Cabr89], are available on conventional workstations.
- In addition, there is a variety of additional Pixar commands to perform a host of shell level, [PixIS], image manipulation operations and to tailor visualization operations as desired.

In order to use these and future Pixar data visualization tools effectively, the user should have a good understanding of the some of the Pixar's basic commands and hardware. As the Pixar is attached to a UNIX[3] host computer (a Sun workstation), this report will describe Pixar commands in a UNIX style. A familiarity with UNIX is assumed.

In this report, we discuss the UNIX environment parameters affecting operation of the Pixar, some basic UNIX shell commands to perform simple visualization tasks, the basic aspects of the Pixar hardware necessary to successfully perform visualization operations and the high-level visualization tools currently available on the Pixar.

2 Basic Knowledge for using the Pixar

As of this writing, the Pixar is hosted by a Sun called "pixczar", and all operations using the Pixar must be done while logged onto this machine.

2.1 Setting the UNIX Environment

Before using the Pixar, some UNIX environment variables, [PixIIP], and shell variables must be set. Do this by putting these lines in a *.login* file on the Pixar host computer.

- *setenv MANPATH /usr/local/man:/usr/man:/usr/pixar/man*
  Sets the path variable for the *man* command for commands related to the Pixar
- *setenv PIXPATH /usr/pixar/demo/pix:/u3/millerm/data:/u3/grant/data*
  Sets the path variable for where to search for pictures called up by the **gt** command.
- *setenv FBDEFS "pxr0 1024 8192 0 pxr0"*
  Defines the default frame buffer, its name and size, used by most of the data visualization tools.

[3] *UNIX* is a trademark of AT&T.

and the following shell variable:

- *set path = ($path /usr/pixar/vip/host/bin /usr/pixar/host/bin /usr/pixar/demo/bin /u3/millerm/bin)*
  Used by the shell to find the executable codes for the requested commands.

These environment and shell variables are useful for execution of Pixar commands as well as obtaining information about them.

2.2 Pixar Help Facility

All commands for the Pixar have a simple help facility associated with them which gives some idea of the options available for the command and how to use them. To obtain this help information, simply type

<command> -help

or just

<command> -

For some commands the above two variations for obtaining help produce different results, so one may want to try both. For the options listed in response to the help command, those in brackets [ ] indicate default values. Remember, when having difficulty remembering the options for a command, **<command> -help** is very helpful. In addition, the standard UNIX *man* facility is also available for all the Pixar standard commands. However, since no *man* pages have been produced for LLNL written Pixar commands, the only way to currently get help information about them, aside from this document, is **<command> -help**.

2.3 Basic Commands

While a host of Shell level, [PixIS], and C callable, [PixII], functions are available for the Pixar, only those most frequently used are summarized below. These and others are described in detail in the Pixar documentation. The most important and useful things to know about the Pixar are the following commands:

- Initialization and Resetting Commands:
  - **pixinit**: Used to initialize all the Pixar hardware. This command will initialize all the Pixar hardware except clear the frame buffer. It is sometimes needed to get out of serious error conditions when nothing seems to be working right.

- **chmap -i**: Used to initialize the Chap (Channel Processor in the Pixar; see Section 3). **chmap -i** is more commonly used than **pixinit** to get out of less serious error conditions, because **chmap -i** only initializes the Chap and is faster. Frequently, the Pixar software fails to free up resources on the Chap especially if Pixar routines are interrupted by a *ctrl-z* or *ctrl-c*. When error messages like the following appear:

  Chad Error: Couldn't allocate resource on the Chap.

  execute the command **chmap -i** to reset the Chap.

- **clr**: Used to clear (set to zero) the Pixar frame buffer. This command can be used to clear the entire frame buffer, or only a section of it, to any specified color.

- Display I/O Commands:

  - **gt**: Used to get a Pixar formatted image file into the frame buffer for further processing and display. This is one of the most frequently used commands and it has several options to select specific channels, set the image offset within the frame buffer or clip the image to a specified window.
  - **gtinfo**: Used to get the header information of a Pixar formatted image file.
  - **get**: Used to get an arbitrary data formatted image, such as Ras, View or raw data, into the frame buffer. This command also has several options to set the image offset within the frame buffer and set various parameters describing the data for raw reads.
  - **sv**: Used to save an image from the frame buffer to Pixar's format. This command has many options including those to save only a particular portion of the frame buffer, a particular set of channels, in dumped mode or run-length encoded, and to save a label with the image header.
  - **ramp**: Used to display color bars. This command allows one to specify any rectangular region of the frame buffer in which a color bar is to appear simply by specifying the four vertices and the color to be associated with the left and right or top and bottom edges. The Pixar then interpolates between the colors on these edges to produce any desired color bar.
  - **loop**: Used to playback a film loop. This command has several options to specify image size, location, number of frames, speed of replay and zoom.
  - **video**: Used to initialize and set up the video controller. The **video** command is used to perform a number of functions with the video controller including zooming the display, displaying a different portion of the frame buffer, displaying only a specific channel of the frame buffer and setting the display hardware to display on an alternative device. The command **video -ntsc** sets the video controller synchronization signals for display on an NTSC monitor. The command **video -hidef** sets the video controller synchronization signals for display on the Pixar or similar monitor.

- Volume Commands

  - **makevol**: Used to create a Pixar Volume from Ras, View or raw data. This command has a number of options including those to specify input and output data volume parameters.

- **vint**: Used as an interactive environment to create Pixar volumes and film loops, and to perform volumetric rendering and other image processing operations.
- **expose**: Used to facilitate use of **vint** by providing a simple interface for making sequences of **vint** commands called *vint scripts*.

It is recommended that the reader also consult [PixIS,PixII] to see the additional shell level commands for image manipulation Pixar has provided.

2.4 File Naming Conventions

The two commands which work with non-Pixar data, **makevol** and **get** assume the following file naming convention.

<path><prfx><sepc><numr>.<extn>

where

- **<path>** is the directory path to the specified file on the disk.
- **<prfx>** is the prefixed file name for a sequence of files or just the file name without the extension.
- **<sepc>** is a field separator character to separate the prefix from the number.
- **<numr>** is a number used to distinguish different files in a sequence or group.
- **<extn>** is the file extension following the '.'.

Any one of these, except **<prfx>**, can be an empty string. Any **<numr>** string is valid. Valid separator characters are

\- , : ; > ! + = ¡ ([{ .

If no '.' exists in the filename, it will be assumed that no extension exists. If there is more than one '.' in the filename all the characters after the *last* '.' are taken to be the extension. In addition, the following file extensions are assumed

- *.vol, .cube,* and *.loop* indicate files containing Pixar Volumes.
- *.sdt* and *.spr* indicate files containing the standard View signal data and signal parameters.
- *.ras* indicates a file containing Ras data.

- *.dat* indicates a file containing raw or dumped data where the entire data set is stored pixel by pixel or voxel by voxel with no coding.
- *.vnt* indicates a file containing a sequence of commands for **vint** known as a *vint script.*
- *.x* indicates a file containing sequence of commands for **expose** known as an *expose script.*

3 Pixar Hardware Overview

The Pixar is a special purpose image manipulation processor which is composed of three main hardware elements:

- The main memory called the *frame buffer.*
- The central processor called the *Chap* or *channel processor.*
- The video controller and its associated display device.

Although the Pixar is designed to handle as many as three Chaps, 48 megabytes of frame buffer memory and two video controllers, the current LLNL configuration utilizes only one Chap, the full 48 megabyte frame buffer and only one video controller/display pair. Each of these elements is described in the sections below in more detail.

3.1 The Frame Buffer

With few exceptions, all the Pixar commands and library routines operate on data contained in the frame buffer, which is usually in the form of a single image or a set of images making up the slices of a Pixar Volume or the frames of a film loop. Since the Pixar was originally designed as an image manipulation engine for fast compositing and matting of images, a great deal of terminology surrounding the Pixar is derived from image processing literature. For example, individual locations, or words, within the Pixar's main memory are referred to as *Pixels* and each pixel is said to be composed of a *Red, Green, Blue* and *Alpha*[4] component. Following the image processing terminology, the physical size of the main memory is often quoted in pixels rather than bytes. The Pixar's main memory provides 8 mega-pixels of storage arranged as a two dimensional clip-board like storage area called the *Frame Buffer* with a width of 1024 pixels and a height of 8192 pixels.

Pixel positions are indexed with x specifying the horizontal position and y specifying the vertical position where $0 \leq x \leq 1023$ and $0 \leq y \leq 8191$ with pixel locations $(0,0)$ and $(1023,8191)$ being the upper left and lower right corners of the frame buffer.

[4]Although red, green and blue are well established components of a color image, since the alpha component is derived from recent developments in image synthesis, it may be unfamiliar. In image synthesis, the alpha component is most often used to represent the fraction of the pixel area covered by the associated color. There, the alpha component is generally used to composite together, or matte, separately synthesized images. Here, just think of the alpha component as another channel for data.

It is important to think of the frame buffer as a two dimensional, clip-board like image memory with pixel positions running from $x = 0$ to $x = 1023$ and $y = 0$ to $y = 8191$.

For example, in order to pin-up a set 4" × 4" color glossy prints from a recent trip to Yosemite on a 16" × 64" cork clip-board, pin the first print in the upper left hand corner of the clip-board. Then place the next print directly to its right, and the next, and the next so that 4 prints would be side-by-side across the top of the board. Then move down one *row* and start by pinning the 5^{th} print directly below the first print of the row above with three more to its right to complete the 2^{nd} row, and so on until the board is filled or there are no more pictures. The Pixar's frame buffer stores images in exactly the same manner.

The only difference between the Pixar's frame buffer and the cork-board example is that if the prints were black and white images, that is single channel, the Pixar would also allow them to be stored with 4 prints directly on top of each other in the red, green, blue and alpha channels of the corresponding pixels. This is allowed because the Pixar is capable of accessing each of the pixel channels independently. Thus, given a short film of 62 black and white (single channel) 320 × 256 frames of a recent free-fall sky-dive off El Capitan, the Pixar would store the first 4 frames in frame buffer locations $0 \leq x \leq 319$ and $0 \leq y \leq 255$ with frame 0 in the red channel, frame 1 in the green channel, frame 2 in the blue channel and frame 3 in the alpha channel as shown in Figure 1.

Frames 4-7 would be stored successively in the rgba channels of the pixel locations directly to the right, namely $320 \leq x \leq 639; 0 \leq y \leq 255$. Frames 8-11 would be stored successively in the rgba channels of pixel locations $640 \leq x \leq 959; 0 \leq y \leq 255$. After frames 0-11 have been stored in this way, there is still some space to the right of the last stack of 4 frames, namely locations $960 \leq x \leq 1023; 0 \leq y \leq 255$. Frames cannot be stored in this space because they would have to be cut to get an exact fit. With only one exception, the Pixar never cuts images to fit them into the frame buffer. So, to continue the example, we are able to store side by side 3 stacks of 4 frames to make one row of 12 across the frame buffer. The Pixar then repeats this process for the next 12 frames and so on until it is done. Since there are a total of 62 images in the film, the Pixar requires $\frac{62}{12} = 5\frac{2}{12}$ rows of 12 frames. Thus, the 6^{th} row is only partially full with 2 images in the red and green channels of the leftmost stack of frames of the row. The area of the frame buffer used to store the data is 3 images wide, which is 3 × 320 = 960 and 6 images high, which is 6 × 256 = 1536 specified by $0 \leq x \leq 959; 0 \leq y \leq 1535$. A general algorithm for computing the area of the frame buffer occupied by a set of images is given in a later section.

One additional note about the frame buffer is that it is divided into 32 pixel square regions called *tiles*. Thus, the 1024 × 8192 pixel frame buffer contains 32 × 256 tiles numbered from 0 to 8191. Tile 0 is in the upper left corner of the frame buffer. Tile 31 is in the upper right and tile 8191 is the lower right. Many Pixar commands require a *start tile* as an option in order to specify the location within the frame buffer of the associated image. When an image is loaded into the frame buffer using **get** or **sv** with an offset in y that is a multiple of 32, the start tile is just the y-offset specified to the **get** or **sv** command; *e.g.*, frame buffer position $(0, 32)$ is tile 32. For example, if some data is loaded into the frame buffer with an offset option of **-o 0 512**, the start tile would be 512. If the offset in y is not a multiple of 32, this association between the offset and start tile does not hold.

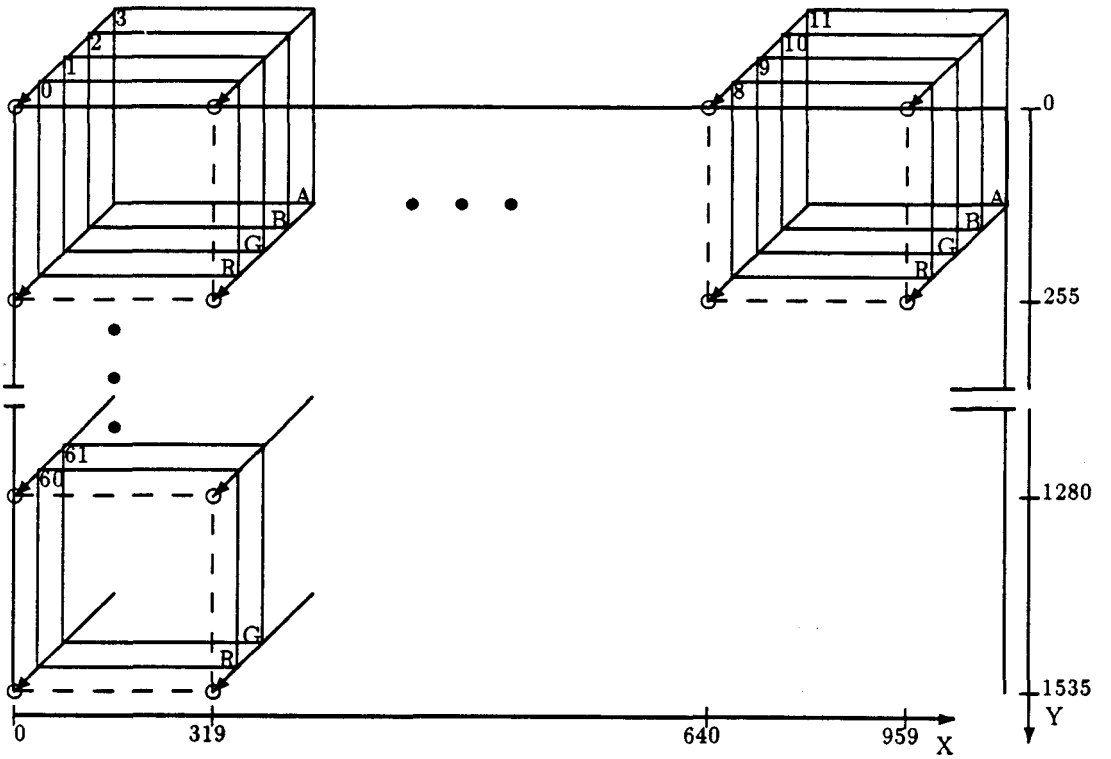


Figure 1: Pixar frame buffer organization (see text for description).

3.2 The Chap

The name Chap is derived from *Cha*nnel *P*rocessor. The Chap is the central processor of the Pixar and has a separate processor for each of the red, green, blue and alpha channels of the frame buffer. Basically, the Chap is four separate processors sandwiched together to act as one. It acts as a slave processor to the host computer. The host is responsible for allocating and freeing resources on the Chap, loading programs and data to and from the Chap, and initiating Chap execution. These functions are performed automatically by the Pixar software. However, frequently the Pixar software fails to free up resources on the Chap especially if Pixar routines are interrupted by a *ctrl-z* or *ctrl-c*. After running the same command several times, error messages saying something like

Chad Error: Couldn't allocate resource on the Chap.

may start to appear. When this happens, execute the command **chmap -i** to reset the Chap and clear its program and data memories as discussed above.

3.3 The Video Controller

The video controller is in charge of accessing those areas of the frame buffer that are to be displayed, and displaying them according to parameters set by the user. The display is 1024 pixel wide and 768 pixels high. In its default state, the video controller displays frame buffer pixel locations $0 \leq x \leq 1023; 0 \leq y \leq 767$. In this state, the upper left and upper right corners of the frame buffer correspond to the upper left and upper right corners of the display. However, the video controller can be set to display any 1024×768 portion of the frame buffer. The video controller is capable of hardware image magnification via pixel replication and a number of other features which are important for display purposes. The video controller is most often used during video productions for displaying specific portions of the frame buffer and/or specific channels as well as changing the video synchronization signals for display on alternative devices.

3.4 An Example Visualization Scenario

Suppose a VHS video tape, which has a display size of about 640×480, is to be made from the El Capitan sky-dive film. Assume the data for the film is stored in the file *skydive.loop*. The following sequence of commands will perform the desired visualization and result in the data being stored in the frame buffer as shown in Figure 2.

1. **clr -w 0 1023 0 767**

2. **gt skydive.loop -o 0 768**

3. **video -ntsc -gamma 2.3**

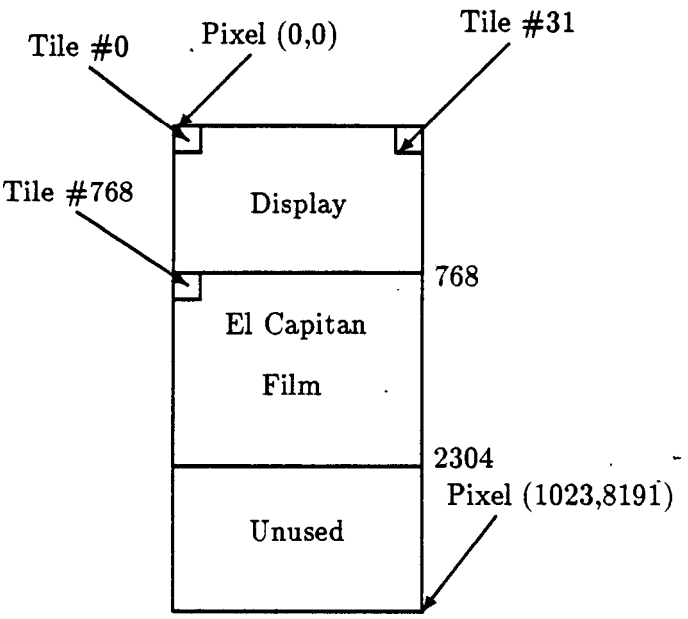


Figure 2: Example of data storage in the Pixar frame buffer.

4. **loop -s 320 256 -frames 62 -vsize 640 480 -o 0 768 -bw -kbd -rock**

The sequence will appear in the upper left corner of the NTSC display. This is due to the fact that the 320 x 256 images cannot be zoomed by an integer factor for the given display size of 640 x 480. For this reason, when display onto NTSC is desired, image sizes of 320 x 240 are best. They can be zoomed by a factor of 3 for display on the Pixar's monitor or a factor of 2 for display on an NTSC device. If, in addition, a tool requiring the start tile as an option was to be used to visualize this data, a start tile of 768 would be specified.

4 Tools

Before the visualization tools are described in detail, some general comments on using the Pixar are necessary. First, when running visualization tools on the Pixar it is best to work at the console in the visualization lab. The Pixar monitor and video taping equipment are there. In addition, all tools requiring the mouse for control of the Pixar work best when operated from the console.

Second, if two users attempt operations which use the Pixar hardware, the first user will be allowed access while the second user will get error messages such as

/dev/chap0: Device Busy

or
/dev/video0: Device Busy

Error messages like these simply indicate that another process is currently using the Pixar hardware. Most of the tools are designed so that two processes cannot access the hardware simultaneously. In that way, two users do not interrupt each other's work, or one user does not interrupt two different processes. On occasion, however, the device busy error messages appear because a prematurely stopped process, stopped by *ctrl-z* or ctrl-c, has not been removed from the host's process table. In this case, the Pixar hardware used by the process is sometimes left in a busy state waiting for a *resume* or *termination* signal from the host. Then, since the process causing the busy state is no longer needed, it can be killed by the command **kill -9 <pid>**, where *pid* is the process identification number of the offending process (see UNIX documentation on "ps" command).

Third, all the Pixar tools utilizing X windows are such that the mouse cursor must be inside the associated window in order to perform the operations. Generally, they are also written so that all operations involve the mouse buttons. In this way, the mouse may be swept across or out of the window without changing the current display parameters. Although, operations with all tools must *begin* inside the window, some may *end* outside the window.

Fourth, all the Pixar visualization tools and many other programs utilize a common storage format for data known as the Pixar Volume. A Pixar Volume, like a generic data volume, is just a sequence of slices of data such that when it is loaded into the frame buffer, **each slice lies on an integral tile region.** This simply means that there are no tiles in the volume which are partially occupied by one or more slices of data. More specifically, this means that slices have dimensions and offsets which are multiples of 32. Thus, slices which are, for example, 256 x 256 at positions $(0,0),(0,256),(0,512),(0,768),(256,0)...$ can be part of a Pixar Volume while slices which are 256 x 256 at positions $(16,0),(16,256),(16,512),(16,768),(272,0)...$ or which are 240 x 240 cannot be part of a Pixar Volume. Thus, to compute the frame buffer area occupied by a volume of N images W pixels wide and H pixels high use the following algorithm.

$$\begin{aligned}
\text{Let } W' &= 32\left\lceil \frac{W}{32} \right\rceil \\
\text{Let } H' &= 32\left\lceil \frac{H}{32} \right\rceil \\
\text{Let } M &= \left\lfloor \frac{1024}{W'} \right\rfloor \\
x_{\text{size}} &= MW' \\
y_{\text{size}} &= \left\lceil \frac{N}{M} H' \right\rceil \quad \text{for color images} \\
y_{\text{size}} &= \left\lceil \frac{N}{4M} H' \right\rceil \quad \text{for black and white images}
\end{aligned}$$

where $\lceil\ \rceil$ and $\lfloor\ \rfloor$ are the ceiling and floor functions respectively and the occupied pixel positions are specified by $0 \le x \le x_{\text{size}} - 1$ and $0 \le y \le y_{\text{size}} - 1$.

4.1 Color Maps and Color Bars

The color maps of the Pixar can be set using the color tool program called **pixcolor** provided by LLNL. The command **pixcolor** runs in X windows on the host computer. The **pixcolor** menu will appear with the left mouse button down while in the window. To adjust the color map of the Pixar, set the desired color and the port it to the Pixar. When not using X windows, color maps can be set in a less friendly manner by the command **pixlut** which displays a limited list of color maps from which to choose. Beware, however, that pixcolor modifies only the lower 512 entries of the total 1024 available in the Pixar's color table. Unless one is programming with the Pixar color tables, this is not a concern.

Once the Pixar color maps have been set, a color bar may be displayed with the command

ramp -w *xmin xmax ymin ymax* **-cl** *left_color* **-cr** *right_color*
ramp -w *xmin xmax ymin ymax* **-ct** *top_color* **-cb** *bottom_color*

where *xmin xmax ymin ymax* specify the rectangle bounding the color bar and *left_color right_color* specify the left and right end colors, respectively, for a horizontal color bar and *top_color bottom_color* specify the top and bottom end colors, respectively, for a vertical color bar. With this command, one can place any desired rectangular color bar anywhere within the Pixar's frame buffer, even in locations which are not currently being displayed by the video controller. Generally, one would specify *xmin xmax ymin ymax* so that the color bar is placed in the portion of the frame buffer currently being displayed by the video controller. When using this command, be careful to ensure that important data in the frame buffer is not over written. For example, if one is using Cubetool with data located at start tile 512, *ymax* should be less than 512. Also, in order to display a color bar during a film loop, a separate color bar must be *painted* into each frame of the loop!

4.2 Vint, Expose and Libvint

The tools **vint** and **expose** [PixIIC] provide a general environment for high-level, visualization programming of the Pixar. **Vint** is an environment for performing sequences of operations on the Pixar such as building Pixar Volumes, volumetric rendering, or limited image processing. Running the Pixar from the **vint** environment is similar to running Pixar from the UNIX shell except that there is greater flexibility of image operations in **vint** and it is a faster interface to the Pixar than the UNIX shell. **Vint** accepts commands entered by the user at the terminal and executes them on the Pixar. It is invoked by the shell command **vint**. However, rarely is **vint** used in its interactive environment. The usual manner of using **vint** is to create a script of commands for **vint** to follow and then pass this script onto **vint** in a non-interactive fashion. For this reason, Pixar also provides an associated command called **expose** which reads an *expose script* to create a sequence of commands to be executed by **vint**. The problem with **expose** is that it is very limited and not easy to use. Nonetheless, the conventional method for running **vint** is to write an expose script, such as *makevol.x*, and then perform the desired **vint** operations by executing the command

expose makevol.x | vint.

Expose reads the script in *makevol.x* and generates a sequence of commands which are piped to **vint**. In summary, **vint** and **expose** provide a working but difficult-to-use environment for visualization programming of the Pixar. A better way of utilizing **vint** is to use a LLNL-written C library called *libvint*.

Libvint is a library of routines which allow the user to easily write C programs which create the scripts that are passed onto **vint**. Libvint was written to conform to Pixar's standard for software libraries. It has a calling sequence similar to LibChad and LibPixar. With an understanding of **vint**, a user can bypass using **expose** to call **vint**, and instead call **vint** routines from C programs. Users who wish to do some simple high-level **vint** programming and who know how to program in C, are encouraged to look at the example program in */u3/millerm/src/libvint/testvint.c* instead of using **expose** and **vint** directly.

4.3 Get

Get is a general purpose program for getting data into the frame buffer of the Pixar. **Get** understands View, Ras and raw data. When using some unfamiliar data format for storing and encoding data, the simplest way to enable **get** to read it is to save the data in dumped or raw mode where the entire data set is stored on a pixel by pixel or voxel by voxel basis without coding using any of the standard data types (byte, unsigned short, short, integer, float, long integer, and double). Another option is to add the new data format to **get** so that **get** will understand the format in the future. This can be done easily by modifying the code in */u3/millerm/src/get/get_data.c* for reading data, and rebuilding **get** by running the command **make get**. It is only necessary to write a routine that will read the new format off disk, decode it, and put it into an array of one of the data types listed above.

Get has many options, most of which are used for specifying a raw data read. When working with data formats known to **get**, these options are not required. Once the image has been read into the frame buffer using **get**, it can be manipulated more completely with shell commands or **vint** and then saved in the Pixar format using **sv**.

4.4 Makevol

Makevol is the main interface between data from the outside world and the Pixar. **Makevol** is used to create Pixar Volumes from View, Ras or raw data sets. **Makevol** uses the *libvint* library, to create a pipe to **vint** and then generate the sequence of **vint** commands necessary to perform the desired operations. It has a number of options, none of which are required in order for it to run correctly when using any of the data formats known to **get**. Also, because **makevol** calls **get** to read the slice images into the frame buffer, any data formats added to **get** as described above, will also be available to **makevol**. If one is using raw data, one must specify several of the data parameters on the command line. Because **makevol** must do a lot of image operations to make a volume, it can take a while to complete. There are some important ways it can be sped up.

First, it is fastest to make a volume using data stored locally. Since there are several non-

local disks mounted on the Pixar, it is easy to make the mistake of calling **makevol** on non-local data. This is very likely to be more time consuming than copying the data to a local disk (by **ftp**) and then running **makevol**, because under certain conditions **makevol** will read the data twice. In order to convert data to Pixar's format, each of the slice images, in general, needs to be intensity scaled. In order to scale all the slices by the same amount, **makevol** must obtain a global minimum and maximum data value for the entire volume. Thus, if the range information is not provided on the command line to **makevol** using the **-range** option, it will have to read the volume off disk twice; once to scan for the maximum and minimum and once to create the volume[5] Furthermore, finding the maximum and minimum in the data is a time-consuming process all by itself. However, while the dimensions of the input data significantly effect the speed with which **makevol** can create a Pixar Volume, the dimensions of the output volume do not.

Makevol defaults to creating 256^3 volumes but can create a volume of any legal dimension. Generally, the size specified depends on what is to be done with the data. If one plans on doing volumetric rendering, one should probably make a small volume at first, say 64^3 to 128^3 which can be used to tune the volumetric rendering to obtain the desired results. A larger volume can then be made for the final images. If one plans on running Slicetool or Cubetool on the volume, a size of 256^3 is fine. If one is making a film loop, a size of 320×240 by the number of frames is recommended. This particular size makes the loop suitable for display on the all the visualization lab devices, while other sizes may not.

4.5 Filmloops

Film loops are probably the second most frequent activity performed on the Pixar. Basically, a sequence of frames must be made into a Pixar Volume[6] and then loaded into the frame buffer using **gt**. Then the shell command **loop** is used to display the image sequence. **loop** has many options which are not, at first, self explanatory. To run **loop**, specify some or all of the following:

- [**-s %d %d**]: The size of the frames in the sequence.
- [**-frames %d**]: The number of frames in the sequence.
- [**-o %d %d**]: The offset of the sequence within the frame buffer.
- [**-bw**]: The sequence is black & white (single channel), otherwise a color sequence is assumed.
- [**-kbd**]: You wish to use the keyboard to control playback otherwise the mouse is assumed. When running X windows, always specify the keyboard since for the mouse, it assumes Suntools.
- [**-rock**] You wish **loop** to rock back and forth through the sequence. Otherwise, it will only loop in a forward direction and jump back to the beginning.

[5] **Makevol** was designed in this way because data sets are becoming so large that non-slice-by-slice operation is impractical.

[6] In general, the **loop** [PixIS] command does not require a Pixar Volume as input. This command will work with a sequence of images of any dimension. However, since **makevol** currently provides the best method for loading a sequence of images into the frame buffer, it is also used to create film loops.

- [**-vsize %d %d**]: The size of the display that the sequence will be played back on. The Pixar display is the default. However, for a different display, one needs to specify its size; a common one is "-vsize 640 480" for playback onto videotape.
- [**-speeds %f %f %f %f**]: The display rates to be used during playback in fractions of seconds. It defaults to 1/24, 1/12, 1/6 and 1/3 seconds per frame.
- [**-zoom %d**]: Specifies a zoom factor to be used during playback.

For example, to display something like the Yosemite El Capitan sky-dive film loop on an NTSC device (which has an approximate display size of 640×480) so that it plays forwards and backwards, use the following commands

gt skydive.loop video -ntsc -gamma 2.3 loop -s 320 256 -frames 62 -vsize 640 480 -bw -kbd -rock

4.6 Interactive Planometric Analysis: Cubetool and Slicetool

There are only two *interactive* programs for analyizing 3D volumetric data. These are Cubetool and Slicetool. Both operate on commands entered by the mouse. Each allows the user to observe a plane of the data volume. Cubetool allows the user to view only planes orthogonal to the primary x, y and z axes, but displays all three simultaneously in orthographic and normal views. Slicetool is more general in that it allows the user to specify an arbitrary slice plane. After an initial analysis with Cubetool, it is more informative to analyze the data with Slicetool.

There are two important differences between Cubetool and Slicetool. Cubetool can really only work with $256 \times 256 \times N$-sized data volumes, where $0 < N < 211$ data volumes while slicetool can work with $K \times M \times N$ data volumes provided K and M are multiples of 32 and N is arbitrary. Since Cubetool was originally written by Pixar for Suntools and then converted to X windows by LLNL, the reason it is not able to handle more than 211 slices is not known. Both Cubetool and Slicetool assume the starting tile for the volume of data is 768. If this is not the case, it must be specified on the command line. However, since Cubetool uses the first 512 tiles for its display, data volumes should be loaded in with a y offset of at least 512 or Cubetool will over-write the volume data in the frame buffer. Since Slicetool uses the first 768 tiles for display, data volumes should be loaded in with a y offset of at least 768.

4.6.1 Cubetool Options and Mouse Commands

Cubetool, [PixIID], displays an orthographic view of the data volume with the front, top and right side faces visible. The data on these facing planes is shown both in the orthographic view and in normal, face-on view. East/west motions with the mouse move the right-facing side of the cube in and out. North/south motions move the front facing side of the cube in and out. North/south motions with the mouse also move the top facing side of the cube up and down. The command **<cubetool> -help** produces the following output.

- Mouse commands:
  - move front plane : middle button down and back/forth motion.
  - move right plane : middle button down and left/right motion.
  - move top plane : left button down and back/forth motion.
  - rotate 90 : click left button.
  - quit : click right button.
- Options:
  - [**-start %d**] set cube starting tile [768]
  - [**-r[ows] %d**] set number of rows in volume [256]
  - [**-c[ols] %d**] set number of columns in volume [256]
  - [**-s[lices] %d**] set number of slices in volume [211]
  - [**-offset %d**] set y offset [0]
  - [**-dest %d %d**] set x y of orthographic projection [0,0]
  - [**-lib %s**] set path to micro-code [/usr/pixar/demo/lib]
  - [**-orient %s %s**] top, front faces from XY0,XYN,XZ0,XZN, YZ0,YZN [XY0 XZN]
  - [**-pixar**] no Pixar computer connected
  - [**-replay %s**] accept mouse input from named file
  - [**-i %s**] accept mouse input from named file
  - [**-record %s**] save mouse input to named file
  - [**-o %s**] save mouse input to named file
  - [**-** [%s]] print usage manual
  - [**-?** [%s]] print usage manual
  - [**-help** [%s]] print usage manual

These options are the same as described in the Slicetool options description. The **-orient** option allows one to specify the default orientation of the cube when it is loaded into the frame buffer. Actually, it does not really load the data in any differently, but instead interprets the mouse commands and traverses the data volume differently so that it appears to be in a different orientation. The **-lib** option should really never be bothered with. It allows one to specify an alternative Pixar library from where the cube traversal code is to be loaded to the Chap. The **-pixar** command allows one to specify that the Pixar is either not available, not connected or does not exist. This option is only used for debugging purposes.

4.6.2 Slicetool Options and Mouse Commands

Slicetool's user interface allows the user to interact with the data as though the data volume is a stationary cube and the slice plane moves about, tangent to a sphere surrounding the cube. The radius of the sphere and the point of tangency of the plane are controlled by motions with the mouse. The mouse motions represent east/west for right/left motion, or north/south for up/down motion. In sweep mode, motion to the west with the mouse sweeps the plane around the sphere to the west. Motion to the east with the mouse sweeps the plane around the sphere to the east and so on. In pivot mode, motion to the west with the mouse, pivots the plane to the west and motion to the east pivots the plane to the east.

The command **Slicetool -help** produces the following list of options for Slicetool.

- Mouse Commands:
  - Sweep Mode:
    - sweep plane left/right : middle button down and left/right motion
    - sweep plane up/down : middle button down and forward/back motion
    - sweep plane in/out : left button down and forward/back motion
  - Pivot Mode:
    - pivot plane left/right : middle button down and left/right motion
    - pivot plane up/down : middle button down and back/forth motion
  - Pivot Point Select:
    - select new point : click middle button
    - cursor u/d/l/r : middle button down and b/f/l/r motion
    - select point : click left button
  - Pivot/Sweep toggle : click left button
  - Quit : click right button
- Options:
  - [**-start %d**] Set Starting tile of volumetric data in frame buffer [768]
  - [**-s[ize] %d [%d %d]**] Set Size of data volume (SLICES, ROWS, COLUMNS) [256x256x256]
  - [**-rep[lay] %s**] accept mouse input from named file
  - [**-i %s**] accept mouse input from named file
  - [**-rec[ord] %s**] save mouse input to named file
  - [**-o %s**] save mouse input to named file
  - [**-minres %d**] Set minimum resolution reduction for progressive rendering [8]
  - [**-vtape**] Adjust display for recording onto videotape
  - [**-reverse[_mouse]**] Reverse the sense of the mouse motions

- [**-offset %d %d**] Set x y offset of slice/cube display (experimental still) [0,0]
- [**-color** [%s] [%f %f %f]] Set the color of lines on the user display
- [**-nodepth[que]**] Set for no depth queueing in user graphics
- - [%s] print usage manual
- **-?** [%s] print usage manual
- **-help** [%s] print usage manual

Since most of the options are explained in the help, only a few are described here. The **-starttile** option specifies the start tile within the frame buffer at which the volume data begins as is described in the hardware section on the frame buffer. The **-record** option allows a user to save the sequence of mouse commands to a file so that they may be replayed later with the **-replay** option. The **-o** and **-i** options, respectively, are synonymous with these. The **-minres** option specifies the minimum resolution the progressive renderer will use to perform progressive rendering. Actually, the number specified as an argument to this option is a resolution reduction factor. If, for example, this argument is 2, the progressive renderer will always start progressive rendering with $\frac{1}{2}$ resolution. If this factor is 8, the progressive renderer will start with $\frac{1}{8}$ resolution. This option gives the user control over the speed slicetool displays data. The higher the number, the faster slicetool will go. Its default value is 8 and this seems to work very well with most data. The **-vtape** option tells Slicetool to format the display for videotaping. This option adjusts the display so that the image data, cube outline and icon are displayed together in the upper left corner of the display. The **-color** options allows the user to specify the color of the lines used for the user graphics portion of the display. The standard View colors are provided and can be specified by name, like magenta or cyan, or the user can set the color by specifying the relative amounts of each of the red, green and blue primaries with the numbers between 0.0, for completely off, and 1.0, for completely on. The default configuration of the user interface allows the user to interact with the data as though the data volume is stationary and the slice plane moves about it. The **-reverse_mouse** option allows to user to invert the sense of mouse motion as though the slice plane is stationary and the data cube is being moved.

5 Summary

This article has only described the most basic and frequently used commands for visualization on the Pixar. A host of additional commands are available and are described in [PixIS]. This article has also introduced the various visualization tools available. All of the visualization tools use data stored as a Pixar Volume which can be created with the command **makevol** from the standard file formats, Ras and View, or from raw or binary dumped data. Although only two interactive visualization programs are currently available, more will be available in the future. For now, film loops provide an adequate method for animating non-interactive visualizations.

References

[Bras88] Brase, JM, Miller, VJ & Wieting, MG "The VIEW Signal and Image Processing System," LLNL Report UCID-21368, March 1988.

[Gran88] Grant, CW *Rasfiles* is an image storage format created by Chuck Grant which is used throughout the Visualization Laboratory.

[Gran89] Grant, CW *Vrender* is a volumetric rendering package created by Chuck Grant which is used throughout the Visualization Laboratory.

[Cabr89] Cabral, B *Cell-Tracer* is a ray-tracing based volumetric rendering package created by Brian Cabral which is used throughout the Visualization Laboratory.

[PixIS] Pixar Image Computer Programmers Manual, Part I, Section on Shell Commands

[PixII] Pixar Image Computer Programmers Manual, Part II.

[PixIID] Pixar Image Computer Programmers Manual, Part II, Section on Demo Programs.

[PixIIP] Pixar Image Computer Programmers Manual, Part II, Section on Public Files and Macros.

[PixIIC] Pixar Image Computer Programmers Manual, Part II, Section on ChapVolume Shell Commands.

[PixIIV] Pixar Image Computer Programmers Manual, Part II, Sections on ChapVolume Tutorial and Technical Summary.