The Stereocam System
Design and Implementation

Kevin Pulo
`kev@zip.com.au`
`http://www.zip.com.au/~kev/`

November, 1999

$Id: stereocam-implementation.lyx,v 1.8 1999/11/25 14:29:32 kev Exp kev $

Abstract

The Stereocam system is a comprehensive tool and modular framework for dealing with stereo imagery in many combinations of input and output devices, for both static and dynamically changing stereo images. It aims to assist in all stages of stereo imagery processing, from input to output. A specific goal is to assist with displaying stereo video, obtained from dual cameras separated by a parallax.

This document details the design and implementation of the Stereocam system in the initial reference implementation. The system is based on the specifications and design detailed in [1], with many alterations and enhancements. This document is a revised version of [1], containing the updated version of the design and details of the implementation itself.

1 Introduction

1.1 Motivation

For some time now, computers have had the ability to display stereo imagery, often referred to as *stereovisuals*[1]. This is a situation where each eye of the viewer is shown a slightly different image to the other eye. The individual images have the same subject, but are observed from two positions offset by a value equivalent to the distance between the viewer's eyes (sometimes called the parallax of the stereo image). The brain of the viewer then combines these left and right eye images in the same way as when viewing any real scene. The result is an image that appears to be "really there" with depth perception, the ability to see how far away an object is without having to consider properties such as its relative size.

Traditionally, support for using stereovisual output in an application required stereo vision to be supported natively within the application itself. However, many applications may output stereovisuals simply by outputting two standard views, each offset by a parallax separation. Tools to process and display these sorts of images are not common, and usually consist of either opaque, inextensible utilities or trivial example programs. As such, general purpose tools for stereovisuals are lacking.

In addition, computers with video cameras attached are becoming increasingly prevalent. The most common type at present are webcam devices (refer to Section 1.4), which are cheap and usually deliver reasonable quality video at good frame rates. SGI now ships webcams as standard across their range of workstations.

[1] By analogy to "audiovisual".

1.2 The Challenge

By far the majority of stereo images have been displaying to date are computer generated images. These are artificial images that, while they may have been based or generated from real data (such as meteorological data), are created within the computer's artificial universe. As such, the idea behind this system is to combine the availability of stereo output devices and video input devices, creating a system that can use dual video cameras to provide a stereo input, coupled with a usual stereo output device.

Further, rather than an application supporting only this mode of stereo vision, a more general-purpose application is envisaged, which provides support for many different combinations of stereo input, output and processing. The application should also be extensible, to allow new types of input, output and processing to be added at any stage.

1.3 Theoretical Requirements

This section details the requirements of any system that deals with stereo vision, usually limitations imposed by the human visual system.

The first requirement for stereovisual images is that both images in the stereo images be "in focus", that is, focused on the same plane in space, the plane of the subject of the image. If they are different, then the image received by one eye will be blurred and out of focus in the same way as a regular photograph can be out of focus. The end result is that the brain becomes confused when trying to combine the crisp image with the blurred one, and cannot properly view the stereo image.

The cameras should be angled inwards so that they are directed towards the subject. This means that as the distance to the subject changes, so must the angle of inclination of the cameras. Subjects at an infinite distance (that is, in the background) would require the cameras to be parallel to each other. Also, subject material closer than the focal length of humans would almost certainly not be properly comprehensible[2], though it would be possible to record. However, the results of these kinds of experiments would be interesting, as a major restriction with humans is the limits of the eye muscles and the way the brain insists on trying to keep the eyes focused on the subject.

When displaying a stereo image the human's eyes are focused on the screen, but the stereo image being displayed may be of a subject at a completely different distance from the viewer. This mismatch is a problem with all stereo displays of this kind, as well as many virtual reality environments, such as first person video games.

All of these problems are physical problems which lie outside the scope of this system[3]. For example, determining the distance to a subject (required to focus the cameras) requires either sophisticated cameras, additional sensory hardware or highly sophisticated artificial intelligence software. However, a system without such features would be substantially limited in the subjects and images that it could view, especially for dynamic images.

1.4 Definitions

The following terms will be used throughout this document, unless stated otherwise they should be interpreted as defined here.

[2] As is the case when being directly observed by a human.

[3] This is not to say that they can be ignored completely, but that the system will ignore them for all intents and purposes. The user will be expected to handle them them properly to obtain satisfactory results.

Image A regular array of pixels, forming a picture.

Left Image An image intended to be seen by the viewer's left eye.

Right Image An image intended to be seen by the viewer's right eye.

Stereo Image A left image and right image that correspond to one another, that is, are intended to be viewed simultaneously.

Component Image The left image or right image of a stereo image.

Frame Synonym for image, especially when applied to an image that is part of a Movie.

Left Frame A frame intended to be seen by the viewer's left eye.

Right Frame A frame intended to be seen by the viewer's right eye.

Stereo Frame A left frame and right frame that correspond to one another, that is, are intended to be viewed simultaneously.

Component Frame The left frame or right frame of a stereo frame.

Movie A sequence of frames that are intended to be viewed as a dynamically changing image, that is, displayed sequentially, exploiting the viewer's persistence of vision (as with televisions, CRTs, and so on).

Left Movie A movie consisting of left frames.

Right Movie A movie consisting of right frames.

Stereo Movie A movie consisting of stereo frames.

Component Movie The left movie or right movie of a stereo movie.

Dynamic Image Synonym for Movie.

Frame Rate The number of frames per second in a movie.

Webcam A common kind of small, usually CCD[4]based video camera, so called because they are often used to display live video on an Internet web site.

Indycam The webcam that SGI ship with their Indy range of workstations.

O_2cam The webcam that SGI ship with their O_2 range of workstations.

2 Preliminary Investigation

2.1 Introduction

Towards the end of 1998 some informal, preliminary experiments with stereo vision were carried out. The experiments were performed with off-the-shelf components (in terms of the hardware and software) with some glue constructed in software to hold the interoperating systems together. The results of the experiments showed promise for the stereovisual techniques that this project aims for, and an indication of the project's feasibility. The results also served to highlight the most highly anticipated areas of difficulty.

[4]CCD stands for *Charge Coupled Device*, a type of video input device that are becoming increasingly popular.

Figure 1: An Indycam.

Figure 2: An O_2cam.

The experiments involved taking still stereo images (that is, stereo versions of standard still photographs), and then displaying them in stereo mode on the computer. While this has a significantly smaller domain to that posed by this system, this simplification has the advantage that preliminary results could be obtained very quickly. Software was sourced to take the images and to subsequently display them in stereo mode, and small shell scripts constructed to glue the various utilities together. The result was a small system that could take two still images on different computers almost simultaneously and then display them in stereo mode.

2.2 Method

The cameras used for the experiment were two of the webcams that SGI shipped with its Indy range of workstations (dubbed *Indycams*, Figure 1). These were preferable those that come with the O_2 workstations (dubbed *O_2cams*, Figure 2) for simple logistical reasons — it's very easy to strap two of the Indy cameras together ("back to back"), with the lenses being separated by a good approximation to a typical human eye separation (Figure 3).[5] The most important detail here is that the cameras are laying on their sides; the images obtained from each need to be rotated 90° clockwise and counter-clockwise as appropriate.

[5]As shown in Figure 3 the cameras were strapped together with rubber bands and placed in a small box to stand them upright, however the base of the cameras has a threaded hole which would allow two cameras to essentially be bolted together for better stability. This could even be extended to have an adjustable bolt controlling the separation of the cameras.

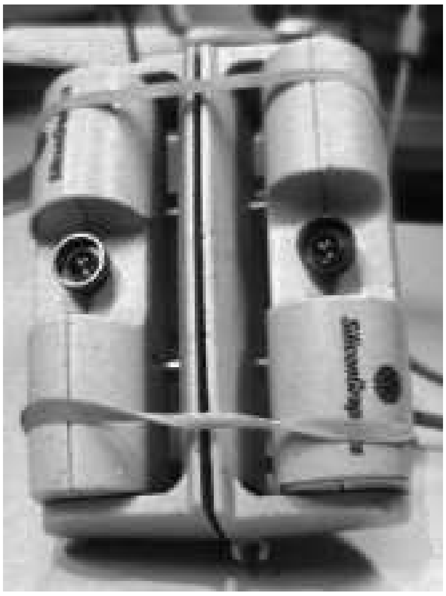

Figure 3: A stereocam formed by two Indycams (Figure 1) strapped back to back.

2.3 Results, Problems Highlighted and Proposed Solutions

Several stereo images were taken with this system, one being particularly good, numerous promising ones and several unsuccessful. The best image was of a box angled to clearly have a corner nearer to the camera than the diagonally opposite corner, in an effort to perceive the depth of the box. Despite one of the cameras being slightly out of focus, the depth of the box was perceived quite well in the (stereo) displayed image.

One notable problem with this early system was that the left and right eye images were often not taken simultaneously. This is because the system was constructed from shell scripts and the Unix `rsh` utility, both of which can be slow and dependent on the system and network loads, thus resulting in uncontrollable (and unpredictable) time differences between the capturing of the left and right images.

2.3.1 Simultaneity

This is the problem of simultaneity between left and right images, and there are two parts. The first is ensuring that left and right images are captured simultaneously (or very close to one another) when the input is a pair of cameras. The second is ensuring that these images are subsequently displayed simultaneously. These problems are together solved by *correlating* frames in a movie.

The latter problem has the property that frames captured at slightly different times may still have a satisfactory appearance. As such, provisions should be considered which allow the output end to decide which frames are to be correlated.

Two main techniques for correlating frames are available — identifying correlated left and right frames with a common tag, and correlating the initial frames and subsequently pairing the i-th left and right frames together. Determining the exact advantages, disadvantages and suitability of these techniques will require experimentation with the system.

When tagging frames the most obvious tag to use is *systime*, the number of seconds since the epoch. Modern systems are able to record this with a suitably high precision (easily supporting 30 frames per second), but this would lower portability somewhat. Also, this would introduce the question of how tags would be embedded into movie files. However, it would allow either the input or output ends to decide how close frames must be in order to be correlated. It would further aid peripheral functions, such as having two cameras begin capturing at the same systime, and determining network latencies in the system.

Another possibility is to have small heartbeat style packets sent across the network between capturing cameras, with each packet having a unique (for example, monotonically increasing integers) identifier that is subsequently used to tag the frames. However, this could place an excess strain on the network in question, possibly degrading the performance of the rest of the system, and it is also overly susceptible to varying network speeds and latencies.

Another option is correlating the first frame, and then assuming that all subsequent frames are correlated (that is, using a "clapper-board" technique). This would be suitable, for example, where the frame rate could be set to a fixed value. Also, the initial frames could be correlated by specifying a systime at which they are taken, or by sending a small heartbeat packet across the network, borrowing ideas from earlier solutions.

2.3.2 Network latency and throughput speed

When the network is involved in the system, it will almost certainly prove to be the system's greatest enemy. There are two parts to the problem — network latency, the delay between sending and receiving data which increases as the amount of traffic on the network increases, and throughput speed, the (total) rate at which data can be sent across the network.

A large network latency will impose an overall latency in a Stereocam system running across a network, as occurs in video-conferencing systems. The latency here is somewhat more sinister, though, as (usually) two frames must be received from different computers, each of which may have different latencies. Large, differing or varying latencies will lower the usefulness of any heartbeat packets in the system and may allow frames to fall out of synchronisation if some kind of tagging technique is not in use to correlate images (as in Section 2.3.1).

Limited network throughput speeds serve to introduce a pseudo-latency when transmitting larger amounts of data (due to the finite size of the data and network speed). This restriction on the size of transmitted data can impose a subsequent restriction on the size of the image being displayed across a network, in terms of its dimensions (including colour depth)[6]. Alternatively, some form of compression may be employed to allow the images to require less data to be transmitted.

2.3.3 Image quality

Naturally, the left and right images of a stereo image must be of comparable quality for the resultant stereo image to be of satisfactory quality. This means that not only must the images be in focus, but factors such as the brightness, contrast, resolution and of lesser importance, aperture and photodetection system, must be preserved between the two images. As several of these factors are dependent on the environment of the images and the construction of the camera, the best way to ensure that these factors are kept as constant as possible is to use identical models of camera when taking the images (as in Section 2.2)[7].

[6] For example, SGI's webcams can capture any of a "full-frame" (640x480), "half-frame" (320x240) or "quarter-frame" (160x120).

[7] The shutter speed of the cameras must also be set identically on both cameras, as this directly influences the amount of light entering the aperture of a camera and hence the brightness of the resultant image. This isn't a problem, as there is rarely any need to alter the shutter speed anyway, but still deserves mentioning.

This is borne out by simply qualitatively observing images taken by an Indycam and an O_2cam — the images from the O_2cam look much crisper, more balanced and much better overall. As such, mixing images from Indycams and O_2cams would almost certainly result in poor quality stereo images.

In addition, there are at least two different models of Indycams, with one producing qualitatively better images.[8] Upon inspection, the two different models have different lenses, and most likely different CCD cameras and perhaps other componentry. Naturally, the better quality Indycams are preferred, and Indycams of the same type should be used to obtain the best overall image quality.

3 Specifications

The overall, ultimate goal of the system is to facilitate stereo vision. That is, to be a system that provides all the facilities required to display stereo images of all kinds, including the input, output and processing of stereo images.

3.1 Requirements Definition

The specific requirements, both functional and non-functional, that the system must fulfill are:

- To support both static and dynamic images, that is, both stereo images and stereo movies.
- To give a guarantee of reasonable image quality. Specifically, this means:
  - Ensuring that only corresponding left and right images (especially frames) are displayed simultaneously, that is, that the left and right component images of stereo images are properly synchronised. This is especially important for stereo movies.
  - Providing a means to couple the quality of the input cameras, in terms of brightness, contrast, frames per second, and so on.
- To support a minimal set of input devices. These are, in order of priority, to be able to read stereo images and stereo movies from:
  1. Disk files.
  2. Video input devices, most notably SGI's webcams and AV input interface.
  3. An instance of the Stereocam system running on a remote computer, across a standard TCP/IP network.
- To support a minimal set of output devices. These are, in order of priority, to be able to write stereo images and stereo movies to:
  1. Disk files.
  2. Stereo video output devices, most notably the CrystalEyes devices on SGI workstations.
  3. An instance of the Stereocam system running on a remote computer, across a standard TCP/IP network.

[8] The SGI Part Number (found on the underside) of the Indycams which produce better quality output is 013-8737-XXX, and those of the poorer quality are 024-0825-001 Rev A.

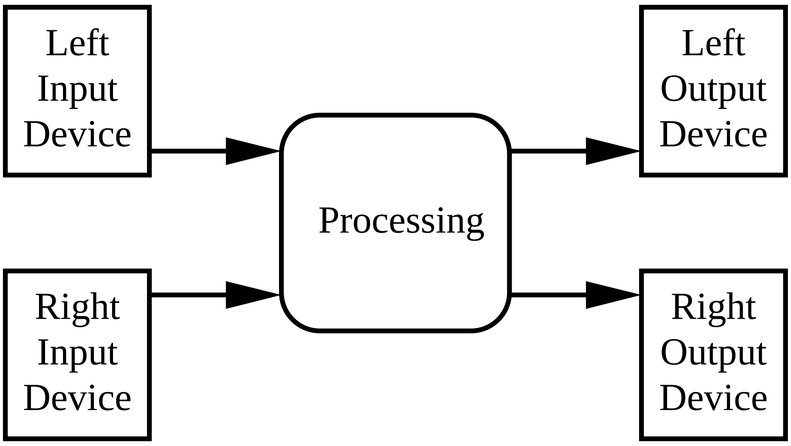

Figure 4: The general flow of data through a Stereocam instance.

- To support any processing required between the input and output stages. This is at least the rotation of images (as highlighted in Section 2.2), but may include other arbitrary forms of processing that may become evident.
- To be of modular construction, allowing new input, output and possibly processing modules to be easily interfaced and substituted for one another[9].
- To be as platform independent and portable as possible, to the point of allowing Stereocam systems running on a network of heterogeneous platforms to be able to work and communicate seamlessly[10].

The symmetry of the input and output devices is important, and will form the basis for a highly coherent and simple object oriented structure for the input and output sections of the system (developed in Sections 4.2.2 and 4.2.3).

The modularity of the system is important as it will allow development to proceed incrementally, obtaining a working base system early in the development, and then subsequently adding modules to improve the base system.

3.2 Data-flow System Model

This model of the system illustrates the ways images flow through and between Stereocam systems.

Internally, images flow from an input stage, through a processing stage and then to an output stage, illustrated in Figure 4. In general, there are two input images and two output images, but as shown in Figures 5 and 6 some instances of the system may deal with only one of the two images, or may treat the two images as a single combined stereo image. The input and output stages may be any general, arbitrary way of inputting or outputting an image (stereo or not), but would most commonly be those listed in Section 3.1.

Figures 5 and 6 show two ways in which several Stereocam instances, running on different hosts, may be used to record and display stereo images and stereo movies.

[9] Dynamic inclusion and exclusion of modules at runtime is not necessary, a modular system requiring recompilation for new modules to be added will suffice. The modularity, however, is essential to allow for expansion during development.

[10] For example, the system should allow a PC with two USB webcams to be used as input, transferring the imagery across a network to an SGI machine for output with the Crystal Eyes system.

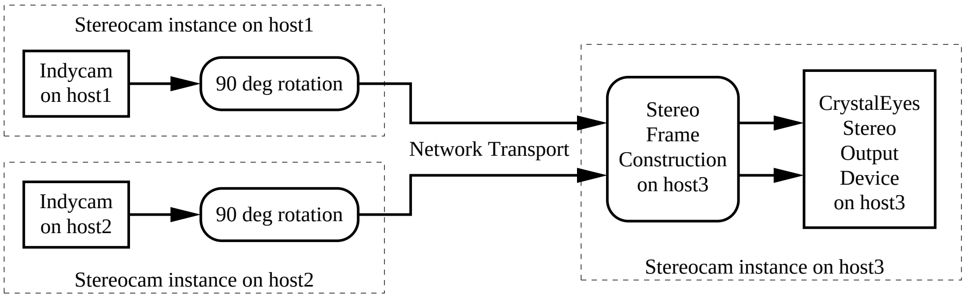

Figure 5: An example of several Stereocam instances using a network to transport images.

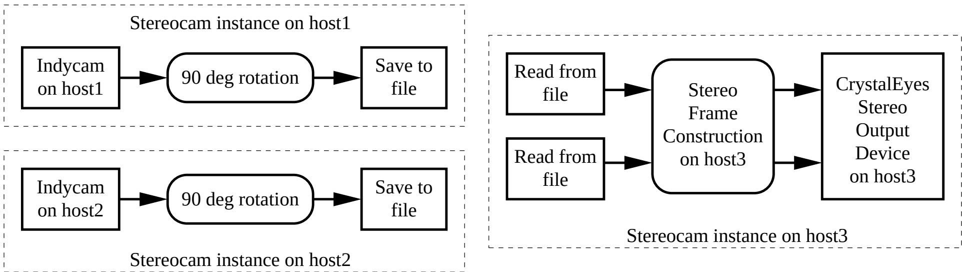

Figure 6: An example of several Stereocam instances using files to transport images.

Figure 5 is envisaged as the most common setup for the system, whereas Figure 6 is the setup described in Section 2.

4 Object Oriented Design

4.1 Class Relationship Diagrams

Figures 7 to 10 illustrate pictorially the various relationships between classes in the system. They will be described in detail in Section 4.2.

The notation used in Figures 7 to 10 is a subset of OMT (Object Modeling Technique), as described in Appendix B of [4]. The key points to note are that methods are followed by empty parenthesis, boldface text is used for class names, italics for abstract classes and methods, a triangle △ indicates inheritance and the symbol ◇—►• indicates containment (with the optional dot next to the arrow indicating the containment of multiple objects of the indicated class).

4.2 Class Descriptions

The techniques and design patterns outlined in [4] have been instrumental in the design of the Stereocam system. Specifically, the *Strategy* pattern is the underlying pattern of the input, output and processing modules. This pattern allows one to define a family of algorithms (for instance, those algorithms that input an image) which are all interchangeable, allowing the algorithm to vary independently from the clients using it. In addition, the *Factory Method* pattern will be used for decisions of exactly which input, output and processing modules should be used. This pattern allows the actual instantiation of objects to be deferred to subclasses, providing the flexibility often required with the Strategy pattern.

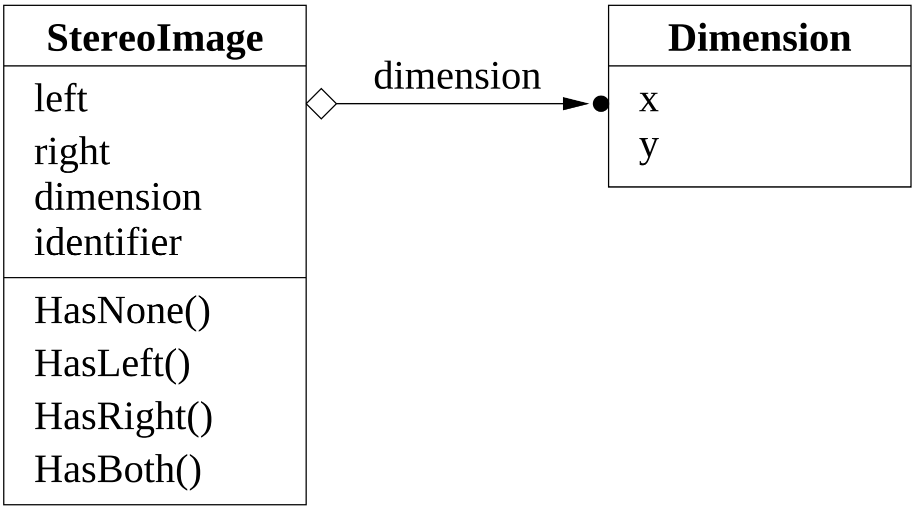

Figure 7: An OMT class diagram showing the relationships of the image related classes.

4.2.1 Image related classes

Figure 7 shows the main image related class, StereoImage, with the Dimension class.

4.2.1.1 StereoImage class

The StereoImage class is a conceptual stereo image as defined in Section 1.4. It contains the following fields:

- Data (possibly raw) for each of the left and right component images. Either or both may be omitted.
- The dimensions of the stereo image ($x \times y$ and possibly also the colour depth). This assumes that the component images are both of this size, which is a necessary condition for stereo images.
- Possibly an identifier, used to tag different stereo images (or their components) for correlation (as described in Section 2.3.1).

As either of both of the left and right component images may be omitted, the class has the HasNone(), HasLeft(), HasRight() and HasBoth() boolean-valued methods.

4.2.1.2 Dimension class

The Dimension class is simply provided as a convenient way of representing an image's dimensions in x and y, and possibly other attributes, such as colour depth.

4.2.2 Input related classes

The main input class is the abstract class Input, shown with its children in Figure 8. This class defines what it is to be an input module, and all input modules will have Input as an ancestor. It defines two abstract methods, GetImage() and GetDimension().

The GetImage() method returns either a StereoImage object which is the next (possibly stereo) image from this input source, or NULL if there are no more images available.

The GetDimension() method returns the dimensions of the images which will be obtained from this input source.

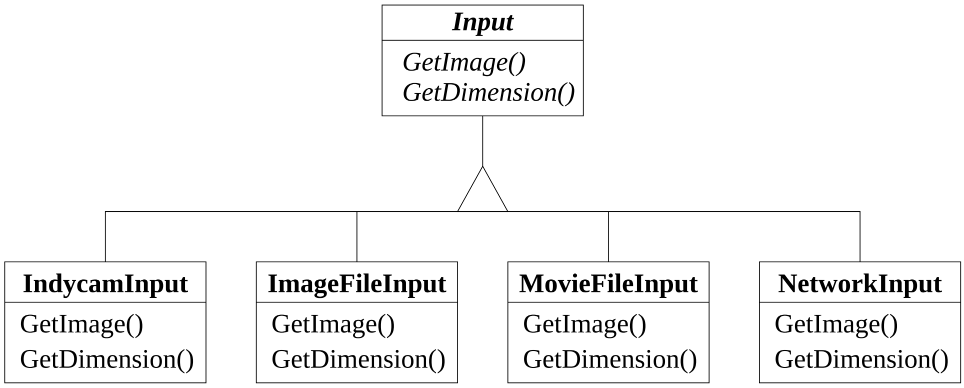

Figure 8: An OMT class diagram showing the relationships of the input related classes.

4.2.2.1 IndycamInput class

The IndycamInput class will interface the required system libraries in order to capture images from the SGI system's Indycam.

Its GetImage() method will never normally return NULL, as subsequent image captures can always be made and only in the case of, for instance, hardware errors will NULL be returned.

Indycams are able to capture images at various resolutions, specifically, fractions of the NTSC frame size, 640×480, and its GetDimension() method will return the resolution the Indycam has been set to.

4.2.2.2 ImageFileInput class

The ImageFileInput class will read a standard image file (in a common file format, such as SGI's RGB format or JPEG format) and then its GetImage() method will return this image the first time it is called, and NULL thereafter. In addition, the possibility of reading several numbered files is available, in which case it will return NULL after the last image file is read.

The GetDimension() method will naturally return the size of the image(s) read in.

4.2.2.3 MovieFileInput class

The MovieFileInput class will read a standard movie or animation file (in a common file format, such as MPEG or Quicktime MOV) and then its GetImage() method will return each of the frames of the movie in order, returning NULL when the sequence of frames is exhausted.

Again, GetDimension() will return the frame size (resolution) of the animation.

4.2.2.4 NetworkInput class

The NetworkInput class will interface with a remote instance of the Stereocam system (specifically, one using the NetworkOutput class, described in Section 4.2.3), and its GetImage() method will return the images sent from the remote Stereocam instance, returning NULL when the supply is exhausted (or the network connection is broken or severed).

The GetDimension() method will return the size of the images, as communicated by the remote NetworkOutput class instance.

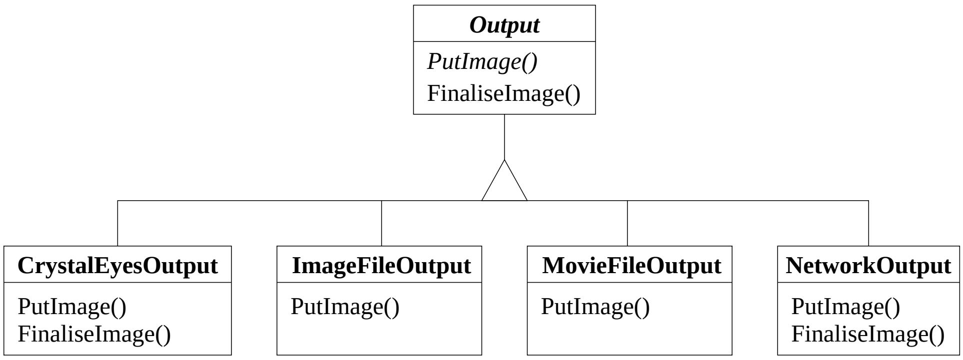

Figure 9: An OMT class diagram showing the relationships of the output related classes.

4.2.3 Output related classes

The main output class is the abstract class Output, shown with its children in Figure 9. This class defines what it is to be an output module, and all output modules will have Output as an ancestor. It defines two methods, the abstract PutImage() and the standard FinaliseImage().

The PutImage() method is called with a StereoImage object as sole parameter, and returns a boolean value indicating if the StereoImage was successfully output. A return value of false indicates that the StereoImage couldn't be output due to normal conditions, specifically, attempting to output more StereoImages than the Output object can handle.

The FinaliseImage() method is called with the final StereoImage object, when the upstream Input module returns NULL. This allows the Output module to perform any final output tasks before its destructor is called. A default implementation, of simply doing nothing, is provided in Output.

When Output objects are created they are passed the dimensions of the images they can expect to receive. This allows some state to be initialised before any images have been output, for example, creating an X11 window of the correct size.

4.2.3.1 CrystalEyesOutput class

The CrystalEyesOutput class will interface the required system libraries in order to output stereo images to the SGI system's CrystalEyes subsystem.

Its PutImage() method will never normally return NULL, as subsequent stereo images can always be displayed to the screen, as with IndycamInput, only hardware errors and the like will cause NULL be returned. The module may display the stereo image in an X11 window if possible[11], in fullscreen mode otherwise. The FinaliseImage() method will allow some degree of interactivity before the program is exited.

4.2.3.2 ImageFileOutput class

The ImageFileOutput class will write the image initially passed to PutImage() to a standard image file, returning true initially and false on subsequent occasions.

In addition, the possibility of writing several numbered files is available, which may subsequently be read in using the ImageFileInput module, or assembled into an animation file.

[11] Specifically, this requires the workstation to support quad-buffering, which is available on modern SGI workstations, such as the O_2.

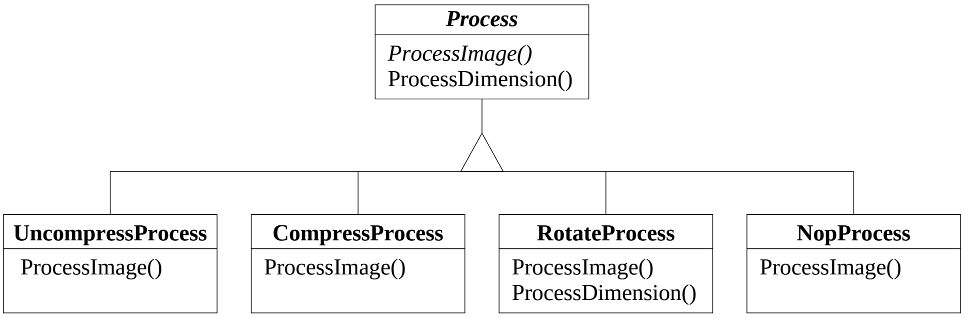

Figure 10: An OMT class diagram showing the relationships of the processing related classes.

4.2.3.3 MovieFileOutput class

The MovieFileOutput class will write the frames passed to PutImage() to a standard movie or animation file. It will return true while it is possible to write these frames and false subsequently (for example, if out of disk space, or if the file was specified at creation time to have a fixed number of frames).

4.2.3.4 NetworkOutput class

The NetworkOutput class will interface with a remote instance of the Stereocam system (specifically, one using the NetworkInput class, described in Section 4.2.2), and will transmit images passed to PutImage() to the remote Stereocam instance. It will return false when the network connection is broken or severed, for instance. The FinaliseImage() method will send a message to the remote Stereocam instance, notifying it that no more images are available.

4.2.4 Processing related classes

The main processing class is the abstract class Process, shown with its children in Figure 4.2.4. This class defines what it is to be a processing module, and all processing modules will have Process as an ancestor. It defines two abstract methods, ProcessImage() and ProcessDimension().

The ProcessImage() method is called with a StereoImage object as the sole parameter and will return the processed image[12]. If a fatal error is encountered, an exception of some kind should be raised, however non-fatal errors should simply cause the passed StereoImage to be returned unmodified.

The ProcessDimension() method is called with a Dimension object, representing an image size, and returns a Dimension object representing the dimensions of the processed image corresponding to the original dimensions given. This is provided to allow the image size obtained by GetDimension() in Input to be correctly adjusted according to any processing before being passed to Output at its creation time.

4.2.4.1 CompressProcess and UncompressProcess classes

The CompressProcess and UncompressProcess classes show a possible implementation for the compression and decompression of images. They also should define a child class of StereoImage, called CompressedStereoImage, to store a compressed stereo image, and subsequently deal with objects of this class where appropriate.[13]

[12] Which may be a new StereoImage object or may be the same StereoImage passed, after being processed.

[13] The compression of stereo images as a whole (that is, CompressedStereoImage) rather than the compression of the component left and right images, may allow significant improvements in compression to be obtained. This results from the observation that the component images of

4.2.4.2 RotateProcess class

The RotateProcess class will rotate the given stereo image's left component 90^{O} clockwise (if it exists) and the right component 90^{O} counterclockwise (if it exists)[14]. This will be used for Stereocams fashioned out of two Indycams strapped together as in Section 2.2.

The ProcessDimension() method should return a Dimension object which is the transpose of the given dimension, that is, has its x and y sizes swapped.

4.2.4.3 NopProcess class

The NopProcess class is the canonical identity operation, which simply does nothing to the input image[15].

4.3 Main Program

Following is a highly simplified rough approximation to the anticipated main loop of the Stereocam program, in C++ syntax. It highlights the two greatest benefits of using an Object Oriented methodology for the system — flexibility and simplicity. Most of the details of the system will be buried in the implementations of the various classes, and these implementations can be changed, added and removed without ever directly affecting other parts of the system.

```
Input *input;
Process *process;
Output *output;
StereoImage *image;
Dimension outputdim;

// Create the desired concrete instantiations. Eg:
input = new MovieFileInput("input.mpg");
process = new NopProcess();
outputdim = process->ProcessDimension(input->GetDimension());
output = new CrystalEyesOutput(outputdim);

while( ! image = input->GetImage() ) {
    image = process->ProcessImage(image);
    if( ! output->PutImage(image) )
        break;
}
```

5 Implementation

This section discusses the reference implementation of the Stereocam system. This implementation is intended to be a functional framework adhering to the design put forth in Sections 3 to 4, with a reasonable amount of useful modules implemented. In addition, the framework of the system is, by design, still fluid, to allow for further structural changes to occur where they are deemed necessary.

---

a stereo image are, almost without exception, very similar. This could be exploited by some sophisticated compression techniques to increase the amount of compression of the images as a whole.

Unfortunately, most of the images transported across the network would tend to be individual images (see Figure 5) and not stereo images.

[14] Or the left component counterclockwise and the right component clockwise, as required and specified at the class's creation time.

[15] The term NOP is derived from NO OPERATION, commonly used in assembly language and other areas of computing.

5.1 Overview

The implementation is centered around SGI's line of Irix based workstations, particularly the Indy and O_2 series.

The system has been written in ANSI C++, using the STL (Standard Template Library), developed using the MIPSpro version 7.3 C++ environment for Irix 6.2 and above.

For storing, manipulating and displaying images, the system is centered around SGI's ImageVision Library version 3.2.1, and for obtaining images from video input sources SGI's Digital Media Video Library version 6.5.4m is used. As such, the system requires recent SGI hardware and software to run. Irix 6.5 on an SGI Indy, Indigo^2 or O_2 (or higher) system is recommended, and the ImageVision, Digital Media and MIPSpro C++ runtime environments[16] must be installed.

5.2 Framework and Structural Implementation

The implementation of the Stereocam system followed the design put forth in Section 4. However, the implementation developed several features which do not form part of the design (that is, they are specific to this implementation). This section highlights these features where they relate to the framework and structure of the system, Section 5.5 deals with the features where they relate to specific classes in the implementation.

5.2.1 General

The first notable difference is that the class names used in the implementation have been "reversed" from those specified in Section 4, so that all the Input modules start with Input, all the Output modules start with Output, and all the Process modules start with Process. Specifically, they are now in bigendian format. This is less intuitive when conversing about the modules, but is easier to work with when programming. For example, the ImageFileInput class is actually InputIlFile in the implementation.

The Param class is used to abstract the parameters which are specified to the system and modules, as discussed in Section 5.4. Parameters are deemed to have a name and optional arguments following, where the name is stored as a string and the arguments as a vector of strings. At all points in the system where strings are used, the C++ STL string class is used, and lists of strings use the C++ STL vector class.

When creating Input, Output and Process classes, arguments are passed as a vector of strings. The specific Input, Output or Process class can then interrogate these arguments to set various options and parameters. The task of instantiating the particular Input, Output or Process class the user has selected is abstracted out to various Factory classes, FactoryInput, FactoryOutput and FactoryProcess. These classes take as input the string the user entered for the name of the module, and return an instance of the corresponding class.

The behaviour of the main program has been modified slightly from that presented in Section 4.3. The previous image is cached, and when the getImage() method of the Input module returns NULL, the FinaliseImage() of the Output module is called with the previous image. This means that when the Input module runs out of input images, the system will "hold" the last image in the output.

[16] More commonly known as an execution only environment, or EOE.

5.2.2 Image related

The StereoImage class directly uses the ImageVision library, by storing the left and right components as private ilImage objects. Accessor methods getLeft(), setLeft(), getRight(), setRight(), getBoth() and setBoth() exist to get and set the components. The hasNone(), hasLeft(), hasRight() and hasBoth() methods exist as described in Section 4.2.1. The dimension of the image is stored as an object of type Dimension, and the identifier stored of type Ident. Again, accessor methods exist to get and set both the StereoImage's dimension and identifier.

The Dimension class simply stores integers for the x and y dimensions, and provides accessor methods for both.

The Ident class is currently non-operative, and is simply defined to be an integer at this point in the project. This may prove adequate, or more sophistication may be required.

5.2.3 Output related

The return value of the putImage() method is swapped, such that it is now a request to exit the program, rather than an indication of success. Returning true indicates to the main program that the Output module is finished, and the main program should exit, whereas a return value of false indicates that the Output module is ready for the next image.

5.2.4 Processing related

The processing stage of the data flow is now constructed by a sequence or "chain" of Process modules, each of which operates on the output of the previous Process. These Process classes are stored in a C++ STL vector, and the main loop of the program is altered to run the Processes in order.

5.3 Compiling and Installing the System

To compile the Stereocam system, you first need to ensure that all the required packages are installed. These packages should be installed by the superuser, usually using the `inst` or Software Manager (`swmgr`) programs.

- MIPSpro version 7.3 C++ compiler execution only environment and development environment, the `c++_eoe` and `c++_dev` packages. MIPSpro 7.2.1 hasn't been tested, and may or may not work. Other environments, such as the GNU `gcc/g++` or `egcs` compilers also haven't been tested.

- ImageVision Library version 3.2.1 execution only environment and development environment, the `il_eoe` and `il_dev` packages. This has the Image Format Library version 1.2.2 as a prerequisite, and so the `ifl_eoe` and `ifl_dev` packages should also be installed. Earlier versions of the ImageVision Library may not work, notably version 2.5 and earlier.

- Digital Media Library version 6.5.4m execution only environment and development environment, the `dmedia_eoe` and `dmedia_dev` packages. This library is only required if the camera Input module is desired, which is highly recommended.

- The Stereocam system requires GNU make to compile. This can be installed as `gmake` by installing the `fw_make` package available from `http://freeware.sgi.com/`, or by obtaining the source to GNU make and compiling and installing it manually. The Stereocam Makefile uses some GNU specific features, and as such the Irix make utility will not work[17].

Once these packages have been installed, the compilation and installation of the Stereocam system should proceed smoothly. Edit the Makefile (and Makefile.IRIX), making changes where necessary. Then compile the system by running gmake. After compilation, a `stereocam` binary should exist which can run on mips3 architecture machines (Indy or better) which are running Irix 6.2 or higher. If desired, this can have its permissions set to allow any user to run it (eg. 755), and copied to a system-wide binary directory, such as `/usr/local/bin`.

5.4 Using the System

Once the system has been compiled and installed as detailed in Section 5.3 it can be used.

Stereocam takes a set of parameters on the command line to specify how it should run. The syntax for running `stereocam` is:

```
stereocam [ --quiet | --verbose | -- help ]
[ --input <Input Module Name> <Input Module Parameters> ]
[ --output <Output Module Name> <Output Module Parameters> ]
[ --process <Process Module Name> <Process Module Parameters> ]
[ ... ]
```

All parameters start with either a single or double dash ('-' or '--'), and are followed by zero or more arguments. Arguments are simply space separated words. Parameters can be specified in any order.

The `--quiet` parameter stops the system from outputting any messages to standard output, similarly, `--verbose` forces the system to output all messages to standard output. The `--help` parameter makes the system output basic help details to standard output.

There should be at most one `--input` parameter. If there is none, the system defaults to a null input module, which never returns any images.

There should be at most one `--output` parameter. If there is none, the system defaults to using the OutputIlStereo class, as described in Section 5.5.3.3.

There may be zero or more `--process` parameters. These parameters specify the chain of processing to be performed, as detailed in Section 5.5.4. The processes will be executed in the order they are specified on the command line.

The parameters for the Input, Output and Processing modules depend upon the module selected, and will be detailed in Section 5.5.

5.5 Class and Module Implementations

This section deals with the specific implementations of the various classes and modules in the system. Refer to Section 5.6 for some examples on how to use the most useful of these modules.

5.5.1 General classes

5.5.1.1 Param

The Param class is an abstraction of a command line parameter. It has a name and a series of arguments which is stored as a vector of strings. It can parse a vector of strings, each of which is a word, into a vector of Param objects, where each Param object represents text of the form `-name argument1 argument2 ...`

The Param class is the basis for all the parameter and option handling in the implementation, and is used in a recursive fashion. For example, the main program

[17] It should be possible to modify the Makefile to have it work with the Irix make utility, if GNU make is unable to be installed.

uses it to break the command line into sections bounded by the parameters acceptable at that level, namely --input, --output, --process, --quiet, --verbose and --help. Subsequently, an Input, Output or Process module may then use it to break its arguments into sections bounded by its acceptable parameters, and so on.

5.5.1.2 FactoryInput, FactoryOutput, FactoryProcess The Factory classes, FactoryInput, FactoryOutput and FactoryProcess are used to abstract the selection of specific Input, Output and Process classes given a string which is considered to be the name of the module.

Specifically, these modules contain a choose() method, which passes the name of the module as a string, and the arguments for the module as a vector of strings. The name is used in a standard if-else-if ladder to select which specific class is required, this is then instantiated using the given arguments and returned to the caller.

Presently, this is only used by the main stereocam program and the InputDual module.

5.5.2 Image modules

5.5.2.1 Input

The Input class is the parent class for all the Input modules in the system, and so it is conceptually abstract — it defines its destructor, the getImage() and getDimension() methods as virtual. However, it provides a minimal null implementation of an Input module, which always returns NULL from getImage() and always returns a dimension of 10×10 pixels from getDimension().

This module is the default when no input module is specified, has the valid name of default, and takes no parameters.

5.5.2.2 InputIlFile

This module reads one or more images from a file, using the ImageVision ilFileImg class. It can read any file format supported by ImageVision, such as SGI RGB, GIF, Jpeg, Tiff, etc.

Valid names for this module are file, image or ilfile.

It requires one or two parameters to be specified, being the filename of the left and/or right image files. If only one filename is specified it is assumed to be the left image file.

Further, it can read a series of numbered files by specifying a printf-style decimal format code in the filename. This will be expanded to the current frame, and the module will continue reading images until no more exist. For example, if the filenames specified are foo-left.%03d.sgi and foo-right.%03d.sgi, then the module will read the files foo-left.000.sgi, foo-right.000.sgi, foo-left.001.sgi, foo-right.001.sgi, foo-left.002.sgi, foo-right.002.sgi, and so on until no more files remain.

5.5.2.3 InputVlCamera

This module captures images from the Indycam or O_2cam attached to the system (if any), using the Digital Media library. It is very versatile and supports many options.

The images as obtained from the Digital Media library are stored as an interlaced array of RGB values in memory, and the ImageVision ilMemoryImg class is used to read this array into a format suitable for the ImageVision library.

One downside to the module is that it captures images using what is known as "discrete" mode, as opposed to "continuous" mode. As the names suggest, discrete mode is suited to capturing single images, and continuous mode is suited to capturing a stream of images for a movie. The use of discrete mode lowers the performance slightly, but makes the code a lot simpler.

Valid names for this module are `camera`, `vlcamera`, `cam`, `indycam`, `o2cam`, `video`, `videocamera`, and `videocam`.

The `-zoom` family of parameters specify the size of the captured image and the default is a half-sized image. A full-sized image can be specified with `-full` or `-framesize full`, a half-sized image with `-half` or `-framesize half` and a quarter-sized image with `-quarter` or `-framesize quarter`. Finally, a zoom fraction can be specified as `-zoom` ***numerator denominator***, where the zoom fraction is $\frac{\text{numerator}}{\text{denominator}}$, or with the shortcut `-zoom` ***denominator***, where the zoom fraction is $\frac{1}{\text{denominator}}$.

The number of frames to capture can be specified with the `-count` ***x***, `-frames` ***x***, `-countframes` ***x***, or `-numframes` ***x***, where x is the number of frames to capture. The shortcuts `-single` and `-one` capture a single image.

Synchronisation The module boasts a synchronisation feature. This is specified by `-sync` ***syncMethod syncOptions***, where *syncMethod* is the name of the synchronisation method desired and *syncOptions* are any options for the synchronisation. Currently, only the `network` synchronisation method is supported, other possible methods of synchronisation are using a serial cable and various external trigger switches.

The synchronisation is implemented such that two remote InputVlCamera modules can connect, and then send each other small messages to indicate when they should capture an image. This is achieved by running a routine which will block the process of execution until it receives a message from the other module, indicating it is time to capture an image. At this point, it send a response message and then both modules capture an image.

The network synchronisation works by designating one module to be the server, and the other to be the client. The server accepts a single optional parameter, the port number to use, which defaults to 12712. The client requires a single parameter, the name of the remote computer to connect to, and optionally also the port number.

Refer to Section 5.6.4 for an example on how to use the synchronisation.

5.5.2.4 InputDual

This module is an excellent example of how elegant a modular, object-oriented solution can be. It utilises the FactoryInput class to allow two Input modules to be used, one for the left image and one for the right. It doesn't allow itself to be used recursively as one of the left or right inputs, and thus avoids any problems with having too many inputs. It also ensures that both the left and right inputs have the same dimensions.

The only valid name for this module is `dual`.

It accepts two optional parameters, `-left` and `-right`, where each is followed by the name and parameters of an Input module (exactly the same as for `--input`). If either `-left` or `-right` is omitted, then no Input module is used for that component.

Its getImage() method is particularly elegant. It simply calls getImage() on its left and right component Input modules (assuming they were specified), and then combines these resultant images. If the left input returns a StereoImage with left and right components, only the left image is used, similarly, if the right input returns a StereoImage with left and right components, only the right image is used. Failing this, it simply uses whichever image is present from the inputs.

This module is very useful for combining different sources of images, for example, from a file and from the camera. It is anticipated that this module will be used to great effect to combine the inputs from remote machines, transmitted across the network with another (network-based) Input module.

5.5.3 Output modules

5.5.3.1 Output

The Output class is the parent class for all the Output modules in the system, and so it is conceptually abstract — it defines its destructor and the putImage() methods as virtual.[18] However, it provides a minimal null implementation of an Output module, which always returns true from putImage(), and hence requests the system exits after the first image, which is discarded.

Valid names for this module are `null`, `none` and `void`, and it takes no parameters.

5.5.3.2 OutputIlFile

This module writes one or more images to files, again using the ImageVision ilFileImg class. It can write any file format supported by ImageVision, such as SGI RGB, GIF, Jpeg, Tiff, etc.

Valid names for this module are `file`, `image`, `ilfile`, `imagefile`, `still`, `stillimage`, or `stillfile`.

It accepts many parameters, all of which are optional.

The `-filename`, `-name`, `-base` and `-filenamebase` options specify the "base" of the output filename, which defaults to `stereocam-out`. This is the left-most portion of the filenames that are used to store the images.

The `-format` option specifies the name of the format to write, and the `-extension` option specifies the extension to use at the end of the filename. If either of these is omitted, the other is used in its place, and both default to `''sgi''`, to use the SGI RGB format.

If the `-multi` or `-many` options are specified, then the module uses multi output mode. In multi output mode, the module continually writes the images passed in putImage() to file, but when not in multi output mode the module only writes the first image given to file.

The final filename used when writing a file depends on two things, whether multi mode is enabled, and whether the StereoImage given has both left and right components or only one. The format is ***filenamebase*[.*xxx*][.left].*extension***, where *filenamebase* is the base filename, *xxx* is the current frame number, padded with zeros, if multi mode is enabled[19], left or right is included as appropriate if the StereoImage has both left and right components, and *extension* is the extension selected, such as `sgi`.

5.5.3.3 OutputIlStereo

This module uses the ImageVision ilViewer class to display the given StereoImages in an X11 window, using CrystalEyes double- or quad-buffering stereo mode.

It supports several interactive features as a result of using ImageVision. Images displayed can be dragged around by using the left mouse button, the edges and corners can be moved to display only a portion of the image, with the portion being viewed can be panned using the middle mouse button. Press the `q` or `Esc` key to quit the module, or the `s` key to toggle stereo mode, the `Up` and `Down` keys to raise and lower individual images and the `Home` key to reset the image to its original size and location.

[18] It also provides a virtual getDimension() method, for convenience, to obtain the dimension of the images the Output module is outputting.

[19] To change the width of this field, specify an integer after `-multi` or `-many` on the command line.

Valid names for this module are `screen`, `display`, `x`, `x11`, `crystaleyes`, `stereo` and `ilstereo`.

It accepts two parameters, both of which are optional.

If `-stereo` is specified, then the module will attempt to change the video mode of the system to one which supports stereo when it initialises. If this isn't specified, but stereo mode is desired, press the `s` key in the stereocam window.

If `-showcomponents` is specified, then the module will display the left and right components separately where they exist in the StereoImages being displayed. (The images will display under each other, so use the mouse to see the lower ones.) Thus if the StereoImage being displayed has only a left or right component it will be displayed, but if the StereoImage has left and right components then all three will be displayed — the individual components and the combined stereo image.

The module contains a standard X-style event loop, which provides the interactivity (for example, the keystrokes above). During standard operation, that is, in the putImage() method, the event loop exits when all pending events have been handled, allowing another StereoImage to be input and processed. After all StereoImages have been displayed, that is, in the finaliseImage() method, the event loop only exits when the user presses the `q` or `Esc` keys. In this fashion, interactivity is provided while the image is updating and when the last frame is being held.

Currently, there is no way to specify the location of the `/usr/gfx/setmon` utility (which is used to change video modes), or to specify the stereo or normal video modes or the X11 display to use.

5.5.3.4 OutputSink

This module is used for debugging purposes. It is similar to the null Output module, except that it always returns false from putImage(), and therefore it continually discards any images it is given.

The only valid name for this module is `sink`, and it takes no parameters.

5.5.4 Processing modules

All the processing modules (except ProcessOrient) rely on various specialised ImageVision classes to perform operations on the left and right components of the StereoImages they process. None of the processes alter the image dimensions unless specifically stated otherwise.

5.5.4.1 ProcessCrop

This module uses the ImageVision ilSubImg class to crop StereoImages. It accepts four parameters, which are, respectively, the upper left and right coordinates of the cropped image (in the coordinate space of the original image), and then the width and height of the cropped image. All the parameters are required, and must be integers.

The valid names for this module are `crop`, `region`, `subimage` and `sub`.

5.5.4.2 ProcessGrey

This module uses the ImageVision ilGreyImg class to convert StereoImages to greyscale. It accepts no parameters, and has valid names `grey`, `greyscale`, `gray` and `grayscale`.

5.5.4.3 ProcessOrient

This module changes the orientation of StereoImages.

The main parameters are `-left`, `-right` and `-both`, each is followed by an orientation. The orientation following `-left` is used on the left component of the StereoImages, that following `-right` is used on the right components and that following `-both` is used on both components. If `-left`, `-right` and `-both` are all omitted, the default is `-both`.

Valid orientations are:

- `hflip` for a horizontal flip
- `vflip` for a vertical flip
- `rot180` or `180` for a rotation about 180 degrees (equivalent to both a horizontal flip and vertical flip, in either order)
- `rot90cw`, `90cw` or `cw` for a 90 degree clockwise rotation
- `rot90ccw`, `90ccw` or `ccw` for a 90 degree counter-clockwise rotation

The module works by utilising the ImageVision iflOrientation property of ilImage objects, which defines the coordinate space of the image. A change in coordinate space, without any change in image data, is equivalent to a change in orientation of the data. Since the image data itself is untouched by this operation it is a very fast operation, and operates when the ilImage's data is accessed in the new orientation (specifically, the data is looped in a different order, corresponding to the new orientation). The module uses a lookup table which maps operations and original image orientations onto the new image orientation, which is then set in the image before it is returned.

The 90 degree rotation orientations transpose the x and y dimensions of the image, but the other orientations do not alter the dimensions of the image.

The valid names for this module are `orient` and `orientate`.

5.5.4.4 ProcessRotZoom

This module uses the ImageVision ilRotZoomImg class to arbitrarily rotate and zoom StereoImages.

The `-rotate` and `-angle` parameters are followed by an arbitrary floating point rotation angle in degrees counter-clockwise to rotate the StereoImage. The default angle is 0.0.

The `-zoom` parameter is followed by the floating point zoom (scaling) factor. Alternatively, the `-horizontal` (or `-horiz`), and `-vertical` (or `-vert`) parameters can be used to specify independent horizontal and vertical zoom factors. The default zoom factor is 1.0.

The image dimensions are scaled according to the specified zoom factors, however, it does not take into account free rotations. This means that initially, some of the rotated image is not viewable, however, all the image data is there.

The valid names for this module are `rotzoom`, `zoomrot`, `rotate`, `rot`, `rotation`, `freerotate`, `zoom` and `scale`.

5.5.4.5 ProcessSaturate

This module uses the ImageVision ilSaturateImg class to alter the saturation of StereoImages.

It accepts a single optional parameter, a floating point number which is the amount to multiply the existing saturation by. The default is 1.0, which is no change. A value of 0.0 has the same effect as ProcessGrey, values between 0.0 and 1.0 will make the colours in the image duller, and values above 1.0 will make the colours more vivid.

The valid names for this module are `saturate`, `saturation` and `sat`.

5.5.4.6 ProcessSharpen

This module uses the ImageVision ilSharpenImg class to apply a sharpening operator StereoImages. It accepts two optional parameters, `-sharpness` and `-radius`.

If `-sharpness` is specified it is followed by a floating point number which specifies the amount of sharpening to be performed, between 0.0 and 1.0, with a default of 0.5.

If `-radius` is specified it is followed by a floating point number which specified the size of the radius in the sharpening operator, with a default of 1.5.

The valid names for this module are `sharp` and `sharpen`.

5.5.4.7 ProcessSobel

This module uses the ImageVision ilSobelImg class to apply a Sobel edge-detection kernel to StereoImages.

It accepts a single optional parameter, a floating point number which is the amount to bias the combination of the horizontal and vertical edge-detection steps. The default is 0.0 which is no bias.

The only valid name for this module is `sobel`.

5.6 Example Usage

5.6.1 Simple usage

To simply display the input from the camera on the screen.

```
stereocam --quiet --input camera --output screen
```

To capture a single, full-sized image.

```
stereocam --quiet --input camera -single -full
--output screen
```

To capture 10 frames from the camera and write them to disk.

```
stereocam --input camera -count 10
--output file -multi -extension jpg testoutput
```

To read a pair of stereo images from a pair of files.

```
stereocam --input file imagefile-left.sgi
imagefile-right.jpg --output screen
```

To read a sequence of stereo images from files.

```
stereocam --input file testinput-left.%03d.sgi
testinput-right.%03d.sgi --output screen
```

5.6.2 Adding a process

To make the colours more saturated, zoom the image and rotate it 60 degrees counter-clockwise.

```
stereocam --quiet --input camera --output screen
--process saturate 2.5 --process rotzoom -zoom 1.5 -angle 60
```

To crop a small region from the previous example.

```
stereocam --quiet --input camera --output screen
--process saturate 2.5 --process rotzoom -zoom 1.5 -angle 60
--process crop 100 100 250 250
```

5.6.3 Dual inputs

To read a pair of stereo images from a pair of files, equivalent to the fourth example in Section 5.6.1.

```
stereocam --input dual
-left file imagefile-left.sgi
-right file imagefile-right.jpg
--output screen
```

To display the left image from the camera and the right image from a set of image files.

```
stereocam --input dual -left camera
-right file prevcapture-right.%03d.sgi
--output screen
```

5.6.4 Synchronisation of cameras

To synchronise two remote cameras to take a 10 images, rotating accordingly and then writing to image files. Assume that the server is named `serverhost` and has the right camera, and the remote computer has the left camera.

First, the server runs the following command.

```
stereocam --input camera -count 10 -sync network
--output file capture-right -multi --process orient cw
```

Then, when it is waiting for the synchronisation connection, the remote computer runs the following command.

```
stereocam --input camera -count 10 -sync network serverhost
--output file capture-left -multi --process orient ccw
```

The two computers will synchronise and attempt to capture the images simultaneously.

If these images are subsequently transferred to a single machine, they can then be played back quite simply.

```
stereocam --input file capture-left.%03d.sgi
capture-right.%03d.sgi --output screen
```

5.7 Adding Modules

There are several steps involved in adding a module to the Stereocam system. These will now be outlined for a hypothetical module, InputFoo. The steps are the same for Output and Process modules, all references to Input should simply be changed to Output or Process, as appropriate.

1. Obtain the module source code. This should be in the form of two files, a C++ header file InputFoo.hh and the source code file InputFoo.cc, which together completely define and implement the class InputFoo which inherits from the Input class. The layout and structure should be similar to existing Input modules and the Input base classes.

   (a) If the module is being written, perhaps the easiest way to obtain this skeletal structure is by copying an existing Input module. The interior code of the old module can then be removed and replaced with the implementation for the new module.

2. Add the module to the Makefile so that it will be included in future compilations. This involves editing the Makefile and adding a line

   ```
   input_obj_files += InputFoo.o
   ```

   near the similar lines. If the module requires extra libraries to be linked in, add them with lines of the form

   ```
   LINK += -llibname
   ```

   where libname is the name of the library to be linked. Similarly, extra library search directories can be added with

   ```
   LIBS += -L/usr/local/package/lib
   ```

   and extra include directories can be added with

   ```
   CFLAGS += -I/usr/local/package/include
   CXXFLAGS += -I/usr/local/package/include
   ```

3. Add the module to the FactoryInput class, to allow it to be selected from the main program. This involves editing the FactoryInput.cc file and added two sections. The first is

   ```
   #include ''InputFoo.hh''
   ```

   to get the declarations for the new module. The second is in the main if-else-if ladder of the FactoryInput::choose routine, where the existing code should be mimicked to provide names which create an instance of the InputFoo class, for example,

   ```
   } else if (name == ''foo'' || name == ''bar'') {
   i = new InputFoo(args);
   }
   ```

4. Run GNU `make` (or `gmake`) to recompile the system; the new module should then be available.

5.7.1 Dynamic runtime inclusion of modules

As this stereocam system is an initial reference implementation, the feature of having modules dynamically linked at runtime wasn't investigated. Naturally, it is the preferred way of using modules, as it would allow the main program to only be recompiled when a structural of framework change is made, and allow modules to be developed separately and included only when needed. Further, the size of the binary program is typically around 4Mb on a standard Irix C++ installation, and this size will soon become prohibitive. Dynamic runtime inclusion of modules can help this by breaking the binary into many, smaller pieces, which are only loaded when called upon.[20]

[20] It also helps where licensing and patent problems arise. For example, it would allow a proprietary, patented image processing algorithm to be implemented in a Process, and distributed in binary form only.

5.8 Problems Encountered

During the initial stages of the implementation, several false starts caused a fair amount lost time. These, along with other major obstacles encountered during the implementation, are documented here.

5.8.1 C++ environment

This is a technical issue, which concerns the C++ development environment. Originally, a less then recent version of the development tools were being used, under the assumption that they would work without any major problems. Unfortunately, this was not the case, particularly where the C++ Standard Template Library (STL), and much time was lost to debugging and isolating quirks in the C++ environment, and subsequently in upgrading the environment.

In addition, the alternative GNU C++ (g++) compiler was tried, along with the corresponding GNU C++ library (libg++). Unfortunately, these were found to be less attractive than the native SGI MIPSpro tools in terms of reliability and code generation.

The solution to this problem was to install the latest versions of the C++ MIPSpro development environment, and to ensure that all the C++ code in the system was strictly ANSI C++ compatible.

5.8.2 Unsuitable libraries and tools — autoconf, automake, imlib, graphapp and xlib

The requirement that the system be as cross-platform as possible induced the initial decision on some tools for cross-platform development and cross-platform libraries. These would also allow the development to proceed at several different sites, rather than the main SGI IRIX site at Sydney Vislab.

The cross-platform development tools desired were GNU autoconf and automake. These are tools which support a system of automatic and cross-platform Makefile generation. They have built-in functions to automatically deduce, use and if required, work-around various quirks in operating system, libraries, tools and so on. They are used in all major GNU projects and many other free and open source projects. However, they also add substantial complexity to all aspects of the Makefile and making the system, and introduce several additional steps to any full build of the system. Ultimately, this was not a goal of the system, and so was abandoned.

Originally, the cross-platform image representation and processing library, imlib, was chosen as the image library of choice. Imlib is very in some respects, but is not particularly suited to C++ programming, or to the kinds of processing desired for this system. In addition, imlib has issues when compiling on IRIX.

The graphapp library is a wrapper library around the native X11 xlib library. It simplifies issues in X11 programming greatly, and exists on X11, Windows and Macintosh platforms, and so it was hoped that it could provide the basis for the image handling sections of the system. Unfortunately, it is substantially less featured than the ImageVision library, and so eventually was removed in favour of a complete ImageVision solution.

5.8.3 Implementing asynchronous event and callback style execution in a synchronous framework

The ImageVision library, Digital Media Video Library and X11 XWindow systems are all based around asynchronous event and callback execution models. In this model, a typical application constructs its user interface, nominates routines to be called when particular events and situations arise, and then relinquishes control to a mainloop, which then handles the user interface and calls the user's callback routines at the appropriate times.

This model of execution is quite different to that of the Stereocam system, which is a synchronous model, without events or callbacks. As such, handling the ImageVision, Digital Media Video and X11 events and callbacks within Stereocam's framework posed quite a challenge. The solution found was to implement a limited, restricted, manual mainloop where a synchronous response was required. For example, interactivity in the X11 window of the OutputIlStereo module is described in Section 5.5.3.3.

Converting the entire Stereocam system to being asynchronous with events and callbacks is a possibility, discussed in Section 6.2.

6 Planned Enhancements

Development of the stereocam system is not complete. There are many improvements and enhancements that may be made and are planned for the system. These are divided into two broad categories, those which related to modules, and those which relate to the framework of the system.

6.1 Module Enhancements and Additions

6.1.1 Streaming StereoImages across a network — InputNetwork and OutputNetwork modules

The system, in its present state, is about ready to handle transferring images across the network. This would require an InputNetwork and OutputNetwork module at each end of the connection, in a similar fashion to how network synchronisation is currently carried out in the InputVlCamera module. The modules would only be required to handle one component of the StereoImages, as the InputDual module could be used at the receiving end.

Other issues include finding a method of serialisation for ilImage objects, as well as portable, network-byte order representation of ilImage objects, to allow them to be transferred across heterogeneous networks.

Finally, the modules should be able to use both the TCP and UDP families of connections. The former provides reliable transfer at the cost of transfer speed, and the latter provides higher transfer speed with no guarantees about the reliability of the transfer.

6.1.2 Registration aid for aligning dual Indycams

Satisfactory stereo images have never been taken with dual Indycams strapped back-to-back as in Section 2.2. This is most likely because no formal registration or calibration techniques have been devised or carried out. Physical aids should be developed in order to correctly align the cameras on the subject, and to attempt to ensure that variables such as image quality, colour saturation, and so on is standardised.

6.1.3 Implementing continuous capture mode in InputVlCamera

Currently the InputVlCamera module uses a discrete mode of image capturing, which introduces substantial overheads into the system for each image captured. The module should be converted to use the continuous mode of image capturing, which is preferable in terms of performance and transfer speed for the system's purpose.

6.1.4 Preloading multiple images in InputIlFile

Currently, the InputIlFile module reads each image on demand, that is, when it is requested. This can be particularly time-consuming (wall-clock) when the images are large or have slow decompression algorithms involved. This results in an animation playback which is somewhat jerky and slow.

A parameter for this module to preload all its images into memory upon initialisation would be highly desirable, as it would allow the animation playback to proceed much faster, with much more aesthetically pleasing results (assuming enough memory is available).

In addition, a "sleep" parameter to control the time between display of subsequent images during playback would be of value in the case of an animation which plays back too quickly.

6.1.5 Additional synchronisation methods in InputVlCamera

Currently the InputVlCamera module only supports synchronisation via a TCP network, some other methods of synchronisation would be desirable. Most notable are:

- Serial connection, which uses a null-modem, RS-232 serial connection between the two computers to communicate the synchronisation messages.
- Serial external trigger, which would use a double-pole, single-throw (probably momentary) switch attached to the serial ports of the computers to trigger when images are taken.
- Standard external trigger, which would use the external trigger found on some webcams (such as the O_2cam) to trigger the capturing of the images.

6.1.6 Synchronisation extension in InputDual

Currently the InputDual module simply pairs the i-th images from each of its left and right Input modules together. A desirable extension would be to have InputDual examine the *identifier* fields in the StereoImages it receives, and then pair stereo images together based on this. This would most notably require handling the case where frames are delayed, out-of-order or simply missing, presumably with some form of caching or interpolation scheme.

6.1.7 Animation file input and output modules

There is currently no support for the input or output of any animation or movie file formats, such as MPEG, Quicktime or SGI movie files. This would be desirable to have, and could probably be provided by the Digital Media Library.

6.1.8 Fullscreen mode in OutputIlStereo

Sometimes the StereoImages, as displayed in the OutputIlStereo module, appear too small in the window. A handy feature would be an option for OutputIlStereo which automatically zooms the image to be displayed using the full-screen.

6.1.9 Investigation of alternative or better video packings in InputVlCamera

Currently, the default 8 bit colour depth, RGBA (Red, Green, Blue, Alpha) format video packing is used when capturing images in InputVlCamera. Video packings haven't been investigated at all, and so it is possible that more suitable or better video packings may exist, and may result in higher quality captured images.

6.1.10 Non-orthodox Processes

There exist a set of "non-orthodox" processes which don't simply apply a process to the left and right components of a StereoImage, but rather use the left and right components together, often to create a single image which uses information from both components combined in an non-standard, often ingenious way.

For example, Paul Cronin from the Physical Optics department of the School of Physics, University of Sydney, has developed an algorithm which allows video phone users to appear to be looking directly at their party. The problem is typically that the video camera in such scenarios is on the side of the screen, but the conversing party is displayed on the screen. When the user looks at their party, the camera records their image from the side, and so to their party the user appears to be looking to one side.

The algorithm takes a webcam on either side of the screen and returns a single image, which is the image of the user looking directly at the screen, reconstructed from the two composite images. This eliminates the problem described above, and would allow both conversing parties to appear to be looking directly at one another.

This is not a traditional piece of processing, as would be implemented in the Stereocam system, yet it still fits into the framework nicely. It would be desirable to have at least one of these kinds of non-orthodox processes, in order to showcase the flexibility and genericity of the Stereocam design and framework.

6.1.11 Stereo JPEG files

There exists a "stereo JPEG file format"[21], which is reasonably standardised and allows a pair of stereo images to be stored in a single JPEG image file. Compression by cross-correlating the two image may or may not be achieved. Extending the ImageVision Library to be able to read and write this format would be of substantial benefit, allowing "standard" stereo JPEG files to be interchanged with other pieces of stereo-JPEG enabled software.

6.1.12 Test Pattern Input Module

The default input module is a null module, that is, it never returns an images. A more useful module, in terms of developing and debugging Process and Output modules, would be an Input module which displays a test pattern, perhaps one which moves over time. This would probably be a simple pattern of coloured vertical bars, similar to a video test pattern.

6.2 Framework and Structural Enhancements

6.2.1 Synchronisation abstraction

Currently, the synchronisation found in the InputVlCamera module has not been implemented in the best or most general possible way. It is not easily extensible and is completely procedural rather than object-oriented as it should be.

[21] The two terms JPEG and JFIF are often confused, to the point of synonymity, and JFIF is almost unheard of. JPEG stands for Joint Photographic Experts Group, and JFIF stands for JPEG File Interchange Format. The JPEG format describes a compression algorithm for lossy compression of continuous-tone images, whereas JFIF describes the file format used to store images compressed with the JPEG technique. This distinction is important for issues such as stereo JFIF files.

The obvious solution, then, is to abstract out the idea of a Synchroniser into a general, abstract class, which then has concrete implementations, such as the already implemented network synchroniser. In fact, it may even be possible to extend the idea further, and to put Synchronisers on the same level as Inputs, Outputs and Processes, where the Synchroniser is specified on the command line in the same fashion, and so on.

6.2.2 Pipelining

Pipelining would be an extremely valuable addition to the system, and would improve performance considerably. Pipelining is the situation where one or more images are input and transferred whilst the current image is being processed and output. The best solution would probably be a three-stage pipeline, with an image in each of the three main stages of the system — input, processing and output — and the images would flow through the "pipe" in that order. This would almost certainly be best achieved with the use of threads of execution[22], with shared memory if necessary to reduce the amount of memory being copied.

6.2.3 Batching

Batching is quite similar pipelining, and the two are actually related and good complements. Batching is a technique where many, possibly all, images are first read directly into memory. They are then processed in-place in memory, and finally output. This would increase performance, particularly where video input is concerned, at the expense of memory and instantaneous output (and therefore also most interactivity, if any).

This idea is a generalised version of the preload feature suggested for the InputIlFile module in Section 6.1.4.

6.2.4 Asynchronous operation with events and callbacks

It has been noted in Section 5.8.3 that problems arise due to Stereocam's synchronous nature, in the asynchronous world of the ImageVision Library, Digital Media Library and the XWindows system.

As such, an enticing possibility is to convert the entire framework and design to be asynchronous. This would allow it to slide seamlessly into the main XWindow event loop, in the same way that the ImageVision Library does currently. The conversion would be fairly straightforward, once the asynchronous design has been devised, since much of the existing code can be rewritten. For example, the code inside the getImage() methods in the Input classes would find a home in some sort of "NewImageReceivedEvent", generated by the Input modules asynchronously as the system calls back the methods for receiving new data from the camera.

6.2.5 Multiple input $\rightarrow$ processing $\rightarrow$ output streams

Currently there is only one input $\rightarrow$ processing $\rightarrow$ output stream present in the system. However, the motivation for having more is that the ilViewer class utilised in the OutputIlStereo module can very easily display any number of arbitrary images. As such, it might be possible to have several of these streams proceeding together, with the output window displaying all of them together. Output module which cannot handle more than one input stream may have their input streams blended in some manner, for example, by tiling the images from the input streams in a mosaic.

[22] Also known as *lightweight processes*.

However, this would require far-reaching changes throughout the framework, including changes to the method of specifying modules and parameters and the way the Images pass through the system in their currently straightforward fashion.

7 Acknowledgements

The author would like to express deepest gratitude and thanks to several people, without whom this project would not have been realised.

- Ben Simons, Systems Manager, Sydney Vislab, for providing the initial inspiration for the project, and immeasurable technical help, guidance and feedback throughout the course of the project.
- Professor Ross McPhedran, Director, Sydney Vislab, for supervising the project and allowing it to be developed at Sydney Vislab.
- Dr Alan Fekete, Science Faculty Talented Student Program (TSP) Coordinator, University of Sydney, for support and always maintaining flexibility in the TSP.

This work was produced using the Sydney Regional Visualisation Laboratory which is supported by the Australian Research Council. In addition, this work constituted a Science Faculty Talented Student Program (TSP) Project in 3rd Year Computer Science in the author's Bachelor of Science (Advanced) degree at the University of Sydney, Australia.

7.1 Legal Notices

References

[1] K. Pulo. *The Stereocam System: Specification and Design.* June 1999.

[2] N. Schweiger (ed), SGI. *ImageVision Library Programming Guide, Reference Manual and Example Programs.* Silicon Graphics, Inc., 1996, Document Number 007-1387-050.

[3] P. Creek and D. Moccia, SGI. *Digital Media Programming Guide, Reference Manual and Example Programs.* Silicon Graphics, Inc., 1996, Document Number 007-1799-060.

[4] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, Reading, Massachusetts, 1995.

[5] F. P. Brooks, Jr. *The Mythical Man Month, Essays on Software Engineering.* Addison-Wesley, Reading, Massachusetts, 1975.

[6] B. W. Kernighan and R. Pike. *The Practice of Programming.* Addison-Wesley, Reading, Massachusetts, 1999.

[7] I. Sommerville. *Software Engineering, Fifth Edition.* Addison-Wesley, Harlow, England, 1995.