SRI
Small Vision System

User's Manual
Software version 4.4d
May 2007

©Kurt Konolige and David Beymer
SRI International
konolige@ai.sri.com
http://www.ai.sri.com/~konolige

1 Introduction

The SRI Stereo Engine is an efficient realization of an area correlation algorithm for computing range from stereo images. Figure 1 shows the results of running the algorithm on a typical scene. The image on the top left is the left image of an original stereo pair, while the one on the top right is a disparity image computed from the stereo pair. In the disparity images, brighter pixels show where the projection of an object diverges between the images (has a high disparity). These are areas that are closer to the cameras. Dark areas have lower disparity, and are further away. Finally, the bottom right shows a view of the 3D reconstruction made from the disparity image.

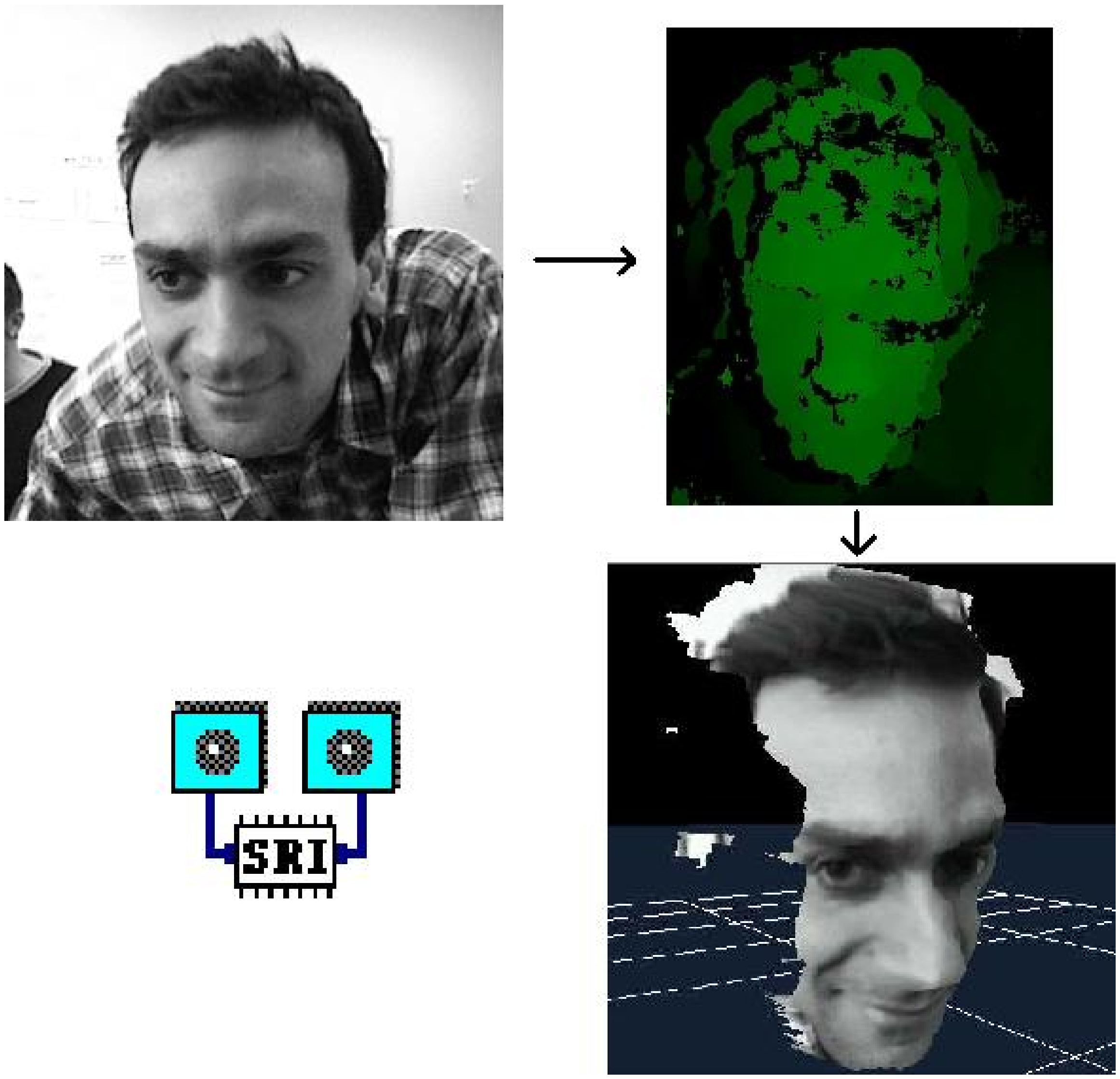


Figure 1-1. An input image and the resultant stereo disparity image. Brighter areas are closer to the camera.

1.1 The SRI Stereo Engine and the Small Vision System

The Stereo Engine exists in several implementations, including embedded, low-power systems and general purpose microcomputers. The embedded systems, or Small Vision Modules (SVMs), contain DSPs or other standalone processors, and produce digital range information. They are meant for end applications where size, cost, and power limitations are critical. SRI will develop embedded SVM systems in partnership with companies who are interested in a particular application.

The Small Vision System (SVS) is an implementation of the Stereo Engine on general-purpose microcomputers, especially PCs running Linux or Windows 95/98/ME/2000/XP. It consists of a set of library functions implementing the stereo algorithms. Users may call these functions to compute stereo results on any images that are available in the PC's memory. Typically, standard cameras and video capture devices are used to input stereo images. The Small Vision System is a development environment for users who wish to explore the possibility of using stereo in an application.

This manual is useful as a source of general information about the Stereo Engine for any implementation, but is also specifically aimed at the development environment of the SVS. It explains the core characteristics of the Stereo Engine, serves as a reference for the stereo function API, and discusses sample applications that use the API. There are also several sample programs, with comments, that illustrate writing programs to the SVS API (*samples* folder). More technical information about stereo processing can be found at www.ai.sri.com/~konolige/svs, including several papers about the stereo algorithms and applications.

SVS includes support for the digital stereo heads from Videre Design (www.videredesign.com), including the DCS series (STH-MDCS/MDCS2/DCSG/STOC), the MEGA-D (STH-MD1), and Dual DCAM. SVS also supports input directly from images in memory. Thus, a user wishing to input video from other cameras, analog or digital, can do so by writing an interface to input the images into memory, and then calling SVS. Sample code for such an interface is included with the distribution.

1.2 The Small Vision System

The Small Vision System (SVS) is meant to be an accessible development environment for experimenting with applications for stereo processing. It consists of a library of functions for performing stereo correlation. Figure 1-2 shows the relationship between the SVS library and PC hardware.

Images come in via a pair of aligned video cameras, called a stereo head. A video capture board or boards in the PC digitizes the video streams into main memory. The SVS functions are then invoked, and given a stereo pair as an argument. These functions compute a disparity image, which the user can display or process further.

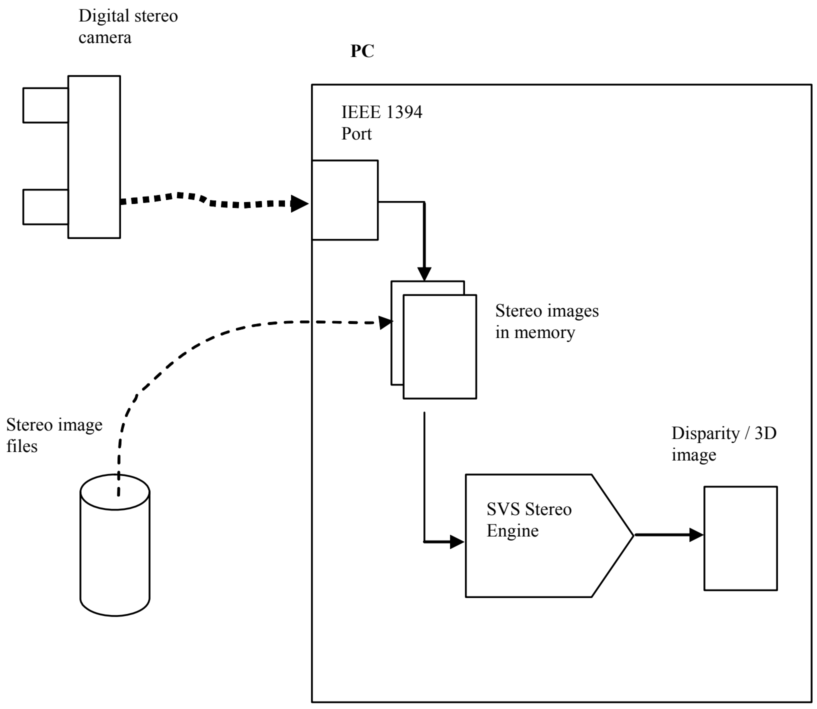


Figure 1-2. The development environment of the Small Vision System.

The SVS environment of Figure 1-2 shows a typical setup for stereo processing of video images. The user may supply his or her own images: the SVS has special processing for dealing with camera distortion and calibrating the stereo image (Section 2.4.2). Special stereo heads are also available from Videre Design (www.videredesign.com). The DCS series (STH-MDCS/MDCS2/DCSG/STOC) are a series of all-digital devices with VGA or megapixel imagers that use the 1394 bus (FireWire) for direct digital input. Other sources of images may also be used, as long as the images can be placed in PC memory. Some examples are images stored on disk, or images obtained from other devices such as scanning electron microscopes.

1.3 Hardware and Software Requirements

The SVS libraries exist for most Unix systems, as well as MS Windows 98SE/ME/2000/XP; that is, on the most common computer platforms available. We have spent considerable effort in optimizing SVS for PCs using the MMX/SSE instruction set, and it will perform best on these platforms, using either Linux or MS Windows. Performance information is in Section 3.7.

Recommended processors are:

- Intel Pentium III, IV, M
- AMD Athlon XP, Athlon 64
- VIA Eden, C3
- TransMeta Efficeon (TM8x series)

Any of these processors support the MMX and SSE instruction sets, which are used by SVS to efficiently perform stereo calculations.

SVS does not currently support 64-bit architectures.

Since MS Windows and Linux are constantly changing, please check the Videre Design website for the latest information about issues involving these operating systems.

1.3.1 Analog Framegrabbers

Analog framegrabbers are no longer directly supported in SVS. Sample code for analog framegrabber interfaces is available in the distribution; the user can modify it to suit his or her needs.

1.3.2 Digital Framegrabbers

The DCS series (STH-MDCS/MDCS2/DCSG/STOC) and the MEGA-D and Daul DCAM stereo heads are all-digital devices that use the IEEE 1394 (FireWire) bus. Some desktops and laptops have 1394 ports integrated directly into their motherboards. Otherwise, a standard 1394 PCI board or PCMCIA card can be used. The card must be OHCI (Open Host Controller Interface) compliant, which almost all boards are.

1.4 The SVS Distribution

The SVS distribution can reside in any directory; normally, it is placed in `c:\svs` (MS Windows systems) or `/usr/local/svs` or a user's directory (Unix systems). Here is the directory structure of the SVS distribution.

```
svs
   readme                 installation guide
   update                 release notes
   docs                   documentation
      smallv.pdf          PDF version of the User Manual
      calibrate.pdf       PDF version of the Calibration Addendum
   bin                    executable and library files
      smallv(.exe)        full-featured GUI client demo
      smallvcal(.exe)     Calibration and firmware tool
      smallvmat(.exe)     GUI client demo with MatLab interface
      stframe(.exe)       simple stereo client example program
      cmat.dll/so         MatLab loadable DLL/so for controlling SVS
      cwrap.dll/so        C interface DLL/so for LabView
      svsXXX.dll,.lib     framegrabber interface fns (Windows OS)
      XXXcap.so           framegrabber interface fns (Linux)
      svs.dll, lib        SVS library (Windows OS)
      svsgrab.dll, lib    SVS library for capture  (Windows OS)
      libsvs.so           SVS library (Linux)
      libsvscap.so        SVS library for capture (Linux)
      fltkdll.dll         Display library (Windows OS)
      libfltk.so          Display library (Linux)
   data                   stereo images
      check.pdf           single-page printable calibration object
      check54-panel.pdf   4-page printable calibration object
      check-54.pdf        54mm single page calibration object (17x22)
      check-108.pdf       108mm single page calibration object (34x44)
      *-L/R/C.bmp         Sample stereo pairs and color files
      *.ini               Sample calibration files
   samples                sample client program sources
   src                    SVS library sources
      svsclass.h, svs.h   main library header (C++, C)
      dcam.h, dcs.h       digital camera headers
      image_io.cpp        file I/O routines
```

2 Getting started with *smallv*

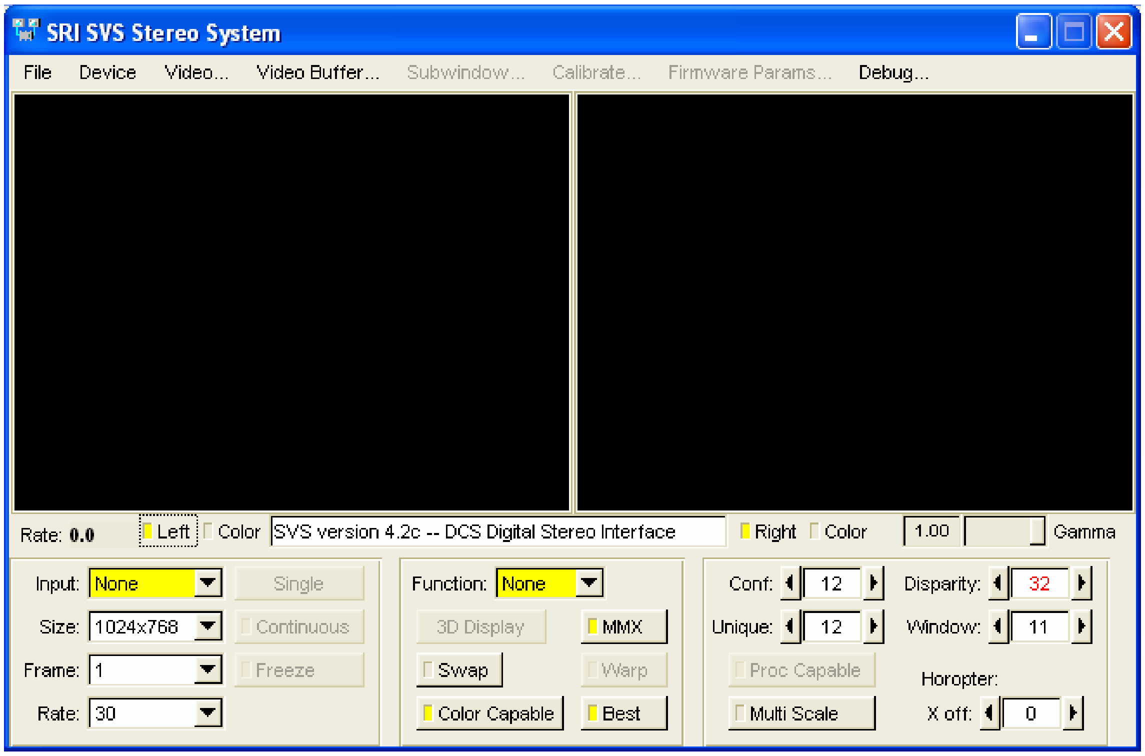
The `smallv` program is a standalone application that exercises the SVS library. It is a GUI interface to the stereo programs, and in addition can load and save stereo image sequences. The `smallv` program is a useful tool for initial development of a stereo application, and can also be used to check out and adjust a stereo camera setup.

The `smallv` program is in the `bin/` directory. It requires shared libraries for the stereo algorithms (`svs`), display (`fltk`), and calibration (various), all of which are in the `bin/` directory. Under MS Windows, these shared libraries (DLLs) must be in the same directory as the `smallv` program, or in the system DLL directory. Under UNIX, the LD_LIBRARY_PATH variable must have the path to the libraries.

Figure 2-1 shows the startup screen of the program. The black windows are for display of image and stereo data. The display programs in SVS use the FLTK cross-platform window interface, and work best in 24 bit video display mode. The version of the program is indicated in the text information area, and the title bar.

`smallv` will accept stereo images from either a live video source, or a stored file. The easiest way to get started with the program is to open a stored stereo sequence. From the `File` menu, choose `Open`, and navigate to the `data/` directory. The file `face320-cal-X.bmp` contains a stereo frame at 320x240 resolution. When you open it, it will show in the display windows. In the `Function` area, pull down the list box and choose `Stereo`. Finally, press the `Continuous` button to compute the stereo disparity and display it. You should see a green pattern representing stereo disparities in the right window. Under the `Horopter` label, click the `X offset` button a few times to see the effect of changing the stereo search area; a value of –4 or so should bring the close parts of the face into range. Clicking the `3D Display` button brings up an OpenGL window with a 3D view of the stereo data.

The rest of this section explains the operation of `smallv`. Since `smallv` exercises most of the functionality of the SVS libraries, it should serve as a general introduction to the SVS functions. If you are



Rate: 0.0

SVS version 4.2c -- DCS Digital Stereo Interface

1.00

Input: None
Size: 1024x768
Frame: 1
Rate: 30
Function: None
Conf: 12
Unique: 12
Disparity: 32
Window: 11
X off: 0

Figure 2-1 Smallv program interface. The two black windows are for display of input images and stereo results.

interested in using a particular framegrabber and set of cameras with `smallv`, please see Section 2.1. The framegrabber interface is indicated in the message area on startup. In this case, it is the IEEE 1394 (Firewire) interface used by the DCS series (STH-MDCS/MDCS2/DCSG/STOC) digital stereo heads.

2.1 Inputting Live Stereo Video

The SVS libraries provide support for live video as stereo input. To input video, you must do the following steps. These instructions assume you are using a DCS series stereo device.

1. Install the IEEE 1394 card, or use a built-in IEEE 1394 port. See the Videre Design website for the latest information on installing drivers for MS Windows or Linux.
2. Start the `smallv(.exe)` program.
3. Set the appropriate video format using the `Video Format` menu.
4. Set the video frame size.
5. Set the input mode to `Video`.

This section gives details necessary for performing these steps.

2.1.1 Stereo heads

Stereo requires two images from different viewpoints. The most common way to get these images is to use two identical cameras separated by a horizontal baseline. It is important the cameras have lenses with the same focal length, and that the pixel elements have the same size.

The baseline is typically from 3 to 8 inches wide, and the cameras are aligned parallel to each other, although other configurations are possible. Figure 2-2 shows a typical stereo camera setup. Two cameras are pointed in the same direction, and they are synchronized so that they capture images at the same time. Synchronization is important if there is any motion in the scene. If the cameras are not synchronized, they can capture the image at slightly different times, and any moving objects will be at a slightly different position in one camera relative to the other, than if it they had taken the image at the same time. If the scene is static, then synchronization is not necessary. The stereo devices from Videre Design (DCS series) achieve synchronization through two different methods – see the caption in Figure 2-2.

A word about monochrome vs. color cameras. If your application does not need color, it is preferable to use monochrome cameras, because stereo relies only on the luminance component of the video signal. Monochrome cameras have much better spatial resolution and dynamic range than color cameras of the same quality, since they do not have to deal with three color channels. Having said this, the MDCS megapixel cameras have such high resolution that using color imagers is generally not a problem, since most applications can use 640x480 or 320x240 image sizes, and the color imagers produce excellent quality by binning (averaging) a set of pixels.

The digital stereo devices produce digital output on the IEEE 1394 (FireWire) bus. The video stream can be input to a PC using a standard IEEE 1394 card, either a PCI card for a desktop PC, or a PC Card (sometimes called a PCMCIA Card) for a laptop. Some desktops and laptops have built-in IEEE 1394 ports, and these do not need an add-in card. The card or built-in IEEE 1394 port should have OHCI (Open Host Controller Interface) capability, which almost all of them do.

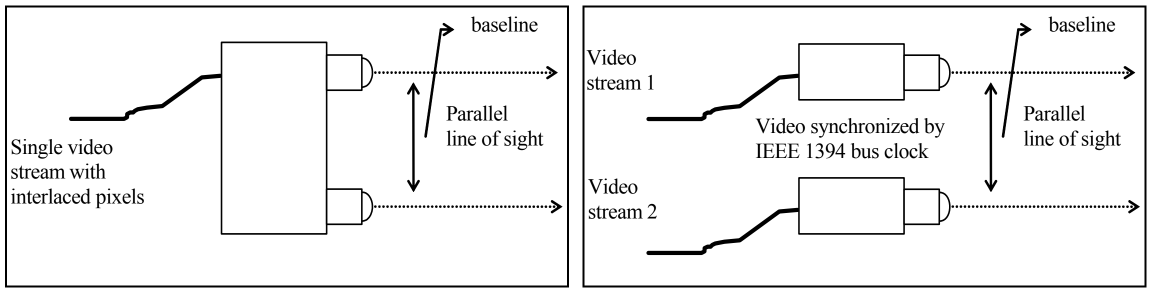


Figure 2-2 Two types of digital stereo cameras. Cameras are positioned with parallel lines of sight. The cameras are synchronized to capture images at exactly the same time. On the left, the integrated electronics of the STH-MDCS/MDCS2/DCSG/STOC series *pixel-locks* the two imagers, so that corresponding pixels are read out at the same time into a single video stream. On the right, the STH-XXX-VAR series has two separate MDCS/DCSG cameras. These cameras synchronize image capture by using the IEEE 1394 bus clock.

PLEASE NOTE: Analog framegrabbers are no longer directly supported under SVS (as of version 3.x). If you wish to use analog framegrabbers, you must write your own interface to the framegrabber, and then present the images to SVS (see Section 6.7.3).

The SVS libraries can work with any size video frame up to 1288 by 1032 pixels. Typically, images are input in standard resolutions:

- 320x240
- 640x480
- 1280x960

2.1.2 Analog Framegrabbers

SVS no longer directly supports analog framegrabbers. The SVS libraries for analog framegrabbers are included in the distribution as a courtesy to those who have used these in previous distributions.

2.1.3 IEEE 1394 (FireWire) Framegrabber

The SVS has an interface to digital stereo heads from Videre Design via the IEEE 1394 serial bus. Any OHCI-compliant IEEE 1394 PCI or PCMCIA card can be used, under MS Windows 98SE/ME/2000/XP or Linux. Please check the stereo head manual and the Videre Design website for instructions on installing the 1394 card and drivers.

The relevant interface libraries are given in Table 2-1 below. By default, the libraries are set up for the current stereo devices, MDCS(2), DCSG, and STOC. To set up a different interface in Linux, copy the library file to `bin/libsvscap.so`. Under MS Windows, execute the setup file in the `bin\` directory

Operating System	Stereo Head	Library	MSW Installation File
Linux	**DCS series** STH-MDCS(2)-XXX STH-DCSG STOC	`dcscap.so`	
	Dual DCAM	`dcamcap.so`	
	MEGA-D (STH-MD1)	`pixcap.so`	
MS Windows 98SE/ME/2000/XP	**DCS series** STH-MDCS(2)-XXX STH-DCSG STOC	`svsdcs.dll`	`setup_dcs.bat`
	Dual DCAM	`svsdcam.dll`	`setup_dcam.bat`
	MEGA-D (STH-MD1)	`svspix.dll`	`setup_megad.bat`

Table 2-1 Interface libraries for Videre Design digital stereo heads.

by double-clicking on it in Windows Explorer.

2.1.4 Selecting Devices

The *Device* menu button lets you tell the SVS library what kind of video input you are using. For Videre Design digital heads (DCS series, MEGA-D and Dual DCAM), you can select among multiple devices attached to any IEEE 1394 card on your computer.

Digital Stereo Devices (DCS series, MEGA-D and Dual DCAM)

A digital stereo head attached to the IEEE 1394 bus is recognized by `smallv`, and the *Device* menu button will drop down a list of these devices when selected. Devices have a *number*, which starts at 1 for the first device encountered. These numbers can change as devices are added or dropped from the bus. Devices also have an *id*, which is a numeric string that is unique to the device. The *Device* list shows both the current device number, and its unique id. The currently selected device is indicated by a checked box;

you can change the current device by selecting any available device. This choice becomes active the next time *Video* input is selected in the input choice box.

Only one type of device, a DCS series, MEGA-D or Dual DCAMs, will be seen by the smallv program. The choice depends on which interface library has been loaded (see Section 2.1.3). It is possible to mix these devices on the same IEEE 1394 bus, but a given application will see only one type of device or the other.

2.1.5 Frame Size

The SVS libraries as delivered can work with frame sizes up to 1288 by 1032. In fact, the SVS algorithms can work with arbitrarily sized frames, but have been restricted so that pre-allocated buffers are not too large.

A subset of frame sizes are supported for video input in the `smallv` application; the following table summarizes them. These are the sizes supported by Format 0, 1 and 2 of the DCAM 1.30 specification.

Video Format	Frame Sizes
1394 (digital) interface	1280x960
	1024x768
	640x480
	512x384
	320x240

Table 2-2 Frame sizes available for video input in smallv.

Video frame size is selected with the `Size` drop list in the `Source` area. Video size can be changed only when frames are not being acquired. Once acquisition starts, the frame size is fixed.

For frame sizes supported by individual devices, please see their documentation.

2.1.6 Image Sampling

Digital devices allow control over sampling of the image array. Sampling can be used to return a full-frame image with less resolution, or to return a *subwindow* of the full frame. Note: subwindowing is available only on MEGA-D and MDCS2 devices, but not MDCS, DCSG, STOC or DCAM devices. See the device manuals for details.

There are several sampling modes. *Decimation* samples the image by removing pixels, so the decimation by 2 removes every other pixel in a line, and every other line. *Binning* samples the image by averaging over a block of four pixels, to produce the same result. Binning produces smoother images with less noise, but it is slower than decimation, which is done by the stereo hardware. Combination sampling modes are available, e.g., “`x4 bin+dec`” samples the image down to ¼ size in horizontal and vertical directions, by decimating by 2 and then binning by 2.

Generally, the user is concerned with the resolution of the image (e.g., 640x480) and the amount of the frame that the image covers. The sampling modes can produce some confusion in this respect. For example, for the STH-MDCS2 device, to get a full-frame 640x480 image, you must specify either decimation by 2 (on-camera) or binning by 2 (on the PC). Setting decimation and binning by 1 will give a format error when attempting to start the video stream.

To help with this problem, there is an alternate way to specify sampling modes, use the *frame division* parameter. This is the parameter that appears on the `smallv` application window. A value of “1” means that the full frame will be returned. A value of “1/2” means that a half-size image (half the width and half the height, so ¼ of the pixels) will be returned, and a value of “1/4” means that a quarter-width and quarter-height image is returned. Using frame division leaves some ambiguity about how to achieve the results – for example, 640x480 at full frame size for the STH-MDCS2 devices can be done either by decimation or binning, as noted. The system will pick an appropriate mode. For more control over the mode, it is always possible to specify the sampling explicitly, using the API.

2.1.7 Image Source

The source for stereo images can be either a memory buffer or a live video stream. The `Source` drop list lets you choose between these, or to stop any input. Buffer input is discussed in Section 2.1.9.

2.1.8 Streaming Mode

Images from video cameras or the buffer can be processed in three acquisition modes. Only one acquisition mode is active at a given time.

- Continuous mode. In this mode, stereo pairs are continuously input, processed, and displayed. The maximum frame rate is up to 80 Hz for the MEGA-D digital system. See Section 3.7 and the manual for the individual stereo heads for performance information. The rate is indicated next to the text information area.
- Single frame mode. In this mode, a single stereo pair is input, processed, and displayed each time the Single button is pressed.
- Freeze mode. In this mode, a single stereo pair is input, then the same frame is continuously processed and displayed. This mode is useful in checking the effect of different stereo parameters on the same image.

2.1.9 Adjusting Video Parameters

The MEGA-D and MDCS/MDCS2 digital stereo heads have manually controlled exposure and gain. Exposure is the time that any given pixel is exposed to light before being read out. Gain is a amplification of the signal that comes out of the pixel. In general, it is best to increase the exposure first, and if necessary, to increase gain once exposure reaches a maximum. The reason for this is that gain will increase the video noise, while exposure increases the pixel's response to light. In some cases, though, short exposure times are desirable for minimizing motion blur, and it may be more convenient to increase gain while exposure is not at a maximum.

The DCS series devices also have an automatic exposure/gain mode. In this mode, both the gain and exposure are controlled to deliver a reasonable light level in the image. The auto algorithm tries to reduce gain as much as possible, increasing exposure first if the image is too dark.

The values of exposure, gain, brightness, and contrast are all represented as a percent.

The colorized version of the MEGA-D and DCS series digital cameras can input color images, and the color balance can be adjusted manually using the red/blue differential gain. More information about color processing is in Section 2.1.12.

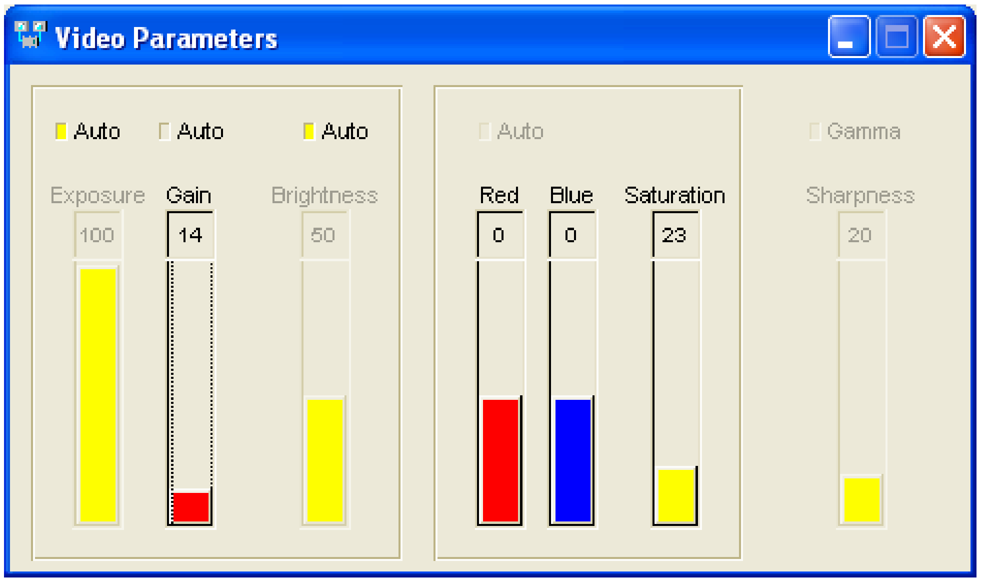


Figure 2-3 Video Parameter dialog box.

2.1.10 Subwindowing

The MEGA-D digital stereo head can send to the host computer just a portion, or *subwindow*, of the stereo image. For example, if the MEGA-D is in x2 sampling mode (full-size image is 640x320), and the image size is chosen to be 320x240, then `smallv` will input only a 320x240 subwindow of the full image. Figure 2-4 shows two of these subwindows, and the original full-size image.

The placement of subwindows is controlled by the vertical (Y) and horizontal (X) offset controls in the `Subwindow` dialog window; the dialog is initiated from the `Subwindow`... menu item in the main window. These parameters can be changed in real time, enabling electronic panning of the live image.

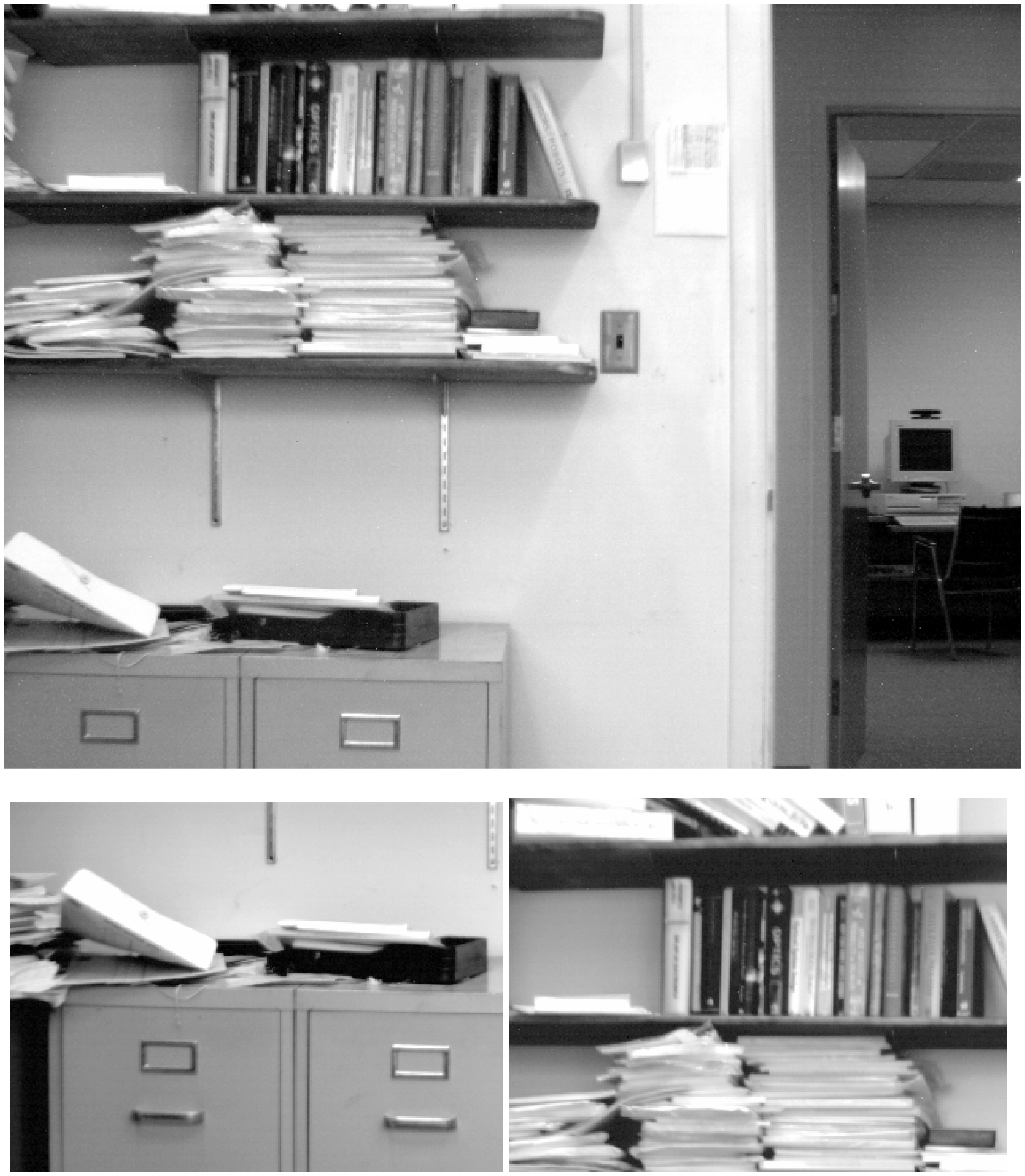


Figure 2-4 Two 320x240 subwindows (bottom) of a 640x480 image (top).

2.1.11 Vergence

When in subwindow mode, the two cameras in a stereo rig generally will have the same X and Y offsets, so that they keep the parallel line-of-sight characteristic of the stereo rig. However, for viewing close objects, it is advantageous to toe-in, or *verge*, the two stereo cameras. In this way, the images of the near object will both contain the object in the center.

Human eyes verge mechanically when viewing close objects. Mechanical vergence for stereo cameras is difficult, however, since it involves complicated motor control, and more importantly, disturbs the calibration that is critical for stereo analysis. Instead, with the subwindow capability of the MEGA-D, it is possible to verge the stereo images electronically, by choosing appropriate horizontal offsets for each image.

Figure 2-5 shows the effects of using electronic vergence. The top stereo pair, of a close object, puts the object into the center of the left frame. In the right frame, the object has a large disparity and is visible in the left side of the frame.

The bottom stereo pair is created by adding vergence to the subwindow process, offsetting the right subwindow horizontally by 120 pixels, relative to the left subwindow. Both frames now have the near object centered.

Vergence of the subwindows is set using the vergence control in the `Subimage` box of the `Subwindow` dialog. It is a real time control, just like the X and Y subwindow offsets.

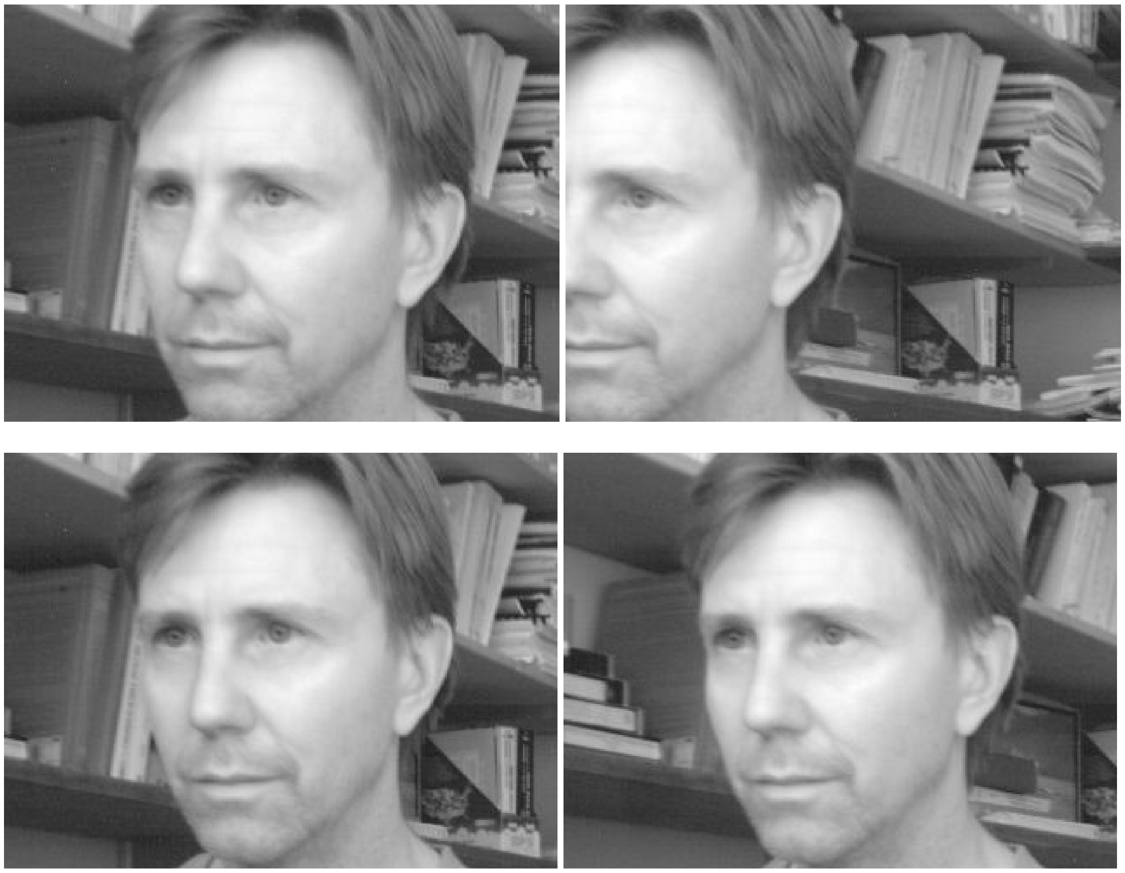


Figure 2-5 Parallel image subwindows (top) and verged image subwindows (bottom), showing a close object.

2.1.12 Color Channels

SVS supports color input and display. Besides the two monochrome left/right stereo channels, there is a third color channel that corresponds to the left image, with images in RGB 32-bit format, and optionally a fourth color channel for the right image. The color channels do not participate in stereo processing, but can be useful in applications that combine color and stereo information, for example, object tracking. Usually only the left color channel is needed, since the left image is the reference image for stereo disparities and 3D information. Both color channels are available for user applications, if desired. When the right color channel is requested, the left color channel is always also provided.

Color information from the STH-MD1-C and DCS series is input as raw colorized pixels, and converted by the interface library into two monochrome and one or two RGB color channels. The main color channel corresponds to the left image, which is the reference image for stereo. The color image can be de-warped, just like the monochrome image, to take into account lens distortion (Section 4). Optionally, a second color channel is available for the right image.

The stereo DCAM device (STH-DCAM) performs color processing on-camera, and sends the results down to the application. De-warping proceeds as for the MEGA-D.

Color information from the camera is input only if the `Color` button is pressed on the main window (Figure 2-1), under the appropriate window. To get the color images in applications, use the `SetColor()` command.

Because the typical color camera uses a colorizing filter on top of its pixels, the color information is sampled at a lower resolution than a similar non-colorized camera samples monochrome information. In general, a color camera has about ¼ the spatial resolution of a similar monochrome camera. To compensate for the reduced resolution, use binning (Section 2.1.6) to increase the fidelity of the image. For example, if you need a 320x240 frame size, use 640x480 and binning x2.

The relative amounts of the three colors, red/green/blue, affects the appearance of the color image. Many color CCD imagers have attached processors that automatically balance the offsets among these colors, to produce an image that is overall neutral (called *white balance*). The STH-MD1-C and STH-MDCS/MDCS2 devices provide manual color balance by allowing variable gain on the red and blue pixels, relative to the green pixels. STH-DCSG and STOC devices do not have on-chip color control – SVS drivers provide for color balance during processing on the PC. Manual balance is useful in many machine vision applications, because automatic white balance continuously changes the relative amount of color in the image. The STH-DCAM allows for either automatic or manual control of color (see Figure 2-3

The manual gain on red and blue pixels is adjusted using the `Video Parameters` window (Section 2.1.9). For a particular lighting source, try adjusting the gains until a white area in the scene looks white, without any color bias.

2.1.13 Color Algorithm

DCS series devices allow two different kinds of color reconstruction in non-binning mode. The reconstruction is done on the PC. When possible, use the binning mode to get the best color fidelity and resolution.

COLOR_ALG_FAST is a fast algorithm that uses bilinear interpolation to give a reasonable and efficient color interpolation. It has the drawback of a "zipper" effect, when viewing strong edges. The zipper effect is more pronounced when decimation occurs on the imager. For example, running the STH-MDCS2-(VAR)-C at 640x480, with decimation x2, will produce the largest effects. The STH-MDCS-(VAR)-C has on-chip binning, so it performs better at this same setting. For the MDCS2 devices, the effects of zippering can be reduced by using the 1/2 frame format at 640x480, that is, the center 640x480 part of the imager (Section 2.1.5). In this mode, the pixels are sampled more densely, and there are fewer artifacts in the reconstruction.

COLOR_ALG_BEST is an adaptive edge-enhancing algorithm that eliminates most of the zipper effect. It consumes more resources, but is the best algorithm for high-quality color reconstruction in non-binned modes. It also improves the grayscale images, giving better stereo results. Again, using the 1/2 frame mode for the MDCS2 devices is best, but reasonable results are produced in full frame also.

The color mode can be selected using the `Best` button on the `smallv` application. In the API, the `color_alg` variable of the `svsVideoImages` class will set the color mode. Note that this variable has no effect when a binning mode is selected.

2.2 Storing, Saving, and Loading Stereo Data

`smallv` provides a basic facility for loading and saving stereo data streams. The file load and store functions are part of the SVS library, and their source code is included. `smallv` exercises these functions, and provides a memory buffer for storing live stereo video.

There are two basic types of storage/playback available. The first, *video storage*, is meant to save a video sequence of stereo images (including color information). Images from live video streaming are captured to an internal buffer of up to 2000 frames (for saving very short sequences), or to a *stream file* on disk (for saving very long sequences). These frames can be replayed and

A second type of storage is still image storage. In this mode, a single frame is stored to a file. The video storage buffer is not involved in this process, and an arbitrary number of such stills can be saved.

2.2.1 Video Buffer Storage

`smallv` has an internal buffer capable of holding up to 2000 stereo pairs (frames). Depending on the size of the images and the amount of memory on the machine, a video sequence of frames can be captured to physical memory without slowing down video capture.

The buffer can be filled from a previously-saved file set, or from live video input. The buffer can also be written out to a file set, and used as the source for stereo processing in `smallv`.

The video buffer is controlled from the `Video Buffer` window, accessed via the menu bar (Figure 2-6).

When the input source is the buffer, the acquisition mode controls (`Continuous`, `Single`, `Freeze`) control the processing of the buffer frames (Section 2.1.8). The frame control can also be used to go to an individual frame when in `Single` acquisition mode.

The `Record` button controls the input of live video into the buffer. `Clear` clears the buffer and resets it to frame 0. Activating the `Record` button starts the input of live video frames into the buffer. The source must be set to `Video`; either `Continuous` or `Single` mode may be used. Frames are stored sequentially until the buffer is full. Pressing `Record` again will also turn off acquisition before the buffer is full.

The size of the buffer can be changed using the `Size` input. It will go up to 2000, in increments of 100. When saving to the buffer, the requisite amount of storage is pre-allocated, so that the storing can proceed quickly. Note that, for larger size buffers, there must be sufficient memory.

As an example, to capture a short video sequence and replay it, perform the following steps.

1. Start acquiring live video in continuous mode.
2. Clear the buffer (`Clear` button).
3. Start buffer storage (`Record` button).

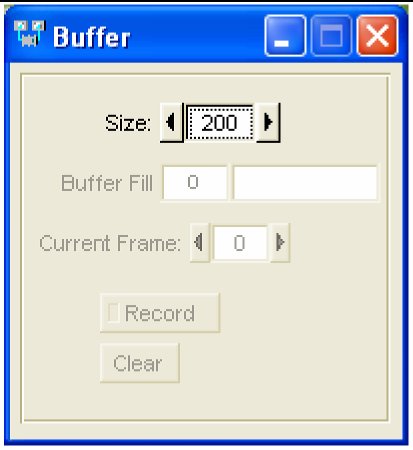

Buffer
Size: 200
Buffer Fill 0
Current Frame: 0

Figure 2-6 Video buffer controls.

4. After a short period, stop buffer storage (`Record` button).
5. Change from `Video` to `Buffer` source.

At this point, the short segment that is in the buffer can be replayed as a short continuous loop. The buffer, or individual images, can be saved to a file. Under the `File` menu, use the commands `Store Video Buffer` and `Load Video Buffer`. A video sequence is stored as a set of BMP files (next subsection), in a sequence starting at 001. You cannot save stereo disparity or 3D data directly from the video buffer. However, the current frame disparity and 3D data can be saved using the still image storage facility, described in the next subsection.

2.2.2 Stream Files

For saving longer sequences, SVS and smallv have a facility for saving directly to disk. The disk files are called *stream files*, because they save the video stream as it comes from the camera. This storage is more efficient than the buffer storage, and also can store an arbitrary length sequence, limited only by the disk storage of the PC. Streaming storage can keep up with a 640x480 stereo stream at 30 Hz, if the disk subsystem of the PC is fast enough.

To initiate a stream file, first open the stereo device using the `Input->Video` drop box. Set the desired image size in the `Size` box. Then, choose `File->Store to stream file` from the File menu. A dialog will ask for the stream file name. Use a name without an extension, e.g., `myStreamFile`. The application will save a parameter file called myStreamFile.ini, and a set of video stream files called `myStreamFile_NN.ssf`, starting with `myStreamFile_00.ssf`. Each of these files will hold up to 2 GB of information, the file limit on 32-bit systems. Then, a new file with the next sequence number will be initiated automatically, so that an arbitrary length of video can be stored (well, at least 100 x 2GB). The amount of storage needed can be large for video sequences – for example, 640x480 @ 30 Hz requires 20 MB/s.

Once the stream file is chosen, the `Stream` button will become enabled. Whenever the `Stream` button is pressed, and continuous video is being input, the input will also be saved to the stream file. It is possible to interrupt and resume saving the video. The stream will stop saving when `Continuous` video is no longer running, or when the `Stream` button is not set. It will save when both `Continuous` and `Stream` are set. The stream file will be closed once `Video` is no longer chosen in the `Input` drop box.

Some hints on saving efficiently:

- Turn off rectification, stereo, and color processing. These processes will just slow down the PC, and their results are not saved to the video stream.
- Make sure no other applications are running, especially ones that use the disk.
- Experiment with the frame rate. On a laptop computer, for example, the disk subsystem may not be fast enough to store 640x480 @ 30 Hz. If not, you will start to see warning messages about frames being skipped. You can set the capture speed lower, to 25 Hz or 15 Hz.

To play back a stream file, use the File->Load from stream file menu item. A dialog will appear asking for a file name – pick the first file in the sequence (with the last part of the file name being "_00.ssf"). The Continuous button can be used to run through the saved video stream. Just as in live video, you can save snapshots (see the section below), as well as perform stereo processing. The sequence will automatically continue through all of the saved files, in sequence.

As of version 4.4c of SVS, there is no way to skip forwards or backwards in the files. This facility will be added at some later date.

2.2.3 Loading and Storing Files

The SVS libraries work with different file types for image storage.

- BMP format. Each BMP file contains a single 8-bit grayscale image, or an RGB 24-bit color image. The color coding for the 8-bit BMP file is 256 shades of gray, with 0 being black and 255 white. By convention, a stereo pair is saved as two files with the linked names `XXX-L.BMP` (left image) and `XXX-R.BMP` (right image). The corresponding left color image is saved as `XXX-C.BMP`, and the right color image as `XXX-Q.BMP`. Finally, the image parameters are stored as a text file `XXX.INI`.

- Text files for disparity images. Disparity images can be saved as a text file, with one line of text for each line of the image (e.g., a 320x240 image will have 240 lines). Each line contains an image row of disparities, as integers. The special values –1 and –2 indicate that the disparities were filtered out, by the texture measure (-1) or the left/right check (-2)..
- Text files for 3D points. 3D point arrays, generated from a disparity image (Section 2.4.2), can be saved as a text file. Each line of the file represents one point of the array. The array has the same format as the image from which it was produced, e.g., if the input image is 320x240, then the file has 320x240 lines, in row-primary order. Each line has 3D X,Y,Z coordinates first, as floating-point numbers, then three integers for the R,G,B values of the pixel at that point. If the disparity at a pixel is filtered out, then the Z value is negative, indicating a filtered value. The text file also contains the image point coordinates and the disparity value.

Still images and still image sequences are loaded using the `File` menu. To load stereo frames, use the `Load Images (BMP)` menu item to bring up a file choice dialog. Choosing either BMP file of a pair automatically loads the other. In addition, if a color file or parameter file (`.ini`) is present, it is also loaded.

A file image sequence is a set of files with a base name `XXXNNN`. For example, `CAL001-L.BMP` is the left image of a stereo pair in a sequence. The sequence can start with any number, but the number must be 3 digits, and must increment sequentially for each stereo pair in the sequence. Choosing `Single` from the `smallv` interface will load the next file in the sequence. `Freeze` reloads the same file continuously, which is useful for changing stereo parameters in `smallv` and seeing their effects. `Continuous` mode is not available when loading a sequence of still images.

Note that loading a still image file using the `Load Images` menu item does not load the images into the video buffer. The only way to load images into the video buffer is with the `Load Video Buffer` menu item.

To save the current stereo image to a file, use the `Store Current (BMP)` menu item. Color information, if present, is saved as a 24-bit BMP file. A sequence of still images can be created by using the correct format for the base name of the stored files, as described above.

If stereo processing is turned on and a disparity image has been produced, then it can be saved as an 8-bit BMP file (`Store Disparity Image (BMP)`), or as a text file (`Store Disparity Image (Text)`). Since the number of bits in the disparity pixels is generally greater than 8, the BMP file only contains the high-order bit information. The text file contains the full value for the disparity.

3D information (X,Y,Z values) can be saved to a file, if stereo processing is selected. Use the `Store 3D Point Array` menu item.

2.3 Display

`smallv` displays two images in its display area. The left display is always the left input image. Input images are displayed in grayscale, unless color information is present: in this case, the left image will be shown in color.

The right display can be either the right input image, or the results of stereo processing. Processing results are always displayed in “greenscale”, using shades of green.

Either display can be turned off by unchecking the box underneath the display area. Turning off the display will let `smallv` run faster.

Images larger than 320x240 are automatically scaled down by factors of 2^n to fit into a 320x240 area. Smaller image sizes are displayed in the original size.

To display properly for human viewing, most video images are formatted to have a nonlinear relationship between the intensity of light at a pixel and the value of the video signal. The nonlinear function compensates for loss of definition in low light areas. Typically the function is x^γ, where γ is 0.45, and the signal is called “gamma corrected.” Digital cameras, such as the MEGA-D, do not necessarily have gamma correction. This is not a problem for stereo processing, but does cause the display to look very dark in low-light areas. You can add gamma correction to the displayed image by choosing an appropriate gamma value in the slider under the right display window (Figure 2-1).

2.4 Stereo Processing and Parameters

In `smallv`, stereo processing takes place in conjunction with the input of stereo images. The basic cycle is:

get stereo pair -> process pair -> display pair

The input is either from live video or the buffer (Sections 2.1 and 2.1.9). In freeze mode, the same pair is processed continuously, so adjustments can be made in stereo parameters.

2.4.1 Stereo Function

Stereo processing is turned on by choosing `Stereo` from the `Function` drop list. The stereo disparity image will appear in the right display. Stereo disparities are encoded by green: brighter green is a higher disparity, and therefore closer to the cameras (see Section 2.4.4 for a technical description of disparity).

Disparities represent the distance between the horizontal appearance of an object in the stereo images. The stereo process interpolates this distance to 1/16 pixel, e.g., a disparity value of 45 represents a displacement of 2 13/16 pixels. The maximum displacement currently supported is 80 pixels, so disparity values range from 0 (no disparity) to 1280. Disparity values are returned as 16-bit (short) integers. The values 0xFFFF and 0xFFFE are reserved for filtering results (Section 2.5)

If `smallv` is running on an MMX processor (Pentium or AMD) then stereo processing is much faster, taking advantage of the parallel data operations. The processor is queried and the MMX box is checked if the instructions are available. You can turn the MMX processing on and off by toggling the box. But, if your system does not have MMX instructions, you will not be able to turn it on.

2.4.2 3D Transformation

A pixel in the disparity image represents range to an object. This range, together with the position of the pixel in the image, determines the 3D position of the object relative to the stereo rig. SVS contains a function to convert disparity values to 3D points. These points can then be displayed in a 3D viewer.

To take the current disparity image and display it in 3D, press the `3D Display` button. An OpenGL window will show the 3D points constructed from the disparity image, and you can change the viewpoint of the window to see the 3D structure (Figure 2-7).

The coordinate system for the 3D image is taken from the optic center of the left camera of the stereo rig. Z is along the optic axis, with positive Z in front of the camera. X is along the camera scan lines, positive values to the right when looking along the Z axis. Y is vertical, perpendicular to the scan lines, with positive values down.

The X and Y position of the viewpoint, as well as rotation around the Z axis, can be changed with the sliders on the left side of the window. The scale of the image can be changed as well. Finally, the viewpoint can be rotated around a point in the image, to allow good assessment of the 3D quality of the stereo processing. The rotation point is selected automatically by finding the point closest to the left camera, near the optic ray of that camera. To rotate the image around this point, put the mouse in the 3D window, and drag the pointer while holding the left button down.

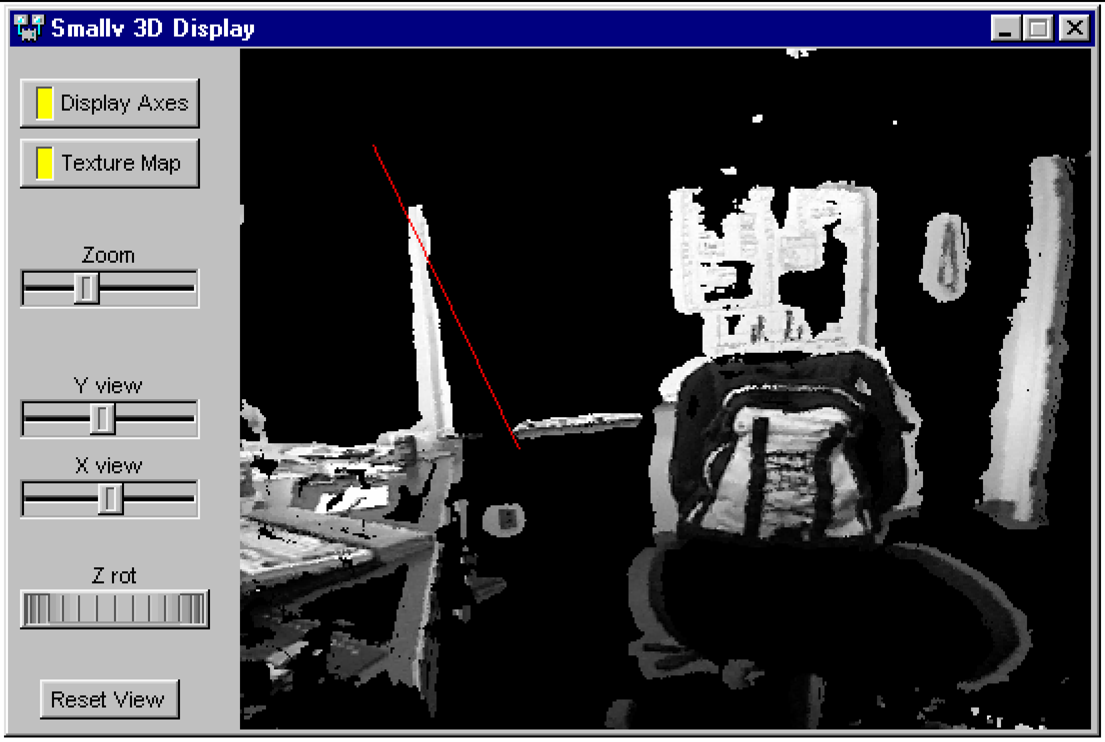


Figure 2-7 3D display window. The red ray is the optic ray from the left camera.

2.4.3 Calibration

For good stereo processing, the two images must be aligned correctly with respect to each other. The process of aligning images is called *calibration*. Generally speaking, there are two parts to calibration: *internal calibration*, dealing with the properties of the individual cameras and especially lens distortion; and *external calibration*, the spatial relationship of the cameras to each other. Both internal and external calibration are performed by an automatic calibration procedure described in Section 4. The procedure needs to be performed when lenses are changed, or the cameras are moved with respect to each other.

From the internal and external parameters, the calibration procedure computes an image warp for rectifying the left and right images. In stereo rectification, the images are effectively rotated about their centers of projection to establish the ideal stereo setup: two cameras with parallel optical axes and horizontal epipolar lines (see Fig. 2-2). Having the epipolar lines horizontal is crucial for correspondence finding in stereo, as stereo looks for matches along horizontal scanlines.

Figure 2-8 shows a pair of images of the calibration target taken with the MEGA-D stereo head and a 4.8 mm wide-angle lens. In the original images on the top, there is lens distortion, especially at the edges of the image: notice the curve in the target. Also, the images are not aligned vertically.

The bottom pair is the result of calibrating the stereo head and then rectifying the two original images. Now the images are aligned vertically, and all scene lines are straight in the images.

Figure 2-9 shows sample disparity images for uncalibrated and calibrated cameras. Without calibration, it is impossible for the stereo algorithms to find good matches.

Calibration parameters, along with other information about the stereo device settings, are stored in a parameter file that ends with the suffix "`.ini`". Parameter files can be loaded using the `File` menu. They can also be stored directly on the stereo device, for any device with a FW firmware version of 2.1 or greater (see the device manuals for information on the firmware). Parameter files stored on the device are automatically loaded into SVS when the device is opened.

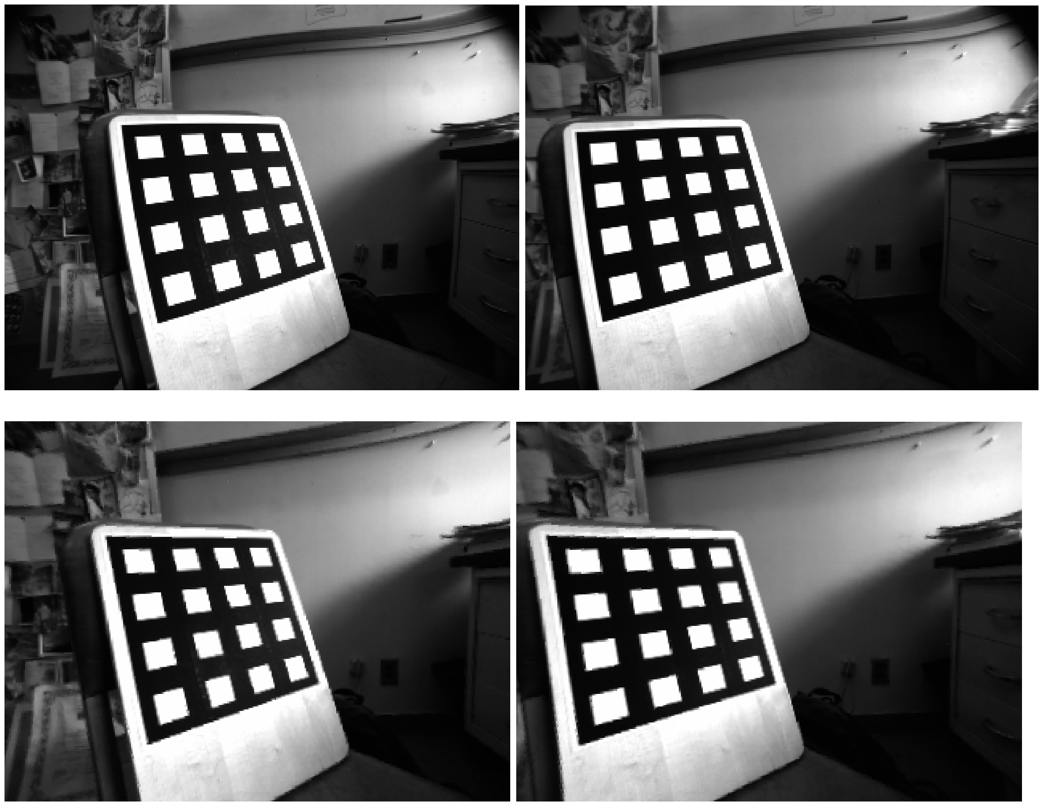


Figure 2-8 Original stereo pair (top) and rectified pair (bottom).

2.4.4 Disparity Search Range

Even with stereo rectification, it may not be possible to match every object in the scene, because the horopter is not large enough. In this case, the horopter can be enlarged by changing the number of disparities searched by the stereo process. This search range can vary from 8 to 128 pixels. Larger search ranges enlarge the horopter, but not in a linear fashion, i.e., a search range of 32 does *not* give twice the horopter range of 16; see Section 4 for technical details.

Changing the disparity search size affects the time it takes to process stereo. A search space of 32 pixels will take about twice as long as a search space of 16 pixels. It will actually take a little less, because there is some fixed overhead in processing the images. Obviously, the smallest search range necessary for the application is the best choice.

Disparities are interpolated to 1/16 pixel, so a search range of 16 means that there are 256 integral disparity values, ranging from 0 (no disparity) to 255 (maximum disparity of 15 15/16 pixels).

The search range is selected using the `Disparities` value in the `Parameters` area. When the range is switched, the disparity image will lighten or darken to reflect the changed values of disparities.

2.4.5 Adjusting the Horopter

The stereo rectification procedure sets up the horopter, or depth of field of stereo, so that objects are matched from infinity to some distance in front of the camera. Objects closer than this near point will not be matched, and will produce random disparity readings. The near point distance is a function of the search size, the stereo baseline, and the focal length of the camera lenses. One can adjust the horopter by adjusting a horizontal X offset, moving the depth range closer to the camera. The depth range desired in the end application would drive the setting of this parameter. For example, if the image does not contain any objects farther than a certain distance, the X offset can be adjusted so that the far point of the horopter is at that distance. Changing the X offset causes the disparity display to get uniformly lighter or darker, as the horopter is shifted and the disparity of an object changes. Adjusting the horopter to cover a specific range of depths is discussed in Section 4.

2.4.6 Pixel Information

SVS will show pixel information when the left button is clicked in either SVS display window. The information is displayed in the text window in the format:

```
(340,270) [v52] [dv480] X62 Y-10 Z1012 u(364,259)
```

The image coordinates of the mouse are given by the `x,y` values. The values in square brackets are the pixel values of the left and right images. If the right image is displaying stereo disparities, then the right value is the disparity value. The `X,Y,Z` values are the real-world coordinates of the image point, in mm. Note that `X,Y,Z` values are calculated only if stereo is being computed, and to be accurate, a good

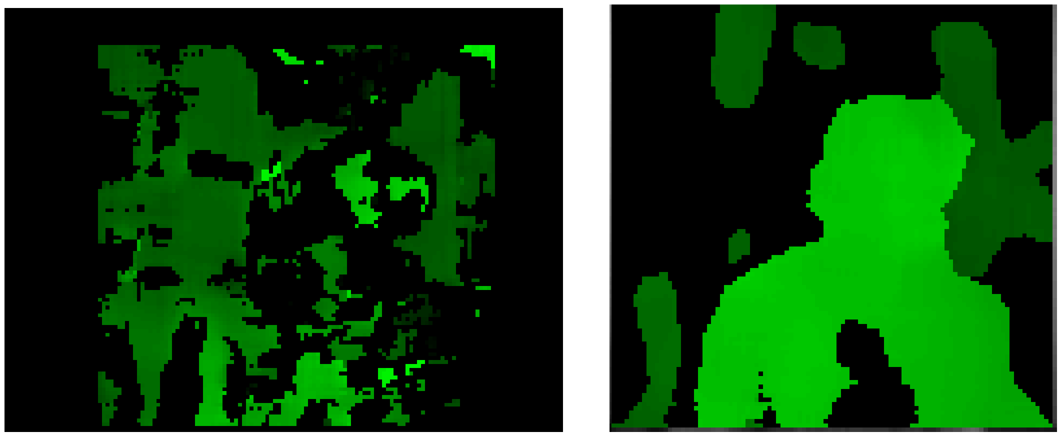


Figure 2-9 Uncalibrated (left) and calibrated (right) disparity images.

calibration file must be input (Section 4). The u() values are the original (non-rectified) image coordinates.

2.4.7 Correlation Window Size

The size of the correlation window used for matching affects the results of the stereo processing. A larger window will produce smoother disparity images, but will tend to "smear" objects, and will miss smaller objects. A smaller window will give more spatial detail, but will tend to be noisy. Typical sizes for the window are 9x9 or 11x11. The window size is selected using the `Sum window` drop list. In the MMX implementation, not all window sizes are supported. More technical information on the correlation window can be found in Section 3.4.

2.4.8 Multiscale Disparity

Multiscale processing can increase the amount of information available in the disparity image, at a nominal cost in processing time. In multiscale processing, the disparity calculation is carried out at the original resolution, and also on images reduced by 1/2. The extra disparity information is used to fill in dropouts in the original disparity calculation (Figure 3-8 in Section 2.4.8).

Multiscale processing is turned on in `smallv` by enabling the `MultiScale` button.

2.5 Filtering

Stereo processing will generally contain incorrect matches. There are two major sources for these errors: lack of sufficient image texture for a good match, and ambiguity in matching when the correlation window straddles a depth boundary in the image. The SVS stereo processing has two filters to identify these mismatches: a confidence measure for textureless areas, and a left/right check for depth boundaries.

Areas that are filtered appear black in the displayed disparity image. To distinguish them from valid disparity values, they have the special values 0xFFFF (confidence rejection) and 0xFFFE (uniqueness rejection).

2.5.1 Confidence Filter

The confidence filter eliminates stereo matches that have a low probability of success because of lack of image texture. There is a threshold, the confidence threshold, that acts as a cutoff. Weak textures give a confidence measure below the threshold, and are eliminated by the filter.

The confidence threshold is adjusted using the `Conf` spin control in the `Parameters` area. A good value can be found by pointing the stereo cameras at a textureless surface such as a blank wall, and starting the stereo process. There will be a lot of noise in the disparity display if the confidence threshold is set to 0. Adjust the threshold until the noise just disappears, and is replaced by a black area.

The computational cost of the confidence filter is negligible, and it is usually active in a stereo application.

2.5.2 Uniqueness Filter

Each stereo camera has a slightly different view of the scene, and at the boundaries of an object there will be an area that can be viewed by one camera but not the other. Such occluded areas cause problems for stereo matches. Fortunately, they can be detected by a consistency check in which the minimum correlation value must be *unique*, that is, lower than all other match values by a threshold. Typically, non-unique mimima will occur near the boundaries of objects.

The uniqueness threshold is controlled by the Unique spin control in the parameter area. It can be turned off completely by setting it to 0. Reasonable values depend on the amount of noise in the scene, and can range from 6 to 20 or so.

2.5.3 Speckle Filter

There are often small disparity regions that are not correct, a sort of salt-and-pepper effect caused by image noise. The speckle filter eliminates these small regions by imposing a region size constraint. There are two parameters:

- *Speckle diff* is the maximum difference between neighboring disparity pixels that are considered to be in the same region. Units are in 1/16 of a pixel (i.e., the same units as disparities). The default value is 4, that is, disparity neighbors that differ by up to 4/16 of a pixel will be considered part of the same region.
- *Speckle area* is the smallest area that will be considered a valid disparity region. The default value is 100 (pixels).

The area parameter can be adjusted in the `Speckle` value control in the smallv application.

2.6 Saving and Restoring Parameters

All of the parameters that control the operation of the SVS Stereo Engine can be saved to a file for later use. Parameter files can be loaded and saved using the File menu: `Load Param File` and `Store Param File`, and with API functions.

The file `data/mdcs-6.ini` contains a sample file for a 6 mm lens on an STH-MDCS stereo rig (see below). It serves as an example of the settings available through parameter files. In practice, these settings are usually computed using the calibration program, and then saved to a file for later use. But, it is also possible to change the settings directly in the file.

For information about the calibration parameters, please see the Calibration Addendum to the Users' Manual.

2.6.1 Saving and Reading Parameters on a Stereo Device

With DCS series devices (firmware version 2.1 or greater), the parameters, including calibration, can be saved to flash storage on the device, using the `File` menu or API calls. These parameters are automatically downloaded into SVS whenever the device is opened, either with the `Video` input pulldown or an API call.

Parameters may be saved to a device using the `File->Upload to Device` menu item of `smallv`. All current parameters are saved. They may be downloaded using the `File->Download from Device` menu item. The download command is used to restore the parameters after they have been changed, since they are automatically downloaded when the device is opened. There are also utility applications for loading and storing parameter files to stereo devices; see the appropriate stereo device manual.

2.6.2 Partial Parameter Files

It is not necessary for a parameter file to contain all of the parameters for a device. For example, saving a parameter file from the `smallvcal` calibration application window saves just the calibration parameters. Partial files are useful when you want to change just a few of the parameters of a device, and leave the rest alone.

Parameter files may be edited with a text editor to add or remove individual parameter values.

2.6.3 Loading Parameter Files

For parameter files to take effect, they must be loaded at the proper time. In general, parameter files should be loaded *after* opening the stereo device (with the `Video` pulldown or an API call). Opening a stereo device initializes default parameters for the device, which can override the effect of the parameters input from a parameter file. These defaults affect parameters in the `[Image]` section of the calibration file.

Calibration and stereo parameters can be loaded at any time, and will not be affected by opening or closing a device (unless the device itself contains an on-board parameter file). These are the parameters in the `[stereo]`, `[external]`, `[left camera]`, and `[right camera]` sections of the parameter file.

2.6.4 Sample Parameter File with Annotations

```
# SVS Engine v 3.2 Stereo Camera Parameter File

[image]               # image frame parameters
max_linelen 1280      # max size of imager
max_lines 960
max_decimation 1      # allowable decimation at imager
max_binning 2         # allowable binning in driver
max_framediv 1        # allowable frame division in driver
gamma 0.850000        # gamma correction for display
```

```
color_right 0          # 0 for monochrome, 1 for color
color 0                # 0 for monochrome, 1 for color
ix 0                   # subwindow offset
iy 0
vergence 0             # vergence of right subwindow
rectified 0            # image rectified already (for files)
width 320              # subwindow size
height 240
linelen 320            # window size
lines 240
decimation 1           # current decimation and binning
binning 2
framediv 1             # current framediv, 0 if turned off
subwindow 0            # 1 for subwindow capability
have_rect 1            # 1 if we have rectification parameters
autogain 0             # 1 if autogain on
autoexposure 1         # 1 if autoexposure on
autowhite 0            # 1 if auto white balance on
autobrightness 0       # 1 if auto brightness on
gain 21                # current gain value [0,100]
exposure 83            # current exposure [0,100]
contrast 0             # current contrast [0,100] (analog only)
brightness 0           # current brightness [0,100]
saturation 0           # current saturation [0,100]
red 0                  # current red gain [-40,40]
blue 0                 # current blue gain [-40,40]


[stereo]               # stereo processing parameters
convx 9                # prefilter kernel size
convy 9
corrxsize 11           # correlation window size
corrysize 11
thresh 212             # confidence threshold value
lr 1                   # left/right filter on (1) or off (0)
ndisp 32               # number of disparities to search
dpp 16                 # subpixel interpolation, do not change!
offx 0                 # horopter offset
offy 0                 # vertical image offset, not used
frame 1.000000         # frame expansion factor, 1.0 is normal


[external]
Tx -89.458214          # translation between left and right cameras
Ty -0.277252
Tz -0.923279
Rx -0.008051           # rotation between left and right cameras
Ry -0.003771
Rz -0.000458


[left camera]
pwidth 640             # number of pixels in calibration images
pheight 480
dpx 0.012000           # effective pixel spacing (mm) for this
dpy 0.012000           # resolution
sx 1.000000            # aspect ratio, analog cameras only
Cx 306.260123          # camera center, pixels
Cy 286.081223
alpha 0.000000         # skew parameter, analog cameras only
```

```
f 511.361705          # focal length (pixels) in X direction
fy 513.710882         # focal length (pixels) in Y direction
kappa1 -0.159408      # radial distortion parameters
kappa2 0.161209
kappa3 0.000000
tau1 0.000000         # tangential distortion parameters
tau2 0.000000
proj                  # projection matrix: from left camera 3D coords
                      #    to left rectified coordinates
  5.240000e+002 0.000000e+000 3.306526e+002 0.000000e+000
  0.000000e+000 5.240000e+002 2.774974e+002 0.000000e+000
  0.000000e+000 0.000000e+000 1.000000e+000 0.000000e+000
rect                  # rectification matrix for left camera
  9.990546e-001 -1.105539e-003 -4.345896e-002
  1.097792e-003 9.999994e-001 -2.021196e-004
  4.345916e-002 1.542196e-004 9.990552e-001


[right camera]
pwidth 640            # number of pixels in calibration images
pheight 480
dpx 0.012000          # effective pixel spacing (mm) for this
dpy 0.012000          # resolution
sx 1.000000           # aspect ratio, analog cameras only
Cx 323.260123         # camera center, pixels
Cy 264.081223
alpha 0.000000        # skew parameter, analog cameras only
f 521.361705          # focal length (pixels) in X direction
fy 523.710882         # focal length (pixels) in Y direction
kappa1 -0.152767      # radial distortion parameters
kappa2 0.142915
kappa3 0.000000
tau1 0.000000         # tangential distortion parameters
tau2 0.000000
proj                  # projection matrix: from right camera 3D coords
                      #    to left rectified coordinates
  5.240000e+002 0.000000e+000 3.306526e+002 -4.659122e+004
  0.000000e+000 5.240000e+002 2.774974e+002 0.000000e+000
  0.000000e+000 0.000000e+000 1.000000e+000 0.000000e+000
rect                  # rectification matrix for right camera
  9.988236e-001 1.777051e-003 -4.845785e-002
  -1.768413e-003 9.999984e-001 2.211362e-004
  4.845816e-002 -1.351825e-004 9.988252e-001
```

3 Stereo Geometry

Stereo algorithms compute range information to objects by using triangulation. Two images at different viewpoints see the object at different positions: the image difference is called *disparity*. This section discusses the basic equations that govern the relationship between disparity and range.

More detailed information on stereo geometry, as well as the process of *rectifying* input images to produce idealized images, can be found in the Calibration Addendum.

3.1 Disparity

The figure below displays a simplified view of stereo geometry. Two images of the same object are taken from different viewpoints. The distance between the viewpoints is called the *baseline* (**b**). The focal length of the lenses is **f**. The horizontal distance from the image center to the object image is **dl** for the left image, and **dr** for the right image.

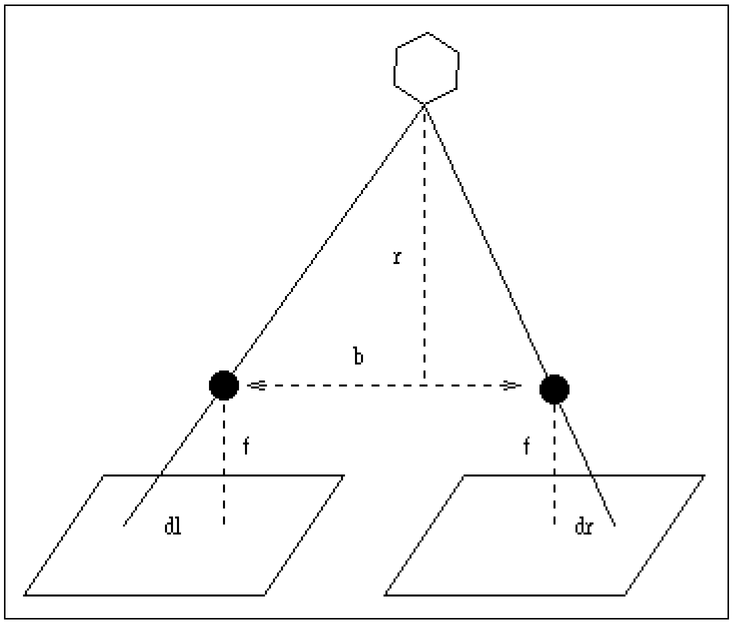


Figure 3-1. Definition of disparity: offset of the image location of an object.

Normally, we set up the stereo cameras so that their image planes are embedded within the same plane. Under this condition, the difference between **dl** and **dr** is called the *disparity*, and is directly related to the distance **r** of the object normal to the image plane. The relationship is:

(1) $\mathbf{r = bf / d}$, where $\mathbf{d = dl - dr}$.

Using Equation 1, we can plot range as a function of disparity for the STH-V1 stereo head. At their smallest baseline, the cameras are about 8 cm apart. The pixels are 14 um wide, and the standard lenses have a focal length of 6.3 mm. For this example, we get the plot in Figure 3-2. The minimum range in this plot is 1/2 meter; at this point, the disparity is over 70 pixels; the maximum range is about 35 meters. Because of the inverse relationship, most of the change in disparity takes place in the first several meters.

The range calculation of Equation (1) assumes that the cameras are perfectly aligned, with parallel image planes. In practice this is often not the case, and the disparity returned by the Stereo Engine will be offset from the ideal disparity by some amount X_0. The offset is explained in the section below on the horopter, and in the section on calibration.

The disparity value can be used to find which pixels correspond in the two images. The left image is considered to be the reference image. Pixels in the left image have higher X coordinates than their corresponding pixels in the right image (the X coordinate in Figure 3-1 goes from left to right – the images are inverted). The Y coordinates are the same. The X coordinates are related by:

(1b) $\mathbf{x_R} = \mathbf{x_L} - 16*\mathbf{d}$.

Disparities are specified in units of 1/16 pixel. Equation (1b) assumes that there is no X offset between the images, and that the calibration was specified as having zero disparity at infinity (the normal case). If there is an offset, (1b) becomes

(1c) $\mathbf{x_R} = \mathbf{x_L} - 16*\mathbf{d} - \mathbf{x_{off}}$

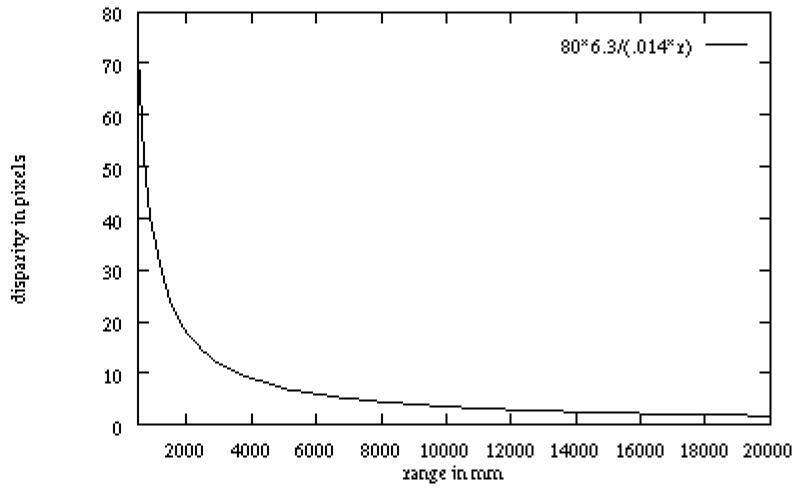


Figure 3-2. Inverse relationship between disparity and range. This plot is for a focal length of 6.3 mm, a baseline of 80 mm, and a pixel width of 14 mm.

See the next section for information about X offsets, and the Calibration Addendum for information about disparities at infinity. .

3.2 Horopter

Stereo algorithms typically search only a window of disparities, e.g., 16 or 32 disparities. In this case, the range of objects that they can successfully determine is restricted to some interval. The *horopter* is the 3D volume that is covered by the search range of the stereo algorithm. The horopter depends on the camera parameters and stereo baseline, the disparity search range, and the X offset. Figure 3-3 shows a typical horopter. The stereo algorithm searches a 16-pixel range of disparities to find a match. An object

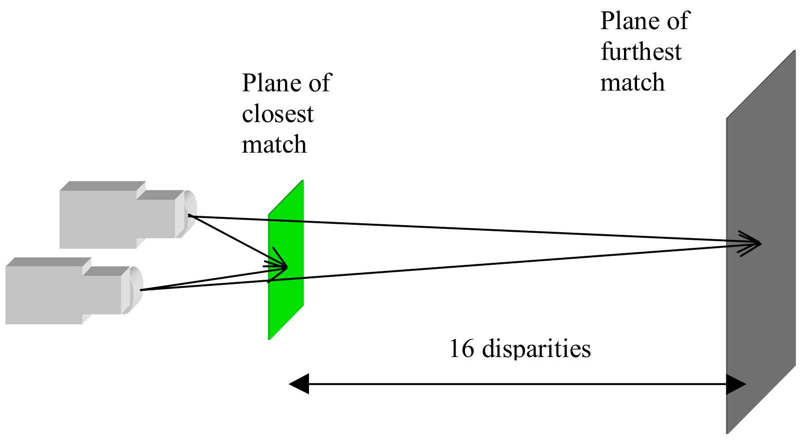


Figure 3-3 Horopter planes for a 16-pixel disparity search.

that has a valid match must lie in the region between the two planes shown in the figure. The nearer plane has the highest disparity (15), and the farthest plane has the lowest disparity (0).

The placement of the horopter can be varied by changing the X offset between the two images, which essentially changes the search window for a stereo match. Figure 3-5 shows the raw disparities for a typical stereo head. The cameras are slightly verged, so a zero disparity plane (where an object appears at the same place in both images) occurs at some finite distance in front of the cameras. If the stereo algorithm is searching 5 disparities, then without any X offset, it will search as shown in the top red arrow, that is, from disparity 0 to disparity 4. By offsetting one image in the X direction by *n* pixels, the horopter can be changed to go from *–n* to *5-n* raw disparities. This search range is indicated by the lower red arrow.

Generally, it is a good idea to set the X offset to compensate for camera vergence or divergence, that is, to set it so that the furthest horopter plane is at infinity. The reason that this is a good idea is because it's usually possible to control how close objects get to the camera, but not how far away. The offset that puts the far horopter plane at infinity is called X_0. With this offset, a disparity of 0 indicates an infinitely far object.

The horopter can be determined from Equation (1). For example, if the disparity search window is 0-31, the horopter (using the graph above) will be from approximately 1 meter to infinity. The search window can be moved to an offset by shifting the stereo images along the baseline. The same 32 pixel window could be moved to cover 10-41 pixel disparities, with a corresponding horopter of 0.8 meters to 2.2 meters.

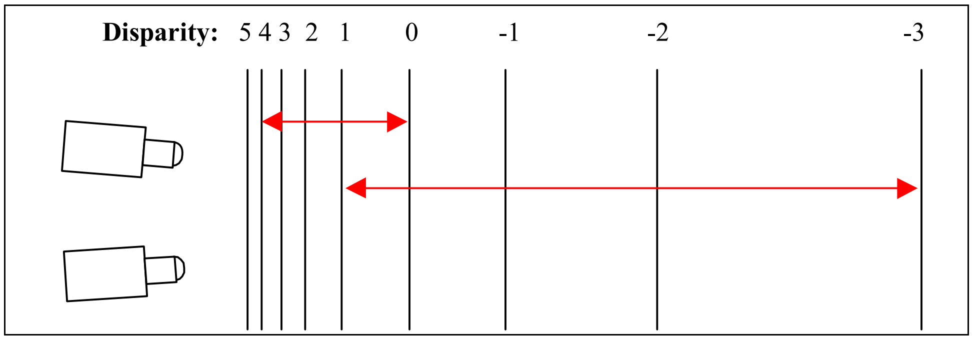


Figure 3-5. Planes of constant disparity for verged stereo cameras. A search range of 5 pixels can cover different horopters, depending on how the search is offset between the cameras.

The location and size of the horopter depends on the application. If an object falls outside the horopter, then its true disparity will not be found, and instead it will get some random distribution of disparities. Figure 3-4 shows what happens when the object's range falls outside the horopter. In the left image, the disparity search window is correctly positioned so that objects from 1 meter to infinity are in view. In the right image, the window has been moved back so that objects have higher disparities. However, close objects are now outside of the horopter, and their disparity image has been "broken up" into a random pattern. This is typical of the disparity images produced by objects outside the horopter.

For a given application, the horopter must be large enough to encompass the ranges of objects in the application. In most cases, this will mean positioning the upper end of the horopter at infinity, and making the search window large enough to see the closest objects.

The horopter is influenced not only by the search window and offset, but also by the camera parameters and the baseline. The horopter can be made larger by some combination of the following:

- Decreasing the baseline.
- Decreasing the focal length (wider angle lenses).
- Increasing pixel width.
- Increasing the disparity search window size.

As the cameras are moved together, their viewpoints come closer, and image differences like disparity are lessened. Decreasing the focal length changes the image geometry so that perceived sizes are smaller, and has a similar effect. It also makes the field of view larger, which can be beneficial in many applications. However, very small focal length lenses often have significant distortion that must be corrected (see the section on calibration). Another way to change the image geometry is to make the pixels wider. This can be done by scaling the image, e.g., from 320x240 to 160x120, which doubles the pixel size. Note that it is only necessary to change the pixel width. Most framegrabbers have hardware scaling to

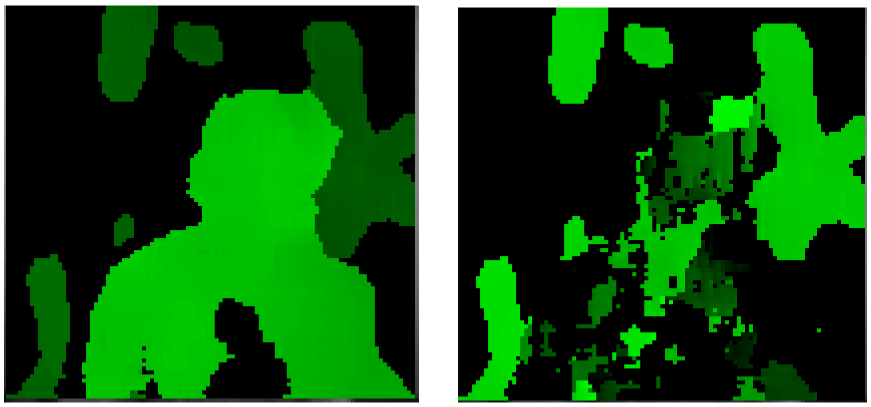


Figure 3-4. Disparity image for all regions withing the horopter (left) and some regions outside the horopter (right).

arbitrary resolutions.

These first three options change the camera geometry, and thus have a corresponding effect on the range resolution, which decreases (see below). The only way to increase the horopter size and maintain range resolution is to increase the disparity search window size, which leads to more computation. Multiresolution methods, which use several sizes of an image, each with its own horopter, are one way to minimize computation (see, for example, the paper by Iocchi and Konolige at *www.ai.sri.com/~konolige/svs*).

3.3 Range Resolution

Often it's important to know the minimal change in range that stereo can differentiate, that is, the *range resolution* of the method. Give the discussion of stereo geometry above, it's easy to see that that range resolution is a function of the range itself. At closer ranges, the resolution is much better than farther ranges.

Range resolution is governed by the following equation.

(2) $\Delta\ \mathbf{r} = (\mathbf{r}^2/\mathbf{bf})\ \Delta\mathbf{d}$

The range resolution, $\Delta\mathbf{r}$, is the smallest change in range that is discernable by the stereo geometry, given a change in disparity of $\Delta\mathbf{d}$. The range resolution goes up (gets worse) as the square of the range. The baseline and focal length both have an inverse influence on the resolution, so that larger baselines and focal lengths (telephoto) make the range resolution better. Finally, the pixel size has a direct influence, so that smaller pixel sizes give better resolution. Typically, stereo algorithms can report disparities with subpixel precision, which also increases range resolution.

The figure below plots range resolution as a function of range for the STH-MD1 (MEGA-D) stereo head, which has a baseline of 9 cm. The Stereo Engine interpolates disparities to 1/16 pixel, so $\Delta\mathbf{d}$ is 1/16 * 7.5 um = 0.08533 um. The range resolution is shown for a sampling of different lens focal lengths. At any object distance, the range resolution is a linear function of the lens focal length.

Equation 2 shows the range resolution of a perfect stereo system. In practice, video noise, matching errors, and the spreading effect of the correlation window all contribute to degrading this resolution.

Range resolution is not the same as range accuracy, which is a measure of how well the range computed by stereo compares with the actual range. Range accuracy is sensitive to errors in camera calibration, including lens distortion and camera alignment errors.

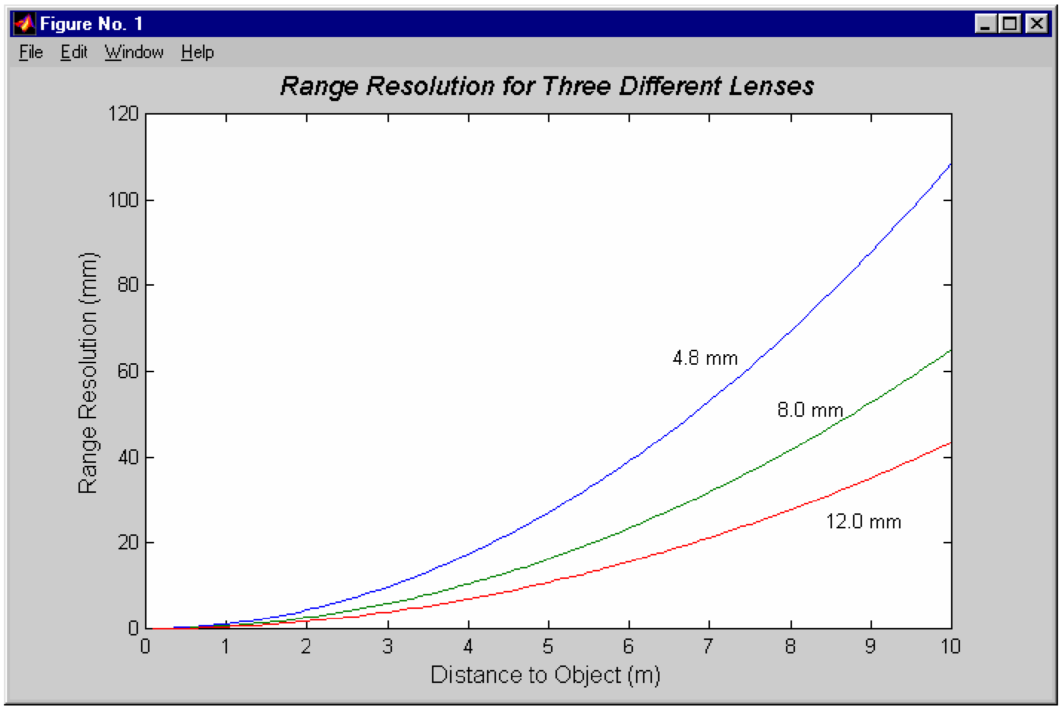


Figure 3-6. Range resolution as a function of range. This plot assumes a baseline of 90 mm, and a pixel size of 7.5 um, with subpixel resolution of 1/16 pixel.

3.4 Area Correlation Window

Stereo analysis is the process of measuring range to an object based on a comparison of the object projection on two or more images. The fundamental problem in stereo analysis is finding corresponding elements between the images. Once the match is made, the range to the object can be computed using the image geometry.

Area correlation compares small patches, or *windows*, among images using correlation. The window size is a compromise, since small windows are more likely to be similar in images with different viewpoints, but larger windows increase the signal-to-noise ratio. Figure 3-7 shows a sequence of disparity images using window sizes from 7x7 to 13x13. The texture filter was turned off to see the effects on less-textured areas, but the left/right check was left turned on.

There are several interesting trends that appear in this side-by-side comparison. First, the effect of better signal-to-noise ratios, especially for less-textured areas, is clearly seen as noise disparities are eliminated in the larger window sizes. But there is a tradeoff in disparity image spatial resolution. Large windows tend to "smear" foreground objects, so that the image of a close object appears larger in the disparity image than in the original input image. The size of the subject's head grows appreciably at the end of the sequence. Also, in the 7x7 the nose can be seen protruding slightly; at 13x13, it has been smeared out to cover most of the face.

One of the hardest problems with any stereo algorithm is to match very small objects in the image. If an object does not subsume enough pixels to cover an appreciable portion of the area correlation window, it will be invisible to stereo processing. If you want to match small objects , you have to use imagers with good enough spatial resolution to put lots of pixels on the object.

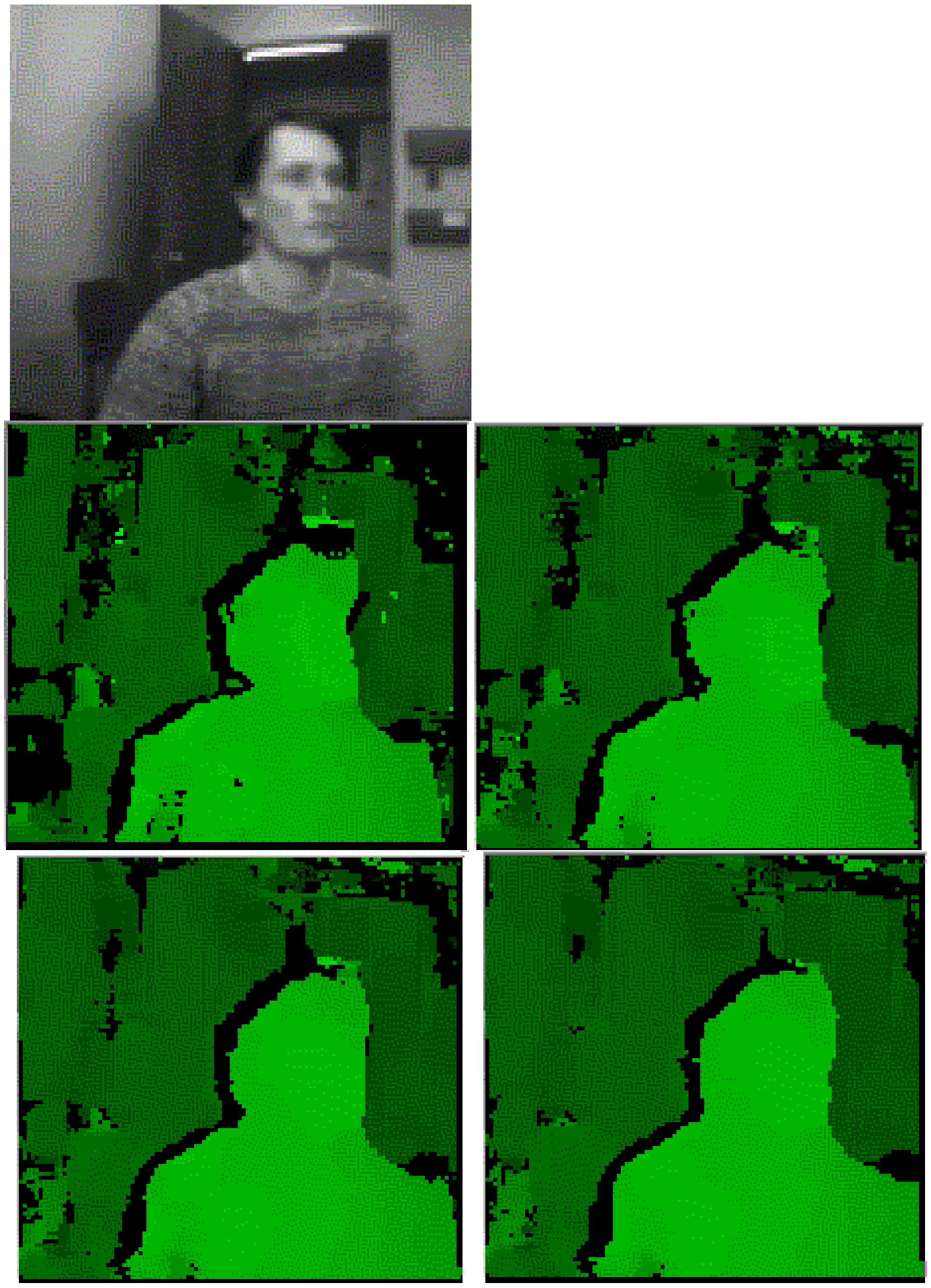
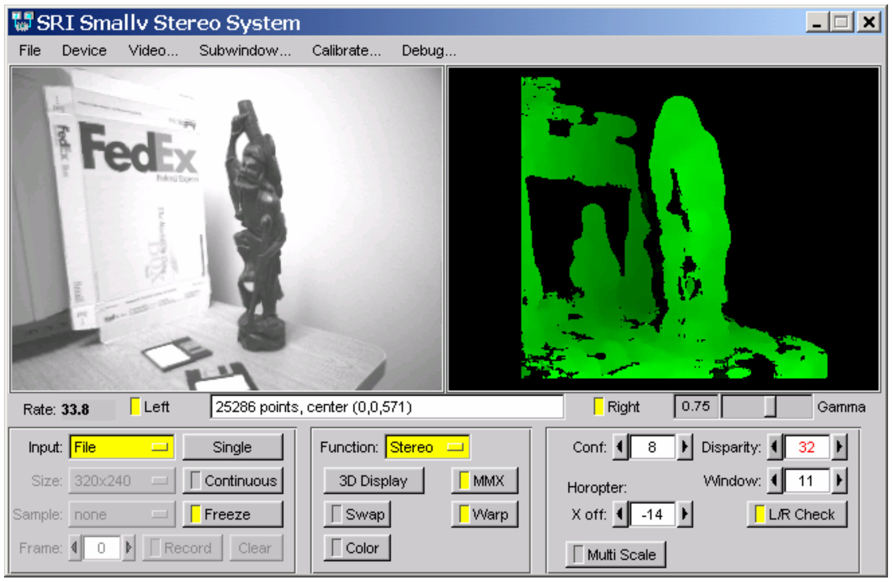


Figure 3-7 Effects of the area correlation window size. At top is the original left intensity image. The greenscale images show windows of 7x7, 9x9, 11x11, and 13x13 windows (clockwise from upper left).

3.5 Multiscale Disparity

Multiscale processing can increase the amount of information available in the disparity image, at a nominal cost in processing time. In multiscale processing, the disparity calculation is carried out at the original resolution, and also on images reduced by 1/2. The extra disparity information is used to fill in dropouts in the original disparity calculation (Figure 3-8).

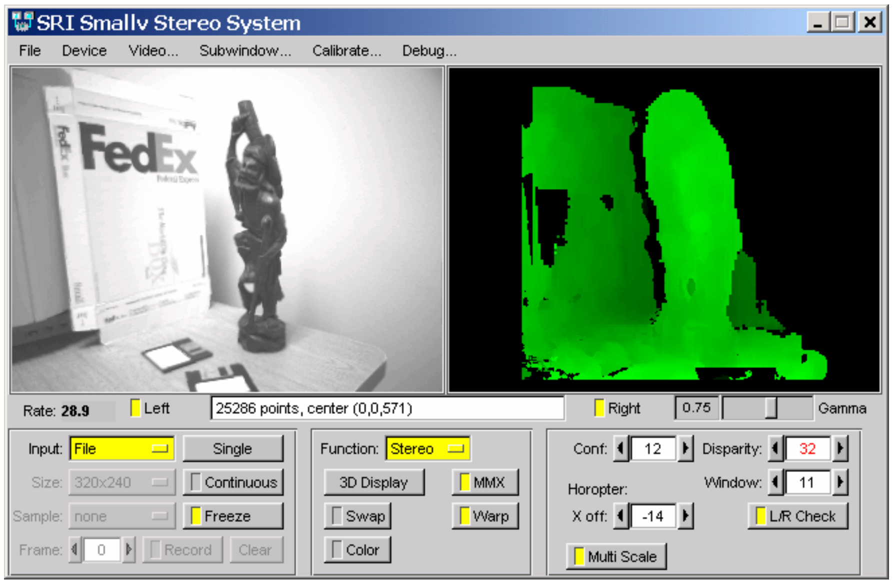


Rate: 33.8 Left 25286 points, center (0,0,571) Right 0.75

Input: File Size: 320x240 Sample: none Frame: 0

Function: Stereo

Conf: 8 Disparity: 32 Window: 11 X off: -14



Rate: 28.9 Left 25286 points, center (0,0,571) Right 0.75

Input: File Size: 320x240 Sample: none Frame: 0

Function: Stereo

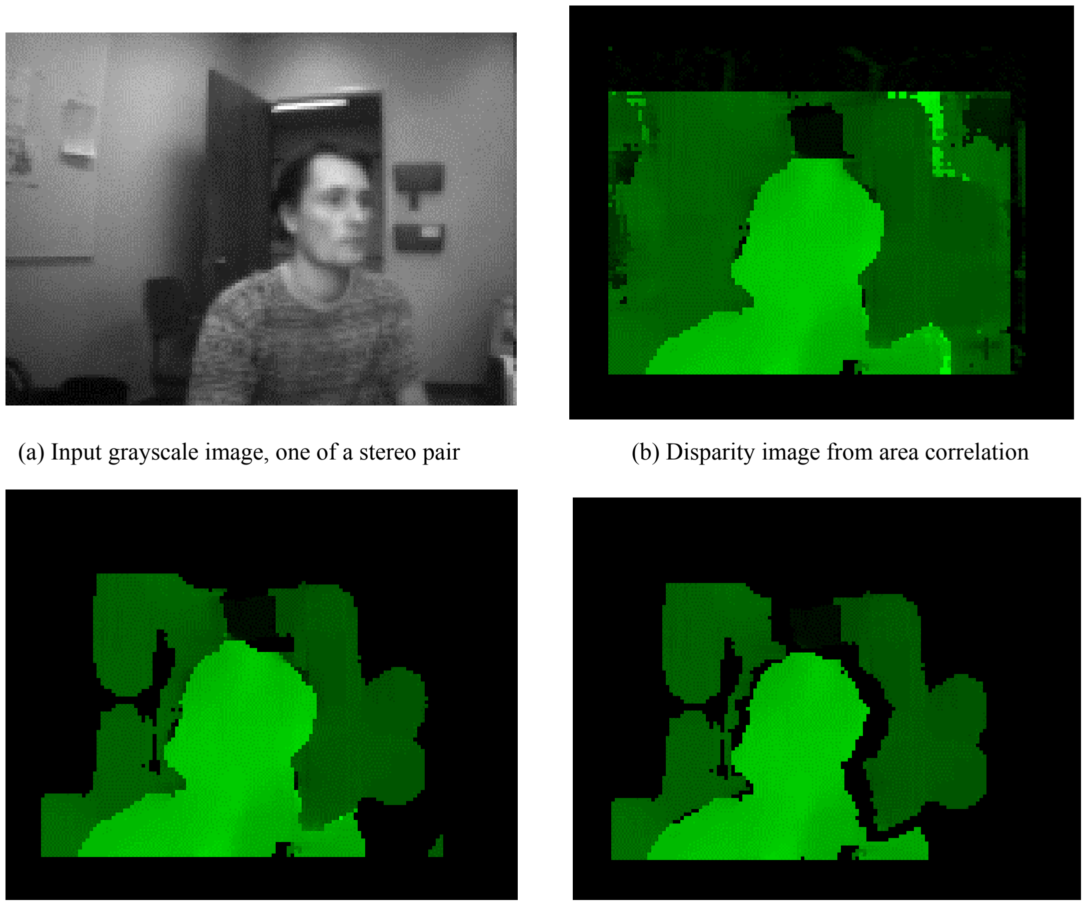
Conf: 12 Disparity: 32 Window: 11 X off: -14

Figure 3-8 Effects of multiscale disparity calculation. Upper figure shows disparity dropouts in a typical scene, where there is not enough texture for correlation to be reliable. Adding disparity information from a ½ resolution image (lower part of figure) shows additional coverage in the disparity image.

3.6 Filtering

Like most vision algorithms, the results of stereo processing can contain errors. In the case of stereo, these errors result from noisy video signals, and from the difficulty of matching untextured or regularly textured image areas. Figure 3-9 shows a typical disparity image produced by the SRI algorithm. Higher disparities (closer objects) are indicated by brighter green (or white, if this paper is printed without color). There are 64 possible levels of disparity; in the figure, the closest disparities are around 40, while the furthest are about 5. Note the significant errors in the upper left and right portion of the image, where uniform areas make it hard to estimate the disparity.

In Figure 3-9(c), the interest operator is applied as a postfilter. Areas with insufficient texture are rejected as low confidence: they appear black in the picture. Although the interest operator requires a threshold, it's straightforward to set it based on noise present in the video input. Showing a blank gray area to the imagers produces an interest level related only to the video noise; the threshold is set slightly above that. Or, more simply, you can use the temporal variance of poorly textured matches to adjust the texture threshold. Observing the disparity image during realtime display, there will usually be areas that flicker rapidly. Adjust the threshold upward until these regions disappear. If there are no such regions, adjust the threshold downward until just before they appear.


(a) Input grayscale image, one of a stereo pair

(b) Disparity image from area correlation

(c) Texture filter applied

(d) Left/right and texture filter applied

Figure 3-9 Post-filters applied to a disparity image. (c) is a texture filter that eliminates textureless areas. (d) is a consistency check between left and right stereo matches.

There are still errors in portions of the image with disparity discontinuities, such as the side of the subject's head. These errors are caused by overlapping the correlation window on areas with very different disparities. Application of a uniqueness check can eliminate these errors, as in Figure 3-9(d).

In practice, the combination of an interest operator and uniqueness check has proven to be effective at eliminating bad matches.

3.7 Performance

NOTE: Version 4.0 of SVS has a new implementation of the correlation algorithms, with much better performance. The table below will be updated shortly.

Using standard PC hardware, running either MS Windows 95/98/ME/2000/XP/NT or Linux, the SVS can compute stereo range in real time. Table 3-1 gives some typical timings for various systems. Because the Stereo Engine has a very small memory footprint, the timings scale almost linearly with increasing processor speed. These timings include the complete stereo algorithm detailed above: disparity computation and interpolation, and post-filtering using a texture filter and left/right filter. Input is a rectified grayscale image pair.

FPS is frames per second. FOM is Figure of Merit, measured in mega-pixels per disparity-second. The FOM is the best judge of the performance of SVS on a processor – higher numbers mean better performance. Note that the Pentium M is the best processor for SVS.

Processor	Speed	OS	Memory	Resolution	Disp	FPS	FOM (Mp/d-s)
Pentium M	1.4 GHz	MSW	500 MB	512x384	48	28	264
Pentium M	1.4 GHz	MSW	500 MB	640x480	64	15	295
Pentium M	2.0 GHz	Linux	1 GB	512x384	48	43	405
Pentium 4	2.5 GHz	Linux	500 MB	640x480	64	15	295

Table 3-1 Processing rates on a Pentium III 500 MHz machine.

3.8 Ideal Stereo Model

For a good understanding of stereo processing, it is necessary to understand more precisely the steps involved. This subsection gives some more detail of the fundamental geometry of stereo, and in particular the relationship of the images to the 3D world via projection and reprojection. A more in-depth discussion of the geometry, and the rectification process, can be found in the Calibration Addendum.

The overall stereo process is described as two stages:

Input images – *rectify* –> **Ideal images –** *stereo correlation* –> **Disparity image**

The rectification step is essential. It converts the input images into an *idealized stereo pair*, with a very particular geometry. This geometry makes it easier for the stereo correlation algorithms to find the correct match (disparity) for each pixel. It also enables SVS to *reproject* a pixel to its three-dimensional coordinates, given the disparity.

Calibration produces information necessary for the rectification step, and also gives the parameters of the resultant ideal stereo pair. When a calibration is present in the SVS system, input images are usually converted to ideal images before they are displayed, and when they are saved to files. In general, the user should interact only with the ideal images, since their relationship to the disparity results is simple. All of the discussion of this section has dealt with idealized images.

Figure 3-10 shows the geometry of the ideal images. The main 3D coordinate system is centered on the focal point (the camera center) of the left camera. The focal point is somewhere inside the left camera lens. Positive Z directions are along the camera principal ray. Positive X is to the right looking along the ray, and positive Y is down. This gives a right-handed coordinate system.

Both images are embedded in a common plane, perpendicular to the principal rays. Also, the image horizontal axes line up, so that the first line of the left image is the same as the first line of the right image.

The principal ray of each camera pierces the image at the coordinates (C_x,C_y). These coordinates are typically not the center of the image, although they are close. They are the normally the same coordinates

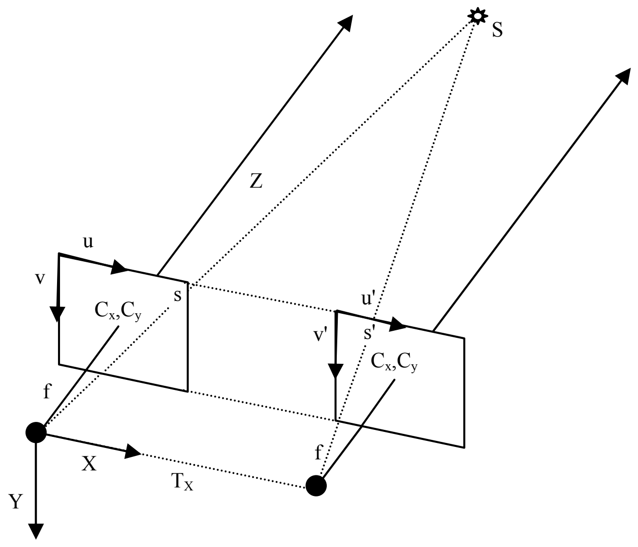


Figure 3-10 Basic stereo geometry. This figure shows the relationship of two *ideal* stereo cameras. The global coordinate system is centered on the focal point (camera center) of the left camera. It is a right-handed system, with positive Z in front of the camera, and positive X to the right. The camera principal ray pierces the image plane at C_x,C_y, which is the same in both cameras. The focal length is also the same. The images are lined up, with *v=v'* for the coordinates of any scene point projected into the images. The distance between the focal points is aligned with the X axis.

in both images; but in exceptional cases, especially for verged images, the C_x values can differ. The focal lengths *f* of both images are the same.

Any 3D point *S* projects to a point in the images along a ray through the focal point. Note that the points *s* and *s'* always have the same *v* coordinate in the two images. The difference in their *u* coordinates is the *disparity* of the 3D point, which is related to its distance from the focal point, and the baseline T_X that separates the focal points.

A 3D point can be projected into either the left or right image by a matrix multiplication, using the *projection matrix* (described in the next subsection). Similarly, a point in the image can be *reprojected* into 3D space using the *reprojection matrix*. Both these operations are supported by several functions in the SVS API (see Section 6).

All of the parameters in Figure 3-10 are found in the projection matrices for the stereo device, produced by calibration.

3.8.1 Projection Matrix

The projection matrix transforms 3D coordinates into idealized image coordinates. The 3D coordinates are in the frame of the left camera (see Figure 3-10). There is a projection matrix for the left camera, and one for the right camera. The form of the 3x4 projection matrix **P** is shown in Table 3-2. A point in 3D $(X,Y,Z)^T$ is represented by homogeneous coordinates $(X,Y,Z,1)^T$ and the projection is performed using a matrix multiply

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \mathbf{P} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

where $(u/w, v/w)$ are the idealized image coordinates. Note that this equation holds for *idealized* images, that is, the coordinates (u,v) are in the rectified image.

The function `svsProject3D` will perform the projection operation, given a calibration parameter set. There is also a member function `Project3D` or the `svsAcquireImages` class, for performing the same operation.

NOTE: The projection matrices contain all of the essential geometry of the idealized stereo pair. In particular, they have the center of projection of the principal ray, the focal length, and the baseline between the cameras. The focal length and image centers are expressed in pixels, and the baseline is in mm. Note that the baseline is part of the 1,4 element of the right image projection matrix. This element is 0 for the left image.

$$\begin{bmatrix} F_x & 0 & C_x & -F_x T_x \\ 0 & F_y & C_y & 0 \\ 0 & 0 & 1 & 0 \\ & & & \end{bmatrix}$$

Table 3-2 Projection matrix for a single camera. *Fx, Fy* is the focal length of the rectified image (pixels), and *Cx,Cy* is the optical center (pixels). *Tx* is the translation of the camera relative to the left (reference) camera. For the left camera, it is 0; for the right camera, it is the baseline times the x focal length (in pixel*mm units). Note that the focal lengths are for the *rectified* images, and thus will be the same; these are *not* the focal lengths *f,fy* given explicitly in the parameter file, which are for the original images.

3.8.2 Reprojection

A point (*u,v*) in the left camera can be re-projected to 3D coordinates, if its disparity is known. The SVS functions `Calc3D` and `CalcPoint3D` in the class `svsStereoProcess` are provided to perform this calculation. Here we give the equations that govern the transformation.

The reprojection transformation is influenced by the *frame* and scaling factors in the calibration (e.g., if the calibration was performed at one resolution, and the input images are at a different resolution). It also depends on any horopter offset. For the following calculation, we assume that the *frame* and scaling are unity, and the horopter offset is zero.

Define the *reprojection matrix* as follows:

$$\mathbf{Q} \equiv \begin{bmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 0 & f \\ 0 & 0 & \frac{-1}{T_x} & \frac{C_x - C'_x}{T_x} \end{bmatrix}$$

where C'_x is from the right projection matrix, and the other parameters are from the left projection matrix.

NOTE: all parameters of **Q** are in the projection matrix of the calibration output – see Table 3-2 and Figure 3-10. In particular, the image center coordinates C_x and C'_x are *not* the center of distortion given explicitly in the calibration parameters; they are the 1,3 element of the projection matrices.

Normally, the calibration produced by SVS will set C'_x equal to C_x, so the last term is 0. Under this condition, the disparity at infinity will be 0. For verged cameras (pointing inwards rather than parallel), it may be useful to have C'_x different from C_x, in order to get the right rectified image to be less offset. In this case, the disparity at infinity will not be zero, it will be negative. The calculation of X,Y,Z coordinates will still be correct, though, using the equation below.

From an image point homogeneous coordinates $(u,v,d,1)^T$, with *d* the disparity, the corresponding 3D point in homogenous coordinates is calculated as:

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = \mathbf{Q} \begin{bmatrix} u \\ v \\ d \\ 1 \end{bmatrix}$$

The actual 3D point is (*X/W, Y/W, Z/W*).

4 Calibration

Most stereo camera setups differ from an ideal setup in which the cameras are perfect pinhole imagers and are aligned precisely parallel. The divergence from ideal causes problems in the quality of the stereo match since epipolar lines are not horizontal. In addition, if the camera calibration is unknown, one does not know how to interpret the stereo disparities in terms of range to an object. Camera calibration addresses these issues by creating a mathematical model of the camera.

SVS incorporates a simple automatic procedure for calibration, using a planar object that can be printed on a standard printer. The calibration is preformed by fitting a model to a number of images taken of a planar calibration object. The user presents the object to the stereo rig in five different (arbitrary) poses. The calibration procedure finds model features in the images, and then calculates a best-fit calibration for the rig. The procedure works for many different combinations of imagers, baselines, and lenses, including wide-angle lenses with severe distortion.

When is it necessary to perform calibration? In general, whenever an action changes the camera intrinsics (lens focal length and center axis) or extrinsics (the cameras move with respect to each other). Here are some actions that would necessitate re-calibration:

- Changing lenses
- Screwing the lenses in or out of their mount
- Zooming, if the lenses are zoom lenses
- Changing the baseline of the cameras
- Any movement or rotation of one camera independent of the other, e.g., severe vibration or shock can change the cameras' relative position
- Changing the lens focus with a focusing ring on the lens
- Changing the lens aperture (can change the rectification)

A rigid mount that keeps the cameras stable with respect to each other is a necessity for a stereo rig. For example, the STH-MDCS2 uses an extruded aluminum frame to stabilize the cameras.

NOTE: There is a Calibration Addendum manual that details the exact steps necessary to perform calibration, and includes troubleshooting information. Please consult that manual for more detailed information about the calibration procedure.

5 Sample Applications

In this section we describe some applications that use the SVS libraries. These applications are bundled with the SVS software.

Currently there are the following applications.

- SMALLV, SMALLVMAT
  These are GUI applications that enable the user to control the full functionality of the stereo device. Full source code and project makefiles are available in the *samples/* directory. SMALLVMAT has a simple interface to the MatLab engine, which allows images to be sent to MatLab arrays under program control. For controlling a stereo device using MatLab commands, try the CMAT application.
- SMALLVCAL
  SMALLVCAL is the only application that has access to the calibration and firmware libraries, which are compiled into this application. SMALLVCAL is distributed only as an application, without the source code.
- STFRAME
  A simple C++ program to acquire, process, and display stereo images from files or from a Videre Design stereo device. This is a good application to study if you are trying to develop your own application.
- LOADER
  A simple application, similar to STFRAME, that shows how to interface SVS to images that are stored in memory. Useful for those developing applications using non-Videre stereo cameras.
- PLANAR
  Extracts a plane from the 3D points using a Hough transform.
- CWRAP
  C function interface to SVS. Useful for MatLab and other applications programs that require a C interface.
- CMAT
  C functions to control a stereo device, acquire and process images, all from a MatLab environment.

5.1 PLANAR Application

The PLANAR application takes the output of the SVS stereo algorithm, and tries to find the dominant plane in the 3D information. It does this by using a 3D version of the Hough transform. Because the Hough transform can be expensive to compute, the search space of the transform can be restricted, i.e., only planes with certain parameters are found. The parameters can be changed by modifying the code, or online using the dialog window.

Source code for the PLANAR application is in the *samples/* directory, and the Visual C++ project is *samples/planar.dsp*. The executable is *bin/planar(.exe)*.

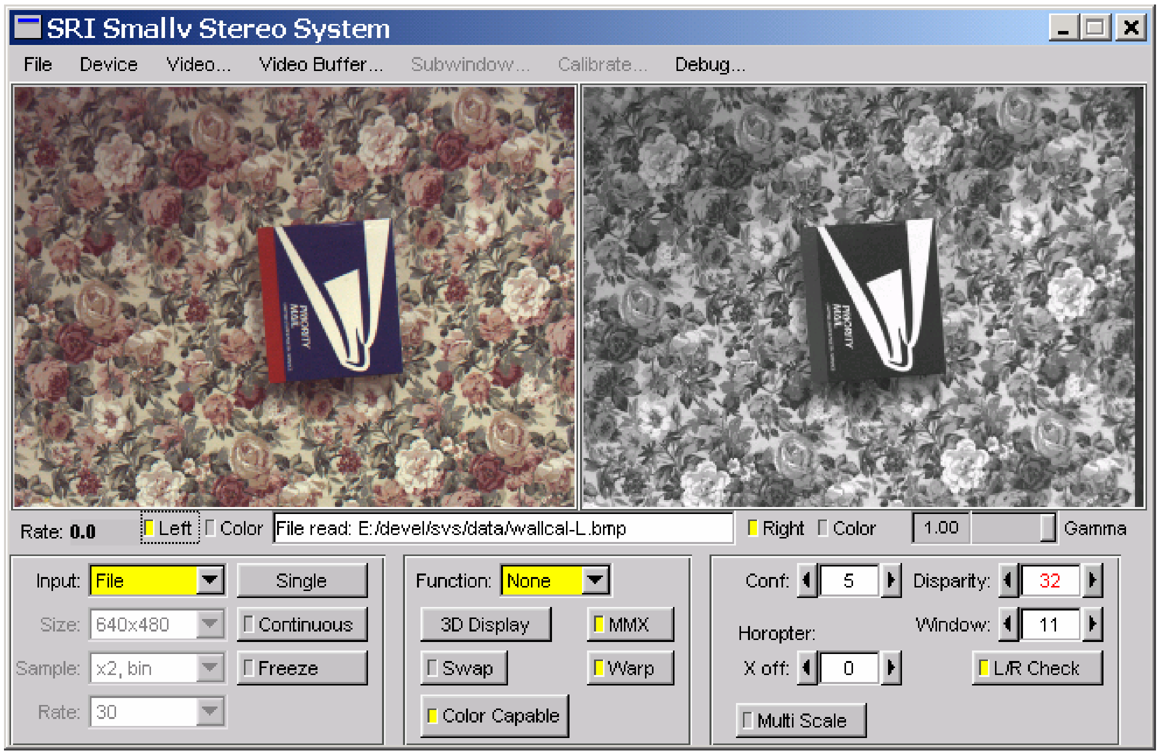
To run the sample application, start the *planar(.exe)* executable. There is no need to have an attached stereo device, since the application can be exercised using stored images. On starting the application, the familiar SVS window will appear. Choose *File->Load Images (BMP)*, and then pick the file *data/wallcal-L.bmp*. Two images will appear in the display windows as in Figure 5-1. This is a stereo pair of a textured wall, with a mailing box placed on the wall. The left image is in color. Both images have been rectified, using the calibration images *data/calN-L/R.bmp*.

Click on the *3D Display* button, which computes stereo information and displays the 3D points in a special 3D graphics window (Figure 5-2). You can rotate and zoom the image in the normal way (right-drag in the window for rotation around the center point of the 3D object).

To calculate the most dominant plane, click the *Display Plane* button. After a few seconds, a light blue plane will appear, about 20 mm behind the wall. The plane is offset so that it won't interfere with the 3D points of the plane. The parameters of the plane are indicated in the Debug window.

In searching for the plane, the Hough transform is limited to about 20 degrees on each side of a center Phi and Theta rotation. The Phi rotation is about the X axis, and the Theta rotation is about the Y axis. Also, the depth perpendicular to the plane is restricted to be from 0.5 to 10 m from the camera. You can change the Phi and Theta centerpoints using the dialog window controls.

Given the dominant plane, it is possible to isolate objects that are not on the plane, by eliminating points that are near the plane. Using the *Filter Range* control, all 3D points within the given range of the



Rate: 0.0
File read: E:/devel/svs/data/wallcal-L.bmp
1.00 Gamma
Input: File
Size: 640x480
Sample: x2, bin
Rate: 30
Function: None
Conf: 5
Disparity: 32
Window: 11
X off: 0

Figure 5-1 PLANAR application main window, with the *data/wallcal* dataset.

plane will not be displayed. For example, setting the range to 30 mm will eliminate the wall points on the plane.

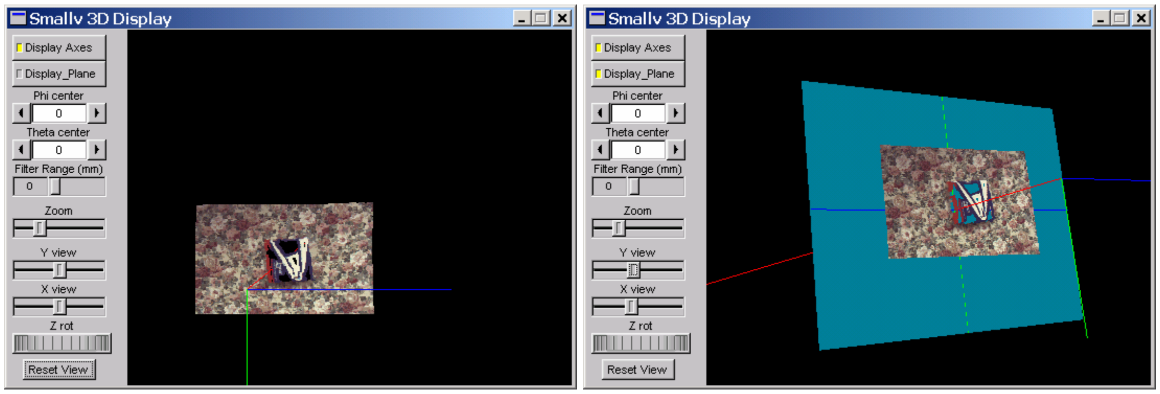


Figure 5-2 3D GUI display of the PLANAR application. In the left view, just the original 3D points are displayed. In the right view, the dominant plane has been found and is displayed 20 mm behind its true position.

5.2 CWRAP Library

The CWRAP application is a library that has a C-language interface to the SVS routines. It maintains a single `svsVideoImages` object to communicate with the SVS API. C functions to control the stereo device, and to process and access images, all refer to this object.

Source code for the CWRAP application is in the *samples/* directory, and the Visual C++ project is *samples/cwrap.dsp*. The library is *bin/cwrap.so(.dll)*

CWRAP is not so much an application as a library. It is an alternate API for SVS, which has less functionality, but can be called completely from C rather than C++. Since the source code is available, the use can modify the library to add more features.

Below is a list of the functions in the library, along with a brief description of their effect. These functions are defined in the header *samples/cwrap.h*. To call the functions, include this header with your program code. The libraries *cwrap.so(.lib)* and optionally *fltk.so* (*fltkdll.lib*) must be linked with your program.

The CWRAP functions include a handy display function, for showing video output of the stereo device. This display is especially useful in debugging. The CWRAP library is distributed with the display routines compiled in, and requires the FLTK libraries as well. If you don't want the display capability, you can compile the CWRAP library without FLTK support, by undefining the `USE_FLTK` symbol at compilation time.

Most of these functions have `void` returns, which some interfaces require. If an error occurs, the variable `c_svsError` is set to -1. If there is no error, it is set to 0.

Function	Action
`c_svsGetVideoObject()`	Initializes the SVS system and sets up a video object for acquiring images. This must be the first call to the API, and it should only be done once.
`c_svsReadParamFile(char *name)`	Reads a parameter file in the video object. Typically this will add calibration information.
`c_svsOpen()`	Opens the first available stereo device. This prepares the device to send video information.
`c_svsSetSize(int width, int height)`	Sets the resolution of the images. Should be called after the `svsOpen()` call, but before video streaming is started.
`c_svsSetRate(int rate)`	Sets the frame rate of the images. Should be called after the `svsOpen()` call, but before video streaming is started. Argument values are 30, 15, 7 (7.5 Hz), and 3 (3.75 Hz). If the 50Hz option has been selected in the camera firmware, then the respective frame rates are lower, but the arguments stay the same, e.g., the argument "30" gives a frame rate of 25 Hz.
`c_svsSetFrameDiv(int div)`	Sets the frame division (subwindow) of the images. Should be called after the `svsOpen()` call, but before video streaming is started. A value of 1 indicates full frame, a value of 2 is 1/2 size frame.
`c_svsSetColor(int left, int right)`	Turns color on (1) or off (0) for the left and right images. Should be called after the `svsOpen()` call. Can be called during video streaming.
`c_svsSetRect(int on)`	Turns rectification on (1) or off (0). Should be called after the `svsOpen()` call. Can be called during video streaming.
`c_svsStart()`	Starts video streaming. Should be called after the `svsOpen()` call.
`c_svsGetImage(int timeout)`	Gets the most recent stereo image, and saves it in an internal variable. Images can be returned by calling various `svsImageXXX` functions. Argument is how long to wait (in milliseconds) before timing out. Should be called after the `svsStart()` call has started video streaming.

`unsigned char *c_svsImageLeft();`
`unsigned char *c_svsImageRight();`
`unsigned long *c_svsImageLeftColor();`
`unsigned long *c_svsImageRightColor();`
`short *c_svsImageDisparity();`
	Returns various images after video streaming has started. The disparity image is automatically calculated if it is requrested.

`c_svsDisplay(int which)`	Starts a display of the video from the left and right cameras. If `which` is 1, then the disparity image is displayed instead of the right image. Display continues until all windows, including the Debug window, are closed.
`c_svsStop()`	Stops video streaming.

Table 5-1 API for the CWRAP C interface.

5.3 CMAT Interface

The CMAT interface is a library that can be loaded into a running MatLab session. It makes the SVS system available from MatLab, and allows the user to acquire and process images from a stereo device.

CMAT provides a basic interface capability. All of the source code and project files are included with the distribution, so the user can add more functionality if it is needed.

Source code for the CMAT interface is in the *samples/* directory (especially *cmat.c* and *cwrap.cpp*), and the Visual C++ project is *samples/cmat.dsp*. The library is *bin/cmat.so(.dll)*

NOTE: you must have MatLab available on your machine to compile this project. Check the project settings to see if the correct path to the MatLab include directory is set up.

5.3.1 Starting and Running the Interface

The interface works with MatLab Version 6.5 Release 13 onward, including MatLab 7.

To run the interface, make sure that the stereo device is plugged in and installed correctly, and that the SVS distribution is correctly installed (version 3.2d or greater). The performance of the stereo device can be verified by running the `smallv(.exe)` application.

MatLab must know where to find the SVS libraries. This can be done in two ways.

1. Add the SVS *bin/* directory to the MatLab search path, and also to the system execution path. For MSW, this is the PATH variable accessible from the Control Panel->System->Advanced->Environment Variables dialog. For Linux, it is the LD_LIBRARY_PATH environment variable.
2. In MatLab, connect to the *bin/* directory, using the `cd` command.

(2) is easier, but requires that MatLab always be connected to the *bin/* directory. (1) will work no matter where MatLab is connected.

In MatLab, the interface is invoked with the `cmat` function. The first argument to `cmat` is a string, specifying the interface action. Other arguments are optional, and depend on the interface action. Here is a typical sequence for getting images.

```
cmat('init')    % starts up the interface, only called once
cmat('open')    % opens the stereo device
cmat('start')   % starts video streaming from the device
cmat('display')   % display the images to verify them;
                  % this function returns only when all
                  % display windows are closed

left = zeros(240,320);  % an array to hold an image
right =  zeros(240,320);  % an array to hold an image
cmat('getImage',left,right)    % return the current left and right
                               % stereo images
image(left);    % displays the image from MatLab
colormap(gray)  % it helps to set the colormap for monochrome images

disparity = zeros(240,320);    % an array to hold an image
cmat('getDisparity',disparity)    % compute and return disparity
                                  % image

cmat('stop')    % stops video streaming from the device
cmat('close')   % closes the stereo device
```

The initialization call must be done just once, before any other calls to `cmat`. After this, the device can be opened and video streaming started. Streaming may be started and stopped as often as desired. When video is no longer desired, the device should be closed. If MatLab is exited without closing the device, a segmentation fault will result.

Errors can occur with all these functions. For example, if there is no attached stereo device, the `Open` call will fail. On error, the `cmat` function returns -1. On success, it return 0.

Images can be returned to MatLab arrays. The format of the arrays must match the format of the images. No bounds checking is performed, so a mismatched array size can cause a segmentation fault. MatLab will catch the fault and continue, so it is not a fatal error.

For grayscale images, the image arrays in MatLab are two-dimensional arrays of doubles, and the pixel values range from 0 to 255. Note that the arrays are set up with the number of *rows* as the first dimension. For color images, the MatLab arrays are three-dimensional double arrays, with the last dimension being 3, e.g., 240x320x3. The pixel values range from 0 to 1, in accord with MatLab conventions for images.

The `getImage` function puts the current stereo left and right images into two arrays. Note that this function changes the elements of arrays that are passed in as arguments, it does not create new arrays. It is more efficient to change elements, since no array creation occurs. If the array is three-dimensional, `getImage` puts a color image; if it is two-dimensional, a grayscale image. It is an error to ask for a color image if the `setColor` command has not been used to turn on color images. Grayscale images are always available.

The disparity image can be calculated and returned after calling `getImage`, using the `getDisparity` function. Its argument is a MatLab two-dimensional array, similar to the grayscale image arrays. The values of the disparity pixels are from 0 to 16x the maximum disparity. The two special values -1 and -2 mean that the pixel is filtered by the texture filter or the left-right filter, respectively.

5.3.2 cmat() Function Call Reference

Table 5-1 is a summary of the available function calls for `cmat`. The source code and project for `cmat` is in the samples/ directory, so it is possible to add new functions, wrapping methods from the C++ API, and making them available.

cmat Function	Arguments	Action
`init`		Initializes the SVS system. This must be the first call to `cmat`, and it should only be done once.
`readParamFile`	File name: string	Reads a parameter file in the video object. Typically this will add calibration information.
`open`		Opens the first available stereo device. This prepares the device to send video information.
`setSize`	width, height	Sets the resolution of the images. Should be called after the `open` call, but before video streaming is started.
`setRate`	rate	Sets the frame rate of the images. Should be called after the `open` call, but before video streaming is started. Values are 30, 15, 7, and 3. These will give 30, 15, 7.5, and 3.75 Hz for 60 Hz devices, and 25, 12.5, 6.25, and 3.125 Hz for 50 Hz devices.
`setFrameDiv`	div	Sets the frame division of the images. Should be called after the `open` call, but before video streaming is started. A value of 1 means the full frame of the imager, a value of 2 means 1/2 frame, i.e., the center subwindow of the image.
`setColor`	left, right	Turns color on (1) or off (0) for the left and right images. Should be called after the `open` call. Can be called during video streaming.
`setRect`	on	Turns rectification on (1) or off (0). Should be called after the `open` call. Can be called during video streaming.
`start`		Starts video streaming. Should be called after the `Open` call.
`getImage`	[left], [right]	Gets the most recent stereo image, and optionally saves it in the arrays `left` and `right`. The arrays should be the same size as the image, with the first dimension the number of rows, e.g., 240 x 320. For color images, use 3-dimensional arrays, e.g., 240 x 320 x 3. To just return the left image, use a single argument. To return no images, use no arguments. Should be called after the `start` call has started video streaming.
`getDisparity`	disparity	Calculates the disparity and returns it in the `disparity` argument. This array should be the same size as the image, with the first dimension the number of rows, e.g., 240 x 320. Should be called after the `getImage` call has returned an image.
`display`	[which]	Starts a display of the video from the left and right cameras. If `which` is not 0, then the disparity image is displayed instead of the right image. Display continues until all windows, including the Debug window, are closed.
`stop`		Stops video streaming.

Table 5-2 Actions for the cmat() function in MatLab.

6 API Reference – C++ Language

In SVS 3.x, the standard programming interface to the SVS libraries is in C++. To add stereo processing to your own programs, you call functions in the Stereo Engine library. These functions are available in `svs.dll` (Windows 98SE/2000/NT/XP) or `libsvs.so` (Unix systems). The header file is `src/svsclass.h`.

Source code samples for the C++ API are in the directory `samples/`. A simple example of the use of these functions is in the sample program `samples/stframe.c`pp. Please review the examples in the `samples/` directory for more explicit information about how to set up projects and makefiles.

Supported versions of C++:

MS Windows	MSVC++ 6.0, Service Pack 5 (SP5 *must* be installed) MSVC++ Net
Linux	GCC 3.3 (preferred) GCC 2.95 (deprecated, supported only to SVS 3.1g)

6.1 Threading and Multiple Stereo Devices

6.1.1 Threading Issues

The SVS core library functions (`svs.dll`, `libsvs.so`) are thread-safe: they can be used in any thread in a process. Of course, the user is responsible for not overlapping calls in different threads, e.g., starting up two competing disparity calculations using the same object in different threads.

The MEGA-D and Dual-DCAM acquisition libraries are also thread-safe, in general. However, there are some known quirks under MS Windows. The most obvious of these is the `Open()` call for the MEGA-D. This call must be made in the main thread. Subsequent accesses using `GetImage()` can be made in any thread.

Graphic window output is handled by the FLTK cross-platform windowing system. This system is not, in general, thread safe. Calls to the FLTK functions must all be made from the same thread; multiple threads are allowed in the application program, as long as all FLTK calls come from the same thread. There is a nascent thread locking mechanism in FLTK, but it is not yet incorporated into SVS.

6.1.2 Multiple Devices

Multiple stereo devices (STH-DCSG, STH-STOC, STH-MDCS2/3) can be accessed simultaneously from a single process, or from multiple processes. Only one process may access a given device, using an `Open()` call. Once this call is made, subsequent calls to `Open()` from other processes will fail until the device is released.

There is more information about multiple devices and the requisite bandwidth in the User Manual for a particular device. There are several restrictions on multiple device usage.

1. When using multiple devices on the same IEEE 1394 bus, it may be necessary to lower the frame rate or frame size before starting streaming video. For example, the STH-MDCS2/3 can use most of the bus bandwidth at 640x480, 30 Hz. Setting the frame rate to 15 Hz, or the frame size to 320x240, will allow more devices to be accessed.

2. Most IEEE1394 host cards have a limitation of 4 receive contexts, which means that only 4 video streams can be open at one time. For STH-XXX-VAR devices, this means only two such devices can be open simultaneously on the IEEE1394 card.

6.2 C++ Classes

There are three main classes for SVS: classes that encapsulate stereo images, classes that produce the images from video or file sources, and classes that operate on stereo images to create disparity and 3D images. These classes are displayed in Figure 6-1. The header file is `src/svsclass.h`.

The basic idea is to have one class (`svsStereoImage`) for stereo images and the resultant disparity images, which performs all necessary storage allocation and insulates the user from having to worry about these issues. Stereo image objects are produced from video sources, stored image files, or memory buffers by the `svsAcquireImages` classes, which are also responsible for rectifying the images according to parameters produced by the calibration routines. Disparity images and 3D point clouds are produced by the stereo processing class `svsStereoProcess` acting on stereo image object, with the results stored back in the stereo image object. Finally, display classes allow for easy display of the images within a GUI.

Figure 6-2 shows a simple example of using the classes to produce and display stereo disparity results. The full program example is in `samples/stframe.cpp`. The basic operations are:

1. Make a video source object and open it. Which video source is used depends on which framegrabber interface file has been loaded: see Section 2.1.2.
2. Make a stereo processing object for producing disparity results from a stereo image.
3. Make some display window objects for displaying images and disparity results.
4. Open the video source.
5. Set the frame size and any other video parameters you wish, and read in rectification parameters from a file.
6. Start the video acquisition.
7. Loop:
    a. Get the next stereo image.
    b. Calculate disparity results.
    c. Display the results.

Image source classes	**Stereo Image and Parameter classes**	**Stereo Processing classes**	**Display classes**
`svsAcquireImages`	`svsStereoImage`	`svsStereoProcess`	`svsWindow`
`svsVideoImages`			`svsGLWindow`
`svsFileImages`	`svsImageParams`		`svsDebugWin`
	`svsRectParams`		
	`svsDistParams`		

Figure 6-1 SVS C++ Classes

```
// Make a video source object, using the loaded framegrabber interface
svsVideoImages *videoObject = getVideoObject();

// Make a stereo processing object
svsStereoProcess *processObject = new svsStereoProcess();

// Open the video source
bool ret = videoObject->Open();
if (!ret) { …error code… }

// Read in rectification parameters
videoObject->ReadParams("../data/megad-75.ini");

// Set up display windows
int width = 320, height = 240;
svsWindow *win1 = new svsWindow(width,height);
svsWindow *win2 = new svsWindow(width,height);
win1->show();
win2->show();

// Start up the video stream
videoObject->SetSize(width, height);
ret = videoObject->Start();
if (!ret) { … error code … }

// Acquisition loop
while (1)
{
   // Get next image
   svsStereoImage *imageObject = videoObject->GetImage(400);
   if (!imageObject)  { … error code …}
   // calculate disparity image
   processObject->CalcStereo(imageObject);
   // display left image and disparity image
   win1->DrawImage(imageObject, svsLEFT);
   win2->DrawImage(imageObject, svsDISPARITY);
}
```

Figure 6-2 A simple program for video acquisition and stereo processing. The full program is in `samples/stframe.cpp`.

6.3 Parameter Classes

`svsImageParams`	Image frame size and subwindow parameters
`svsRectParams`	Image rectification parameters
`svsDispParams`	Image stereo processing (disparity) parameters

Parameter classes contain information about the format or processing characteristics of stereo image objects. Each stereo image object contains an instance of each of the above classes. Application programs can read these parameters to check on the state of processing or the size of images, and can set some of the parameters, either directly or through class member functions.

6.3.1 Class svsImageParams

Frame size and subwindow parameters for stereo images. In general, the only way these parameters should be changed is through member functions of the appropriate objects, e.g., using `SetSize` in the `svsVideoImages` class.

6.3.2 Class svsRectParams

Rectification parameters for stereo images. They are used internally by the rectification functions. Application programs should not change these parameters, and will have few reasons to look at the parameter values. Rectification parameters are generated initially by the calibration procedure, then written to and read from parameter files, or internal storage in the stereo device.

6.3.3 Class svsDispParams

Disparity parameters control the operation of stereo processing, by specifying the number of disparities, whether left/right filtering is on, and so on. Most of these parameters can be modified by application programs.

6.4 Stereo Image Class

`svsStereoImage`	Stereo image class

The stereo image class encapsulates information and data for a single stereo image pair, along with any of its processed results, e.g., disparity image or 3D point cloud.

Stereo image objects are usually produced by one of the image acquisition classes (`svsVideoImages` or `svsFileImages`), then processed further by an `svsStereoProcess` object.

An `svsStereoImage` object holds information about its own state. For example, there are Boolean flags to tell if there is a valid set of stereo images, whether they are rectified or not, if a valid disparity image has been computed, and so on.

The `svsStereoImage` class handles all necessary allocation of buffer space for images. User programs can access the image buffers, but should be careful not to de-allocate them or destroy them.

6.4.1 Constructor and Destructor

```
svsStereoImage();
~svsStereoImage();
```

Constructor and destructor for the class. The constructor initializes most image parameters to default values, and sets all image data to NULL.

```
char error[256];
```

If a member function fails (e.g., if `ReadFromFile` returns `false`), then `error` will usually contain an error message that can be printed or displayed.

6.4.2 Stereo Images and Parameters

```
bool haveImages;         // true if we have good stereo images
bool haveColor;          // true if left image color array present
bool haveColorRight;     // true if right image color array present
svsImageParams ip;       // image format, particular to each object
unsigned char *Left();   // left image array
unsigned char *Right();  // right image array
unsigned long *Color();  // left-color image array
unsigned long *ColorRight();  // right-color image array
```

These members describe the stereo images present in the object. If stereo images are present, `haveImages` is `true`. The stereo images are always monochrome images, 8 bits per pixel. Additionally, there may be a color image, corresponding to the left image, and a color image for the right imager, if requested. Color images are in RGBX format (32 bits per pixel, first byte red, second green, third blue, and fourth undefined). If the left color image is present, `haveColor` is `true`. The color image isn't used by the stereo algorithms, but can be used in post-processing, for example, in assigning color values to 3D points for display in an OpenGL window. Similarly, if the right color image is present, `haveColorRight` is true. The color images may be input independently of each other.

Frame size parameters for the images are stored in the variable `ip`. The parameters should be considered read-only, with one exception: just before calling the `SetImage` function.

The `Left`, `Right`, and `Color` functions return pointers to the image arrays. User programs should not delete this array, since it is managed by the stereo object.

6.4.3 Rectification Information

```
bool isRectified; // have we done the rectification already?
bool haveRect;    // true if the rectification params exist
svsRectParams rp; // rectification params, if they exist
```

The images contained in a stereo image object (left, right and left-color) can be *rectified*, that is, corrected for intra-image (lens distortions) and inter-image (spatial offset) imperfections. If the images are rectified, then the variable `isRectified` will be `true`.

Rectification takes place in the `svsAcquireImage` classes, which can produce rectified images using the rectification parameters. The rectification parameters can be carried along with the stereo image object, where they are useful in further processing, for example, in converting disparity images into a 3D point cloud.

If rectification parameters are present, the `haveRect` variable is true. The rectification parameters themselves are in the `rp` variable.

6.4.4 Disparity Image

```
bool haveDisparity;      // have we calculated the disparity yet?
svsDisparityParams dp;   // disparity image parameters
short *Disparity();      // returns the disparity image
```

The disparity image is computed from the stereo image pair by an `svsStereoProcess` object. It is an array of short integers (signed, 16 bits) in the same frame size as the input stereo images. The image size can be found in the `ip` variable. It is registered with the left stereo image, so that a disparity pixel at X,Y of the disparity image corresponds to the X,Y pixel of the left input image. Values of –1 and –2 indicate no disparity information is present: -1 is for low-texture areas, and –2 is for disparities that fail the left/right check.

If the disparity image has been calculated and is present, then `haveDisparity` is `true`. The parameters used to compute the disparity image (number of disparities, horopter offset, and so on) are in the parameter variable `dp`.

The disparity image can be retrieved using the `Disparity()` function. This function returns a pointer to the disparity array, so it is very efficient. User programs should not delete this array, since it is managed by the stereo object.

6.4.5 Confidence Image

```
bool doConfidence;       // should we return a confidence image?
short *Confidence();     // returns the confidence image
```

For some research applications, it is desirable to know the *confidence* of the stereo result at each pixel. The confidence measure is computed from the visual texture around each pixel. Normally, the confidence is thresholded using the confidence parameter in the stereo parameters `dp`, and the confidence image is not returned directly.

The confidence image is computed from the stereo image pair by an `svsStereoProcess` object, at the same time that the disparity image is calculated. The boolean variable `doConfidence` must be set to true to produce the confidence image. The confidence image is returned with the `Confidence()` function; the buffer is managed internally, so user programs should not delete this buffer. It is an array of short integers (signed, 16 bits) in the same frame size as the input stereo images. The confidence measure is always positive. Higher values indicate more texture. The image size can be found in the `ip` variable.

It is registered with the left stereo image, so that a confidence pixel at X,Y of the confidence image corresponds to the X,Y pixel of the left input image.

There is currently no direct function for outputting the confidence image to a file.

6.4.6 3D Point Array

```
bool have3D;              // do we have 3D information?
int numPoints;            // number of points actually found
svs3Dpoint *pts3D;        // 3D point array
typedef struct {int A; float X,Y,X} svs3Dpoint; // 3D point structure
```

The 3D point array is an array of 3D point structures that correspond to each pixel in the left input image. The array has the same size (width and height) as the input stereo images. The 3D point array is computed from the disparity image using the external camera calibration parameters stored in `rp`. An `svsStereoProcess` object must be used to compute it.

Each point is represented by a coordinate (X,Y,Z) in a frame centered on the left camera focal point. The Z dimension is distance from the point perpendicular to the camera plane, and is always positive for valid disparity values. The X axis is horizontal and the positive direction is to the right of the center of the image; the Y axis is vertical and the positive direction is down relative to the center of the image (a right-handed coordinate system). The A value is an integer indicating the status of the point:

A = 0	Out of bounds of disparity image
A < 0	Filtered point (confidence or uniqueness)
A > 0	Valid point, A is the disparity value

NOTE: The units of X,Y,Z are in *meters*.

If the 3D array is present, then `have3D` is true. The actual number of 3D points present in the arrays is given by `numPoints`.

All structures in the 3D array are aligned on 16-byte boundaries, for efficient processing by MMX/SSE instructions.

6.4.7 File I/O

```
bool SaveToFile(char *basename);    // saves images and params to files
bool ReadFromFile(char *basename);  // gets images and params from files
bool ReadParams(char *name);        // reads just params from file
bool SaveParams(char *name);        // save just params to file
```

Images and parameters in a stereo object can be saved to a set of files (`SaveToFile`), and read back in from these files (`ReadFromFile`). The basename is used to create a file set. For example, if the basename is `TESTIMAGE`, then the files set is:

```
TESTIMAGE-L.bmp   // left image, if present
TESTIMAGE-R.bmp   // right image, if present
TESTIMAGE-C.bmp   // left color image, if present
TESTIMAGE.ini     // parameter file
```

Just the parameters can be read from and written to a parameter file, using `ReadParams` and `SaveParams`. These functions take the explicit name of the file, e.g., `TESTIMAGE.ini`. Parameter files have the extension `.ini`, by convention.

For storing and reading parameters from a stereo device, see Section 6.6.6.

6.4.8 Copying Functions

```
void SetImages(unsigned char *left, // Sets images from user data
               unsigned char *right,
               unsigned char *color,
               unsigned char *color_right,
               svsImageParams *ip = NULL,
               svsRectParams *rp = NULL,
               bool rect = false,
               bool copy = false);
void CopyFrom(svsStereoImage *si);  // copies contents of si to object
```

These functions are not used in typical applications, since they manipulate the stereo object buffers. User programs can insert buffer data into a stereo image object using the above functions. These functions are generally useful for making memory buffers of sequences of images, rather than for initial input of images. For example, if you want to input images from your own stereo rig, with images stored in memory, it is recommended to use the `svsStoredImages` acquisition class, which will produce `svsStereoImage` samples. Acquisition classes can perform rectification operations, while the `svsStereoImage` class cannot.

Optional parameter information can be supplied with the images, via the parameter arguments; otherwise, the parameters already present in the object remain the same. If any of the image arguments is NULL, then no image data is inserted for that image. If the `copy` argument is `true`, then the buffer contents are copied onto the stereo image object's own buffers. If not, then the input buffers are used internally.

6.5 Acquisition Classes

`svsAcquireImages`	Base class for all acquisition
`svsVideoImages`	Acquire from a video source
`svsFileImages`	Acquire from a file source
`svsStoredImages`	Acquire from a memory source

Acquisition classes are used to get stereo image data from video or file sources, and put it into `svsStereoImage` structures for further processing. During acquisition, images can be *rectified*, that is, put into a standard form with distortions removed. Rectification takes place automatically if the calibration parameters have been loaded into the acquisition class.

The two subclasses acquire images from different sources. `svsVideoImages` uses the capture functions in the loaded `svsgrab` DLL to acquire images from a video device such as the MEGA-D stereo head. `svsFileImages` acquires images from BMP files stored on disk.

6.5.1 Constructor and Destructor

```
svsAcquireImages();
virtual ~svsAcquireImages();
```

These functions are usually not called by themselves, but are implicitly called by the constructors for the subclasses `svsVideoImages` and `svsFileImages`.

6.5.2 Rectification

```
bool HaveRect();
bool SetRect(bool on);
bool GetRect();
bool IsRect();
bool ReadParams(char *name);
bool SaveParams(char *name);
```

These functions control the rectification of acquired images. `HaveRect()` is true when rectification parameters are present; the normal way to input them is to read them from a file, with `ReadParams()`. The argument is a file name, usually with the extension `.ini`. If the acquisition object has rectification parameters, they can be saved to a file using `SaveParams()`.

Rectification of acquired images is performed automatically if `HaveRect()` is true, and rectification processing has been turned on with `SetRect()`. Calling `ReadParams()` will also turn `on SetRect()`. The state of rectification processing can be queried with `GetRect()`.

If the current image held by the acquisition object is rectified, the `IsRect()` function will return true.

```
bool RectImagePoint(double *u, double *v, double x, double y, int which)
bool UnrectImagePoint(double *x, double *y, double u, double v,
                      int which)
```

These utility functions return the point in the rectified image (u,v) that corresponds to the input image point (x,y), or vice versa. The argument `which` is either `svsLEFT` or `svsRIGHT`. If there is no image or rectification, the function returns false.

6.5.3 Projection

```
bool Project3D(double *x, double *y, double X, double Y, double Z,
               int which)
```

This function can be used to project a 3D point to its corresponding point in the rectified image. The argument `which` is either `svsLEFT` or `svsRIGHT`. If there is no image or rectification, the function returns false.

6.5.4 Controlling the Image Stream

```
bool CheckParams()
bool Start()
bool Stop()
svsStereoImage *GetImage(int ms)
```

An acquisition object acquires stereo images and returns them when requested. These functions control the image streaming process.

`CheckParams()` determines if the current acquisition parameters are consistent, and returns true if so. This function is used in video acquisition, to determine if the video device supports the modes that have been set. If the device is not opened, `CheckParams()` returns `false`.

`Start()` starts the acquisition streaming process. At this point, images are streamed into the object, and can be retrieved by calling `GetImage()`. `GetImage()` will up to ms milliseconds for a new image before it returns; if no image is available in this time, it returns NULL. If an image is available, it returns an `svsStereoImage` object containing the image, rectified if rectification is turned on. The `svsStereoImage` object is controlled by the acquisition object, and the user program should not delete it. The contents of the `svsStereoImage` object are valid until the next call to `GetImage()`.

`Start()` returns false if the acquisition process cannot be started. `Stop()` will stop acquisition.

NOTE: `GetImage()` returns a pointer to an `svsStereoImage` object. This object can be manipulated by the user program until the next call to `GetImage()`, at which point its data can change. The user program should not delete the `svsStereoImage` object, or any of its buffers. All object and buffer allocation is handled by the acquisition system. If the application needs to keep information around across calls to `GetImage()`, then the `svsStereoImage` object or its buffers can be copied.

6.5.5 Error String

```
char *Error()
```

Call this function to get a string describing the latest error on the acquisition object. For example, if video streaming could not be started, `Error()` will contain a description of the problem.

6.6 Video Acquisition

The video acquisition classes are subclasses of `svsAcquireImages`. The general class is `svsVideoImages`, which is referenced by user programs. This class adds parameters and functions that are particular to controlling a video device, e.g., frame size, color mode, exposure, and so on.

Particular types of framegrabbers and stereo heads have their own subclasses of svsVideoImages. In general, the user programs won't be aware of these subclasses, instead treating it as a general svsVideoImage object.

A particular framegrabber interface class is accessed by copying a DLL file to `svsgrab.dll`. For example, for the STH-MDCS stereo head and IEEE 1394 interface, copy `svsdcs.dll` to `svsgrab.dll` (see Section 2.1). Every such interface file defines a subclass of `svsVideoImages` that connects to a particular type of framegrabber and its associated stereo head.

To access the `svsVideoImages` object for the interface file, the special function `svsGetVideoObject()` will return an appropriate object.

6.6.1 Video Object

`svsVideoImages *svsGetVideoObject()`

Returns a video acquisition object suitable for streaming video from a stereo device. The particular video object that is accessed depends on the video interface file that has been loaded; see Section 2.1, This function creates a new video object on each call, so several devices can be accessed simultaneously, if the hardware supports it.

6.6.2 Device Enumeration

`int Enumerate()`
`char **DeviceIDs()`

Several STH-MDCS, MEGA-D and Dual DCAM stereo devices can be multiplexed on the same computer. Any such device connected to an IEEE 1394 card is available to SVS. The available devices are enumerated by the `Enumerate()` function, which returns the number of devices found, and sets up an array of strings that have the identifiers of the devices. The ID array is returned by the `DeviceIDs()` function. The user application should not destroy or write into the array, since it is managed by the video object. The strings are unique to the device, even when plugged into different machines.

The `Enumerate()` function rescans the bus each time it is called, so whenever devices are plugged or unplugged, it can be called to determine which devices are available. `DeviceIDs()` should only be called after at least one `Enumerate()`.

`Enumerate()` is called automatically when the video object is first returned from `svsGetVideoObject()`.

STH-MDCS, MEGA-D and Dual DCAM devicees can be intermixed on the same bus. However, SVS loads only a single video driver, so any particular SVS application will see only the STH-MDCS, MEGA-Ds or the Dual DCAMs.

6.6.3 Opening and Closing

`bool Open(char *name = NULL)`
`bool Open(int devnum)`
`bool Close()`

Before images can be input from a stereo device, the device must be opened. The `Open()` call opens the device, returning true if the device is available. An optional name can be given to distinguish among

Device Type	Identifying String

STH-MDCS2/3(-C)
STH-DCSG(-C)
STH-STOC(-C)
	SSSS or SSSSS
	last 4 or 5 digits of serial number

STH-MDCS2/3-VAR(-C) STH-DCSG-VAR(-C)	SSSS or SSSSS last 4 or 5 digits of serial number of either left or right camera
MEGA-D	Exact string as returned by `DeviceIDs()`
Dual-DCAM	#LLLL:#RRRR The "#" and ":" are mandatory. The LLLL string is any unique substring of the left camera name, the RRRR string is any unique substring of the right camera name. These strings can be any size.

Table 6-1 Identifier strings in Open() for different device types.

several existing devices. The naming conventions for devices depend on the type of device; typically it is a serial number or other identifier; see Table 6-1. These identifiers are set up by the `Enumerate()` call, and reside in the `DeviceIDs()` structure. Alternatively, a number can be used, giving the device in the order returned by the `DeviceIDs()` function, i.e., 1is the first device, 2 is the second, and so on. A value of 0 indicates any available device.

Upon opening, the device characteristics are set to default values. To set values from a parameter file, use the `ReadParams()` function.

A stereo device is closed and released by the `Close()` call.

6.6.4 Image Framing Parameters

```
bool SetSize(int w, int h)
bool SetSample(int decimation, int binning)
bool SetFrameDiv(int div)
bool SetRate(int rate)
bool SetOffset(int ix, int iy, int verge)
bool SetColor(bool on, bool onr = false)
bool CheckParams()
int  color_alg     [COLOR_ALG_FAST, COLOR_ALG_BEST]
```

See Sections 2.1.4, 2.1.5, and 2.1.6 for more information on frame sizes and sampling modes.

These functions control the frame size and sampling mode of the acquired image. `SetSize(w,h)` sets the width and height of the image returned by the stereo device. In most cases, this is the full frame of the image. For example, most analog framegrabbers perform hardware scaling, so that almost any size image can be requested, and the hardware scales the video information from the imager to fit that size. In most analog framegrabbers, the sampling parameters (decimation and binning) are not used, and a full-frame image is always returned, at a size given by the `SetSize()` function.

The digital stereo devices allow the user to specify a desired frame rate. Typically the device defaults to the fastest frame rate allowed, and the application can choose a different one to minimize bus traffic. The MEGA-D differs from the STH-MDCS and Dual-DCAM in its interpretation of the frame rate parameter; see Table 6-2.

Some stereo devices, such as the MEGA-D, allow the user to specify a *subwindow* within the image frame. The subwindow is given by a combination of sampling mode and window size. The sampling mode can be specified by `SetSample()`, which sets binning and decimation for the imager. The MEGA-D supports sub-sampling the image at every 1, 2 or 4 pixels; it also supports binning (averaging) of 1 or 2 (a 2x2 square of pixels is averaged). For example, with binning =2 and decimation = 2, the full frame size is 320 x 240 pixels. Using `SetSize()`, a smaller subwindow can be returned. The offset of the subwindow within the full frame comes from the `SetOffset()` function, which specifies the upper left corner of the subwindow, as well as a *vergence* between the left and right images.

An alternative and simpler way to set sampling modes, for most devices, is with the `SetFrameDiv()` function. A value of "1" means that the full frame will be returned. A value of "1/2" means that a half-size image (half the width and half the height, so ¼ of the pixels) will be returned, and a value of "1/4" means that a quarter-width and quarter-height image is returned. Using frame division leaves some ambiguity about how to achieve the results – for example, 640x480 at full frame size for the STH-MDCS devices can be done either by decimation or binning, as noted. The system will pick an appropriate mode. For more control over the mode, it is always possible to specify the sampling explicitly, using `SetSample()`.

`SetColor()` turns color on the left image on or off. Additionally, some applications require color from the right imager also, and setting the second argument to true will return a color image for the right imager. Generally, returning color requires more bus bandwidth and processing, so use color only if necessary.

The type of color algorithm used for color reconstruction in MDCS and MDCS2 color devices can be selected with the `color_alg` variable. This variable has effect only in non-binning modes. See Section 2.1.13 for information on choosing a color algorithm.

The video frame parameters can be set independently, and not all combinations of values are legal. The `CheckParams()` function returns `true` if the current parameters are consistent.

None of the frame or sampling mode parameters can be changed while images are being acquired, except for the offset parameters. These can be changed at any time, to pan and tilt the subwindow during acquisition.

6.6.5 Image Quality Parameters

`bool SetExposure(bool auto, int exposure, int gain)`

Device	**Rate Parameter**	**Frame Rate**
MEGA-D	0, 1	Normal
	2	Normal / 2
	3	Normal / 3
	4	Normal / 4
Dual-DCAM STH-MDCS2/3 STH-DCSG STH-STOC	30	30 Hz
	15	15 Hz
	7	7.5 Hz
	3	3.75 Hz

Table 6-2 Frame rates as a function of the SetRate() parameter. MEGA-D frame rates are determined from the base rate by clock division. Dual-DCAM and STH-MDCS rates are determined directly as frames/second.

```
bool SetExposure(int exposure, int gain, bool auto_exp, bool auto_gain)
bool SetBalance(bool auto, int red, int blue)
bool SetBrightness(bool auto, int brightness)
bool SetLevel(int brightness, int contrast)
```

See Section 2.1.9 and 2.1.12 for more information about video quality parameters.

These functions set various video controls for the quality of the image, including color information, exposure and gain, brightness and contrast. Not all stereo devices support all of the various video modes described by these parameters.

In general, parameters are normalized to be integers in the range [0,100]. See individual device manuals for the interpretation of the parameters for the device.

`SetExposure()` has two forms. For manual control, either form can be used, with the auto parameters set to *false*. For auto exposure and gain control, the first form sets auto mode for both gain and exposure. The second form allows more control over whether exposure or gain is auto controlled. In auto mode, the manual parameters are ignored.

`SetBalance()` sets the color balance for the device. Manual parameters for red and blue differential gains are between –40 and 40. If auto is chosen and available for the device, the manual parameters are ignored.

`SetBrightness()` sets the brightness control for digital devices. Brightness can be set between 0 and 100, with 30 being a typical value. If auto is chosen and available for the device, the manual parameters are ignored.

`SetLevel()` sets the brightness and contrast for analog framegrabbers. It is no longer used.

These functions can be called during video streaming, and their effect is immediate.

6.6.6 Stereo Device Parameter Upload and Download

```
bool SaveParams();       // save params to currently open device
bool ReadParams();       // read params from currently open device
```

Parameters in a video object can be saved to a stereo device (`SaveParams`), and read back in from the device (`ReadParams`). The device must have a FW firmware version of 2.1 or greater. All the current parameters of the device are saved.

The device must already be opened with a call to `Open()` in order to save parameters. Any stored parameters are automatically loaded when the device is opened.

Parameters may be saved to a device using the `File->Upload to Device` menu item of `smallv`. They may be downloaded using the `File->Download from Device` menu item. There are also utility applications for loading and storing parameter files to stereo devices; see the stereo device manual for your device.

6.6.7 Disparity Processing Parameters

```
IMPORT void SetNDisp(int n); // set number of disparities, 8 to 128
IMPORT void SetLR(bool on);   // set LR check, legacy
IMPORT void SetThresh(int n); // set texture filter threshold
IMPORT void SetUnique(int n); // set uniqueness filter threshold
IMPORT void SetCorrsize(int n); // set correlation window size, 7 to 21
IMPORT bool SetHoropter(int n); // set horopter (X offset), in pixels
IMPORT void SetSpeckleSize(int n); // set min disparity region size
IMPORT void SetSpeckleDiff(int n); // set disparity region neighbor diff
```

See Sections 2.4.4, 2.4.5, 2.4.7, 2.5.1, and 2.5.2 for more information on disparities and their parameters.

These functions control disparity processing. `SetNDisp` sets the number of disparities. `SetThresh` and `SetUnique` set the thresholds for post-filtering. `SetCorrsize` sets the correlation window size. `SetHoropter` sets the X offset of the right image relative to the left, for close-in processing. Positive values verge the two images, negative values de-verge them. `SetSpeckleSize` and `SetSpeckleDiff` control speckle elimination in postprocessing.

6.6.8 STOC Processing

```
bool SetProcMode(proc_mode_type mode);     // set STOC processing mode
enum proc_mode_type
  {
    PROC_MODE_OFF = 0,
    PROC_MODE_NONE,
    PROC_MODE_TEST,
    PROC_MODE_RECTIFIED,
    PROC_MODE_DISPARITY,
    PROC_MODE_DISPARITY_RAW
  };
```

The Stereo-on-a-Chip device has on-chip processing modes that perform rectification and disparity calculations. See the STOC User's Manual for more information about these modes.

The mode may be changed while the video stream is running.

6.6.9 Controlling the Video Stream

See the functions in Section 6.5.4.

6.7 File and Memory Acquisition

The file and memory acquisition classes are subclasses of `svsAcquireImages`. This classes are used to input stereo images from files, or from arrays in memory, and present them for processing. Users who have their own stereo devices, and acquire images into memory, can use these classes to perform stereo processing with the SVS libraries.

While images are not streamed from files in the same way as from a video source, the function calls are similar. After opening the file, `the GetImage()` function is called to retrieve the stored image. A single image file set is repeatedly opened and read in by successive calls to `GetImage()`.

A file sequence consists of file sets whose basenames end in a 3-digit number. Opening a file set that is part of a sequence causes the rest of the sequence to be loaded on successive calls to GetImage().

6.7.1 File Image Object

```
svsFileImages()
~svsFileImages()
```

Constructor and destructor for the file acquisition object.

6.7.2 Getting Images from Files

```
bool Open(char *basename)
svsStereoImage *GetImage(int ms)
bool Close()
```

The `Open()` function opens a file set and reads it into the object; see Section 2.2.2 for information about file sets. The file set can include calibration parameters that describe the rectification of the images.

The image is automatically read in on the call to open; it isn't necessary to call `GetImage()`. To re-read the file images, or to read the next image in a sequence, call `GetImage()`.

The file is closed with the `Close()` function.

6.7.3 Stored Image Object

```
svsStoredImages()
~svsStoredImages()
```

Constructor and destructor for the file acquisition object.

6.7.4 Setting Images from Memory

```
bool Load(int width, int height,
          unsigned char *lim, unsigned char *rim,
          unsigned char *cim = NULL, unsigned char *cimr = NULL,
          bool rect = false, bool copy = false);
```

The `Load()` function sets images from memory into the acquisition object, after which they can be read out (and optionally rectified) using `GetImage()`. The left and right monochrome images must be loaded; the color images are optional. If the images are already rectified, set `rect` to `true`.

Normally, user images are passed into the acquisition object as pointers; `GetImage()` will output these pointers unless rectification is performed. If rectification is performed, then the object manages and outputs its own buffers, which the user should not delete.

User images can also be copied into the object's buffers, if `copy` is set to `true`. Here, the object manages all buffers, and the user program should not destroy them.

Typically, a use program will have images stored in memory, and will want to rectify them as part of the stereo process. To do so, first use `Load()` to attach the stored images to the object. Next, load a parameter file into the acquisition object, using `ReadParamFile`. Finally, turn on rectification by calling `DoRect(true)` and call `GetImage()` to return the rectified images.

There is a sample program, `loader(.exe)`, that illustrates the use of the `svsStoredImage` class. The source file `samples/loader.cpp` shows the sequence of calls needed to use images in memory.

6.8 Stereo Processing Classses

`svsStereoProcess`	Stereo processing class
`    svsMultiProcess`	Multiscale stereo processing class

The stereo processing classses perform stereo processing on stereo images encapsulated in an `svsStereoImage` object. The results are stored in the stereo image object. All relevant parameters, such as calibration information and stereo parameters, are also part of the stereo image object.

The processing class `svsStereoProcess` handles basic disparity calculation, as well as conversion of the disparity image into 3D points.

The processing class `svsMultiProcess` extends stereo processing to perform multiple scale stereo processing in computing the disparity image. Multiscale processing adds information from stereo processing at reduced image sizes.

6.8.1 Stereo and 3D Processing

```
svsStereoProcess()
~svsStereoProcess()
bool CalcStereo(svsStereoImage *si)
bool Calc3D(svsStereoImage *si,
            int x = 0, int y = 0, int w = 0, int h = 0,
            svs3Dpoint *dest = NULL, float *trans = NULL,
            double dcutoff = 0,
            svs3Dpoint *mins = NULL, svs3Dpoint *maxs = NULL)
bool CalcPoint3D(int x, int y, svsStereoImage *si,
               double *X, double *Y, double *Z)
void svsReconstruct3D(float *X, float *Y, float *Z, float x, float y,
              float disp, svsSP *sp, svsTransform *loc = NULL)
void svsProject3D (float *x, float *y, float X, float Y, float Z,
            svsSP *sp, int which, svsTransform *loc = NULL)
```

`CalcStereo()` calculates a disparity image and stores it in `si`, assuming `si` contains a stereo image pair and its `haveDisparity` flag is `false`. To recalculate the stereo results (having set new stereo processing parameters), set the `haveDisparity` flag to `false` and call `CalcStereo`().

`Calc3D()` calculates a 3D point array from the disparity image of `si`. If `si` does not have a disparity image, then it is first calculated, and then the point array is computed. The point array is stored in the stereo image object, and the `have3D` flag is set.

Valid values for 3D points are indicated by the A value in the returned points.

There are some options to `Calc3D` to make it more flexible; all of them have defaults that allow the normal processing of `Calc3D`. `x,y,w,h` define a rectangle in the disparity image, for processing just a portion of the disparity image. If w or h is zero, the whole image is used.

An optional 3D point buffer `dest` can be passed in as an argument, to be filled as output. The user is responsible for reserving memory in this buffer.

An optional transformation `trans` can be specified. `trans` should be a 4x4 homogenous transformation array, with the first four elements as the first row of the transform, the second four as the second row, etc.

A cutoff for Z distance can be given. This cutoff defines the maximum Z value that will be considered a valid return; it is useful for getting rid of high Z values. Note that this distance is always computed in the camera frame, not the transformed frame.

The minimum and maximum XYZ values can be returned in the `mins` and `maxs` structures.

For some applications, computing the whole 3D array is not necessary; only certain points are needed. In this case, the function `CalcPoint3D()` is provided. This function returns `true` if the disparity at image point x,y exists, and puts the corresponding 3D values into the X,Y,Z variables. Otherwise, it returns `false` and does not change X,Y, or Z.

These next two functions are standalone functions, not associated with the `svsStereoProcess` class.

In some cases, it is useful to compute XYZ coordinates from a user-supplied disparity. The function `svsReconstruct3D` supplies this functionality. Given an image position (which can be a subpixel position), and a disparity (in 1/16's of a pixel), and a parameter list (from the `svsStereoImage` object), it will return the 3D position. There is also an optional 3D transformation that will be applied to this 3D position.

The inverse operation, projecting a 3D point into the image, is done with the function `svsProject3D`. This function returns the image point corresponding to a given 3D point. The argument `which` should be one of `svsLEFT` or `svsRIGHT`.

There is also a member function of the `svsAcquireImages` class for computing a projection – see the definition of this class above.

Stereo parameters – number of disparities, correlation window size, horopter offset, etc. – can be adjusted using member functions of the `svsAcquireImages` class and its subclasses: see Section 6.6.7.

6.8.2 Multiscale Stereo Processing

`svsMultiProcess()`
`~svsMultiProcess()`
`bool doIt`

This multiscale class computes stereo disparity at the input image resolution, and also at a x2 reduced image size, then combines the results. Multiscale processing adds additional information, filling in parts of the disparity image that may be missed at the higher resolution.

The `svsMultiProcess` class subclasses `svsStereoProcess`, and is used in exactly the same way. The boolean variable `doIt` turns the multiscale processing on or off.

6.9 Window Drawing Classes

`svsWindow`	Window class for drawing 2D images
`svsDebugWin`	Window class for printing output

The window drawing classes output 2D stereo imagery to the display. The display window relies on the FLTK cross-platform windowing system (www.fltk.org), and provides basic graphical object drawing in addition to image display. The `svsDebugWin` class is for text output, useful when debugging programs.

It is also possible to output 3D information, especially point clouds formed from the 3D stereo reconstruction functions. This display relies on the OpenGL window capabilities of FLTK. For more information, see the example code in `samples/svsglwin.cpp`.

6.9.1 Class svsWindow

This class outputs 2D images to a display window. It can output monochrome, color, and false-color disparity images. In addition, there is an overlay facility for drawing graphical objects superimposed on the image.

`svsWindow` objects will downsize the displayed images to fit within their borders, using factors of 2. For example, a 640x480 image displayed in a 320x200 window will be decimated horizontally by 2 (to 320 columns), and vertically by 4 (to 120 rows). Images smaller than the display window are not upsampled; they are simply displayed in their normal size in the upper-left corner of the window.

Graphical overlays for the window can be drawn using FLTK drawing functions on the window, e.g., lines, circles, etc. `svsWindow` subclasses the FLTK `Fl_Window` class.

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

Constructor and destructor. The constructor creates a new `svsWindow` object, displayed at position `x,y` of any enclosing FLTK object, and with width `w` and height `h`. Typically `w` and `h` are multiples of 160x120.

The window will not be visible until `show()` is called on it.

```
DrawImage(svsStereoImage *si, int which = svsLEFT,
                void *ovArg = NULL);
ClearImage()
```

Main drawing function. Draws a component of the stereo image object si. The argument which specifies which component is drawn, according to the following table.

`svsLEFT, svsRIGHT`	Left and right monochrome images
`svsLEFTCOLOR, svsRIGHTCOLOR`	Left and right color images
`svsDISPARITY`	Disparity image (displays in green false color)

The optional last argument is passed to any assigned overlay drawing function (see `svsDrawOverlay` below).

To clear an image from the window, and reset it to black, use `ClearImage()`.

```
virtual DrawOverlay(svsImageParams *ip, void *ovArg);
DrawOverlayFn(void (*fn)(svsWindow *, svsImageParams *, void  *ovArg))
```

These functions draw overlay information on the image displayed by `svsWindow`. There are two ways to draw overlays. One is to subclass `svsWindow`, overriding the `DrawOverlay()` function. Then, the subclass can perform any FLTK drawing within the subclass function.

Another way to use overlays is to assign an overlay function to the `svsWindow` object, with `DrawOverlayFn`. This overlay function is called every time the overlay needs to be drawn.

The last argument to these functions is passed in from the argument specified in the `DrawImage()` function.

6.9.2 Class svsDebugWin

This class displays text in a scrollable window.

`svsDebugWin(int x, int y, int w, int h, char *name = NULL)`
`~svsDebugWin`

Constructor and destructor. The `x,y` arguments specify placement of the upper-left corner of the window; `w` and `h` give the width and height. An optional title (`name`) can be given.

The window will not be displayed until the `show()` function is called.

`Print(char *str)`

Prints a string on the debug window. Each string is printed on a new line, and the window scrolls to that line.

7 Update Log

Version 4.4d
May 2007

Added seeking to disk stream interface
Added STOC horopter offset for close-in stereo
*** NOTE *** Changed the sign of the horopter `offx` parameter
Storing user info on camera

Version 4.4c
March 2007

Nice colormap for disparities
Image streaming to disk
Added support for (STH)-MDCS3
Fixed svsProject3D, had factor of 0.001

Version 4.4a
November 2006

More efficient image storage on calibration
Image sizes increased to 2560x2048
Disparity images now saved in grayscale
Larger number of frames available in buffer storage

Version 4.3g
October 2006

Added STOC processing to stframe example
Added right imager to register write functions in Firmware dialog
Modified setup_XXX.bat fns to change driver in bincal directory
CalcPoint3D, for old-style calibrations, is now in m rather than mm
Added speckle processing to STOC disparity returns

Version 4.3f
August 2006

Converted MSW version to Matlab 7.1 for all Matlab projects
Added src/image_io.cpp, which was missing in MSW version
Added samples/smallv.sln file for MSVC 2003 - all samples
  projects are now converted to MSVC 2003
Max size now 2560x2048
Changed calculation of unique filter, better results

Version 4.2e
June 2006

Single Image" button removed in smallv and friends
Fixed problems with number of squares in calibration target
Fixed segfault in smallvcal when user did not have root permission -
  it was trying to save a points file
smallvcal now accepts images up to 2048x1920 in size
Version 4.2d
February 2006

Firmware dialog only in smallvcal
User reconfiguration of STOC

Version 4.2c
February 2006

DCSG/STOC button in calibration dialog
PROC_MODE_OFF allows pass-through processing of stereo data for STOCs
svsWrite3DArray and svsWrite3DCloud output wrong 3D points and would
  sometimes crash. Fixed.
Removed unneeded warptabgen.cpp file from smallv

Version 4.2b
January 2006

More support for STOC devices, including color
Fixed bug in origAddr, UnrectImagePoint
Fixed bug in adding more than 10 calibration images

Version 4.2a
December 2005

Initial support for STOC devices
Fixed bug in autoexposure for VAR devices
Upgraded to CMU MSW drivers version 6.3
Calibration targets now can accept different numbers of squares

Version 4.1g
December 2005

Firmware 3.x and 4.x now show up in local parameter dialog
SetExposure() with three arguments has been eliminated

Version 4.1f
November 2005

Disparity range not divisible by 16 gave errors
  Showed in multi-scale stereo at 48 disparities
Fixed problems with pixcap.so and smallvcal
Fixed hanging problem with MEGA-D devices on latest 2.6 kernels
Support for Matrox Meteor analog cards discontinued
flwin displays images in correct aspect ratio

Fixed problem with BMP image width not being divisible by 4
  generates new size internally
BMP files can now be 24 bpp for L/R grayscale, converted internally
  added bw parameter to svsReadFileBMP
Added VNET++ 2003 projects
Calibration load sequence can start on any number, clears all old images
Fixed some problems with deleting calibration images
Memory error in Calc3D zeroing fixed
samples/planar example works
Removed X,Y,Z,V arrays from svsStereoImage
Version 4.1e
September 2005

Initialized all border pts in Calc3D
In Calc3D, pt->A holds disparity value (integer). It is 0 or negative
  for filtered points
Calc3D disparity cutoff changed to distance cutoff
Fixed bug in larger disparity resolution (> 1Mx1M)
Fixed bug in parameter loading - string bounds
Fedora Core 4 works, using stdlib++.so.5
Fixed segfault in Linux smallv when asking for firmware parameters of
  a -VAR device
Dual-DCAM devices are now recognized

Version 4.1d
August 2005

320x240 mode works for DCSG-VAR devices
Added features to calc3D for max/min, disparity cutoff, arbitrary
  transform
Changed svs3Dpoint structure: element A is an integer, set to
  1 for valid disparity, 0 for invalid

Version 4.1c
August 2005

Shift in disparity image corrected
Fixed problem with max decimation in color DCSG
Disabled 320x240 decimated format for DCSG VAR models

Version 4.1b
June 2005

Fixed MEGA-D drivers for MSW and Linux - wouldn't accept higher than
  320x240 resolution
Fixed problem with offchip autogain/autoexposure in MDCS driver

Version 4.1a
May 2005

Release version

Fixed MEGA-D interface for 2.6 kernels - still needs patch on IEEE1394
  driver
Added SetGamma function to svsVideoImages class
Fixed problems with initialization of video parameters after start of
  streaming, under MSW
Support for DCSG devices

Version 4.0d
May 2005

Got rid of MSVCRTD dll in svs.dll

Version 4.0c
April 2005

Fixed display of XYZ values in smallv windows
Fixed vertical lines in non-MMX disparity calculation

Version 4.0b
April 2005

Warping (rectification) was extremely slow because of unnecessary
  computations
closeVideoObject() added to bttvcap.so
Bug fixes

Version 4.0a
March 2005

Basic changes to stereo algorithm
  - speed increased by ~x2
  - new uniqueness check, instead of L/R check
  - better fill-in for left side of disparity image
  - better quality of disparity results for horizontal and
    diagonal features
New functions and data structures for conversion to 3D points: X,Y,Z
  structures rather than arrays. NOTE: UNITS IN METERS, not MM
Fixes to display of odd-sized images in svsWindow (some caused
  crashes)
Reverted to standard FLTK libraries, no custom changes (FLTK 1.1.6)

Version 3.3a
February 2005

svsCheckMMX fully implemented
Changes to basic stereo correlation, now runs faster with SSE2
  instructions
closeVideoObject added to allow driver to clean up

Version 3.2g
November 2004

Edge-aware color interpolation algorithm available - slower but much better images. Added parameter color_alg to svsVideoImages.
Added setRate and setFrameDiv commands to cwrap/cmat code.
Fixed bug in RectImagePoint(), was returning FALSE when rectification present.

Version 3.2f
October 2004

Parameter files now change the resolution of the video image.
Fixed problem with VAR models not getting monochrome image during color transfer of left (reversed) image
Fixed annoying revision of directory name in first call to Load and other file functions.

Version 3.2e
October 2004

MatLab 7 now works with the CMAT sample program, and with Linux.
Added max_framediv and framediv parameters to parameter file.
Changed settings in samples/ MSVC++ projects, so they point to local DLLs and create local executables.
Rationalized the use of framediv, decimation, and binning parameters.
STH-MDCS2, interpolation on decimation for smoother images.

Version 3.2d
October 2004

MatLab interface extended - MatLab can now load functions to control a Videre Design device from within MatLab itself, in samples/cmat.
SVS Users' Manual updated with information about the MatLab interface.

Version 3.2c
September 2004

Confidence images can be returned from the stereo calculation: see the API under CalcStereo().
Example of using images in memory to run stereo - samples/loader

Version 3.2b
July 2004

New calibration parameter for X-offset calibrations (e.g., verged cameras). Disparity at infinity can be different from 0.

cwrap library for C syntax access to the SVS libraries.
Download/upload of parameter files to stereo devices with latest
firmware.

Version 3.2a
June 2004

Added SetFrameDiv() function as an alternative to SetSample(), to make
it easier to keep frame sizes while changing resolution.
Support for 2.6 kernels in Linux.
Multiscale is working again.
Support for Linux 2.6 kernels
Added acqTime variable to svsAcquireImages, give frame time in ms from
start of system open.
New SetExposure() method, allows auto gain on/off. Old one still works.

Version 3.1j
May 2004

Added support for local parameters, including vertical offset.
VADJUST tool for variable baseline devices, requires latest firmware.
Added Enumerate() and DeviceIDs() member functions to Dual DCAM
interface; can open a particular set of devices by giving an argument
to Open(char *devname) of the form "#LLLLL:#RRRRR".
Menu item in Debug window for saving to file.
Linux version support gamma value in color display svsWindow.
Fixed minor bug in DCS Open() routine, would sometimes not open a VAR
pair if another camera was on the bus.
DCS interface now allows serial numbers in the call to Open("xxxx"),
to open a specific device.
DCS interface always calls Enumerate() when videoObject is created.
Fixed problem with badpix driver enumeration for some early devices.

Version 3.1i
April 2004

Fixed bug in color interpolation for STH-MDCS, left-color non MMX
mode.
Debug window now can be resized, saved, copied.
Added functions for original->rectified coords, 3D->rectified coords.
Dual-DCAM SetRate() function fixed.

Version 3.1h
April 2004

Added autoexposure parameter to .ini files. Cameras can now assert
both autogain and autoexposure.
More efficient way of calculating X,Y,Z from x,y,d.
Dual-DCAM crash fixed.

Version 3.1g
March 2004

Selection of stereo device now allowed in dcscap.so.
Rectified image pixel is zeroed if there is no corresponding pixel in the input image.
Changed symbols in DCAM library to have dcam1394 prefix instead of dc1394, interfered with the libdc1394 functions.
Deleted unnecessary ReadFromFile() function in svsFileImages.

Version 3.1f
February 2004

smallvcal.exe was segfaulting in MSW.

Version 3.1e
January 2004

Support for image-reversed STH-MDCS-VAR, with smaller baseline (5 cm).
Fixed bug in non-binning mode for STH-MDCS-VAR-C color processing.
Display gamma setting work correctly for MDCS-type cameras - setting the gamma in the svsDCSAcquireVideo object will set it for the returned stereo images.
Fixed bug in MEGA-D drivers, not setting IP params correctly, and so bombed out on warping.

Version 3.1d
December 2003

Odd bug in correlation and mswpix code under MSVC++ 6.0 - EBX frame pointer. Should have affected the algorithm in previous versions (???). Fixed.

Version 3.1c
December 2003

Linux handles multiple OHCI cards using the /dev/video1394/0,1,2... interface. Automatically uses correct card based on detected cameras. Old interface /dev/video1394 still works for single OHCI card.

Version 3.1b
November 2003

Bug-fix release
Windows XP now has correct default for device drivers
Linux by default prints to the debug window
16 byte alignment for images

Version 3.1a
October 2003

Revised version of MSW camera lookup, now just checks the registry

Version 3.0h
September 2003

Allow calibration images that are not multiples of 320x240
Allow fixed/nonfixed aspect ratio in calibration
Fixed rectification offsets for highly-distorted images
Still better color processing on STH-MDCS
Control over auto exposure using auto_bias
FLTK 1.1.3
Fixed bug in Video startup in dCamera that caused a crash

Version 3.0g
August 2003

Added smooth disparity interpolation fix.
Fixed bug in extra disparity code.

Version 3.0f
July 2003

Fixing a nasty bug in the Linux version of smallvcal, which caused an initial segfault -- whoops, not fixed yet...

Changed the calibration procedure to always rectify with 0 disparity at infinity.

Version 3.0e
July 2003

Modified Linux MEGA-D drivers to work with kernels up to 2.4.20
In 2.4.21+ kernels, MEGA-D is shut down by broadcast packet. Kernel patch for IEEE1394 driver must be applied.

Version 3.0d
July 2003

Confidence control level increased for MMX routines
Color gains implemented on host for MDCS cameras
PLANAR.EXE program added, finds strongest plane in 3D

Version 3.0c

June 2003

Auto-exposure for MDCS cameras, implemented in host software
Bug fixes for frame sizes in Linux

Version 3.0b
May 2003

Version 3.0a and 3.0b are a new release sequence, with capabilities for the new MDCS line of stereo video cameras.

The MDCS cameras (STH-MDCS(-C,-VAR) and the monocular MDCS) all require SSE instructions, which are on Pentium III processors but not Pentium II. Other PIII clones will also work - Athlon, Transmeta, Eden.

Version 2.4a
First issue of the 2.4 version, with a new calibration routine.

Version 2.3i
Added video buffer capability for storing a sequence of video frames in smallv (up to 200). Limit can be changed by recompiling smallv.
Fixed bug in which only 99 images in a sequence were read by the file routines.

Version 2.3h
September 2002
Right color image fixes for Linux, now up-to-date with MSW versions
Re-installed dual-framegrabber mode for bttv driver

Version 2.3g
September 2002
All known color and video parameter problems fixed for the MEGA-D MSW drivers. Right image color processing not yet ported to Linux for non-binning modes.

Version 2.3f
August 2002

Full color handling for MSW versions of Stereo DCAM and MEGA-D devices; for Linux, full color handling for Stereo DCAMs.
Updated color modes for DCAMs -- 30 fps 640x480 YUV411 now available.

Version 2.3
December 2001

Full C++ version, with auto buffer handling
Completely rewritten API

Version 2.2d
August 2001

Added Dual DCAM interface
Multiple digital stereo heads available from a single application:
svsVideoImages::Enumerate() function
Right color image available to user programs

Version 2.2c
July 2001

Added scaling feature in calibration
Bug fixes

Version 2.2a
June 2001

Completely re-written in C++, new API
Automatic buffer handling

Version 2.1c
March 2001

1. Last stable release before C++ version 2.2a
2. OpenGL window now has mouse-drag rotation, better rotation center
3. Minor changes to the interface

Version 2.1b
February 2001

1. Warping code now works with subwindows (no vergence yet)
2. Calibration procedure updated to include standard Videre Design stereo head parameters
3. Added Debug Window for debuggin feedback
4. svsSP structure revised to have subwindow warping offsets
5. Color interpolation for non-binning modes
6. Linux fully in sync with MSW version

Version 2.1a
December 2000

1. Latest 1394/FireWire updates to stereo head firmware, images are stable under all pan/tilt motions
2. Calibration software uses checkerboard target, more distortion

parameters
3. Revised svsSP structure has all info about cameras

Version 2.0
July 2000

1. Platform-independent windowing system, FLTK, for display
2. Color support
3. OpenGL support
4. Digital framegrabber (1394) support
5. Shared libraries in Unix

Version 1.4
December 1999

1. Added calibration software using planar target
2. Added support for 3D transformation of disparities

Version 1.3
May 1999

1. Various bug fixes

Version 1.2
August 1998

1. Added more LOG bits on Windows side
2. Added support for more framegrabbers, use svsgrab.dll in Windows
3. Added framegrabber left/right swap for line interlace
4. Added warping pre-filter and internal parameter code
5. Fixed .ssi file save bug

Version 1.1
April 1998
Bug fix release

1. Fixed bug with pb_1 constant uninitialized
2. Added more bits in LOG for Unix side, need to do it for Windows

Version 1.0
April 1998

First public release