Videre Design
DCAM Digital Camera Capture Software

User's Manual
Version 1.0c
July 2001

? Kurt Konolige
Videre Design
kurt@videredesign.com
http://www.videredesign.com

1 Introduction

The DCAM is Videre Design's VGA-format progressive-scan digital camera, with an IEEE 1394 (Firewire) bus interface. While IEEE 1394 hardware is a well-established standard, the software interface for image processing is not. The DCAM Capture Software is an easy-to-use C++ API that offers cross-platform access to video images from DCAM devices, and gives user programs full program control over video parameters. Its intended use is for programmers who want to capture uncompressed video images into memory for further processing.

Features:

- ?? Conforms to the IEEE Digital Camera (DC) specification
- ?? Efficient access to video images via DMA transfer (no CPU involvement)
- ?? Capture 640x480 frames at 30 fps in monochrome mode, 15 fps in RGB color mode
- ?? Automatically enumerates accessible devices on the IEEE 1394 bus; up to 10 devices accessible from the same program
- ?? Capture simultaneous video streams, up to the 400 Mbps IEEE 1394 limit
- ?? Select frame size and video mode (color, monochrome) under program control
- ?? Auto and manual modes for exposure, gain, brightness, and white balance
- ?? Manual control of color saturation, sharpness, and gamma

The DCAM Capture Software comes with 2 applications with full source code. *dcam(.exe)* is a full-featured GUI that allows immediate viewing and control of DCAM devices, including the saving of images to files. *ddisp(.exe)* is a simple application that shows how to access the DCAM video stream and display it.

The Capture Software is available for MS Windows 98/ME/2000, and Linux 2.2.16 and higher kernels. Table 1-1 describes the main components of the Capture Software.

Capture Software Component		
MS Windows	**Linux**	**Description**
dCamera.dll, .lib *dcam.h*	*libdcap.so* *dcam.h*	Driver interface file. Contains the dSystem and dCamera classes that are the interface to the IEEE 1394 drivers for the DCAM. The class definitions are in *dcam.h*.
fltkdll.dll, .lib	*libftlk.so.1*	FLTK windowing library. Cross-platform display classes for images and GUI widgets.
dcam.exe	*dcam*	Full-featured GUI application. Enumerates the available cameras, presents video imagery from one camera in a window, saves single images to files, full camera control
ddisp.exe	*ddisp*	Simple capture program. Illustrates the basic operation of opening a camera, grabbing images from it, and displaying the images.

Table 1-1 Capture Software components

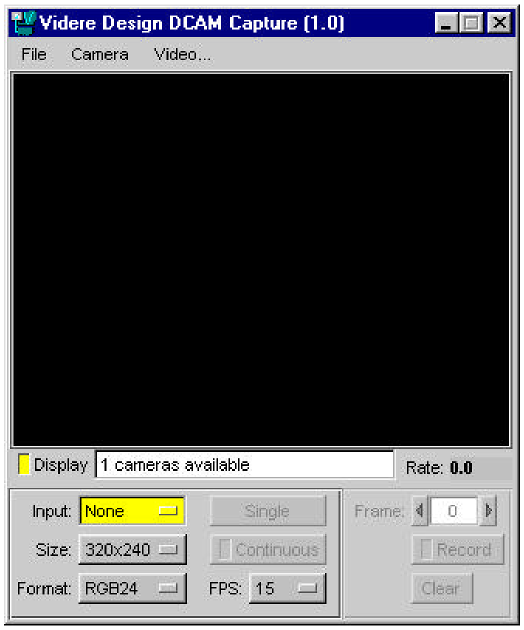
2 Getting started with *dcam*

The *dcam(.exe)* program is a standalone application that exercises the DCAM Capture Software. It is a GUI interface to the software, and in addition can save single images. The *dcam* program is a useful tool for checking out your digital cameras.

The *dcam* program is in the *bin* directory. It requires the Capture Software component shared libraries (Table 1-1), all of which are in the *bin* directory. Under MS Windows, these shared libraries (DLLs) must be in the same directory as the *dcam.exe* program, or in the system DLL directory. Under Linux, the LD_LIBRARY_PATH variable must have the path to the libraries.

Figure 2-1 shows the startup screen of the program. The black window is for display of the video image. The display programs in *dcam* use the FLTK cross-platform window interface, and work best in 24 bit mode. The version of the program is indicated in the title bar.

The rest of this section explains the operation of *dcam*. Since *dcam* exercises all of the functionality of the DCAM Capture Software libraries, it serves as a general reference for camera functions. The rest of this Section explains the operation of *dcam*.



1 cameras available

Rate: 0.0

Input: None

Size: 320x240

Format: RGB24

FPS: 15

Frame: 0

Figure 2-1 *dcam* program interface. The black window is for display of video.

2.1 Enumerating Devices

Before starting up *dcam*, plug in a DCAM digital camera to an available IEEE 1394 port. You must have installed an IEEE 1394 PCI or PCMCIA card, and the low-level drivers, as detailed in the installation instructions for the DCAM (www.videredesign.com/support_dcam.htm). Then, when *dcam* is started, it will enumerate all of the available DCAMs attached to the IEEE 1394 bus. The text information window will show the number of cameras found. If there is a problem with recognizing the DCAM, no cameras will be found. Please check the installation instructions to determine what to do.

Assuming there is at least one recognized camera, you can choose which camera to control by using the *Camera* pull-down menu. The current one will be marked.

Individual cameras are identified by a 64-bit identifier, parsed as two 32-bit hexadecimal integers. The API function *dSystem::InitCamera()* can open a particular camera based on this identifier, or based on the order of its enumeration by *dSystem*.

2.2 Inputting Live Video

The DCAM Capture libraries provide support for live video input. To start a live video stream, follow this procedure:

1. Choose a camera using the *Camera* menu.
2. Initialize the camera for input by pulling down *Video* from the *Input:* chooser. After a short pause, the application will display *Camera initialized*, and the *Single* and *Continuous* buttons will become active. If there is a problem with the camera, it will not be initialized.
3. Select appropriate Size, Format, and FPS (frames-per-second) from the drop-down lists in the application. Default values will work.
4. Press the *Continuous* button. If the camera does not support the selected modes, then the message *Unsupported modes* will appear in the information window. Otherwise, video imagery will appear in the video window. This window always shows a 320x240 frame, even when the camera video image is 640x480.

Size, format, and frames-per-second cannot be changed while the video image is live. The image stream must first be halted, the mode changed, and then restarted.

2.2.1 Video Format

Video format is the format of the pixels in the video image. The DCAM Capture Software libraries support two video formats: RGB24 and monochrome. In RGB24, each pixel is 24 bits, with 8 bits of red, 8 of green, and 8 of blue. In monochrome, each pixel is 8 bits or grayscale information. Video format is selected from the drop-down *Format* list.

Since each DCAM is limited to a maximum of 200 Mbps over the IEEE 1394 bus, the video format helps determine the maximum frame size and rate combinations available. Consult Table 2-1 for a complete listing.

2.2.2 Frame Size

There are two frame sizes, 320x240 and 640x480. In both of these, the image is scanned progressively, that is, the camera captures a frame all at once, and scans it out one line at a time. Frame size is selected from the drop-down *Size* list.

The DCAM imager has 640x480 pixels, in a Bayer color pattern – each pixel returns a Red, Green, or Blue color value. The DCAM camera interpolates to give a complete set of RGB color values at each pixel in 640x480 mode, which produces a somewhat blurred color image. At 320x240, the color image is much more crisp, because no interpolation is needed.

Another advantage to of a 320x240 size is that is uses a lot less bus and memory bandwidth than 640x480. Each DCAM is limited to a maximum of 200 Mbps over the 1394 bus; consult Table 2-1.

2.2.3 Frame Rate

DCAM Capture Software supports 4 different frame rates: 30, 15, 7.5, and 3.75 frames per second. Not all of these frame rates can be used with all the frame sizes and video formats; consult Table 2-1.

Selecting slower frame rates has two advantages:

1. Exposure times can be made longer for low-light situations.
2. Lower bus traffic means more cameras can be active simultaneously

Video Format	Frame Size	Frame Rates
RGB24 (24 bit pixels)	320x240	30, 15, 7.5, 3.75 fps
	640x480	15, 7.5, 3.75 fps
Y800 (monochrome, 8 bit pixels)	320x240	30, 15, 7.5, 3.75 fps
	640x480	30, 15, 7.5, 3.75 fps

Table 2-1 Compatible video modes

Frame rates are selected from the *FPS* drop-down list.

2.2.4 Display Output and Actual Frame Rate

The video display can be turned on or off using the *Display* button. Displaying video with in the FLTK window can use significant system resources, and it is sometimes useful to stop it, while continuing the input of video. The displayed frame is always 320x240, even when the video image is 640x480.

The actual frame rate, determined by timing the last 10 frames, is indicated in an output box.

2.3 Video Parameters

The DCAM Capture Software allows direct control of all of the video parameters that are available on the DCAM imagers. These include:

?? Exposure and gain
?? Brightness
?? White balance
?? Color saturation
?? Sharpness
?? Gamma

Some of these parameters can be controlled automatically by the DCAM itself. All parameters can be adjusted manually under program control. In manual mode, the parameters are adjusted by giving a value between 0 and 100 (except Gamma, which is either on or off).

The video parameters are set using the Video Parameter dialog (Figure 2-2). Invoke this dialog by choosing the *Video* menu item in the *dcam* application window.

A note about Manual modes. The DCAM camera has no provision for reading back values set by Auto modes. As soon as Auto mode is switched to Manual mode, the parameter value given by the application takes effect. It is not possible to use Auto mode to set a parameter, then switch to Manual mode to freeze that parameter.

2.3.1 Exposure and Gain

Exposure is the amount of time the DCAM imager is exposed to light on each video frame. *Gain* is the amount of amplification applied to the charge accumulated by each pixel. In general, larger exposures mean better images because the signal to noise ratio is increased. Larger gain, which is necessary for low-light situations, amplifies noise as well, and tends to lead to a noisier image.

In *Auto* mode, the DCAM adjusts exposure and gain to give an image with reasonable levels of light and dark. The algorithm tries to maximize exposure and minimize gain, in keeping with minimizing image noise.

In Manual mode, the exposure and gain can be adjusted independently.

2.3.2 Brightness

Brightness is the offset of the video signal when no light is present. Normally this should be left in

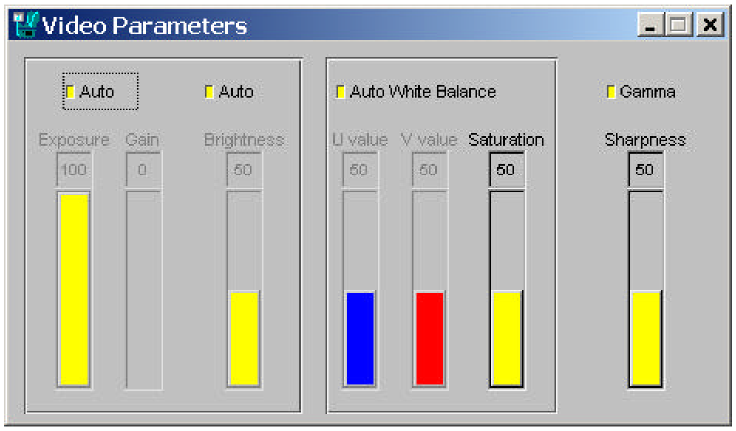


Figure 2-2 Video Parameter Dialog Box

Auto mode, where the imager looks at a set of pixels that are not exposed to light and adjusts the offset accordingly.

2.3.3 White Balance

The relative amounts of red, green, and blue present in the video image can be adjusted by differentially adjusting the gain on the red and blue pixel values, relative to green. In *Auto* mode, the DCAM camera tries to make the image have an overall balance of these colors. As the lighting and scene changes, it constantly adjusts the picture so that the relative amounts of these colors are the same. For many image processing applications, this leads to unacceptable changes in the color balance, and Manual mode should be used.

In Manual mode, the gains of the U,V color channels can be adjusted using the sliders. For a particular lighting source, try adjusting the gains until a grey area in the scene looks grey, without any color bias.

2.3.4 Color Saturation

Color saturation is the overall amount of color present in the image. This value is always set manually. A value of 0 yields equal amounts of red, green, and blue, that is, a monochrome image. A value of 100 gives the largest differential in the colors, but will look odd. The default value is 27.

2.3.5 Sharpness

The 640x480 color images appear slightly blurred, because the DCAM color processor is interpolating the color values. To compensate, it provides a sharpness filter that will emphasize image edges. Moderate amounts of sharpness can make the image look better, but also emphasizes any noise present in the image. Low values of sharpness actually blur the image. Most image processing applications will prefer to have a *neutral* sharpness value, which is 20 (the default setting).

2.3.6 Gamma

The output response of most monitors is nonlinear, and the display of an image with linear brightness values will be seen as overly dark and contrasty. Applying a gamma correction to the video image make the image display more naturally, by emphasizing dark values. For many image processing applications, however, gamma correction is undesirable.

3 DCAM Capture Software API

The capture interface is a set of C++ classes whose member implement the functionality described in the previous Section. By including the appropriate capture interface library (Table 1-1), user programs can access a DCAM and import video images into memory.

The capture classes are described in the header file *src/dcam.h*. The two classes are *dSystem* and *dCamera*. *dSystem* is the class for initializing the IEEE 1394 bus, enumerating and selecting a camera. The *dCamera* class controls an individual camera, inputting video and changing video modes and parameters.

Two windowing classes are provided for display and debugging. The *dcamWin* class can draw monochrome or color images in a window. The *dcamDebugWin* class provides a simple output browser window for displaying text message to the user. Both windows are based on the FLTK windowing system (www.fltk.org), and can be used in either MS Windows or Linux environments.

See the source code for the *ddisp(.exe)* application (Section 3.5) for typical use of these classes.

3.1 *dSystem Class*

This class initializes and controls the IEEE 1394 bus.

```
class dSystem
```

3.1.1 Initialization Function

```
dSystem *dSysInit()
dSystem *dSys
```

This function is called once, at the start of the application, to initialize the IEEE 1394 bus and enumerate the DCAM devices on it. It returns a *dSystem* class instance that can be used to access the cameras. This is the preferred way to instantiate a *dSystem* object. The global variable *dSys* is set to the value of the *dSysInit()* call, so it is always available. If the call fails, because there are no nodes on the IEEE 1394 bus (i.e., no IEEE 1394 cards are found), then it returns NULL.

3.1.2 Camera Enumeration

```
int NumCameras()
```

Returns the number of cameras found on the IEEE 1394 bus.

```
char **Names()
U64V *ChipIDs()
```

Each camera has a name, which is the string *Videre Design DCAM* for the DCAM. Individual cameras are distinguished by their chip ids, which are *U64V* structures: an array of two 32-bit integers. The camera id is part of the camera firmware, and does not change across instantiations of *dSystem*.

```
dCamera *InitCamera(char *)
dCamera *InitCa mera(int n)
dCamera *InitCamera(U64V *id)
```

Once cameras are enumerated on the IEEE 1394 bus, they can be opened (or *initialized*) using one of these functions. They return an instance of the *dCamera* class, which can then be used to input video into buffers and otherwise control the camera. If there is a problem opening the camera, the member function returns NULL.

The first form opens the first camera which matches the name. The second opens the *n*th enumerated camera, starting with index 0. The third form opens the camera with id *id*.

3.1.3 Video Streaming

```
bool Start()
bool Stop()
```

These functions start and stop the video streaming of all opened cameras, at the same time. They are mostly for convenience; applications will typically start or stop video streaming by using the individual camera object.

The return value is *true* if the call is successful, and *false* if not. The reason for the error can be found using the *Error()* function.

3.1.4 Error Codes

```
char *Error()
```

Functions that return errors, such as *Start()* and *Stop()*, usually place a reason for the error into a buffer that can be accessed with the *Error()* function.

3.2 dCamera Class

This class controls individual cameras, providing an interface for grabbing images into buffers. It also provides functions for controlling video modes and parameters. See Section 2 for an explanation of video modes and parameters.

```
class dCamera
```

A camera object should be instantiated using one of the *InitCamera* member functions of the *dSystem* object.

3.2.1 Video Modes

```
bool SetFormat(dISIZE size, dITYPE type, dISPEED speed)
dISIZE Size()
dITYPE Type()
dISPEED Speed()
```

The *SetFormat* function sets the frame size, video format (*type*), and frame rate (*speed*). The values for modes are enums, and can be found at the beginning of the *dcam.h* header file. If a particular video mode is achievable, it returns *true*; else it returns *false*, and the error reason can be retrieved with the *Error()* function. Video modes cannot be changed during video streaming, and *SetFormat* will return an error in this case.

The three other functions return the current values for the video modes.

3.2.2 Video Parameters

```
void SetExposure(int val, bool auto_flag = false)
void SetGain(int val)
void SetBrightness(int v al, bool auto_flag = false)
void SetWhiteBalance(int  uval, int vval, bool auto_flag = false)
void SetSaturation(int val)
void SetGamma(bool on)
void SetSharpness(int val)
```

These functions set the corresponding video parameters. All take a value from 0 to 100, except for *SetGamma*, which takes a Boolean. Video parameters can be changed at any time, even during video streaming.

3.2.3 Video Streaming

```
bool Start()
bool Stop()
```

These functions start or stop video streaming from the camera. They return *true* if successful, and *false* if not. The most likely cause of not being able to start a video stream are

1. Incompatible video modes, and
2. Insufficient IEEE 1394 bandwidth (with multiple cameras)

3.2.4 Grabbing Images

```
bool GetImage(unsigned char **buf, int ms = 0,  int *frame = NULL,
              unsigned long *time = NULL);
bool ReadyImage(int ms);
```

These functions control grabbing of images into memory buffers.

The *GetImage* function returns an image into the pointer *buf* provided by the caller. The buffer itself is generated and managed by the capture interface, and should not be freed up by the caller. But, the caller is free to write into or copy the buffer. The buffer contents are guaranteed not to be changed until the next call to *GetImage*.

The image is packed into the buffer in a form that depends on the video format. For Y800 (monochrome), each pixel occupies one byte. For RGB24 images, each pixel occupies 3 bytes, with the R, G, and B components appearing in that order. The pixels are packed in each line, so that a line of a 320x240 frame occupies 960 bytes, and a line of a 640x480 frame occupies 1920 bytes.

The *ms* argument is an optional timeout; the *GetImage* function will wait up to *ms* milliseconds for the camera to return a new image, and return *false* if there isn't one available within that time.

Information about the particular frame returned is found in the optional *frame* and *time* arguments. If present, the *frame* argument is set to the frame number. Frame numbers start at 1 when the camera is initially opened, and increment for each frame received during streaming mode. So, a user application can tell if a frame has been skipped by checking the frame number.

The time at which a frame is captured (that is, at which the full buffer is received by the host) can be returned in the *time* variable. The value is a system time in milliseconds. The absolute value does not mean much (except if you are using the system clock for other purposes), but the relative times tell how much time has elapsed during frame capture. For example, at 30 fps, the *time* variable will be 33 or 34 between successive frames.

The *ReadyImage* function can be used to check for the availability of a new image, without returning it.

3.2.5 Error Codes

```
char *Error()
```

Functions that return errors, such as *Start()* and *Stop()*, usually place a reason for the error into a buffer that can be accessed with the *Error()* function.

3.3 *dcamWin Class*

This class provides a graphics window for drawing the images returned from the DCAM. Both color and monochrome images are supported. The image is decimated by a factor of 2 horizontally and vertically to fit within the graphics window. For example, if the window is 400 x 300, then an image of size 640 x 480 is decimated to 320 x 240 before displaying.

```
class dcamWin
```

3.3.1 Constructor and Destructor

```
dcamWin(int x, int y, int h, int w)
~dcamWin()
```

Constructs a *dcamWin* object of size *w x h*, and puts its left-hand corner at position *x,y* with respect to its parent window. Generally, a *dcamWin* object will be the child of an *Fl_Window* object; see Section 3.5 for an example.

3.3.2 Drawing Images

```
void DrawImage(unsigned char *im, dCamera *)
void ClearImage()
```

These functions control the display of images in the *dcamWin* window. To draw a particular image held in a buffer, use the *DrawImage* function. The image is typically returned by the *GetImage* function of a *dCamera* object. The *dCamera* object is included as an argument so the *DrawImage* function can tell the image dimension and pixel format (monochrome vs. RGB color).

An image persists in the *dcamWin* display until it is displaced by another image with the *DrawImage* function, or until *ClearImage* is called to clear the window.

3.4 *dcamDebugWin Class*

Often it is useful to have a text display window for printing debugging information from an application program. The *dcamDebugWin* class provides a simple output browser for printing text strings for user perusal.

```
class dcamDebugWin
```

3.4.1 Constructor and Destructor

```
dcamDebugWin(int x, int y, int w, int h, char *nam e = 0L)
~dcamDebugWin()
```

Constructs an output window browser that can be written to. The size is *w x h*, and the offset from its parent window is *x,y*. Generally, debug windows won't have parent windows, and it won't be necessary to specify nonzero offsets.

3.4.2 Printing Text

```
 void Print(char *str)
```

Prints the string *str* on the debug window, and scrolls the screen up so that *str* is visible in the window.

3.5 DDISP *Example Application*

This section presents a simple application that exercises the DCAM API. Some excerpts from the program illustrate the basic ideas for capturing and displaying images.

3.5.1 Display Window

The display window is an FLTK window, with a graphics subwindow for displaying images.

```
Fl::visual(FL_RGB8);         // try to use 24 -bit graphics
Fl_Window *mainw = new Fl_Window(327, 276, "Videre Design DCAM");
dcamWin *win = new dcamWin(3, 23, 320, 240);
mainw->show(0,NULL);         // show the window
```

3.5.2 Camera Initialization

The IEEE 1394 system and the cameras are initialized, and the camera parameters are set up. After this,

```
dSysInit();                  // sets global var dSys
if (dSys == NULL)            // oops, problems initializing
  {
    …error…
  }

dcam = dSys->InitCamera(0); // get first one found
if (!dcam)
  {
    …error…
  }

// set format to  320x240, RGB color, 15 fps
if (!dcam->SetFormat(S_320x240, S_RGB24, S_15))
  {
    …error…
  }
```

3.5.3 Video Streaming

Next, we start video streaming

```
if (!dcam->Start())
  {
    …error…
  }
```

3.5.4 Grabbing and Displaying Images

Once video streaming is started, images are available to the application with the *GetImage* function. This function can also return information about the frame number and the time it was captured, although this capability isn't used here.

The following loop checks for windowing events, then

```
while (1)
  {
    if (!Fl::check())        // process any window events
      return 0;              // exit button pressed

    unsigned char *im;
    if (dcam->GetImage(&im, 500))  // get the latest image
      win->DrawImage(im, dcam);    // and draw i t
  }
```