TRICLOPS

Software Development Kit (SDK)

Version 3.1

User’s guide and command reference

Triclops Software License Agreement

The Triclops Stereo Vision Software Development Kit (the "Software") is owned and copyrighted by Point Grey Research, Inc. All rights are reserved. The Original Purchaser is granted a license to use the Software subject to the following restrictions and limitations.

1. The license is to the Original Purchaser only, and is nontransferable unless you have received written permission of Point Grey Research, Inc.
2. The Original Purchaser may use the Software only with Point Grey Research, Inc. cameras owned by the Original Purchaser, including but not limited to, Digiclops® and Bumblebee Camera Modules.
3. The Original Purchaser may make back-up copies of the Software for his or her own use only, subject to the use limitations of this license.
4. Subject to s.5 below, the Original Purchaser may not engage in, nor permit third parties to engage in, any of the following:
   A. Providing or disclosing the Software to third parties.
   B. Making alterations or copies of any kind of the Software (except as specifically permitted in s.3 above).
   C. Attempting to un-assemble, de-compile or reverse engineer the Software in any way.
   D. Granting sublicenses, leases or other rights in the Software to others.
5. Original Purchasers who are Original Equipment Manufacturers may make Derivative Products with the Software. Derivative Products are new software products developed, in whole or in part, using the Software and other Point Grey Research, Inc. products. Point Grey Research, Inc. hereby grants a license to Original Equipment Manufacturers to incorporate and distribute the libraries found in the Software with the Derivative Products. The components of any Derivative Product that contain the Software libraries may only be used with Point Grey Research, Inc. products, or images derived from such products.

5.1 By the distribution of the Software libraries with Derivative Products, Original Purchasers agree to:

   A. not permit further redistribution of the Software libraries by end-user customers;
   B. include a valid copyright notice on any Derivative Product; and
   C. indemnify, hold harmless, and defend Point Grey Research, Inc. from and against any claims or lawsuits, including attorney's fees, that arise or result from the use or distribution of any Derivative Product.

Point Grey Research, Inc. reserves the right to terminate this license if there are any violations of its terms or if there is a default committed by the Original Purchaser. Upon termination, for any reason, all copies of the Software must be immediately returned to Point Grey Research, Inc. and the Original Purchaser shall be liable to Point Grey Research, Inc. for any and all damages suffered as a result of the violation or default.

Chapter 1: Introduction

This chapter presents the features of the Triclops Stereo Vision Software Development Kit (SDK). It also gives the reader pointers to other resources useful for mastering stereo vision technology.

Stereo Vision Technology

Stereo vision technology allows range measurement using triangulation between slightly offset cameras. The Digiclops™ Stereo Vision System consists of a three-camera module and a software system that performs range measurements. The camera module generates three gray-scale, or color, images that are digitized and stored in the memory of the computer. The software system analyzes the images and establishes correspondence between pixels in each image. Based on the camera's geometry and the correspondences between pixels in the images, it is possible to determine the distance to points in the scene.

Consider the example in Figure 1. In this example, the camera module has simultaneously obtained three images of the scene. While the images appear quite similar, closer inspection reveals a shift between closer objects and those that are further away. Based on the amount of shift, the system is able to determine the distance to the objects in the scene. Figure 1 also shows a depth image that is the result of the Triclops Stereo Vision SDK. In this image, closer objects are represented with brighter shades of gray while objects further away are represented with darker shades of gray.

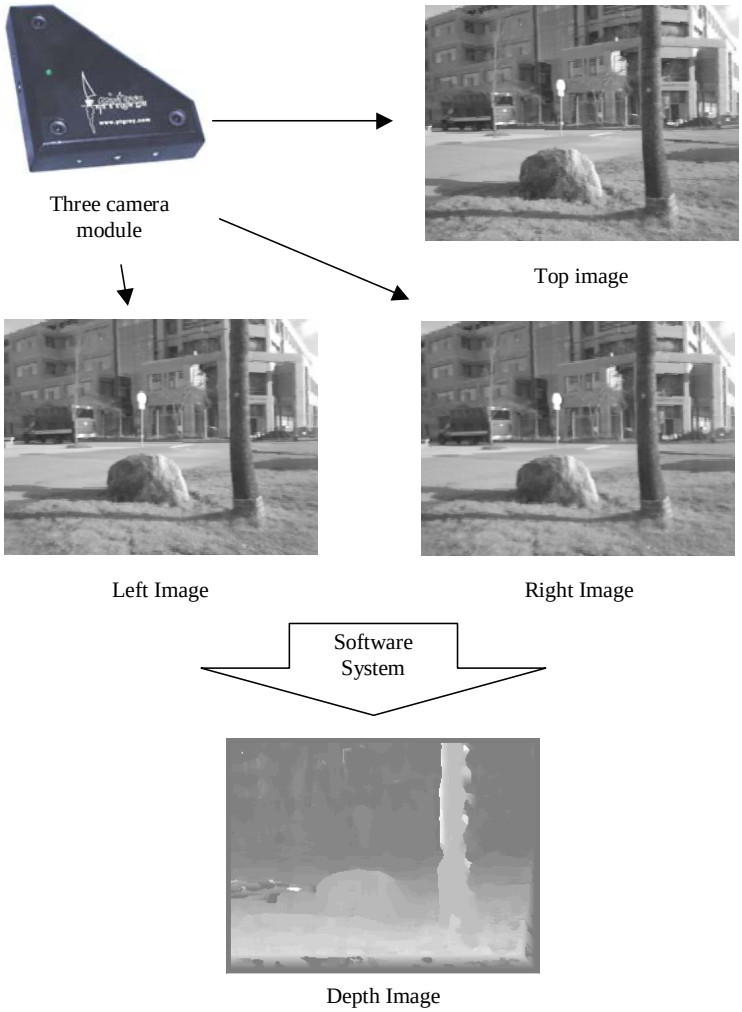

Figure 1: Stereo Processing;
From images to distance measurements

With the depth image it is possible to determine the 3D position of every point in the image relative to the camera module. This section was written to give the reader intuition about the functionality of the system. A more detailed description of the process and programming details will be discussed in further chapters.

Who Should Read this Manual

This manual is intended for personnel responsible for installing the Triclops Stereo Vision SDK and for programmers developing applications with the Triclops system.

Installers must have a working knowledge of the Windows 98 or Windows 2000 operating system at the system administrator level.

Developers will find familiarity with the concepts of stereo vision very useful in utilizing the Triclops system and making sense of the stereo results. For a general background in stereo vision, see Chapter 7: Stereo Vision Details.

System Requirements

To use the Triclops Stereo Vision SDK, you must have an IBM-compatible PC with:
A Pentium MMX or Pentium II processor of 166MHz or faster
Windows 98 or Windows 2000
Recommended at least 16 MB of RAM

Online Resources

For the latest demos and sample code written for the Triclops Stereo Vision SDK, visit our web page at:
http://www.ptgrey.com

Chapter 3: Installing the Software

This chapter discusses installing the software and setting up the operating system.

Installation of Triclops

The Triclops software system is currently distributed on a CD-ROM disk. Please consult the README file on the disk for a detailed description of the installation procedure. In addition to the CD-ROM, there will also be a floppy disk included with the system. This disk contains the calibration file specific to your Digiclops camera unit. In addition to the calibration file, there may also be updated libraries on this floppy. Please consult the README file on the floppy disk for details of contents and installation.

This section assumes the target operating systems is Microsoft Windows 98/2000.

To install the Triclops software:

1. Place the CD-ROM installation disk into your CD-ROM drive.
2. The "Point Grey Research, Inc." message box will appear.
3. Select the "Install Triclops" button and follow the prompts given by the Setup Wizard. It is recommended that you install the Triclops software in the default location suggested by the install shield. If not, the example program project file settings will have to be changed to include the correct SDK paths for include and library files.

The "Point Grey Research, Inc." message box also has an "Explore CD" button. When this is selected, you will be presented with the CD contents that contains:

Subdirectory	Contents
Bin	This subdirectory contains PGRView, a program which allows one to view point cloud files generated by Example7 of the provided source code example programs.
Src	This subdirectory contains Triclops example source code. The Triclops example source code is discussed in Chapter 5: Programming with the Triclops Application Programming Interface (API).
Include	This subdirectory contains the include files required to compile using the Triclops library. The primary include file is the triclops.h. This subdirectory also contains the pnmutils.h file, which includes utilities for reading and writing PGM and PPM format images.
Lib	This subdirectory contains the Triclops and pnmutils libraries. It also contains the triclops.dll, which is the dynamic link library for triclops.lib.
Doc	This subdirectory contains the Triclops Manual, and the PGRView User Manual.

Image Viewing

At this time it is a good idea to also install an image-viewing program. The Triclops library supports the PGM image format. Paint Shop Pro is a good choice for viewing PGM files. The Triclops or the Digiclops™ installation disk has on it the option of installing this program.

Chapter 4: Getting Started

This chapter goes through a number of exercises designed to ensure that the system is properly installed and to give the user an initial feel for the system's capabilities.

This chapter is for Microsoft Windows 98/2000 users

Running the Demo Program

After installing the Triclops SDK and an appropriate stereo device (i.e.: a Digiclops stereo vision system), it is a good idea to test the device and the stereo installation by running the interactive demonstration programs. Please refer to the documentation for your stereo vision system installation for a description of the different demonstration programs available and their operation.

For further information on the meaning of stereo parameters that can be set in the demonstration programs, please refer to Chapter 7: Stereo Vision Details.

Compiling a Triclops Program

Now that you know that the system is installed correctly, you can try compiling a sample program and verify that the Triclops library and include files are properly set. For this step, we assume that your development environment is Microsoft Visual C++. In the following steps, the variable %TRICLOPS_PATH% represents the directory in which the Triclops SDK has been installed. The default location for this directory is C:\Program Files\Point Grey Research\Triclops Stereo Vision SDK.

1. Create a new directory outside of the %TRICLOPS_PATH% directory sub-tree.
2. Copy the directory %TRICLOPS_PATH%\examples to your newly created directory.
3. Start Visual C++ and open the examples.dsw workspace in your newly created directory.
4. Set the library path to %TRICLOPS_PATH%\lib.
5. Set the preprocessor include path to %TRICLOPS_PATH%\include
6. Set the active project to example 1.
7. Build the program.
8. Run the program.
9. View the images, disparity.pgm and reference.pgm that are created by the program in your newly created directory with Paint Shop Pro.

This program reads a raw stereo image from a file, performs stereo processing and saves the gray-scale images and depth map into separate files.

If you are using a compiler other than Visual C++ you will have to specify the library path and the include path in order for the compiler to find the appropriate files.

Chapter 5: Programming with the Triclops Application Programming Interface (API)

This chapter presents the Triclops API and explains the engine behind the functions. Several programming examples are presented in order to illustrate the API.

Programming with the Triclops Stereo Vision SDK

The goal of the Triclops Stereo Vision SDK is to provide the user with accurate and fast depth map generation. Therefore, the ultimate result of the Triclops Application Programming Interface (API) is a depth map. Depth maps can be produced in a number of different ways. Further characteristics of depth maps may vary depending on a number of settings. Triclops system allows specifying the following characteristics of the stereo processing:

Stereo Parameters	Description of the Parameter
Image size/resolution	The API allows the user to specify the size of the image that is to be used as the initial gray-scale image.
Disparity range	Allows the user to specify the range that distance measurements are to be done in.
Mask size	Allows the user to specify the coarseness of features that are to be matched between images.
Preprocessing	Specifies whether matching should be done on gray-scale or preprocessed images, such as edge images.
Validation	Specifies the methods used for verifying the correctness of matched-between images.
Regions Of Interest	Specifies the region of the image that processing should be done on. This feature allows processing speedup.
Subpixel Interpolation	This feature allows establishing correspondences to subpixel accuracy allowing generation of more precise distance measurements.

For detailed descriptions of stereo vision parameters please refer to Chapter 7: Stereo Vision Details.

Triclops Context

The Triclops software system allows specifying all characteristics of stereo processing discussed above. Furthermore, the software system allows the specification of multiple stereo processing that may occur on a single set of images. To enable efficient stereo processing of different kinds on the same set of images, the concept of Triclops Contexts is introduced.

By using the Triclops context it is possible to encapsulate all of the information required for a specific kind of stereo processing. Furthermore, multiple Triclops contexts allow for the sharing of data and processing with minimal effort on the user's part. Triclops contexts store camera configuration, parameters of stereo processing, input data and results.

A Triclops context must first be initialized using the configuration of the camera module. Configuration contains information about the number and geometry of the cameras, as well as the intrinsic and extrinsic parameters of the cameras.

A Triclops context is then configured for the specific kind of stereo processing. Parameters such as the processing resolution, disparity range, validation, and subpixel interpolation may be specified.

Next, a Triclops context must be assigned images that are to be processed by the stereo kernel. This is done by passing into a context information obtained from the stereo device, or by loading the image information from the file.

The Triclops context is then used for performing image pre-processing and the stereo processing. The results of stereo processing can then be retrieved from the context.

Triclops Example Source Code

In order to quickly understand the Triclops library, a number of examples are provided with the software distribution. All examples can be found in the %TRICLOPS_PATH%\examples directory. It is a good idea for the user to make a copy of the examples and compile them in a new location.

Important: Please check error codes!

It is important to note that the following examples, with the exception of #6, do not check for error codes. We have chosen to do this to ensure that the examples are not cluttered, and are therefore easier to follow. When creating your own source code, it is important that you include error checking. A list of different TriclopsErrors can be found at the beginning of chapter 6.

All Triclops Stereo Vision SDK functions return a TriclopsError enumerated type error code. This error code can be used to determine the reason for the function failure. An example of checking the error code and formatting an error string is:

```
#include “stdafx.h”
#include “triclops.h”

int
main( int argc, char ** argv )
{
   . . .
   TriclopsError e;
   e = triclopsStereo( context );
   if ( e != ok )
   {
      char szErrorString[1024];           // the error message buffer
      sprintf( szErrorString,
   “triclopsStereo() error : %s\n”,
               triclopsErrorToString( e ) );
      // pop up an error dialog with the formatted error string
      MessageBox( 0, szErrorString, “Triclops Error”, 0 );
   }
   . . .
}
```

In addition, the Triclops Stereo Vision SDK works with several different stereo devices. This includes Triclops, Color Triclops, Digiclops and Color Digiclops. The examples provided are designed to work with either a Digiclops or a Color Digiclops. For other stereo acquisition devices, image acquisition function need to be changed.

Example 1: Generating a Depth Map Image

```
/*
 * Example 1:
 *
 * Loads three stereo images from a PPM file and performs stereo
processing
 * to create a disparity image which is then output to depth.pgm.
 *
 * This program assumes there is a camera calibration file in the same
directory
 * named 'config'.
 */

#include <stdio.h>
#include <stdlib.h>
#include "triclops.h"
#include "pnmutils.h"

int
main( int argc, char **argv )
{
   TriclopsContext     context;
   TriclopsImage       depthImage;
   TriclopsInput       inputData;
   TriclopsError       tErr;

   // get the camera module configuration
   tErr = triclopsGetDefaultContextFromFile( &context, "config" );
   if ( tErr != ok )
   {
      printf( "Can't open calibration file 'config'\n" );
   }

   // Load images from file
   ppmReadToTriclopsInput( "input.ppm", &inputData );

   // set up some stereo parameters:
   // set to 320x240 output images
   tErr = triclopsSetResolution( context, 240, 320 );
   // set disparity range
   tErr = triclopsSetDisparity( context, 5, 60 );

   // set the display mapping
   // note: disparity mapping corrupts the disparity values so that
making
   // distance measurements is more difficult and less accurate.
   // Do not use it when you intend to actually use disparity values for
   // purposes other than display
   tErr = triclopsSetDisparityMapping( context, 128, 255 );
   tErr     = triclopsSetDisparityMappingOn( context, 1 );
   // set the validation mappings to 0 (black)
   tErr = triclopsSetUniquenessValidationMapping( context, 0 );
   tErr = triclopsSetTextureValidationMapping( context, 0 );

   // Preprocessing the images
   tErr = triclopsPreprocess( context, &inputData );
```

```
   // stereo processing
   tErr =  triclopsStereo( context );

   // retrieve the depth image from the context
   tErr = triclopsGetImage( context, TriImg_DISPARITY, TriCam_REFERENCE,
&depthImage );

   // save the depth image
   tErr = triclopsSaveImage( &depthImage, "depth.pgm" );

   // clean up memory allocated in context
   freeInput( &inputData );
   tErr = triclopsDestroyContext( context );


   return 0;
}
```

This example generates a depth map from a PPM image file.

1. The example performs stereo processing on the file “input.ppm”, and saves the depth map into a file. It is not necessary to worry about grabbing and displaying for this example.
2. After loading the camera configuration file from a file named “config” in the same directory, the input PPM file is read and is stored in a TriclopsInput data structure.
3. The resolution, disparity range, and pixel mapping values are then set in the program.
4. The rest of the stereo parameters are left with the default values loaded from the configuration file.
5. Image preprocessing and stereo processing is then done.
6. The disparity image associated with the reference camera is retrieved, and saved to a gray-scale pgm file named “depth.pgm”.

Example 2: Grabbing images and stereo processing

```
/*
 * Example 2:
 *
 * Gets input from the Digiclops, and performs stereo processing
 * to create a disparity image. A raw image from the reference camera
 * and the disparity image are both written out.
 */

#include <stdio.h>
#include <stdlib.h>
#include "triclops.h"
#include "digiclops.h"

int
main( int argc, char **argv )
{
   TriclopsContext     triclops;
   TriclopsImage       depthImage;
   TriclopsImage       refImage;
   TriclopsInput       inputData;
   DigiclopsContext    digiclops;

   // open the Digiclops
   digiclopsCreateContext( &digiclops );
   digiclopsInitialize( digiclops, 0 );

   // get the camera module configuration
   digiclopsGetTriclopsContextFromCamera( digiclops, &triclops );

   // set the Digiclops resolution
   // use 'HALF' resolution when you need faster throughput, especially
for
   // color images
   // digiclopsSetImageResolution( digiclops, DIGICLOPS_HALF );
   digiclopsSetImageResolution( digiclops, DIGICLOPS_FULL );

   // start grabbing
   digiclopsStart( digiclops );

   // grab an image
   digiclopsGrabImage( digiclops );
   digiclopsExtractTriclopsInput( digiclops, STEREO_IMAGE, &inputData );

   // set up some stereo parameters:
   // set to 320x240 output images
   triclopsSetResolution( triclops, 240, 320 );

   // set disparity range
   triclopsSetDisparity( triclops, 5, 60 );

   // set the display mapping
   triclopsSetTextureValidationMapping( triclops, 0 );
   triclopsSetUniquenessValidationMapping( triclops, 0 );
```

```
   // Preprocessing the images
   triclopsPreprocess( triclops, &inputData );

   // stereo processing
   triclopsStereo( triclops );

   // retrieve the depth image from the triclops context
   triclopsGetImage( triclops, TriImg_DISPARITY, TriCam_REFERENCE,
&depthImage );
   // retrieve the raw image from the triclops context
   triclopsGetImage( triclops, TriImg_RAW, TriCam_REFERENCE, &refImage
);

   // save the depth and reference images
   // NOTE: because we did not use "disparity mapping" these images will
appear
   // very dark.  This is because the disparity values have not been
scaled for
   // viewing.
   triclopsSaveImage( &depthImage, "depth.pgm" );
   triclopsSaveImage( &refImage, "reference.pgm" );

   // close the digiclops
   digiclopsStop( digiclops );
   digiclopsDestroyContext( digiclops );

   // destroy the triclops context
   triclopsDestroyContext( triclops ) ;

   return 0;
}
```

This example is a modification of the previous one. Instead of accepting input from a user-supplied PPM file, images are obtained from the first Digiclops device on the 1394 bus.

The example works as follows:

1. Digiclops™ is opened and initialized.
2. The camera module configuration is retrieved.
3. The resolution for Digiclops™ is set; to maximize throughout or for colour images, set the resolution to HALF.
4. The camera starts to grab images.
5. The Triclops resolution is set to 240 x 320.
6. The disparity range of the Triclops is set to (5, 60)
7. The texture validation mapping and the uniqueness validation are turned off.
8. The images are preprocessed and stereo-processed.
9. The depth image is retrieved and saved to the file “depth.pgm”.
10. The raw image is retrieved and saved to “reference.pgm”.
11. The images can be viewed in an image viewer such as Paint Shop Pro, using “reference.pgm” for comparison.

Example 3: Subpixel interpolation and depth calculation

```
#include <stdio.h>
#include "triclops.h"
#include "digiclops.h"

int
main( int argc, char **argv )
{
   TriclopsContext     triclops;
   TriclopsImage16     depthImage16;
   TriclopsImage       refImage;
   TriclopsInput       inputData;
   DigiclopsContext     digiclops;
   int                 i, j;
   float               x, y, z;
   FILE *              pointFile;
   FILE *              vrmlFile;
   int                 pixelinc;

   // open the Digiclops
   digiclopsCreateContext( &digiclops );
   digiclopsInitialize( digiclops, 0 );

   // get the camera module configuration
   digiclopsGetTriclopsContextFromCamera( digiclops, &triclops );

   // set the digiclops to deliver the stereo image only
   digiclopsSetImageTypes( digiclops, STEREO_IMAGE );

   // set the Digiclops resolution
   // use 'HALF' resolution when you need faster throughput, especially
for
   // color images
   // digiclopsSetImageResolution( digiclops, DIGICLOPS_HALF );
   digiclopsSetImageResolution( digiclops, DIGICLOPS_FULL );

   // start grabbing
   digiclopsStart( digiclops );

   // grab an image
   digiclopsGrabImage( digiclops );
   digiclopsExtractTriclopsInput( digiclops, STEREO_IMAGE, &inputData );
```

```
  // set up some stereo parameters:
   // set to 320x240 output images
   triclopsSetResolution( triclops, 240, 320 );
   // set disparity range
   triclopsSetDisparity( triclops, 10, 60 );

   // turn on sub-pixel interpolation
   triclopsSetSubpixelInterpolation( triclops, 1 );

   // preprocessing the images
   triclopsPreprocess( triclops, &inputData );

   // stereo processing
   triclopsStereo( triclops );

   // retrieve the interpolated depth image from the triclops context
   triclopsGetImage16( triclops, TriImg16_DISPARITY, TriCam_REFERENCE,
&depthImage16 );
   triclopsSaveImage16( &depthImage16, "disp16.pgm" );

   // save points to disk
   pointFile = fopen( "points.txt", "w+" );
   vrmlFile = fopen( "points.wrl", "w+" );

   fprintf( vrmlFile, "#VRML V1.0 ascii\n" );
   fprintf( vrmlFile, "Material {diffuseColor 0 1 0 }\n" );
```

```
   // determine the number of pixels spacing per row
   pixelinc    = depthImage16.rowinc/2;
   for ( i = 0; i < depthImage16.nrows; i++ )
   {
      unsigned short * row     = depthImage16.data + i * pixelinc;
      for ( j = 0; j < depthImage16.ncols; j++ )
      {
       int disparity = row[j];

       // filter invalid points
       if ( disparity < 0xFF00 )
       {
          // convert the 16 bit disparity value to floating point x,y,z
          triclopsRCD16ToXYZ( triclops, i, j, disparity, &x, &y, &z );

          // look at points within a range
          if (z > 0.25 && z < 1.0)
          {
             fprintf( pointFile, "%f %f %f 255 255 255\n", x, y, z );
             fprintf( vrmlFile, "Separator {\n" );
             fprintf( vrmlFile, "   Translation "
              "{ translation %f %f %f }\n",
              x, y, z );
             fprintf( vrmlFile, "   Cube { width 0.005 height 0.005
depth 0.005 }\n" );
             fprintf( vrmlFile, "}\n" );
          }
       }
      }
   }
   fclose( pointFile );
   fclose( vrmlFile );

   // retrieve three rectified images from the triclops context
   triclopsGetImage( triclops, TriImg_RECTIFIED, TriCam_L_RIGHT,
&refImage );
   triclopsSaveImage( &refImage, "rect_right.pgm" );

   triclopsGetImage( triclops, TriImg_RECTIFIED, TriCam_L_TOP, &refImage
);
   triclopsSaveImage( &refImage, "rect_top.pgm" );

   triclopsGetImage( triclops, TriImg_RECTIFIED, TriCam_L_LEFT,
&refImage );
   triclopsSaveImage( &refImage, "rect_left.pgm" );

   // close the digiclops
   digiclopsStop( digiclops );
   digiclopsDestroyContext( digiclops );

   // destroy the triclops context
   triclopsDestroyContext( triclops ) ;

   return 0;
}
```

This example demonstrates the use of subpixel interpolation, 16-bit images, and the calculation of three-dimensional XYZ coordinates from stereo images.

1. After grabbing the image from the camera, the "triclopsSetSubpixelInterpolation()" call turns on subpixel interpolation.
2. The relevant stereo options are set, and then the images are preprocessed and stereo processed. Due to interpolation, a 16-bit depth image can then be retrieved from the context and saved. For each pixel, the most significant byte is the disparity in whole pixels and the least significant byte is the fractional portion of the disparity.
3. To convert from 16-bit disparity values to floating point disparity values, one can use the following C-code fragment:

```
unsigned short disparity16 = <some value from the depth map>;
float          disparityFloat;
// convert disparity16 to disparityFloat
disparityFloat = (float) disparity16 / 256.0;
```

   16-bit disparity data can be accessed for every valid interpolated pixel in this image (with invalid pixels having a disparity value greater than 0xFF00), and given as a parameter to the "triclopsRCD16ToXYZ()" function, which returns the corresponding three-dimensional XYZ coordinates. Any data within 0.25 to 1.0 meters of the camera is written to a text file ("points"). It should be noted that subpixel resolution images do not follow the mapping schemes that regular images do. Validation mapping is applied only to the least significant byte. Therefore, if texture validation mapping is set to 0, and uniqueness validation mapping is set to 1, the 16-bit invalid texture pixel will be 0xFF00 and the 16-bit invalid uniqueness pixel will be 0xFF01.

4. Finally, the three rectified images for each camera are saved to disk, and the frame grabber is closed.

   Note that "triclopsSetDisparityMapping()" is not called in this example. Therefore, disparity values will not be scaled for better contrast and brightness for human viewers. This is important, as the disparity values should not be scaled if one of the triclopsRCDToXYZ family functions (triclopsRCD8ToXYZ, triclopsRCD16ToXYZ, triclopsRCDFloatToXYZ) will be used. In cases where the scaling is extremely desirable, use triclopsRCDMappedToXYZ().

5. The points file can be viewed with the PGRView program.

Example 4: Regions of Interest

```
#include <stdio.h>
#include "triclops.h"
#include "digiclops.h"

int
main( int argc, char **argv )
{
   TriclopsContext     triclops;
   TriclopsImage       depthImage;
   TriclopsImage       refImage;
   TriclopsInput       inputData;
   DigiclopsContext     digiclops;
   TriclopsROI         *rois;
   int                 maxRois;

   // open the Digiclops
   digiclopsCreateContext( &digiclops );
   digiclopsInitialize( digiclops, 0 );

   // get the camera module configuration
   digiclopsGetTriclopsContextFromCamera( digiclops, &triclops );

   // set the digiclops to deliver the stereo image only
   digiclopsSetImageTypes( digiclops, STEREO_IMAGE );

   // set the Digiclops resolution
   // use 'HALF' resolution when you need faster throughput, especially
for
   // color images
   // digiclopsSetImageResolution( digiclops, DIGICLOPS_HALF );
   digiclopsSetImageResolution( digiclops, DIGICLOPS_FULL );

   // start grabbing
   digiclopsStart( digiclops );
```

```
// grab an image
   digiclopsGrabImage( digiclops );
   digiclopsExtractTriclopsInput( digiclops, STEREO_IMAGE, &inputData );

   // set up some stereo parameters:
   // set to 640x480 output images
   triclopsSetResolution( triclops, 480, 640 );
   // set the stereo mask size to maximum
   triclopsSetStereoMask( triclops, 11 );
   // set disparity range
   triclopsSetDisparity( triclops, 5, 60 );

   // set the display mapping
   triclopsSetTextureValidationMapping( triclops, 0 );
   triclopsSetUniquenessValidationMapping( triclops, 0 );

   // get the pointer to the regions of interest array
   triclopsGetROIs( triclops, &rois, &maxRois );

   if (maxRois >= 4) {
      // set up four regions of interest:

      // entire upper left quadrant of image
      rois[0].row   = 0;
      rois[0].col   = 0;
      rois[0].nrows = 240;
      rois[0].ncols = 320;

      // part of the lower right
      rois[1].row   = 240;
      rois[1].col   = 320;
      rois[1].nrows = 180;
      rois[1].ncols = 240;

      // centered in upper right quadrant
      rois[2].row   = 60;
      rois[2].col   = 400;
      rois[2].nrows = 120;
      rois[2].ncols = 160;

      // small section of lower left
      rois[3].row   = 300;
      rois[3].col   = 30;
      rois[3].nrows = 80;
      rois[3].ncols = 80;

      triclopsSetNumberOfROIs( triclops, 4 );
   }
```

```
   // Preprocessing the images
   triclopsPreprocess( triclops, &inputData );

   // stereo processing
   triclopsStereo( triclops );

   // retrieve the depth image from the triclops context
   triclopsGetImage( triclops, TriImg_DISPARITY, TriCam_REFERENCE,
&depthImage );
   // retrieve the raw image from the triclops context
   triclopsGetImage( triclops, TriImg_RAW, TriCam_REFERENCE, &refImage
);

   // save the depth and reference images
   triclopsSaveImage( &depthImage, "depth.pgm" );
   triclopsSaveImage( &refImage, "reference.pgm" );

   // close the digiclops
   digiclopsStop( digiclops );
   digiclopsDestroyContext( digiclops );

   // destroy the triclops context
   triclopsDestroyContext( triclops ) ;

   return 0;
}
```

This example is another modification of example 2. It demonstrates the use of Regions of Interest (ROIs). As before, input is retrieved from the Digiclops and stereo processing options are set up. Note this time that a larger image (640x480) is used, so the stereo mask should also be increased.

Since a larger image is being used, stereo processing time will be longer. We can speed this up by specifying Regions of Interest. This is done by getting a pointer to an array of TriclopsROI structures with the triclopsGetROIs function call, and setting individual elements of this array to specify the ROIs. The triclopsSetNumberOfROIs function informs the Triclops library how many Regions of Interest have been set up inside the array. Preprocessing and stereo processing then take place as before, and the depth and reference images are saved. The depth image will have data in each of the four specified Regions of Interest.

This example works as follows:

1. The Digiclops™ is opened and initialized.
2. The camera module configuration is retrieved and the image resolution is set.
3. The camera is set to deliver stereo images only.
4. The camera begins to grab images.
5. The Triclops resolution, disparity range, and stereo mask is set.
6. The display mapping is set up.
7. Four regions of interest are mapped out on the image, in this case, the entire upper left quadrant, part of the lower right quadrant, centred in the upper right quadrant, and a small section of the lower left quadrant.
8. The image is preprocessed and stereo processed.
9. The image depth is retrieved and saved to the file “depth.pgm”.
10. The raw image is retrieved and saved to the file “reference.pgm” for comparison.

Example 5: Finding the depth of the center of the image

```
#include <stdio.h>
#include "triclops.h"
#include "digiclops.h"

int
main( int argc, char **argv )
{
   TriclopsContext     triclops;
   TriclopsImage       depthImage;
   TriclopsInput       inputData;
   DigiclopsContext    digiclops;
   TriclopsROI         *rois;
   int                 maxRois;
   int                 disparity;
   int                 i, j, nPoints;
   float               x, y, z;
   float               avgZ;


   // open the Digiclops
   digiclopsCreateContext( &digiclops );
   digiclopsInitialize( digiclops, 0 );
   // get the camera module configuration
   digiclopsGetTriclopsContextFromCamera( digiclops, &triclops );
   // set the digiclops to deliver the stereo image only
   digiclopsSetImageTypes( digiclops, STEREO_IMAGE );
   // set the Digiclops resolution
   // use 'HALF' resolution when you need faster throughput, especially
for
   // color images
   // digiclopsSetImageResolution( digiclops, DIGICLOPS_HALF );
   digiclopsSetImageResolution( digiclops, DIGICLOPS_FULL );
   // start grabbing
   digiclopsStart( digiclops );

   // set up some stereo parameters:
   // set to 320x240 output images
   triclopsSetResolution( triclops, 240, 320 );

   // set disparity range
   triclopsSetDisparity( triclops, 5, 60 );
```

```
   // set invalid pixel value
   triclopsSetTextureValidationMapping( triclops, 0 );
   triclopsSetUniquenessValidationMapping( triclops, 0 );

   // get the pointer to the regions of interest array
   triclopsGetROIs( triclops, &rois, &maxRois );
   if (maxRois >= 1)
   {

      // set up a 40x30 region of interest in the center of the image
      rois[0].row = 104;
      rois[0].col = 139;
      rois[0].nrows = 30;
      rois[0].ncols = 40;

      triclopsSetNumberOfROIs( triclops, 1 );
   }

   printf( "Center  \t\tAverage\n" );

   while (1)
   {
      unsigned char  *row;

      // grab an image
      digiclopsGrabImage( digiclops );
      digiclopsExtractTriclopsInput( digiclops, STEREO_IMAGE, &inputData
);

      // Preprocessing the images
      triclopsPreprocess( triclops, &inputData );

      // stereo processing
      triclopsStereo( triclops );

      // retrieve the depth image from the context
      triclopsGetImage( triclops, TriImg_DISPARITY, TriCam_REFERENCE,
&depthImage );

      // get the depth of the center pixel (at 159, 119)
      disparity = depthImage.data[119 * depthImage.rowinc + 159];
```

```
      // invalid pixel?
      if ( disparity == 0 )
      {
       printf("INVALID \t\t");
      }
      else
      {
       triclopsRCD8ToXYZ( triclops, 119, 159, disparity, &x, &y, &z );
       printf("%f\t\t", z);
      }

      // calculate depth average of the entire ROI
      avgZ = 0;
      nPoints = 0;
      for ( i = 104; i < 134; i++ )
      {
       row = depthImage.data + depthImage.rowinc * i;
       for ( j = 139; j < 179; j++ )
       {
          disparity = row[ j ];
          if ( disparity != 0 )
          {
             nPoints++;
             triclopsRCD8ToXYZ( triclops, i, j, disparity, &x, &y, &z );
             avgZ += z;
          }
       }
      }
      if ( nPoints == 0 )
      {
       printf( "INVALID \r" );
      }
      else
      {
       avgZ /= nPoints;
       printf( "%f\r", avgZ );
      }

      fflush(stdout);
   }

   // close the digiclops
   digiclopsStop( digiclops );
   digiclopsDestroyContext( digiclops );

   // destroy the context
   triclopsDestroyContext( triclops ) ;

   return 0;
}
```

Example 5 combines depth finding with the use of a region of interest; it continuously grabs an image from the Digiclops, calculates the depth for a pixel at the center as well as the average depth for an area around the same pixel, and prints the values.

Once all stereo parameters have been set, a 40 by 30 region of interest in the center is set up. The code then enters an infinite loop. Within this loop, an image is taken from the Digiclops and the region of interest is stereo processed. The disparity data at the center pixel is tested for validity. If it is valid, it is used to calculate the depth data (the Z value returned by the triclopsRCD8ToXYZ function). The same is also done for every other pixel in the region of interest, except that these Z values are averaged together. These two values are printed, and the loop starts again.

This example can be tested by holding a large flat object in front of the Digiclops device (for example, a sheet of newspaper), and moving it towards and away from the camera. The depth values that are displayed should change accordingly.

The example works as follows:

1. Digiclops™ is opened and initialized.
2. The camera module configuration is obtained.
3. Digiclops™ is set to return the stereo image only.
4. The resolution is set (set to HALF for faster throughput; i.e. colour images).
5. The resolution is set to 240 x 320.
6. The disparity range is set to (5, 60).
7. The texture validation mapping and the uniqueness validation mapping are turned off (sets an invalid pixel value).
8. The centre of the image is found.
9. The image is grabbed, preprocessed, and stereo processed.
10. The depth image is retrieved and the depth at the centre pixel is obtained.
11. The depth image at the centre of the image is calculated.

Example 6: Printing configuration information

```
#include <stdio.h>
#include <stdlib.h>
#include "triclops.h"
#include "digiclops.h"

int
main( int argc, char **argv )
{
   TriclopsError       e;
   TriclopsContext     triclops;
   float               baseline;
   int                 nCameras;
   int                   nrows, ncols;
   float               focalLength;
   TriclopsCameraConfiguration config;
   DigiclopsContext    digiclops;

   // to extract the TriclopsContext, we will briefly open the connected
   // Digiclops.  You can also do this from a file using
   // "triclopsGetDefaultContextFromFile()"
   // open the Digiclops
   digiclopsCreateContext( &digiclops );
   digiclopsInitialize( digiclops, 0 );
   // get the camera module configuration
   digiclopsGetTriclopsContextFromCamera( digiclops, &triclops );
   // close the digiclops
   digiclopsStop( digiclops );
   digiclopsDestroyContext( digiclops );


   printf( "Triclops Version  : %s\n", triclopsVersion() );

   // get the camera configuration (TriCfg_L, TriCfg_BACKWARDS_L, etc)
   e = triclopsGetCameraConfiguration( triclops, &config );
   if ( e != ok )
   {
      printf( "Error getting camera configuration: %s\n",
triclopsErrorToString( e ) );
      exit ( 1 );
   }
   printf( "Config            : %d\n", config );

   // get the number of cameras
   e = triclopsGetNumberOfCameras( triclops, &nCameras );
   if ( e != ok )
   {
      printf( "Error getting number of cameras: %s\n",
triclopsErrorToString( e ) );
      exit ( 1 );
   }
```

```
printf( "Number of Cameras : %d\n", nCameras );

   // get the baseline
   e = triclopsGetBaseline( triclops, &baseline );
   if ( e != ok )
   {
      printf( "Error getting baseline: %s\n", triclopsErrorToString( e )
);
      exit ( 1 );
   }
   printf( "Baseline          : %f\n", baseline );

   // get the focal length
   e = triclopsGetFocalLength( triclops, &focalLength );
   if ( e != ok )
   {
      printf( "Error getting focal length: %s\n", triclopsErrorToString(
e ) );
      exit ( 1 );
   }
   triclopsGetResolution( triclops, &nrows, &ncols );

   printf( "Focal Length      : %f pixels for %d x %d image\n",
      focalLength, ncols, nrows ) ;

   // destroy the context
   triclopsDestroyContext( triclops ) ;

   return 0;
}
```

This example demonstrates how to access Triclops configuration information by printing the Triclops library version, the camera configuration, the number of cameras, the camera baseline, and the camera focal length. The configuration information is retrieved from a Digiclops device attached to the 1394 bus.

This example also shows error handling and the use of the triclopsErrorToString function. Most Triclops functions return an error value, which can be tested.
The example works as follows:

1. The Digiclops™ is opened and initialized
2. The camera module configuration is retrieved.
3. The Digiclops™ is closed.
4. The version of Triclops is obtained and printed to screen.
5. The number of cameras is obtained and printed to screen.
6. The baseline is retrieved and printed to screen.
7. The focal length is retrieved and printed to screen.

Example 7: Generating a 3D image (point cloud) file

```
#include <stdio.h>
#include <stdlib.h>
#include "triclops.h"
#include "digiclops.h"
#include "pnmutils.h"

int
main( int argc, char **argv )
{
   TriclopsInput       stereoData;
   TriclopsInput       colorData;
   TriclopsImage16     depthImage16;
   TriclopsColorImage  colorImage;
   TriclopsContext     triclops;
   DigiclopsContext    digiclops;
   float           x, y, z;
   int                   r, g, b;
   FILE*           pointFile;
   int                   nPoints = 0;
   int                   pixelinc ;
   int                   i, j, k;
   unsigned short *    row ;
   int                   disparity ;

   // open the Digiclops
   digiclopsCreateContext( &digiclops );
   digiclopsInitialize( digiclops, 0 );

   // get the camera module configuration
   digiclopsGetTriclopsContextFromCamera( digiclops, &triclops );

   // set the digiclops to deliver the stereo image and right (color)
image
   digiclopsSetImageTypes( digiclops, STEREO_IMAGE | RIGHT_IMAGE );

   // set the Digiclops resolution
   // use 'HALF' resolution when you need faster throughput, especially
for
   // color images
   // digiclopsSetImageResolution( digiclops, DIGICLOPS_HALF );
   digiclopsSetImageResolution( digiclops, DIGICLOPS_FULL );

   // start grabbing
   digiclopsStart( digiclops );
```

```
   // set up some stereo parameters:
   // set to 320x240 output images
   triclopsSetResolution( triclops, 240, 320 );

   // set disparity range
   triclopsSetDisparity( triclops, 1, 100 ) ;

   triclopsSetStereoMask( triclops, 11 ) ;
   triclopsSetEdgeCorrelation( triclops, 1 ) ;
   triclopsSetEdgeMask( triclops, 11 ) ;

   // lets turn off all validation except subpixel and surface
   // this works quite well
   triclopsSetTextureValidation( triclops, 0 );
   triclopsSetUniquenessValidation( triclops, 0 );

   // turn on sub-pixel interpolation
   triclopsSetSubpixelInterpolation( triclops, 1 ) ;
   // make sure strict subpixel validation is on
   triclopsSetStrictSubpixelValidation( triclops, 1 );

   // turn on surface validation
   triclopsSetSurfaceValidation( triclops, 1 );
   triclopsSetSurfaceValidationSize( triclops, 200 );
   triclopsSetSurfaceValidationDifference( triclops, 0.5 );

   // grab the image set
   digiclopsGrabImage( digiclops );
   // grab the stereo data
   digiclopsExtractTriclopsInput( digiclops, STEREO_IMAGE, &stereoData
);
   // grab the color image data
   // (note: if you are using a B&W Digiclops, this will of course be
   // in B&W)
   digiclopsExtractTriclopsInput( digiclops, RIGHT_IMAGE, &colorData );

   // preprocessing the images
   triclopsPreprocess( triclops, &stereoData ) ;

   // stereo processing
   triclopsStereo( triclops ) ;

   // retrieve the interpolated depth image from the context
   triclopsGetImage16( triclops, TriImg16_DISPARITY,
                  TriCam_REFERENCE, &depthImage16 );
   triclopsRectifyColorImage( triclops, TriCam_REFERENCE,
                        &colorData, &colorImage );
```

```
   // save points to disk
   pointFile = fopen( "out.pts", "w+" );

   // determine the number of pixels spacing per row
   pixelinc    = depthImage16.rowinc/2;
   for ( i = 0, k = 0; i < depthImage16.nrows; i++ )
   {
      row     = depthImage16.data + i * pixelinc;
      for ( j = 0; j < depthImage16.ncols; j++, k++ )
      {
       disparity = row[j];

         // filter invalid points
       if ( disparity < 0xFF00 )
       {
          // convert the 16 bit disparity value to floating point x,y,z
          triclopsRCD16ToXYZ( triclops, i, j, disparity, &x, &y, &z ) ;

            // look at points within a range
          if ( z < 5.0 )
          {
             r = (int) colorImage.red[k];
             g = (int) colorImage.green[k];
             b = (int) colorImage.blue[k];
             fprintf( pointFile, "%f %f %f %d %d %d\n", x, y, z, r, g, b
);
             nPoints++;
          }
       }
      }
   }
   fclose( pointFile );
   printf("Points in file: %d\n", nPoints );

   return 0;

}
```

This example generates a 3-D point cloud image file. This example uses subpixel interpolation for more accurate stereo depth extraction. It also introduces the use of surface validation, a method for removing noise from depth reconstructions. Each valid depth pixel is converted into a 3d XYZ position and output to a point file. In addition, the color of the point is also written out through the use of a rectified color image. The colored point cloud file can be viewed in the PGRView program.

This example generates a 3-dimensional point cloud image file. It works as follows:

1. Digiclops™ is opened and initialized.
2. Digiclops™ is set to deliver a stereo, colour image
3. The resolution for the Digiclops™ is set; for faster throughput (i. e. for colour images), the resolution can be set to HALF.
4. Digiclops™ starts to grab images.
5. The Triclops settings are configured to the following settings:
    - Set Triclops resolution to 320 x 240
    - Set the disparity range to (1, 100)
    - Set the stereo mask to 11
    - Turn the edge correlation on
    - Turn the edge mask on
    - Turn the texture validation off
    - Turn the uniqueness validation off
    - Turn subpixel interpolation and strict subpixel validation on
    - Turn surface validation on and set it to size 200 and differential 0.5
6. Digiclops™ grabs the image set as well as the corresponding stereo and colour image data.
7. The image is preprocessed and then stereo processed
8. The interpolated depth image is retrieved and rectified
9. The points are saved to the file “out.pts”
10. The pixel spacing is determined and invalid points are filtered out.
11. The valid points are displayed to the screen.

Example 8: Performs Stereo Using a 2 Camera Kernel

```
include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <assert.h>

#include "digiclops.h"
#include "triclops.h"
#include "pnmutils.h"


int
main( int argc, char** argv )
{
   TriclopsContext   triclopsContext;
   DigiclopsContext  digiclopsContext;

   TriclopsInput     triclopsInput;

   TriclopsImage16   imageDepth16;

   TriclopsError  te;
   DigiclopsError de;


   // open and initialize the Digiclops.
   de = digiclopsCreateContext( &digiclopsContext );
   assert( de == DIGICLOPS_ok );

   de = digiclopsInitialize( digiclopsContext, 0 );
   assert( de == DIGICLOPS_ok );


   // get the camera module configuration.
   de = digiclopsGetTriclopsContextFromCamera( digiclopsContext,
&triclopsContext );
   assert( de == DIGICLOPS_ok );

   de = digiclopsSetImageTypes( digiclopsContext, ALL_IMAGES );
   assert( de == DIGICLOPS_ok );

   de = digiclopsSetImageResolution( digiclopsContext, DIGICLOPS_FULL );
   assert( de == DIGICLOPS_ok );
```

```
   // start grabbing.
   de = digiclopsStart( digiclopsContext );
   assert( de == DIGICLOPS_ok );

   de = digiclopsGrabImage( digiclopsContext );
   assert( de == DIGICLOPS_ok );

   de = digiclopsExtractTriclopsInput( digiclopsContext, STEREO_IMAGE,
&triclopsInput );
   assert( de == DIGICLOPS_ok );


   // set up some stereo parameters.
   te = triclopsSetEdgeCorrelation( triclopsContext, 1 );
   assert( te == ok );

   // set disparity range.
   te = triclopsSetDisparity( triclopsContext, 1, 16 );
   assert( te == ok );

   te = triclopsSetSubpixelInterpolation( triclopsContext, 1 );
   assert( te == ok );

   // do stereo in 2-camera mode.
   te = triclopsSetCameraConfiguration( triclopsContext,
TriCfg_2CAM_HORIZONTAL );
   assert( te == ok );


   // preprocess and stereo.
   te = triclopsPreprocess( triclopsContext, &triclopsInput );
   assert( te == ok );

   te = triclopsStereo( triclopsContext );
   assert( te == ok );


   // extract the 16-bit disparity image.
   te = triclopsGetImage16( triclopsContext, TriImg16_DISPARITY,
TriCam_REFERENCE, &imageDepth16 );
   assert( te == ok );

   te = triclopsSaveImage16( &imageDepth16, "disparity16-dual.pgm" );
   assert( te == ok );


   // close the digiclops
   digiclopsStop( digiclopsContext );
   digiclopsDestroyContext( digiclopsContext );

   // destroy the triclops context
   te = triclopsDestroyContext( triclopsContext ) ;
   assert( te == ok );

   return 0;
}
```

This example grabs an image from Digiclops and performs stereo on the image using a two camera kernel instead of the three camera kernel. The use of the two camera kernel results in faster processing but sacrifices some of the accuracy of the three camera kernel.

Example 8 works in the following manner:

1. The Digiclops™ camera is opened and initialized.
2. The camera model configuration is obtained.
3. The Digiclops™ camera begins to grab images.
4. Edge-based correlation is turned on.
5. The disparity range is set to (1, 16) with 1 being the minimum and 16 being the maximum.
6. The camera configuration is set to 2-camera mode.
7. The image is preprocessed by unpacking the image, smoothing it, rectifying it, and detecting the edges.
8. Stereo is performed on the image by doing stereo processing, image validation, and subpixel interpolation.
9. The 16 bit stereo image is extracted and saved to “disparity16-dual.pgm”.
10. The context is destroyed and the example program terminates.

Example 9: Using Wide Baseline Stereo Processing

```
/*
 * Example 9 (Widebaseline example):
 *
 * Create a widebaseline context based on the context and 3d
 * transformation
 * information of two separate digiclops camera and then perform stereo
 * processing based on the widebaseline input created from the same pair
 * of cameras.
 *
 */

#include <stdio.h>
#include <math.h>

#include "pnmutils.h"
#include "triclops.h"


const int nrows = 480;
const int ncols = 640;


int main( int argc, char ** argv )
{
   char * configFile1   = 0;
   char * configFile2   = 0;
   char * inputFile1    = 0;
   char * inputFile2    = 0;
   char * transformFile1= 0;
   char * transformFile2= 0;
   TriclopsError  e;
   TriclopsContext      triclops1, triclops2, wideTriclops;
   TriclopsTransform    triclopsTransform;
   TriclopsInput  input1, input2;
   TriclopsInput        wideInput;


   // retrieve arguments from command line
   switch( argc )
   {
      default:
       printf( "testwidebase "
             "<config1> <input1.ppm> <transform1> "
             "<config2> <input2.ppm> <transform2> "
             "\n" );
       return 1;
      case 7:
       configFile1      = argv[1];
       inputFile1 = argv[2];
       transformFile1   = argv[3];
       configFile2      = argv[4];
       inputFile2 = argv[5];
       transformFile2   = argv[6];
```

```
        break;
    }
    // get camera 1's triclops context
    e  = triclopsGetDefaultContextFromFile( &triclops1, configFile1 );
    if ( e != TriclopsErrorOk )
    {
       printf( "Can't read config file '%s'\n", configFile1 );
       return 1;
    }

    // get camera 2's triclops context
    e  = triclopsGetDefaultContextFromFile( &triclops2, configFile2 );
    if ( e != TriclopsErrorOk )
    {
       printf( "Can't read config file '%s'\n", configFile1 );
       return 1;
    }

    if( ( e = triclopsGetTransformFromFile( transformFile1,
&triclopsTransform ) )
           != TriclopsErrorOk )
    {
       printf( "Can't read transform file '%s'\n", transformFile1 );
       return 1;
    }
    if( ( e = triclopsSetTriclopsToWorldTransform( triclops1,
triclopsTransform ) )
           != TriclopsErrorOk )
    {
       printf( "Fail to set camera1 transform.'\n");
       return 1;
    }

    if( ( e = triclopsGetTransformFromFile( transformFile2,
&triclopsTransform ) )
           != TriclopsErrorOk )
    {
       printf( "Can't read transform file '%s'\n", transformFile2 );
       return 1;
    }
    if( ( e = triclopsSetTriclopsToWorldTransform( triclops2,
triclopsTransform ) )
           != TriclopsErrorOk )
    {
       printf( "Fail to set camera2 transform.'\n");
       return 1;
    }


    // create a widebaseline context based on that of camera1 (right
unit) and camera2 (left unit)
    e = triclopsCreateWidebaselineContext( triclops1, triclops2,
&wideTriclops );
```

```
   if ( e != TriclopsErrorOk )
   {
      printf( "Failed to create widebaseline context\n" );
   }



   // set up some of the standard stereo processing flags
   triclopsSetResolution( wideTriclops, nrows, ncols );
   triclopsSetStereoMask( wideTriclops, 13 );
   triclopsSetDisparity( wideTriclops, 1, 200 );
   triclopsSetSubpixelInterpolation( wideTriclops, 0 );

   // Load camera1's images from file
   if( !ppmReadToTriclopsInputRGB( inputFile1, &input1 ) )
   {
      printf( "Can't read input file '%s'\n", inputFile1 );
      return 1;
   }

   // Load camera2's images from file
   if( !ppmReadToTriclopsInputRGB( inputFile2, &input2 ) )
   {
      printf( "Can't read input file '%s'\n", inputFile2 );
      return 1;
   }

   // create the widebaseline input based on the two cameras' input
   e = triclopsCreateWidebaselineInput( input1, input2, &wideInput );
   if( e != TriclopsErrorOk )
   {
      printf( "Failed to create wide baseline input.\n" );
      return 1;
   }

   TriclopsImage image;
   TriclopsImage rightRectified;
   TriclopsImage leftRectified;

   // Perform widebaseline stereo processing
   e = triclopsPreprocess( wideTriclops, &wideInput );
   e = triclopsStereo( wideTriclops );

   // retrieve and save the right raw image
   e = triclopsGetImage( wideTriclops, TriImg_RAW, TriCam_REFERENCE,
&image );
   e = triclopsSaveImage( &image, "right-raw.pgm" );
```

```
   // retrieve and save the left raw image
   // Note : Two camera stereo requires the non-reference camera to be
named TriCam_L_TOP, even
   //     though in this case we are refering to the camera that is on
the left
   e = triclopsGetImage( wideTriclops, TriImg_RAW, TriCam_L_TOP, &image
);
   e = triclopsSaveImage( &image, "left-raw.pgm" );

   // retrieve and save the right rectified image
   e = triclopsGetImage( wideTriclops, TriImg_RECTIFIED,
TriCam_REFERENCE, &rightRectified );
   e = triclopsSaveImage( &rightRectified, "right-rect.pgm" );

   // retrieve and save the left rectified image
   // Note : Two camera stereo requires the non-reference camera to be
named TriCam_L_TOP, even
   //     though in this case we are refering to the camera that is on
the left
   e = triclopsGetImage( wideTriclops, TriImg_RECTIFIED, TriCam_L_TOP,
&leftRectified );
   e = triclopsSaveImage( &leftRectified, "left-rect.pgm" );

   // retrieve and save the resulting disparity image
   e = triclopsGetImage( wideTriclops, TriImg_DISPARITY,
TriCam_REFERENCE, &image );
   e = triclopsSaveImage( &image, "disp3.pgm" );

   // clean up memory allocated for the two camera inputs
   // Note : the wideInput is just a wrapper around the two camera
input,
   //     and hence it does not occupy any dynamically allocated memory
   //     and will not require clean up here
   freeInput( &input1 );
   freeInput( &input2 );

   // clean up memory allocated for the contexts
   e = triclopsDestroyContext( triclops1 );
   e = triclopsDestroyContext( triclops2 );
   e = triclopsDestroyContext( wideTriclops );

   return 0;
```

This example creates a widebaseline context based on the context and 3d transformation information of two separate digiclops camera and then perform stereo processing based on the widebaseline input created from the same pair of cameras.

The example works as follows:

Two separate configuration files are used to create two contexts.

Two tranform files are used to create transforms to correspond to the contexts that were created

Create a wide baseline context using the two contexts.

Set the resolution, stereo mask, disparity, and subpixel interpolation.

Load the images from each camera.

Create a wide baseline input from the two separate contexts

Perform stereo processing on the wide baseline input

Save the following images: right raw image, left raw image, right rectified image, left rectified image, and the resulting disparity image.

Free the allocated memory associated with this example.

Chapter 6: Triclops Application Programming Interface (API) Function Reference

This chapter presents a detailed description of each function in the API.

In this chapter you will find detailed descriptions of all functions and types used by the Triclops library. The first part of the chapter presents the enumerated types, type structures and a summary of functions. The second part of the chapter presents detailed descriptions of the Triclops library functions.

Enumerations

TriclopsCamera

This enumerated type identifies individual cameras given a specific camera configuration.

Declaration

```
enum TriclopsCamera
{
   TriCam_REFERENCE,
   TriCam_RIGHT,
   TriCam_TOP,
   TriCam_LEFT,

   TriCam_L_RIGHT       = TriCam_RIGHT,
   TriCam_L_TOP            = TriCam_TOP,
   TriCam_L_LEFT        = TriCam_LEFT,

   TriCam_COLOR,
   TriCam_L_COLOR = TriCam_COLOR

}
```

Elements

TriCam_COLOR	note: TriCam_COLOR is only for use with the Color Triclops. It is not used for any Digiclops or Bumblebee product TriCam_COLOR may be phased out in future releases of Triclops SDK
TriCam_LEFT	
TriCam_L_COLOR	
TriCam_L_LEFT	
TriCam_L_RIGHT	these are kept here as legacy code for now... should be replaced in user code to not include the configuration specific "_L_" these values may be phased out in future releases of Triclops SDK
TriCam_L_TOP	
TriCam_REFERENCE	
TriCam_RIGHT	
TriCam_TOP	

TriclopsCameraConfiguration

This enumerated type defines the camera configuration. The symbols in the table represent the cameras as they would be seen when the camera module is viewed from the front. This type is either read from the camera or is set manually to indicate 2-Camera mode.

Declaration

```
enum TriclopsCameraConfiguration
{
```

```
   TriCfg_L,
   TriCfg_2CAM_HORIZONTAL,
   TriCfg_2CAM_VERTICAL

}
```

Elements

TriCfg_2CAM_HORIZONTAL	2 Camera Unit or 3 Camera Unit in 2 camera stereo mode.
TriCfg_2CAM_VERTICAL	2 Camera Vertical Unit or 3 Camera Unit in 2 camera vertical mode.
TriCfg_L	3 Camera L-Shaped Triclops or Digiclops.

TriclopsError

All Triclops functions return an error value that indicates whether an error occurred, and if so what error. The following enumerated type lists the kinds of errors that may be returned.

Declaration

```
enum TriclopsError
{
   TriclopsErrorOk = 0,
   ok = 0,
   TriclopsErrorNotImplemented,
   TriclopsErrorInvalidSetting,
   TriclopsErrorInvalidContext,
   TriclopsErrorInvalidCamera,
   TriclopsErrorInvalidROI,
   TriclopsErrorInvalidRequest,
   TriclopsErrorBadOptions,
   TriclopsErrorCorruptConfigFile,
   TriclopsErrorNoConfigFile,
   TriclopsErrorUnknown,
   TriclopsErrorNonMMXCpu,
   TriclopsErrorInvalidParameter,
   TriclopsErrorSurfaceValidationOverflow,
   TriclopsErrorCorruptTransformFile,
   TriclopsErrorSystemError

}
```

Elements

TriclopsErrorBadOptions	Some options are illegal or contradictory.
TriclopsErrorCorruptConfigFile	The configuration file is corrupted, missing mandatory fields, or is for the wrong Triclops version.
TriclopsErrorCorruptTransformFile	An error has occurred during a system call. Check 'errno' to determine the reason. This includes the system running out of memory.
TriclopsErrorInvalidCamera	The specified camera is not valid for this camera configuration.
TriclopsErrorInvalidContext	The TriclopsContext passed in was corrupt or invalid.
TriclopsErrorInvalidParameter	An invalid parameter has been passed.
TriclopsErrorInvalidROI	An impossible Region Of Interest was specified. For

	example, a region of interest with negative height or width.
TriclopsErrorInvalidRequest	Given the current specified options, the request is not valid. For example, requesting the disparity image from a context that has not executed 'triclopsStereo()'.
TriclopsErrorInvalidSetting	User specified input to the function that was not a valid value for this function.
TriclopsErrorNoConfigFile	Can not find the configuration file.
TriclopsErrorNonMMXCpu	A function that requires MMX was called on a non-MMX enabled CPU.
TriclopsErrorNotImplemented	User requested an option that is not yet implemented.
TriclopsErrorOk	Function succeeded.
TriclopsErrorSurfaceValidationOverflow	A call to TriclopsSetSurfaceValidation has caused an overflow.
TriclopsErrorSystemError	Can not find the transform file or its contents is invalid
TriclopsErrorUnknown	An indeterminable error occurred.
ok	Depreciated "ok" Value for downward compatibility.

TriclopsImage16Type

This enumerated type defines the various 16bit image types.

Declaration

```
enum TriclopsImage16Type
{
   TriImg16_DISPARITY = 0

}
```

Elements

TriImg16_DISPARITY	Disparity Image: This is the 16-bit resultant depth image after stereo processing with subpixel on.

TriclopsImageType

The enumerated type TriclopsImageType indicates what kind of image is either being requested or returned. It also indicates the size of each pixel in the image.

Declaration

```
enum TriclopsImageType
{
   TriImg_DISPARITY = 0,
   TriImg_RAW,
   TriImg_RECTIFIED,
   TriImg_EDGE,
   TriImg_PACKED

}
```

Elements

TriImg_DISPARITY	Disparity image: This is the resultant depth image after stereo processing.
TriImg_EDGE	Edge image: A Bandpass filter has been applied to this image. This image has values that range about 128. A value of 128 indicates no edge. Values greater and less than 128 indicate an edge with strength relative to the difference between 128 and the pixel value.
TriImg_PACKED	This image type indicates the image is actually multiple images interleaved. This corresponds to the [B G R U] format that many RGB framegrabbers deliver data in.
TriImg_RAW	Raw unrectified image: This is an image with the aspect ratio that was supplied by the input. There is no correction for lens distortion or camera misalignment.
TriImg_RECTIFIED	Rectified image: This is an image that has been corrected for lens distortion and camera misalignment to fit a pinhole camera model.

TriclopsInputType

This enumerated type identifies the format of the input to the Triclops library. This input has generally either been read from a set of files (for off line processing) or has just been delivered by a frame grabber. There are two formats of input data that are currently being supported. RGB format indicates that the data is supplied with a separate buffer for each R, G and B channel. RGB packed indicates that the 3 channels are interleaved.

Declaration

```
enum TriclopsInputType
{
   TriInp_NONE,
   TriInp_RGB_32BIT_PACKED,
   TriInp_RGB

}
```

Elements

TriInp_NONE	This is used to mark that the input has not yet been set.
TriInp_RGB	An array of separated bands with the following ordering: [R R R....][G G G...][B B B...].
TriInp_RGB_32BIT_PACKED	An array of pixels with color bands interleaved in the following order: [B G R U] [B G R U].

Types

TriclopsBool

Definition for Boolean variables.

Declaration

```
typedef int TriclopsBool
```

TriclopsColorImage

This structure is used for image output from the Triclops system for color images. The structure is the same as TriclopsImage except that the data field is replaced by three color bands; 'red', 'green' and 'blue'. Each band is a complete image for that particular color band. In this case, rowinc is the row increment for each color band.

Declaration

```
struct TriclopsColorImage
{
   int           nrows;
   int           ncols;
   int           rowinc;
   unsigned char*    red;
   unsigned char*    green;
   unsigned char*    blue;

}
```

Elements

blue	The pixel data for blue band of the image.
green	The pixel data for green band of the image.
ncols	The number of columns in the image.
nrows	The number of rows in the image.
red	The pixel data for red band of the image.
rowinc	The row increment of the image.

TriclopsContext

Triclops context is a pointer to an internal structure that maintains all image and bookkeeping information necessary to the Triclops library. It is the container for all parameters

Declaration

```
typedef void*  TriclopsContext
```

TriclopsImage

This structure is used both for image input and output from the Triclops system.

Declaration

```
struct TriclopsImage
{
   int   nrows;
   int   ncols;
   int   rowinc;
   unsigned char* data;

}
```

Elements

data	The area for the pixel data. This must be numRows * numCols bytes large.
ncols	The number of columns in the image.
nrows	The number of rows in the image.
rowinc	This is the pitch, or row increment for the image. Data must be contiguous within each row, and the rows must be equally spaced. Rowinc indicates the number of bytes between the beginning of a row and the beginning of the following row.

TriclopsImage16

This structure is used for image output from the Triclops system for image types that require 16-bits per pixel. This is the format for subpixel interpolated images. The structure is identical to the TriclopsImage structure except that the data contains unsigned shorts rather than unsigned chars. Rowinc is still the number of bytes between the beginning of a row and the beginning of the following row (NOT number of pixels).

Declaration

```
struct TriclopsImage16
{
   int           nrows;
   int           ncols;
   int           rowinc;
   unsigned short*   data;

}
```

Elements

data	The pixel data of the image.
ncols	The number of columns in the image.
nrows	The number of rows in the image.
rowinc	The number row increment of the image.

TriclopsImage3d

This structure defines the format of an image consisting of 3d points.

Declaration

```
struct TriclopsImage3d
{
   int nrows;
   int ncols;
   int rowinc;
   TriclopsPoint3d*     points;

}
```

Elements

ncols	The number of columns in the image.
nrows	The number of rows in the image.
points	The area for the pixel data. This must be numRows * numCols bytes large.
rowinc	The number of bytes between the beginning of a row and the beginning of the following row

TriclopsInput

TriclopsInput structure contains image input to the Triclops library. The library accepts two formats: 32-bit packed, and RGB separate buffer inputs. Field u contains a pointer to one of the two typed structures that contain the image data. The inputType field indicates which type of input is actually present in the structure.

Declaration

```
struct TriclopsInput
{
   TriclopsInputType inputType;
   TriclopsTimestamp timeStamp;
   int   nrows;
   int   ncols;
   int   rowinc;

   union
   {
      TriclopsInputRGB         rgb;
      TriclopsInputRGB32BitPacked   rgb32BitPacked;
   } u;

}
```

Elements

inputType	The input type indicating packed or unpacked data.
ncols	The number of columns in the input images.
nrows	The number of rows in the input images.
rowinc	The row increment of the input images.
timeStamp	The timestamp on the image data (generated by the camera.)
u	The actual image data is either packed or unpacked and can be accessed with either TriclopsinputRGB or TriclopsInputRGB32BitPacked.

TriclopsInputRGB

This structure consists of three separate buffers, each containing nrows * ncols pixels for each of the RGB bands.

Declaration

```
struct TriclopsInputRGB
{
   void*    red;
   void*    green;
   void*    blue;

}
```

Elements

blue	The blue band.
green	The green band.
red	The red band.

TriclopsInputRGB32BitPacked

This structure contains RGB packed data.

Declaration

```
struct TriclopsInputRGB32BitPacked
{
   void*    data;

}
```

Elements

data	a pointer to an array of nrows*ncols*4 pixels. The pixels are organized in the following fashion [RGBU][RGBU] ...

TriclopsPackedColorImage

This structure defines the format of a 32bit packed color image.

Declaration

```
struct TriclopsPackedColorImage
{
   int   nrows;
   int   ncols;
   int   rowinc;
   TriclopsPackedColorPixel*   data;

}
```

Elements

data	A pointer to the pixel data.
ncols	The number of columns in the image.
nrows	The number of rows in the image.

rowinc	The number of bytes in each row of the image.

TriclopsPackedColorPixel

This structure defines the format for a 32bit color pixel.

Declaration

```
struct TriclopsPackedColorPixel
{
   unsigned char     value[4];

}
```

Elements

value	The 32 bit pixel data.

TriclopsPoint3d

This structure defines a single 3d pixel. It is only used in the TriclopsImage3d structure.

Declaration

```
struct TriclopsPoint3d
{
   float point[3];

}
```

Elements

point	The 3 values for the (x,y,z) point in order x = 0, y = 1, z = 2.

TriclopsROI

This structure describes a Region Of Interest for selective processing of the input images.

Declaration

```
struct TriclopsROI
{
   int   row;
   int   col;
   int   nrows;
   int   ncols;

}
```

Elements

col	The column value for the upper-left corner of the region of interest.
ncols	The width of the region of interest.
nrows	The height of the region of interest.

row	The row value for the upper-left corner of the region of interest.

TriclopsTimestamp

This structure describes the format by which time is represented in the Triclops Stereo Vision SDK.

Declaration

```
struct TriclopsTimestamp
{
   long sec;
   long u_sec;

}
```

Elements

sec	The number of seconds since the epoch.
u_sec	The number of microseconds within the second.

TriclopsTransform

Declaration

```
typedef struct
{
   double   matrix[4][4];

} TriclopsTransform
```

Debugging and Error Reporting

triclopsErrorToString

Converts a Triclops error into a meaningful string.

This function returns a string that describes the given TriclopsError. This allows error reporting software to report meaningful error messages to the user.

Declaration

```
char*
triclopsErrorToString( TriclopsError error )
```

See Also

TriclopsError

triclopsGetDebug

Retrieves the state of the debug message printing of the stereo kernel.

Declaration

```
TriclopsError
triclopsGetDebug( const TriclopsContext context,
                  TriclopsBool* on )
```

Parameters

context	The context.
on	A pointer that contains the current value of the debug flag.

See Also

triclopsSetDebug()

triclopsSetDebug

Controls the debug message printing of the stereo kernel.

Turning debug on enables the Triclops library to print detailed debugging messages to stderr.

Declaration

```
TriclopsError
triclopsSetDebug( TriclopsContext context,
                  TriclopsBool on )
```

Parameters

context	The context.
on	Flag turning debug on or off.

See Also

triclopsGetDebug()

TriclopsContext Manipulation

triclopsCopyContext

Copies the context parameters and images. Care must be taken when using triclopsCopyContext() as contexts will share image buffers.

This function creates a copy of the input context. This allows the user to have two contexts with the same options and also to share the same image buffers. Care must be taken with this function. The ensuing contexts will share image buffers. This allows them to share some of the previous processing, however, any changes in resolution or calls to set image buffers will create new buffers to be used. These buffers will no longer be shared and thus programs may not operate as expected at this time. The recommendation is to use this operation when one wants to run stereo with different stereo options. After the preprocessing step, the context can be copied and different stereo options set before the call to triclopsStereo().

Note: This only copies the options, not the images

Declaration

```
TriclopsError
triclopsCopyContext( const TriclopsContext contextIn,
                     TriclopsContext* contextOut )
```

Parameters

contextIn	A context that has already been constructed.
contextOut	A copy of the input context.

See Also

triclopsGetDefaultContext(), triclopsDestroyContext()

triclopsDestroyContext

Destroys the given context.

This function destroys a context and frees all associated memory.

Declaration

```
TriclopsError
triclopsDestroyContext( TriclopsContext context )
```

triclopsGetDefaultContextFromFile

Setup the initial context with data obtained from a file.

This function reads in the default option list and the camera calibration data from a file. If the specified file is not found, contains invalid fields, or is for the wrong product, the value of CorruptConfigFile will be returned.

Declaration

```
TriclopsError
triclopsGetDefaultContextFromFile( TriclopsContext* defaultContext,
                                   char* filename )
```

See Also

TriclopsGetDefaultContextFromFile(), triclopsWriteDefaultContextToFile()

triclopsWriteDefaultContextToFile

This writes the default calibration file from the TriclopsContext to a file.

This function writes the default context parameters and calibration data to a file. It does not write the current configuration, rather the original configuration that would obtained when getting the default from either a file or a device.

Declaration

```
TriclopsError
triclopsWriteDefaultContextToFile( TriclopsContext* context,
                                   char* filename )
```

Parameters

context	The TriclopsContext
filename	The name of the file to be written

See Also

TriclopsGetDefaultContextFromFile()

Validation Support

triclopsGetStrictSubpixelValidation

Retrieves the value that appears in the disparity image for pixels that fail uniqueness validation.

Declaration

```
TriclopsError
triclopsGetStrictSubpixelValidation( TriclopsContext context,
                                     TriclopsBool* on )
```

Parameters

context	The context.
on	A pointer to a Boolean that will contain the current setting.

See Also

triclopsSetStrictSubpixelValidation(), triclopsSetSubpixelInterpolation(), triclopsGetSubpixelInterpolation()

triclopsGetSubpixelValidationMapping

Retrieves the value that appears in the disparity image for pixels that fail subpixel validation.

Declaration

```
TriclopsError
triclopsGetSubpixelValidationMapping( const TriclopsContext context,
                                      unsigned char* value )
```

Parameters

context	TriclopsContext for the operation.
value	A pointer that will hold the subpixel validation mapping value.

See Also

triclopsSetSubpixelValidationMapping

triclopsGetSurfaceValidation

Gets the current surface validation setting.

This function is used to obtain the current setting of surface validation.

Declaration

```
TriclopsError
triclopsGetSurfaceValidation( const TriclopsContext context,
                              TriclopsBool* on )
```

Parameters

context	TriclopsContext for the operation.
on	A pointer for the returned value of the current setting of the surface validation flag.

See Also

triclopsSetSurfaceValidation ()

triclopsGetSurfaceValidationDifference

Returns the maximum disparity difference between two adjacent pixels that will still allow the two pixels to be considered part of the same surface.

Declaration

```
TriclopsError
triclopsGetSurfaceValidationDifference(
               const TriclopsContext context,
               float* diff )
```

Parameters

context	TriclopsContext for the operation.
diff	A pointer that returns the current value of the surface validation difference parameter.

triclopsGetSurfaceValidationMapping

Retrieves the current surface validation mapping.

This function returns the current setting for the surface validation parameter.

Declaration

```
TriclopsError
triclopsGetSurfaceValidationMapping( const TriclopsContext context,
                                     unsigned char* value )
```

Parameters

context	TriclopsContext for the operation.
value	A pointer that will return the current value of the surface validation mapping parameter.

triclopsGetSurfaceValidationSize

Retrieves the current validation size.

This function is used to extract the surface size parameter for surface validation.

Declaration

```
TriclopsError
triclopsGetSurfaceValidationSize( const TriclopsContext context,
                                  int* size )
```

Parameters

context	TriclopsContext for the operation.
size	A pointer that returns the current value of the size parameter.

See Also

triclopsSetSurfaceValidationSize

triclopsGetTextureValidation

Retrieves the state of the texture validation flag.

Declaration

```
TriclopsError
triclopsGetTextureValidation( const TriclopsContext context,
                              TriclopsBool* on )
```

See Also

TriclopsSetTextureValidation()

triclopsGetTextureValidationMapping

Gets the value that appears in the disparity image for pixels that fail texture validation.

Declaration

```
TriclopsError
triclopsGetTextureValidationMapping( const TriclopsContext context,
                                     unsigned char* value )
```

Parameters

context	The TriclopsContext to get the texture validation mapping from.
value	The current texture validation mapping setting.

triclopsGetTextureValidationThreshold

Retrieves the texture validation threshold.

Declaration

```
TriclopsError
triclopsGetTextureValidationThreshold( const TriclopsContext context,
                                       float* value )
```

Parameters

context	The context.
value	The location for the retrieved threshold.

See Also

TriclopsSetTextureValidationThreshold()

triclopsGetUniquenessValidation

Retrieves the state of the uniqueness validation

Declaration

```
TriclopsError
triclopsGetUniquenessValidation( const TriclopsContext context,
                                 TriclopsBool* on )
```

Parameters

context	The context.
on	A pointer to a Boolean.

See Also

TriclopsSetUniquenessValidation()

triclopsGetUniquenessValidationMapping

Retrieves the value that appears in the disparity image for pixels that fail uniqueness validation.

Declaration

```
TriclopsError
triclopsGetUniquenessValidationMapping(
                const TriclopsContext context,
                unsigned char* value )
```

Parameters

context	The context.
value	A pointer to an unsigned char that will contain the current value of the mapping.

See Also

TriclopsSetUniquenessValidationMapping()

triclopsGetUniquenessValidationThreshold

Retrieves the uniqueness validation threshold.

Declaration

```
TriclopsError
triclopsGetUniquenessValidationThreshold(
                const TriclopsContext context,
                float* value )
```

Parameters

context	The context.
value	A pointer to a float indicating the current threshold.

See Also

TriclopsSetUniquenessThreshold()

triclopsSetStrictSubpixelValidation

Sets the state of the subpixel validation setting.

The strict subpixel validation option is used when the subpixel interpolation flag is turned on. Strict subpixel validation enables a more restrictive validation method for subpixel validation. With strict subpixel validation on, there will be less data in the output image, but it will be more reliable.

Declaration

```
TriclopsError
triclopsSetStrictSubpixelValidation( TriclopsContext context,
                                     TriclopsBool on )
```

Parameters

context	The context
on	A Boolean value indicating whether validation is on or off.

See Also

triclopsGetStrictSubpixelValidation(), triclopsSetSubpixelInterpolation(), triclopsGetSubpixelInterpolation()

triclopsSetSubpixelValidationMapping

Sets the value that appears in the disparity image for pixels that fail subpixel validation.

Strict subpixel validation verifies that the disparity values contribute to the subpixel interpolation make sense. By setting the mapping value to 0x??, an invalid pixel that failed this validation step will be marked 0xFF??.

Declaration

```
TriclopsError
triclopsSetSubpixelValidationMapping( TriclopsContext context,
                                      unsigned char value )
```

Parameters

context	TriclopsContext for the operation.
value	The new subpixel validation mapping value. The range is between 0 and 255.

triclopsSetSurfaceValidation

Enables or disables surface validation.

This function is used to enable or disable surface validation.

Declaration

```
TriclopsError
triclopsSetSurfaceValidation( TriclopsContext context,
                              TriclopsBool on )
```

Parameters

context	TriclopsContext for the operation.
on	A Boolean flag indicating if surface validation should be enabled or disabled.

triclopsSetSurfaceValidationDifference

Set the maximum disparity difference between two adjacent pixels that will still allow the two pixels to be considered part of the same surface.

Declaration

```
TriclopsError
triclopsSetSurfaceValidationDifference(
               TriclopsContext context,
               float diff )
```

Parameters

context	TriclopsContext for the operation.
diff	The maximum disparity difference between two adjacent pixels that will still allow the two pixels to be considered part of the same surface.

triclopsSetSurfaceValidationMapping

Sets the current surface validation mapping.

Surface validation is a noise rejection method. By setting the mapping value to 0x??, an invalid pixel that failed this validation step will be marked 0xFF??.

Declaration

```
TriclopsError
triclopsSetSurfaceValidationMapping( TriclopsContext context,
                                     unsigned char value )
```

Parameters

context	TriclopsContext for the operation.
value	Mapping value for pixels that fail surface validation. The range is between 0 and 255.

triclopsSetSurfaceValidationSize

Sets the minimum number of pixels a surface can cover and still be considered valid.

This function is used to set the minimum number of pixels a surface can cover and still be considered valid. The larger the number is, the fewer surfaces will be accepted. The lower the number is, the more surfaces will be accepted. Common parameter values range from 100 to 500, depending on image resolution.

Declaration

```
TriclopsError
triclopsSetSurfaceValidationSize( TriclopsContext context,
                                  int size )
```

Parameters

context	TriclopsContext for the operation.
size	The minimum number of pixels a surface can cover and still be considered valid.

triclopsSetTextureValidation

Turns texture validation on or off.

Sets the context texture validation flag. When set to true, texture-based validation is enabled. Pixels that do not pass the validation test will be marked with the texture validation mapping value in the disparity image.

Declaration

```
TriclopsError
triclopsSetTextureValidation( const TriclopsContext context,
                              TriclopsBool on )
```

Parameters

context	The context.
on	Texture validation flag.

See Also

triclopsGetTextureValidation()

triclopsSetTextureValidationMapping

Sets the value that appears in the disparity image for pixels that fail texture validation.

Declaration

```
TriclopsError
triclopsSetTextureValidationMapping( TriclopsContext context,
                                     unsigned char value )
```

Parameters

context	The TriclopsContext to set the texture validation mapping for.
value	The new value to map invalid pixels to. The range is from 0 to 255.

triclopsSetTextureValidationThreshold

Sets the texture validation threshold.

Sets the texture validation threshold. This threshold allows one to tune the texture-based rejection of pixels. Values range from 0.0 (no rejection) to 128.0 (complete rejection) but good operating range is between 0.0 and 2.0.

Declaration

```
TriclopsError
triclopsSetTextureValidationThreshold( TriclopsContext context,
                                       float value )
```

Parameters

context	The context
value	The new texture validation threshold.

See Also

TriclopsGetTextureValidationThreshold()

triclopsSetUniquenessValidation

Turns uniqueness validation on or off.

Turns uniqueness validation on or off. Uniqueness validation verifies that the match of a given pixel to its corresponding pixels in the top and left images is unique enough to be considered the definite correct match. If pixels at other disparities have almost as good matching to the given pixel as the best disparity, the pixel can be rejected as an unconfident match.

Declaration

```
TriclopsError
triclopsSetUniquenessValidation( TriclopsContext context,
                                 TriclopsBool on )
```

Parameters

context	The context.
on	A Boolean value indicating whether validation is on or off.

See Also

TriclopsGetUniquenessValidation()

triclopsSetUniquenessValidationMapping

Sets the value that appears in the disparity image for pixels that fail uniqueness validation.

Sets the unique matching validation mapping value. This is the value that is assigned to pixels in the disparity image that fail the uniqueness validation test. The range is between 0 and 255.

Declaration

```
TriclopsError
triclopsSetUniquenessValidationMapping(
                TriclopsContext context,
                unsigned char value )
```

Parameters

context	The context.
value	The value to which failing pixels are mapped.

See Also

TriclopsGetUniquenessValidationMapping()

triclopsSetUniquenessValidationThreshold

Sets the uniqueness validation threshold.

This function sets the uniqueness matching criteria threshold. This value can range from 0.0 to 10.0. The larger the number, the less rejection occurs. Common operating ranges are between 0.5 and 3.0.

Declaration

```
TriclopsError
triclopsSetUniquenessValidationThreshold(
                TriclopsContext context,
                float value )
```

Parameters

context	The context.
value	A float indicating the new threshold.

See Also

TriclopsGetUniquenessValidationThreshold()

General

triclopsGetImage

Retrieves a specified type of image associated with the specified camera.

This function supplies the data to a TriclopsImage. The user is responsible for allocating the TriclopsImage structure. The data value of the structure will point to memory within the stereo kernel working space. Therefore, if a permanent copy of this image is desired, the user must copy it out.

Declaration

```
TriclopsError
triclopsGetImage( const TriclopsContext context,
                  TriclopsImageType imageType,
                  TriclopsCamera camera,
                  TriclopsImage* image )
```

Parameters

context	The context.
imageType	The image type requested.
camera	The camera that generated the requested image.
image	A TriclopsImage with the image data.

See Also

triclopsSetImageBuffer()

triclopsGetImage16

Retrieves a specified 16-bit image associated with the specified camera.

This function performs the same action as triclopsGetImage(), except that it retrieves a 16-bit image type. Currently the only supported 16-bit image is the resultant disparity image from subpixel interpolation.

Declaration

```
TriclopsError
triclopsGetImage16( const TriclopsContext context,
                    TriclopsImage16Type imageType,
                    TriclopsCamera camera,
                    TriclopsImage16* image )
```

Parameters

context	The context.
imageType	The image type requested.
camera	The camera that generated the requested image.

image	A TriclospImage16 with the image data.

See Also

triclopsGetImage()

triclopsGetROIs

Retrieves a handle to an array of Regions Of Interest.

This function returns a pointer to an array of region of interest (ROI) structures. The user may set requested region of interest boundaries in this array. The Regions Of Interest that will be valid must begin with the 0th element in the array and continue in a contiguous manner. After having set the Regions Of Interest, the user must call triclopsSetNumberOfROIs() to indicate to the stereo kernel how many ROIs he/she wishes processed. Currently, ROIs are only applied during the stereo processing, not during preprocessing.

Declaration

```
TriclopsError
triclopsGetROIs( TriclopsContext context,
                 TriclopsROI** rois,
                 int* maxROIs )
```

Parameters

context	The context.
rois	A pointer to a 1D array of ROI structures.
maxROIs	The maximum number of ROIs allowed.

See Also

triclopsSetNumberOfROIs(), triclopsStereo(), TriclopsROI

triclopsGetResolution

Retrieves the resolution of the resultant images. This includes rectified, disparity and edge images.

This returns the current resolution for output images of the given context.

Declaration

```
TriclopsError
triclopsGetResolution( const TriclopsContext context,
                       int* nrows,
                       int* ncols )
```

Parameters

context	The context.
nrows	Number of rows in the output images.

ncols	Number of columns in the output images.

See Also

triclopsSetResolution()

triclopsSaveColorImage

Saves an image to the specified filename. The file format currently supported is PGM format.

This function saves the input image to the requested file. Currently, this function will not detect if the file could not be opened, and will always return successful. Color images are saved in PPM format.

Declaration

```
TriclopsError
triclopsSaveColorImage( TriclopsColorImage* image,
                        char* filename )
```

Parameters

image	The TriclopsColorImage to be saved.
filename	The file name in which to save the image.

See Also

triclopsSaveImage()

triclopsSaveImage

Saves an image to the specified filename. The file format currently supported is PGM format.

This function saves the input image to the requested file. Currently, this function will not detect if the file could not be opened, and will always show successful return.

Declaration

```
TriclopsError
triclopsSaveImage( TriclopsImage* image,
                   char* filename )
```

Parameters

image	The TriclopsImage to be saved.
filename	The file name in which to save the image.

triclopsSaveImage16

Saves an image to the specified filename. The file format currently supported is PGM format.

This function saves the input image to the requested file. Currently, this function will not detect if the file could not be opened, and will always return successful.

Declaration

```
TriclopsError
triclopsSaveImage16( TriclopsImage16* image,
                     char* filename )
```

Parameters

image	The TriclopsImage16 to be saved.
filename	The file name in which to save the image.

See Also

triclopsSaveImage()

triclopsSavePackedColorImage

Allows the user to save a packed color image to the given file.

Declaration

```
TriclopsError
triclopsSavePackedColorImage( TriclopsPackedColorImage* image,
                              char* filename )
```

Parameters

image	A pointer to the buffer containing the image.
filename	The name of the file to be written to.

triclopsSetColorImageBuffer

Allows the user to set separate buffers to which individual bands of the processed color image are written to. In this case, the "processed" image means the rectified image that is rectified when triclopsRectifyColorImage() is called.

Declaration

```
TriclopsError
triclopsSetColorImageBuffer( TriclopsContext context,
                             TriclopsCamera nCamera,
                             unsigned char* red,
                             unsigned char* green,
                             unsigned char* blue )
```

Parameters

context	The TriclopsContext to set the buffer for.
nCamera	The camera buffer to set.
red	A pointer to the red buffer.
green	A pointer to the green buffer.

blue	A pointer to the blue buffer.

See Also

triclopsRectifyColorImage()

triclopsSetImage16Buffer

This function allows the user to set the location to which 16bit (generally subpixel) depth images are written to once they are processed.

Declaration

```
TriclopsError
triclopsSetImage16Buffer( TriclopsContext context,
                          unsigned short* buffer,
                          TriclopsImage16Type imageType,
                          TriclopsCamera camera )
```

Parameters

context	The TriclopsContext to set the buffer in.
buffer	A pointer to the buffer.
imageType	The type of image to be written to the buffer.
camera	The camera to write from.

triclopsSetImageBuffer

Sets the internal image buffer for the specified camera and image type to be the buffer supplied by the user.

This function allows the user to specify directly what memory he/she wishes the output images to be deposited into. This memory will be used by the stereo kernel as working space. This has the advantage of saving a copy for tasks such as displaying to the screen. The user may simply set the output image buffer to his/her display buffer. However, since this memory will be used by the stereo kernel as working space, the contents of the buffer may change with each call of triclopsPreprocess() or triclopsStereo(). If the results are to be saved, it is the user's responsibility to do so. In addition, the user is responsible to allocate sufficient memory for the buffer, and to de-allocate the buffer after it is no longer needed. Before de-allocating the buffer, the user should call triclopsUnsetImageBuffer(). If the user requests an invalid image, such as the disparity image, when the context stereo flag is set to false, or the edge image when edge correlation is set to false, an error of invalid request will be returned. Disparity images are always associated with the reference camera.

Declaration

```
TriclopsError
triclopsSetImageBuffer( TriclopsContext context,
                        unsigned char* buffer,
                        TriclopsImageType imageType,
                        TriclopsCamera camera )
```

Parameters

context	The context.
buffer	A user allocated buffer of sufficient size.
imageType	The image type.
camera	The camera.

See Also

triclopsGetImage(), triclopsUnsetImageBuffer()

triclopsSetNumberOfROIs

Sets the number of Regions Of Interest the user currently wants active.

This function indicates to the stereo kernel how many ROIs in the ROI array will be processed. Before calling this function, the user must first call triclopsGetROIs() and set up the ROIs he/she intends to use. If nrois is set to 0, the entire image will be processed. This is the default setting.

Declaration

```
TriclopsError
triclopsSetNumberOfROIs( TriclopsContext context,
                         int nrois )
```

Parameters

context	The context.
nrois	Number of ROIs.

See Also

triclopsGetROIs()

triclopsSetPackedColorImageBuffer

Allows the user to set the buffer to which the processed color image is written to.

Declaration

```
TriclopsError
triclopsSetPackedColorImageBuffer( TriclopsContext context,
                                   TriclopsCamera nCamera,
                                   TriclopsPackedColorPixel* buffer )
```

Parameters

context	The TriclopsContext to set the buffer for.
nCamera	The camera buffer to set.

buffer	A pointer to a buffer of TriclopsPackedColorPixels.

triclopsSetResolution

Sets the resolution of the resultant images. This includes rectified, disparity and edge images.

This function sets the desired resolution of the output images. These images include the rectified, disparity and edge images. This resolution must maintain the 640x480 columns to rows ratio. If the user wishes to have an image of a different aspect ratio, he/she must use Regions Of Interest to control the size of the image that is being processed.

Declaration

```
TriclopsError
triclopsSetResolution( TriclopsContext context,
                       int nrows,
                       int ncols )
```

Parameters

context	The context.
nrows	Number of rows in the output images.
ncols	Number of columns in the output images.

See Also

triclopsGetResolution(), triclopsGetROIs()

triclopsSetResolutionAndPrepare

Sets the resolution of the resultant images. This includes rectified, disparity and edge images.

This function sets the desired resolution of the output images and also immediately constructs the rectification tables. For large images, the construction of the rectification can take a while. This function allows you to control when the construction takes place, otherwise it will occur during the first call to triclopsPreprocess(). The resolution of the input images must be specified at this time, as this is necessary for the construction of the tables. The output images include the rectified, disparity and edge images. This requested resolution must maintain the 640x480 columns to rows ratio. If the user wishes to have an image of a different aspect ratio, he/she must use Regions Of Interest to control the size of the image that is being processed.

Declaration

```
TriclopsError
triclopsSetResolutionAndPrepare( TriclopsContext context,
                                 int nrows,
                                 int ncols,
                                 int nInputRows,
                                 int nInputCols )
```

Parameters

context	The context.

nrows	Number of rows in the output images.
ncols	Number of columns in the output images.
nInputRows	Number of rows in the input images.
nInputCols	Number of columns in the input images.

See Also

TriclopsGetResolution(), triclopsSetResolution(), triclopsSetRectify()

triclopsUnsetColorImageBuffer

This releases the user specified internal color image buffer for the specified camera. The next time this buffer is required by the system, it will allocate a new one for internal use.

If the user has already called triclopsSetColorImageBuffer for a particular camera, the stereo kernel will be using that buffer when rectifying a color image. If the user no longer wants to have the supplied buffer used by the stereo kernel, he/she may use this function to inform the stereo kernel that it is no longer available. A new buffer will be created the next time it is required by the stereo kernel.

Declaration

```
TriclopsError
triclopsUnsetColorImageBuffer( TriclopsContext context,
                               TriclopsCamera camera )
```

Parameters

context	The context.
camera	The camera.

See Also

triclopsRectifyColorImage(), triclopsSetColorImageBuffer()

triclopsUnsetImage16Buffer

This releases the user specified internal image buffer for the specified camera and image type. The next time this buffer is required by the system, it will allocate a new one for internal use.

If the user has already called triclopsSetImage16Buffer for a particular camera and image type, the stereo kernel will be using that buffer for internal processing. If the user no longer wants to have the supplied buffer used by the stereo kernel, he/she may use this function to inform the stereo kernel that it is no longer available. A new buffer will be created the next time it is required by the stereo kernel.

Declaration

```
TriclopsError
triclopsUnsetImage16Buffer( TriclopsContext context,
                            TriclopsImageType imageType,
                            TriclopsCamera camera )
```

Parameters

context	The context.
imageType	The image type.
camera	The camera.

See Also

triclopsGetImage(), triclopsSetImage16Buffer()

triclopsUnsetImageBuffer

This releases the user specified internal image buffer for the specified camera and image type. The next time this buffer is required by the system, it will allocate a new one for internal use.

If the user has already called triclopsSetImageBuffer for a particular camera and image type, the stereo kernel will be using that buffer for internal processing. If the user no longer wants to have the supplied buffer used by the stereo kernel, he/she may use this function to inform the stereo kernel that it is no longer available. A new buffer will be created the next time it is required by the stereo kernel.

Declaration

```
TriclopsError
triclopsUnsetImageBuffer( TriclopsContext context,
                          TriclopsImageType imageType,
                          TriclopsCamera camera )
```

Parameters

context	The context.
imageType	The image type.
camera	The camera.

See Also

TriclopsGetImage(), triclopsSetImageBuffer()

triclopsUnsetPackedColorImageBuffer

This releases the user specified internal color image buffer for the specified camera. The next time this buffer is required by the system, it will allocate a new one for internal use.

If the user has already called triclopsSetPackedColorImageBuffer for a particular camera, the stereo kernel will be using that buffer when rectifying a color image. If the user no longer wants to have the supplied buffer used by the stereo kernel, he/she may use this function to inform the stereo kernel that it is no longer available. A new buffer will be created the next time it is required by the stereo kernel.

Declaration

```
TriclopsError
triclopsUnsetPackedColorImageBuffer( TriclopsContext context,
                                     TriclopsCamera camera )
```

Parameters

context	The context.
camera	The camera.

See Also

triclopsSetPackedColorImageBuffer(), triclopsRectifyPackedColorImage()

triclopsVersion

Returns a string with the Triclops library version.

This function returns internally-handled memory. The caller should not free the returned pointer.

Declaration

```
const char*
triclopsVersion()
```

Remarks

Input:

Returns: char* - A string containing the version information for the context.

Preprocessing

triclopsGetEdgeCorrelation

Retrieves the state of the edge based correlation flag.

Declaration

```
TriclopsError
triclopsGetEdgeCorrelation( const TriclopsContext context,
                            TriclopsBool* on )
```

Parameters

context	The context.
on	A pointer to a Boolean that will contain the current setting.

See Also

triclopsSetEdgeCorrelation()

triclopsGetEdgeMask

Retrieves the edge detection mask size.

Declaration

```
TriclopsError
triclopsGetEdgeMask( const TriclopsContext context,
                     int* masksize )
```

Parameters

context	The context.
masksize	A pointer to an integer that will contain the current mask size.

See Also

triclopsSetEdgeMask()

triclopsGetLowpass

Retrieves the state of the low-pass filtering feature.

Declaration

```
TriclopsError
triclopsGetLowpass( const TriclopsContext context,
                    TriclopsBool* on )
```

Parameters

context	The context.
on	A pointer to a Boolean variable that will store the current setting.

See Also

triclopsSetLowpass()

triclopsGetRectify

Retrieves the state of the rectification feature.

Declaration

```
TriclopsError
triclopsGetRectify( const TriclopsContext context,
                    TriclopsBool* on )
```

Parameters

context	The context.
on	A pointer to a Boolean that will store the current setting.

See Also

triclopsSetRectify()

triclopsPreprocess

Does image unpacking, smoothing, rectification and edge detection, as specified by parameters.

This function does all necessary preprocessing on the input data. It unpacks the data, which strips individual channels from 32-bit packed data and puts them into 3 TriImg_RAW channels. It applies a low-pass filter on these channels if requested, and corrects for lens distortion and camera misalignment, saving these images into TriImg_RECTIFIED. Finally it performs 2nd derivative Gaussian edge processing on the rectified images and saves them into TriImg_EDGE internal images.

Declaration

```
TriclopsError
triclopsPreprocess( TriclopsContext context,
                    TriclopsInput* input )
```

Parameters

context	The context.
input	The image to be processed.

See Also

triclopsStereo(), triclopsSetResolution(), triclopsSetEdgeCorrelation(), triclopsSetRectify(), triclopsSetLowpass()

triclopsSetEdgeCorrelation

Turns edge based correlation on or off.

Edge based correlation is required for texture and uniqueness validation to be used.

Declaration

```
TriclopsError
triclopsSetEdgeCorrelation( TriclopsContext context,
                            TriclopsBool on )
```

Parameters

context	The context.
on	A Boolean value indicating whether correlation should be turned on or off.

See Also

triclopsGetEdgeCorrelation(), triclopsSetTextureValidation(), triclopsSetUniquenessValidation()

triclopsSetEdgeMask

Sets the edge detection mask size.

Declaration

```
TriclopsError
triclopsSetEdgeMask( TriclopsContext context,
                     int masksize )
```

Parameters

context	The context.
masksize	The new mask size, which is valid between 3 and 11.

See Also

triclopsGetEdgeMask()

triclopsSetLowpass

Turns low-pass filtering before rectification on or off.

Declaration

```
TriclopsError
triclopsSetLowpass( TriclopsContext context,
                    TriclopsBool on )
```

Parameters

context	The context.
on	A Boolean value indicating whether it should be turned on or off.

See Also

triclopsGetLowpass()

triclopsSetRectify

Turns rectification on or off.

Declaration

```
TriclopsError
triclopsSetRectify( TriclopsContext context,
                    TriclopsBool on )
```

Parameters

context	The context.
on	A Boolean indicating whether rectification should be turned on or off.

See Also

triclopsGetRectify()

Stereo

triclopsGetDisparity

Retrieves the disparity range from the given context.

Declaration

```
TriclopsError
triclopsGetDisparity( const TriclopsContext context,
                      int* minDisparity,
                      int* maxDisparity )
```

Parameters

context	The context
minDisparity	A pointer to an integer that will store the current value.
maxDisparity	A pointer to an integer that will store the current value.

triclopsGetDisparityMapping

Retrieves the disparity range from the given context.

Declaration

```
TriclopsError
triclopsGetDisparityMapping( const TriclopsContext context,
                             unsigned char* minDisparity,
                             unsigned char* maxDisparity )
```

Parameters

context	The context.
minDisparity	The disparity range in the output disparity image minimum.
maxDisparity	The disparity range in the output disparity image maximum.

See Also

triclopsSetDisparityMapping(), triclopsSetDisparity()

triclopsGetDisparityMappingOn

Retrieves the current setting.

This function is used to extract the current disparity mapping setting.

Declaration

```
TriclopsError
triclopsGetDisparityMappingOn( TriclopsContext context,
                               TriclopsBool* on )
```

Parameters

context	TriclopsContext for the operation.
on	A pointer that will contain the current value of this flag.

See Also

triclopsSetDisparityMappingOn.

triclopsGetDoStereo

Retrieves the state of the stereo processing.

Declaration

```
TriclopsError
triclopsGetDoStereo( const TriclopsContext context,
                     TriclopsBool* on )
```

Parameters

context	The context.
on	A pointer to a Boolean that will store the current setting.

See Also

triclopsSetDoStereo()

triclopsGetStereoMask

Retrieves the stereo correlation mask size.

Declaration

```
TriclopsError
triclopsGetStereoMask( const TriclopsContext context,
                       int* size )
```

Parameters

context	The context.
size	A pointer to an integer that will store the current value.

triclopsGetSubpixelInterpolation

Retrieves the state of the subpixel interpolation feature.

Declaration

```
TriclopsError
triclopsGetSubpixelInterpolation( const TriclopsContext context,
                                  TriclopsBool* on )
```

Parameters

context	The context.
on	A pointer to a Boolean that will store the current setting.

See Also

triclopsSetSubpixelInterpolation(), triclopsSetStrictSubpixelValidation()

triclopsSetAnyStereoMask

Allows the user to set stereomask to any size.

This function allows you to set a stereo correlation mask of any size. There is a danger that an internal counter in the stereo correlation engine may overflow if mask sizes over 15 are provided. Thus, the current limit for triclopsSetStereoMask is 15. However, in practice, much larger mask sizes can be used successfully. If an overflow results, it may not affect the stereo result, and it will generally only happen for pathological cases. Therefore, this function is provided as an 'experimental' function to allow users to set any mask size they wish.

Declaration

```
TriclopsError
triclopsSetAnyStereoMask( TriclopsContext context,
                          int size )
```

Parameters

context	TriclopsContext for the operation.
size	The size for a new stereo correlation mask.

See Also

triclopsSetStereoMask()

triclopsSetDisparity

Sets the disparity range for stereo processing.

Declaration

```
TriclopsError
triclopsSetDisparity( TriclopsContext context,
                      int minDisparity,
                      int maxDisparity )
```

Parameters

context	The context.
minDisparity	The disparity range minimum.
maxDisparity	The disparity range maximum.

triclopsSetDisparityMapping

Sets the disparity range for stereo processing.

This function sets the disparity mapping values. The disparity mapping values control what range of pixels values appear in the output disparity image. The true disparity ranges between the minimum and maximum values set with triclopsSetDisparity(). The output image has its pixel values linearly scaled from minimum => maximum disparity to minimum => maximum disparity mapping. This is primarily used when one wishes to use a screen display buffer as the disparity image buffer. Note: it is advisable to set the disparity mapping to the exact values of the input disparities if one is not using the screen buffer display feature.

Declaration

```
TriclopsError
triclopsSetDisparityMapping( TriclopsContext context,
                             unsigned char minDisparity,
                             unsigned char maxDisparity )
```

Parameters

context	The context.
minDisparity	The disparity range in the output disparity image minimum. The range is between 0 and 255.
maxDisparity	The disparity range in the output disparity image maximum. The range is between 0 and 255.

See Also

triclopsGetDisparityMapping(), triclopsSetDisparity(), triclopsRCD8ToXYZ(), triclopsRCD16ToXYZ(), triclopsRCDMappedToXYZ()

triclopsSetDisparityMappingOn

Enables or disables disparity mapping.

This function is used to enable or disable disparity mapping. See the comments section on this subject in Chapter 7 in the Triclops manual for why disparity mapping can cause trouble.

Declaration

```
TriclopsError
triclopsSetDisparityMappingOn( TriclopsContext context,
                               TriclopsBool on )
```

Parameters

context	TriclopsContext for the operation.
on	A Boolean flag indicating if the mapping should be on or off.

triclopsSetDoStereo

Turns stereo processing on or off.

Declaration

```
TriclopsError
triclopsSetDoStereo( TriclopsContext context,
                     TriclopsBool on )
```

Parameters

context	The context.
on	A Boolean indicating whether stereo processing should be turned on or off.

See Also

triclopsGetDoStereo()

triclopsSetStereoMask

Set the stereo correlation mask size.

Declaration

```
TriclopsError
triclopsSetStereoMask( TriclopsContext context,
                       int masksize )
```

Parameters

context	The context.
masksize	The new correlation mask size, which is valid between 1 and 15.

triclopsSetSubpixelInterpolation

Turns subpixel interpolation stereo improvements on or off.

Declaration

```
TriclopsError
triclopsSetSubpixelInterpolation( TriclopsContext context,
                                  TriclopsBool on )
```

Parameters

context	The context.
on	A Boolean indicating whether it should be on or off.

See Also

triclopsGetSubpixelInterpolation(), triclopsSetStrictSubpixelValidation()

triclopsStereo

Does stereo processing, validation, and subpixel interpolation, as specified by parameters.

This function performs the stereo processing and validation, and generates the internal image TriImg_DISPARITY.

Declaration

```
TriclopsError
triclopsStereo( TriclopsContext context )
```

Parameters

context	The context.

See Also

triclopsPreprocessing(), triclopsSetDoStereo(), triclopsEdgeCorrelation(), triclopsSetTextureValidation(), triclopsSetUniquenessValidation(), triclopsGetROIs(), triclopsSetSubpixelInterpolation()

Configuration

triclopsGetBaseline

Retrieves the baseline of the cameras.

This function retrieves the baseline of the cameras in meters. The stereo context must have already been read.

Declaration

```
TriclopsError
triclopsGetBaseline( TriclopsContext context,
                     float* base )
```

Parameters

context	The context.
base	The baseline in meters.

See Also

triclopsGetDefaultContextFromFile()

triclopsGetCameraConfiguration

Retrieves the current configuration of the stereo camera. This configuration is the configuration that specifies the stereo algorithm used. For example, 2CAM_HORIZONTAL configuration can be used to do 2 camera stereo on a 3 camera device.

Declaration

```
TriclopsError
triclopsGetCameraConfiguration( const TriclopsContext context,
                                TriclopsCameraConfiguration* config )
```

Parameters

context	The context.

config	A pointer that will hold the result.

See Also

TriclopsCameraConfiguration, TriclopsSetCameraConfiguration(), TriclopsGetDeviceConfiguration()

triclopsGetDeviceConfiguration

This function returns the physical configuration of the stereo device. This allows the user to determine what algorithms they have available on the current device.

Declaration

```
TriclopsError
triclopsGetDeviceConfiguration( const TriclopsContext context,
                                TriclopsCameraConfiguration* config )
```

Parameters

context	The context.
config	The physical CameraConfiguration.

See Also

TriclopsCameraConfiguration, TriclopsGetCameraConfiguration(), TriclopsSetCameraConfiguration()

triclopsGetFocalLength

Retrieves the focal length of the cameras.

This function returns the focal length of the system. The focal length is in 'pixels' for the current selected output resolution. All cameras' rectified images have the same focal length. The default stereo context must have been read before this call can be made.

Declaration

```
TriclopsError
triclopsGetFocalLength( const TriclopsContext context,
                        float* focallength )
```

See Also

triclopsGetDefaultContextFromFile(), triclopsSetResolution()

triclopsGetImageCenter

Returns the optical center for pinhole calculations.

It is important that the context already has the resolution set. If triclopsSetResolution is not set, the returned value cannot be used for calculations. This image center can be used as the position in the image plane of the optical center for pinhole camera calculations.

Declaration

```
TriclopsError
triclopsGetImageCenter( TriclopsContext context,
                        float* centerRow,
                        float* centerCol )
```

Parameters

context	TriclopsContext for the operation.
centerRow	A pointer that will contain the row position of the image center for the current resolution.
centerCol	A pointer that will contain the column position of the image center for the current resolution.

See Also

triclopsSetResolution()

triclopsGetSerialNumber

This function returns the serial number of the Digiclops Stereo Vision System associated with the given TriclopsContext.

Declaration

```
TriclopsError
triclopsGetSerialNumber( const TriclopsContext context,
                         int* serialNumber )
```

Parameters

context	The context to extract the serial number from.
serialNumber	The serial number of the Digiclops Stereo Stereo Vision System associated with the given context.

triclopsSetCameraConfiguration

Sets the configuration of the cameras. This configuration determines the configuration for the stereo algorithm. For example, a three camera stereo device may be set into "2 camera horizontal" mode for faster stereo processing.

Declaration

```
TriclopsError
triclopsSetCameraConfiguration( const TriclopsContext context,
                                TriclopsCameraConfiguration config )
```

Parameters

context	The context.
config	The new CameraConfiguration.

See Also

TriclopsCameraConfiguration, TriclopsGetCameraConfiguration(), TriclopsGetDeviceConfiguration()

Conversions

triclopsGetTransformFromFile

Loads the contents of a TriclopsTransform from the input file.

This function fills in the provided TriclopsTransform structure based on the contents read from the specified file. If the specified file is not found, contains invalid fields, the value of CorruptTransformFile will be returned.

Declaration

```
TriclopsError
triclopsGetTransformFromFile( char* fileName,
                              TriclopsTransform* transform )
```

Parameters

fileName	name of the file from which to load the transform
transform	the 4x4 homogenous transform

See Also

triclopsRCDToWorldXYZ(), triclopsRCDFloatToWorldXYZ(), triclopsRCD8ToWorldXYZ(), triclopsRCD16ToWorldXYZ(), triclopsWorldXYZToRCD(), triclopsSetTriclopsToWorldTransform(), triclopsGetTriclopsToWorldTransform(), triclopsWriteTransformToFile

triclopsGetTriclopsToWorldTransform

Gets the Triclops to World transform.

This function fills in the provided TriclopsTransform structure with the current contents of the Triclops to World transform for this TriclopsContext

Declaration

```
TriclopsError
triclopsGetTriclopsToWorldTransform( TriclopsContext context,
                                     TriclopsTransform* transform )
```

Parameters

context	the TriclopsContext
transform	the 4x4 homogenous transform

See Also

triclopsRCDToWorldXYZ(), triclopsRCDFloatToWorldXYZ(), triclopsRCD8ToWorldXYZ(), triclopsRCD16ToWorldXYZ(), triclopsWorldXYZToRCD(), triclopsSetTriclopsToWorldTransform() triclopsGetTransformFromFile(), triclopsWriteTransformToFile()

triclopsRCD16ToWorldXYZ

Converts image coordinates and a 16-bit disparity into a world 3D point.

This function takes a 16-bit disparity value and converts it to XYZ coordinates in world coordinates. The world coordinates are determined by transforming the point from the Triclops coordinate system to the world coordinate system based on the TriclopsContext transform.

Note: It is up to the user to supply valid disparity values

Declaration

```
TriclopsError
triclopsRCD16ToWorldXYZ( TriclopsContext context,
                         int row,
                         int col,
                         unsigned short disp,
                         float* x,
                         float* y,
                         float* z )
```

Parameters

context	The stereo context.
row	The row of the input pixel.
col	The column of the input pixel.
disp	The disparity value of the input pixel.
x	The x coordinate of the corresponding 3D point
y	The y coordinate of the corresponding 3D point
z	The z coordinate of the corresponding 3D point

See Also

triclopsRCDFloatToXYZ(), triclopsRCDMappedToXYZ(), triclopsRCD8ToXYZ(), triclopsRCD16ToXYZ(), triclopsSetDisparity(), triclopsSetDisparityMapping() triclopsRCDToWorldXYZ(), triclopsRCDFloatToWorldXYZ(), triclopsRCD8ToWorldXYZ(), triclopsSetTriclopsToWorldTransform(), triclopsGetTriclopsToWorldTransform()

triclopsRCD16ToXYZ

Converts image coordinates and a 16-bit disparity into true 3D points.

When using this function, you should ensure that the values for the Triclops disparity mapping feature are the same as the disparity range.

Note: it is up to the user to supply valid pixel locations. Pixels which have been invalidated may give negative results.

Declaration

```
TriclopsError
triclopsRCD16ToXYZ( TriclopsContext context,
```

```
                    int row,
                    int col,
                    unsigned short disp,
                    float* x,
                    float* y,
                    float* z )
```

Parameters

context	The stereo context.
row	The row of the input pixel.
col	The column of the input pixel.
disp	The disparity value of the input pixel.
x	The x coordinate of the point represented by the input row column disparity in the camera coordinate system.
y	The y coordinate of the point represented by the input row column disparity in the camera coordinate system.
z	The z coordinate of the point represented by the input row column disparity in the camera coordinate system.

See Also

triclopsRCDFloatToXYZ(), triclopsRCDMappedToXYZ(), triclopsRCD8ToXYZ(), triclopsSetDisparity(), triclopsSetDisparityMapping()

triclopsRCD8ToWorldXYZ

Converts image coordinates and an 8-bit disparity into a world 3D point.

This function takes an 8-bit disparity value and converts it to XYZ coordinates in world coordinates. The world coordinates are determined by transforming the point from the Triclops coordinate system to the world coordinate system based on the TriclopsContext transform.

Note: It is up to the user to supply valid disparity values

Declaration

```
TriclopsError
triclopsRCD8ToWorldXYZ( TriclopsContext context,
                        int row,
                        int col,
                        unsigned char disp,
                        float* x,
                        float* y,
                        float* z )
```

Parameters

context	The stereo context.
row	The row of the input pixel.
col	The column of the input pixel.

disp	The disparity value of the input pixel.
x	The x coordinate of the corresponding 3D point
y	The y coordinate of the corresponding 3D point
z	The z coordinate of the corresponding 3D point

See Also

triclopsRCDFloatToXYZ(), triclopsRCDMappedToXYZ(), triclopsRCD16ToXYZ(), triclopsSetDisparity(), triclopsSetDisparityMapping() triclopsRCDToWorldXYZ(), triclopsRCDFloatToWorldXYZ(), triclopsRCD16ToWorldXYZ(), triclopsSetTriclopsToWorldTransform(), triclopsGetTriclopsToWorldTransform()

triclopsRCD8ToXYZ

Converts image coordinates and an *-bit disparity into true 3D points.

When using this function, you should ensure that the values for the Triclops disparity mapping feature are the same as the disparity range.

Note: it is up to the user to supply valid pixel locations. Pixels which have been invalidated may give negative results.

Declaration

```
TriclopsError
triclopsRCD8ToXYZ( TriclopsContext context,
                   int row,
                   int col,
                   unsigned char disp,
                   float* x,
                   float* y,
                   float* z )
```

Parameters

context	The stereo context.
row	The row of the input pixel.
col	The column of the input pixel.
disp	The disparity value of the input pixel.
x	The x coordinate of the point represented by the input row column disparity in the camera coordinate system.
y	The y coordinate of the point represented by the input row column disparity in the camera coordinate system.
z	The z coordinate of the point represented by the input row column disparity in the camera coordinate system.

See Also

triclopsRCDFloatToXYZ(), triclopsRCDMappedToXYZ(), triclopsRCD16ToXYZ(), triclopsSetDisparity(), triclopsSetDisparityMapping()

triclopsRCDFloatToWorldXYZ

Converts an image location and a floating-point disparity value into a world 3D point.

This function takes a floating-point disparity value and converts it to XYZ coordinates in world coordinates. The world coordinates are determined by transforming the point from the Triclops coordinate system to the world coordinate system based on the TriclopsContext transform.

Note: It is up to the user to supply valid disparity values

Declaration

```
TriclopsError
triclopsRCDFloatToWorldXYZ( TriclopsContext context,
                            float row,
                            float col,
                            float disp,
                            float* x,
                            float* y,
                            float* z )
```

Parameters

context	The stereo context.
row	The row of the input pixel.
col	The column of the input pixel.
disp	The disparity value of the input pixel.
x	The x coordinate of the corresponding 3D point
y	The y coordinate of the corresponding 3D point
z	The z coordinate of the corresponding 3D point

See Also

triclopsRCDMappedToXYZ(), triclopsRCD8ToXYZ(), triclopsRCD16ToXYZ() triclopsRCDToWorldXYZ(), triclopsRCD8ToWorldXYZ(), triclopsRCD16ToWorldXYZ(), triclopsSetTriclopsToWorldTransform(), triclopsGetTriclopsToWorldTransform()

triclopsRCDFloatToXYZ

Converts image coordinates and a floating-point disparity value into true 3D points.

This function takes a floating-point disparity value and converts it to XYZ coordinates.

Note: It is up to the user to supply valid pixel locations. Pixels which have been invalidated may give negative results.

Declaration

```
TriclopsError
triclopsRCDFloatToXYZ( TriclopsContext context,
                       float row,
                       float col,
                       float disp,
```

```
                        float* x,
                        float* y,
                        float* z )
```

Parameters

context	The stereo context.
row	The row of the input pixel.
col	The column of the input pixel.
disp	The disparity value of the input pixel.
x	The x coordinate of the point represented by the input row column disparity in the camera coordinate system.
y	The y coordinate of the point represented by the input row column disparity in the camera coordinate system.
z	The z coordinate of the point represented by the input row column disparity in the camera coordinate system.

See Also

triclopsRCDMappedToXYZ(), triclopsRCD8ToXYZ(), triclopsRCD16ToXYZ()

triclopsRCDMappedToWorldXYZ

Converts image coordinates with disparity values that have been mapped using the disparity mapping function into world 3D points.

This function takes disparity values that have been scaled by the disparity mapping feature. If you have set "Disparity Mapping" on you should use this function. However, it is less efficient than the other XYZ conversion functions and may have some round-off errors. It is preferable to set the disparity mapping off. This function will transform the point into a "world" coordinate system based on the transform recorded in the TriclopsContext.

Note: It is up to the user to supply valid disparity values, invalid disparity values (as taken from an invalid pixel in the a disparity image) may give negative results.

Declaration

```
TriclopsError
triclopsRCDMappedToWorldXYZ( TriclopsContext context,
                             int row,
                             int col,
                             unsigned char disp,
                             float* x,
                             float* y,
                             float* z )
```

Parameters

context	The stereo context.
row	The row of the input pixel.
col	The column of the input pixel.

disp	The disparity value of the input pixel.
x	The x coordinate of the point represented by the input row column disparity in the camera coordinate system.
y	The y coordinate of the point represented by the input row column disparity in the camera coordinate system.
z	The z coordinate of the point represented by the input row column disparity in the camera coordinate system.

See Also

triclopsRCDFloatToXYZ(), triclopsRCD8ToXYZ(), triclopsRCD16ToXYZ() triclopsRCDToWorldXYZ(), triclopsRCDFloatToWorldXYZ(), triclopsRCD8ToWorldXYZ(), triclopsRCD16ToWorldXYZ(), triclopsSetTriclopsToWorldTransform(), triclopsGetTriclopsToWorldTransform()

triclopsRCDMappedToXYZ

Converts image coordinates with disparity values that have been mapped using the disparity mapping function into true 3D points.

This function takes disparity values that have been scaled by the disparity mapping feature. If you do not have the disparity mapping values set to the same as the disparity values, you should use this function. However, it is less efficient than the other XYZ conversion functions and may have some round-off errors. It is preferable to set the disparity mapping range to the same as the disparity range and use one of the other conversion functions.

Note: It is up to the user to supply valid pixel locations. Pixels that have been invalidated may give negative results.

Declaration

```
TriclopsError
triclopsRCDMappedToXYZ( TriclopsContext context,
                        int row,
                        int col,
                        unsigned char disp,
                        float* x,
                        float* y,
                        float* z )
```

Parameters

context	The stereo context.
row	The row of the input pixel.
col	The column of the input pixel.
disp	The disparity value of the input pixel.
x	The x coordinate of the point represented by the input row column disparity in the camera coordinate system.
y	The y coordinate of the point represented by the input row column disparity in the camera coordinate system.
z	The z coordinate of the point represented by the input row column disparity in the camera coordinate system.

See Also

triclopsRCDFloatToXYZ(), triclopsRCD8ToXYZ(), triclopsRCD16ToXYZ()

triclopsRCDToWorldXYZ

Converts image coordinates and disparity values to world 3D points.

This function takes a pixel location and matching disparity value and calculates the 3D position that this combination represents. The position is calculated in the "world" coordinate system. That is to say, the position is calculated in the Triclops coordinate system and then transformed by the TriclopsContext transform to a new coordinate system.

Declaration

```
TriclopsError
triclopsRCDToWorldXYZ( TriclopsContext context,
                       float row,
                       float col,
                       float disp,
                       float* x,
                       float* y,
                       float* z )
```

Parameters

context	The stereo context.
row	The row of the input pixel.
col	The column of the input pixel.
disp	The disparity value of the input pixel.
x	The x coordinate of the corresponding 3D point in the world coordinate system
y	The y coordinate of the corresponding 3D point in the world coordinate system
z	The z coordinate of the corresponding 3D point in the world coordinate system

Remarks

It is up to the user to supply valid disparity values. Values taken from invalid pixels in the disparity image give negative results.

See Also

triclopsRCDFloatToXYZ(), triclopsRCD8ToXYZ(), triclopsRCD16ToXYZ(), triclopsRCDFloatToWorldXYZ(), triclopsRCD8ToWorldXYZ(), triclopsRCD16ToWorldXYZ(), triclopsSetTriclopsToWorldTransform(), triclopsGetTriclopsToWorldTransform()

triclopsRCDToXYZ

Converts image coordinates with disparity values that have been mapped using the disparity mapping function into true 3D points.

This function takes disparity values that have been scaled by the disparity mapping feature. If you do not have the disparity mapping values set to the same as the disparity values, you should use this function. However, it is less efficient than the other XYZ conversion functions and may have some round-off errors. It is preferable to set the disparity mapping range to the same as the disparity range and use one of the other conversion functions.

Note: It is up to the user to supply valid pixel locations. Pixels that have been invalidated may give negative results.

Declaration

```
TriclopsError
triclopsRCDToXYZ( TriclopsContext context,
                  float row,
                  float col,
                  float disp,
                  float* x,
                  float* y,
                  float* z )
```

Parameters

context	The stereo context.
row	The row of the input pixel.
col	The column of the input pixel.
disp	The disparity value of the input pixel.
x	The x coordinate of the point represented by the input row column disparity in the camera coordinate system.
y	The y coordinate of the point represented by the input row column disparity in the camera coordinate system.
z	The z coordinate of the point represented by the input row column disparity in the camera coordinate system.

See Also

triclopsRCDFloatToXYZ(), triclopsRCD8ToXYZ(), triclopsRCD16ToXYZ()

triclopsRectifyColorImage

Rectifies a TriclopsInput structure into a TriclopsColorImage. This function is used for TriclopsInput's that have been obtained from a Color Digiclops or a Color Triclops. If the system is a Color Triclops, nCamera should be set to TriCam_COLOR.

Declaration

```
TriclopsError
triclopsRectifyColorImage( TriclopsContext context,
                           TriclopsCamera nCamera,
                           TriclopsInput* input,
                           TriclopsColorImage* output )
```

Parameters

context	The context.
nCamera	A TriclopsCamera enumerated value indicating which camera the input image came from.
input	The raw color image encoded into a TriclopsInput structure.
output	The resultant rectified color image.

triclopsRectifyPackedColorImage

This function rectifies a packed color image. This function will only rectify a packed TriclopsInput (a TriclopsInput of type TriInp_RGB_32BIT_PACKED). It is useful for creating rectified color images for display, since bitmap displays commonly require the data format to be packed.

Declaration

```
TriclopsError
triclopsRectifyPackedColorImage( TriclopsContext context,
                                 TriclopsCamera nCamera,
                                 TriclopsInput* input,
                                 TriclopsPackedColorImage* output )
```

Parameters

context	The TriclopsContext to use to rectify the color image.
nCamera	The camera from which this TriclopsInput originated. If the system is a Color Triclops, nCamera should be set to TriCam_COLOR.
input	The color image to be rectified.
output	The rectified color image.

triclopsRectifyPixel

Converts a pixel coordinate location from the unrectified image coordinates to rectified image coordinates.

Declaration

```
TriclopsError
triclopsRectifyPixel( const TriclopsContext context,
                      TriclopsCamera camera,
                      float rowIn,
                      float colIn,
                      float* rowOut,
                      float* colOut )
```

Parameters

context	The context.
camera	The camera for which the pixel should be rectified.
rowIn	The location of the pixel to rectify.

colIn	The location of the pixel to rectify.
rowOut	The location of the rectified pixel.
colOut	The location of the rectified pixel.

See Also

triclopsPreprocess(), triclopsRectifyColorImage()

triclopsSetTriclopsToWorldTransform

Sets the Triclops to World transform.

```
This function sets the internal TriclopsContext transform to match the
provided transform.  There are several things to note:
```

- the transform is the Triclops-to-World transform. ie: when this transform is applied to a Triclops point, it will change it to a world point
- the transform must be of the approved format. This means it has a 3x3 rotational component that is orthonormal and the bottom row must be of format (0 0 0 n). This function will try to normalize the rotational component and clean up the bottom row of the matrix. One can verify whether modifications to the transform were necessary by obtaining the current transform using triclopsGetTriclopsToWorldTransform() and comparing.

Declaration

```
TriclopsError
triclopsSetTriclopsToWorldTransform( TriclopsContext context,
                                     TriclopsTransform transform )
```

Parameters

context	the TriclopsContext
transform	the 4x4 homogenous transform

See Also

triclopsRCDToWorldXYZ(), triclopsRCDFloatToWorldXYZ(), triclopsRCD8ToWorldXYZ(), triclopsRCD16ToWorldXYZ(), triclopsWorldXYZToRCD(), triclopsGetTriclopsToWorldTransform() triclopsGetTransformFromFile(), triclopsWriteTransformToFile()

triclopsUnrectifyPixel

Converts a pixel coordinate location from the rectified image coordinates to unrectified image coordinates.

Declaration

```
TriclopsError
triclopsUnrectifyPixel( const TriclopsContext context,
                        TriclopsCamera camera,
                        float rowIn,
                        float colIn,
                        float* rowOut,
                        float* colOut )
```

Parameters

context	The context.
camera	The camera for which the pixel should be unrectified.
rowIn	The location of the pixel to unrectify.
colIn	The location of the pixel to unrectify.
rowOut	The location of the unrectified pixel.
colOut	The location of the unrectified pixel.

Remarks

This version will accommodate input rectified pixel locations that are outside of the normal rectified image bounds, as long as the corresponding unrectified location is within its own image bounds.

See Also

triclopsRectifyPixel()

triclopsWorldXYZToRCD

Converts world 3D points into image coordinates.

This function takes as input the XYZ position of a point in the World coordinate system, moves the point to the Triclops coordinate system (as described by the TriclopsContext transform), and determines what row, column, and disparity value would result from the resulting point.

Declaration

```
TriclopsError
triclopsWorldXYZToRCD( TriclopsContext context,
                       float x,
                       float y,
                       float z,
                       float* row,
                       float* col,
                       float* disp )
```

Parameters

context	TriclopsContext set up for desired resolution.
x	X value of a point in the World coordinate system.

y	Y value of a point in the World coordinate system.
z	Z value of a point in the World coordinate system.
row	The row in a disparity image.
col	The column in a disparity image.
disp	The disparity value that would match the point specified in XYZ.

See Also

triclopsRCDFloatToXYZ(), triclopsRCDMappedToXYZ(), triclopsRCD8ToXYZ(), triclopsSetDisparity(), triclopsSetDisparityMapping() triclopsRCDToWorldXYZ(), triclopsRCDFloatToWorldXYZ(), triclopsRCD8ToWorldXYZ(), triclopsRCD16ToWorldXYZ(), triclopsXYZToRCD(), triclopsSetTriclopsToWorldTransform(), triclopsGetTriclopsToWorldTransform()

triclopsWriteTransformToFile

Saves the contents of a TriclopsTransform to the output file.

This function saves the contents of the specified transform to an external file.

Declaration

```
TriclopsError
triclopsWriteTransformToFile( char* fileName,
                              TriclopsTransform* transform )
```

Parameters

fileName	name of the file to which to save the transform
transform	the 4x4 homogenous transform to save

See Also

triclopsRCDToWorldXYZ(), triclopsRCDFloatToWorldXYZ(), triclopsRCD8ToWorldXYZ(), triclopsRCD16ToWorldXYZ(), triclopsWorldXYZToRCD(), triclopsSetTriclopsToWorldTransform(), triclopsGetTriclopsToWorldTransform(), triclopsGetTransformFromFile

triclopsXYZToRCD

Converts true 3D points into image coordinates.

This function takes as input the XYZ position of a point in the Triclops coordinate system, and determines what row, column, and disparity value would result from a sensed point at XYZ.

Declaration

```
TriclopsError
triclopsXYZToRCD( TriclopsContext context,
                  float x,
                  float y,
```

```
                    float z,
                    float* row,
                    float* col,
                    float* disp )
```

Parameters

context	TriclopsContext set up for desired resolution.
x	X value of a point in the Triclops coordinate system.
y	Y value of a point in the Triclops coordinate system.
z	Z value of a point in the Triclops coordinate system.
row	The row in a disparity image.
col	The column in a disparity image.
disp	The disparity value that would match the point specified in XYZ.

Ungrouped Objects

TRICLOPS_VERSION

Declaration

```
#define TRICLOPS_VERSION 3101
```

Remarks

The version of the library.

Widebaseline Stereo

triclopsCreateImage3d

Allocates a TriclopsImage3d to the correct size as specified by the resolution of the TriclopsContext

Declaration

```
TriclopsError
triclopsCreateImage3d( TriclopsContext context,
                       TriclopsImage3d** ppimage )
```

Parameters

context	The current TriclopsContext.
ppimage	Pointer to the address of the TriclopsImage3d structure to allocate memory for.

See Also

triclopsDestroyImage3d(), triclopsExtractImage3d(), triclopsExtractWorldImage3d()

triclopsCreateWidebaselineContext

Creates a widebaseline context based on the contexts of the two underlying stereo devices.

This function creates a widebaseline context based on the contexts of the two underlying stereo devices. The two input contexts, beside having been properly initialized (and loaded) must have their transformation set previous to this operation. Furthermore, these transforms both relate back to the same world reference frame so that they can be used to determine how the two cameras are positioned relative to one another. Once the widebaseline context has been created, stereo processing for the virtual widebaseline camera can proceed by providing it with the corresponding widebaseline input.

Declaration

```
TriclopsError
triclopsCreateWidebaselineContext( TriclopsContext triclops1,
                                   TriclopsContext triclops2,
                                   TriclopsContext* triclops3 )
```

Parameters

triclops1	The context for the stereo devive which forms the right camera for the stereo pair.
triclops2	The context for the stereo devive which forms the left camera for the stereo pair.
triclops3	The new context for stereo between the two devices

See Also

TriclopsCreateWidebaselineInput

triclopsCreateWidebaselineInput

Creates a widebaseline input based on the two camera inputs.

Create a widebaseline input based on the TriclopsInput of the two underlying stereo devices. The resulting input can then be used to perform stereo processing with the corresponding widebaseline context.

Declaration

```
TriclopsError
triclopsCreateWidebaselineInput( TriclopsInput input1,
                                 TriclopsInput input2,
                                 TriclopsInput* input3 )
```

Parameters

input1	The input from the stereo devive which forms the right camera for the stereo pair.
input2	The input from the stereo devive which forms the left camera for the stereo pair.
input3	The new input constructed from that of the two underlying devices

See Also

triclopsCreateWidebaselineContext

triclopsDestroyImage3d

Deallocates a TriclopsImage3d allocated by triclopsCreateImage3d()

Declaration

```
void
triclopsDestroyImage3d( TriclopsImage3d** ppimage )
```

Parameters

ppimage	Pointer to the address of the TriclopsImage3d structure to destroy.

See Also

triclopsCreateImage3d(), triclopsExtractImage3d(), triclopsExtractWorldImage3d()

triclopsExtractImage3d

Creates a 3D image given the current disparity result of a TriclopsContext

Declaration

```
TriclopsError
triclopsExtractImage3d( TriclopsContext context,
                        TriclopsImage3d* pimage )
```

Parameters

context	The current TriclopsContext.
pimage	Pointer to the TriclopsImage3d structure.

Remarks

Invalid points will be assigned the value of (0,0,0) in the returned image.

See Also

triclopsDestroyImage3d(), triclopsCreateImage3d(), triclopsExtractWorldImage3d()

triclopsExtractWorldImage3d

Creates a 3D image given the current disparity result of a TriclopsContext that is transformed to the world coordinate system

Declaration

```
TriclopsError
triclopsExtractWorldImage3d( TriclopsContext context,
                             TriclopsImage3d* pimage )
```

Parameters

context	The current TriclopsContext.

pimage	Pointer to the TriclopsImage3d structure.

Remarks

Invalid points will be assigned the value of (0,0,0) in the returned image.

See Also

triclopsDestroyImage3d(), triclopsCreateImage3d(), triclopsExtractImage3d()

Chapter 7: Stereo Vision Details

This chapter presents an overview of stereo vision technology. After reading this chapter you will have a better understanding of the data flow in the library and all tunable parameters. This will allow you to customize the system to particular tasks.

The purpose of stereo vision is to perform range measurements based on images obtained from slightly offset cameras. There are three steps in performing stereo processing:

Establish correspondence between image features in different views of the scene.
Calculate the relative displacement between feature coordinates in each image.
Determine the 3D location of the feature relative to the cameras, using the knowledge of the camera geometry.

Consider the following example. Figure 2 shows an image pair obtained from the horizontally displaced cameras of the Triclops camera module. We can identify two points A and B in both images. The point A_{left} corresponds to the point A_{right}. Similarly, point B_{left} corresponds to the point B_{right}.

Figure 2: Example of matching points between stereo images

Using a ruler, if you measure out the horizontal distance between the left edge of the images and the points, you will find that distances in the left image are greater than the distance to the corresponding point in the right image. For example, the distance to the phone handle from the left edge of the image is greater than the distance to the phone handle in the right image. Based on this distance (also called disparity) it is possible to determine the distance to the phone handle from the camera module.

We will define the disparity as the difference between the coordinates of the same features in the left and right image. You will find that the distances from the top of the image to the matching features are exactly the same in both images. This is because the cameras are horizontally aligned, therefore only the horizontal displacement is relevant.

Disparity for the feature A will be defined as $D(A) = x(A_{left}) – x(A_{right})$ and the disparity of point B will be derived as $D(B) = x(B_{left}) – x(B_{right})$, where $x(A_{left})$ is the x coordinate of the point A_{left}.

If you calculated D(A) and D(B) you will find that D(B) > D(A) this indicated that point B in the scene is closer than point A.

Establishing Correspondence

The Triclops library establishes correspondence between images using the Sum of Absolute Differences correlation method. The intuition behind the approach is to do the following:

For every pixel in the image
Select a neighborhood of a given square size from the reference image
Compare this neighborhood to a number of neighborhoods in the other image (along the same row)
Select the best match
End
Comparison of neighborhoods or masks is done using the following formula:

$$\min_{d=d_{\min}}^{d_{\max}} \sum_{i=-\frac{m}{2}}^{\frac{m}{2}} \sum_{j=-\frac{m}{2}}^{\frac{m}{2}} \left| \mathrm{I}_{right}[x+i][y+j] - \mathrm{I}_{left}[x+i+d][y+j] \right|$$

where:

$d_{\min}$ and $d_{\max}$ are the minimum and maximum disparities.

m is the mask size.

I_{right} and I_{left} are the right and left images.

Equation 1: Sum of absolute differences

Calculating Distances

The distances from the cameras are determined using the displacement between images and the geometry of the cameras. The position of the matched feature is a function of the displacement, the focal length of the lenses, resolution of the CCD and the displacement between cameras.

The Triclops library provides a function that converts depth maps into distance images.

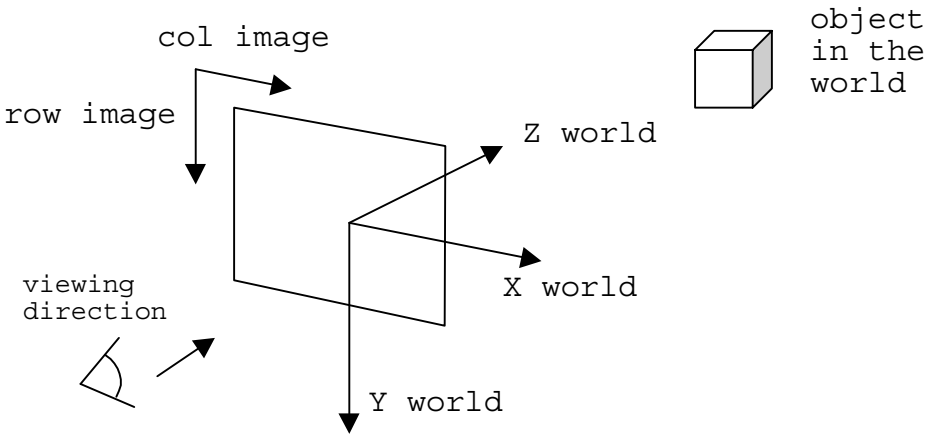

Figure 3: Image and world coordinate systems in the Triclops library

Figure 7 illustrates the coordinate system in which the Triclops library represents images and the world measurements. The origin of the image is in the top left corner of the upright image. The origin of the world measurements is in the pinhole of the reference camera.
Multiple Cameras

The Digiclops™ Stereo Vision System consists of three cameras. The reason for this is to make matching between images more robust. The principle behind using multiple cameras is to combine the matching results in order to establish the most correct match. In the case of the Triclops library, the results of correlation are added up for the equivalent displacements in the vertical and horizontal direction. By combining the correlation values it is possible to encase the valleys in which the correct match is found.

Triclops Library Data Flow

Figure 4 shows the data flow in the Triclops library. The library takes raw images obtained from the Triclops camera module and produces depth images. There are two main processing blocks in the library. The first processing block is the image pre-processing block that applies a low-pass filter, rectifies the images and performs edge detection. The second processing block does stereo matching, validation of results and subpixel interpolation. The result of the library is a depth image.

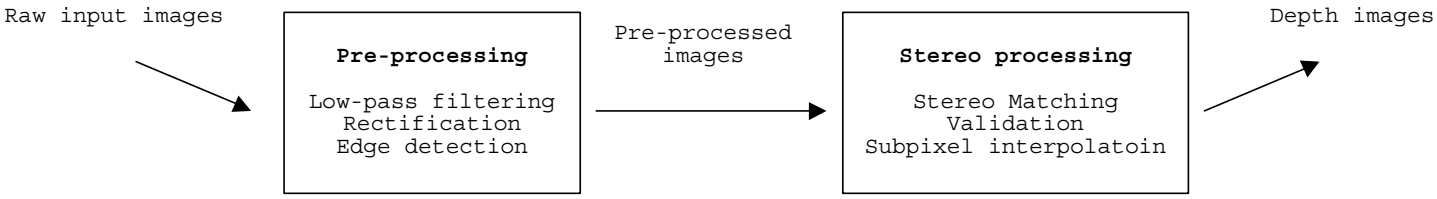

Figure 4: Triclops library data flow

Preprocessing

The pre-processing module of the Triclops library prepares the raw images for stereo processing. Pre-processing allows specification of the processing resolution, and the following functionality:
Low-Pass Filtering

In order to rectify the images it is important to smooth them. If image rectification is done it is a good idea to turn on the low pass filtering. Rectification can still be done with low pass filtering disabled but the rectified images will exhibit aliasing effects. The user has the option of disabling the low-pass filtering in order to speed up processing.

Rectification

Rectification is the process of correcting input images for the distortions of the lenses. Lenses often cause distortions in raw images. For example, straight lines in the scene will often appear curved in the raw images. This effect will be particularly evident in the corners of the images. Rectified images will be corrected for these kinds of distortions.

Further, rectified images will be corrected so that the rows of images digitized from horizontally displaced cameras are aligned, and similarly that the columns of images obtained from vertically displaced cameras are aligned. Without this feature, searching along the rows and columns will not produce the correct results.

Edge Detection

Edge detection is an optional feature that allows matching on the changes in the brightness rather than the absolute values of the pixels in the images. This feature is useful because the cameras in the Triclops camera module have auto gain control. If the auto gains in the cameras do not change identically, the absolute brightness between images may not be the same. While absolute brightness are not the same, the change in the intensity stays constant. Therefore, using edge detection will help in environments where the lighting conditions change significantly.

While edge detection may improve results, there is an additional processing cost that is associated with it. Therefore the user needs to evaluate the result improvement when choosing to turn edge detection on.

Note that validation is only available in the edge detection mode.

Stereo Processing

The stereo processing module applies the Sum of Absolute Differences algorithm described in previous sections. There are a number of parameters that determine the kind of depth image produced:

Disparity Range

Disparity range is the range of pixels that the stereo algorithm searches in order to find the best match. In the Triclops system a disparity of zero pixels corresponds to an infinitely far away object. The maximum disparity defines the closest position of an object that is to be determined. Users are allowed and encouraged to set the disparity range that is the most suitable for the task at hand. Reducing the disparity range will allow the system to run faster and will reduce the chance of a mismatching.

Correlation Mask

Correlation mask is a square neighborhood around the pixel that the system is trying to find the match for. The user is allowed to specify the size of the correlation mask. The correlation mask controls the coarseness of features compared between images. Larger masks will produce depth maps that are denser and smoother, however, they may lack precision in identifying the position of depth discontinuities. On the other hand smaller masks will produce sparser and more noisy depth images, but the localization of depth discontinuities will be much better.

To produce similar results the size of the mask must be proportional to the resolution of the images processed. Thus, in order to produce comparable results, a 5x5 mask on a 160x120-pixel image should be increased to a 9x9 mask for a 320x240-pixel image. Mask sizes must be odd numbers. Valid mask sizes are 3x3, 5x5, 7x7, and invalid mask sizes are 4x4, 6x6, 8x8. The Triclops is capable of supporting a maximum 15x15 mask to a 1x1 minimum. In addition, a new experimental function has been added: triclopsSetAnyStereoMask. This function allows the user to set any correlation mask size for stereo. This function is classified as an 'experimental' function, and care should be taken in its use. For more information, see the description of 'triclopsSetAnyStereoMask' in the Detailed Experimental Function Descriptions section.

Validation

In some cases, such as occlusions and lack of texture, it is not possible to establish correspondence between images. If the correspondence is not correct, the obtained measurements can not be correct. In order to avoid incorrect measurements, two validation methods are introduced:
Texture validation determines whether disparity values are valid based on levels of texture in the correlation mask. If the amount of texture is not sufficient to produce correct matches, the pixel will be declared invalid.
Uniqueness validation determines whether the best match for a particular pixel is significantly better than other matches within the correlation mask. Even if the correlation mask has enough texture, the correct match may not exist due to an occlusion. If the correlation result is not strong enough, the pixel will be declared invalid.

The user can specify two thresholds that control the strictness of validation - one for texture and one for uniqueness.

Better Calibration

The camera calibration has been improved to give greater overlap between cameras. This means a wider range of focal length lens can be used in the Triclops family of stereo vision cameras. Triclops SDK 2.2 is the first Triclops release that fully takes advantage of the new advances in calibration.

Subpixel Interpolation

The Triclops library allows matching between images to subpixel accuracy. The library takes advantage of the matching results of the neighboring pixels of the resulting disparity to determine an approximation that is within a fraction of a pixel. Accurate calibration between cameras allows an accuracy of 0.2 of a pixel.

This function marginally increases the computation time. If precise 3D position information is not required, it may be omitted.

Surface Validation

Triclops SDK 2.5 supports 'surface validation'. This is a filtering process designed to remove noise from the disparity image. The kind of noise removed with this process is 'spikes'. Spikes are characteristic of mismatches in correlation-based stereo vision. The spikes can often cover a connected region of many pixels. This noise is not zero-mean, random, evenly distributed or Gaussian. The result is that this noise is difficult to remove with standard filtering techniques, as it appears to be a valid signal, instead of noise. Surface validation is a method to validate regions of a disparity image based on an assumption that they must belong to a likely physical surface in the image. The method segments the disparity image into connected regions. Any region that is less than a given size, is suspect and removed from the disparity image. See also in the Detailed Function Descriptions section: triclopsSetSurfaceValidation, triclopsGetSurfaceValidation, triclopsSetSurfaceValidationSize, triclopsGetSurfaceValidationSize, triclopsSetSurfaceValidationDifference, triclopsGetSurfaceValidationDifference, triclopsSetSurfaceValidationMapping, triclopsGetSurfaceValidationMapping.

Subpixel Validation Mapping

In previous versions of the Triclops SDK, the validation mapping did not work when the subpixel interpolation flag was on. Now, validation mapping values are used during subpixel interpolation. Pixels which are invalid when performing subpixel validation are marked with values of 0xFF00 + mv, where 'mv' is the mapping value for the particular validation check that has failed.

Region Of Interest and Subpixel

Unfortunately, although ROI processing does work with subpixel stereo, it does not improve the speed of processing. Therefore, we recommend that users only bother with ROI processing if they are not using the subpixel functionality of the stereo engine.

Disparity Mapping

Disparity mapping is a method to automatically scale the output disparity image between some fixed 'mapping' values. This is a convenience, especially for demos when one wants the disparity images to be scaled within the entire display range of brightness values. The problem with disparity mapping is that it can cause a lot of trouble when trying to use mapped information to extract 3D information. The RCDToXYZ family of functions are designed to extract 3D information from disparity values. If these functions have to first correct for disparity mapping, a significant performance reduction has incurred.

To make it easy to simply turn off and ignore disparity mapping, we have added a new flag for the TriclopsContext, the DisparityMappingOn flag. This is accessed through triclopsSet/GetDisparityMappingOn() functions. The default value will be 'false'. This means that most

users of the Triclops SDK can ignore disparity mapping. Users who are using disparity mapping must change their programs to call triclopsSetDisparityMappingOn(context, 1) to enable disparity mapping.

Disparity mapping does not currently work for subpixel stereo, and Point Grey Research, Inc. (PGR) currently has no plans to extend disparity mapping to subpixel stereo. PGR would like to discourage users from depending on this functionality as this may be removed in later versions of the SDK.

Contacting Point Grey Research

We encourage you to visit the online Q & A forums on our web site if you encounter any problems with the Triclops Stereo Vision SDK. For any problems not resolved on our web site, or if you have any comments for us, you can contact us via the methods listed below.

For any questions, concerns or comments please contact us via the following methods:

Email:

info@ptgrey.com

sales@ptgrey.com

support@ptgrey.com

Telephone:

(604) 730-9937

Fax:

(604) 732-8231

Mail:

Point Grey Research, Inc.

305-1847 W. Broadway

Vancouver, BC

V6J 1Y6

Canada

Index

A

B

C

D

E

G

I

L

M

U

V

W