The Stereocam System: Specification and Design

Kevin Pulo, mailto:kev@zip.com.au

June, 1999
$Id: stereocam-design.lyx,v 1.10 1999/06/28 04:25:53 kev Exp $

Abstract

The Stereocam system aims to be a comprehensive system for dealing with stereo imagery for almost all combinations of input and output devices and for both static and dynamically changing stereo images.

This document details the requirements, specifications and design of the Stereocam system. It is intended to be a mostly complete guide to an implementor of such a system. It describes the most important high-level aspects of the system, while leaving much of the low-level specifics as implementation details.

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.

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.

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 1st 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.

[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.

1.4 Definitions

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

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.

[4]CCD stands for *Charge Coupled Device*, which are becoming increasingly popular in sensors of all kinds.

Figure 1: An Indycam.

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.

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

[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 2: An O_2cam.

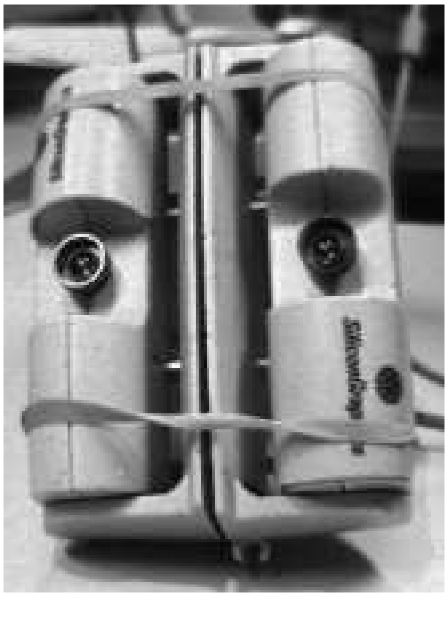

Figure 3: A stereocam formed by two Indycams (Figure 1) strapped back to back.

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.

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 bright-

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.

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.

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.

ness 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.

3. An instance of the Stereocam system running on a remote computer, across a standard TCP/IP network.

- 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[8].

- 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.[9]

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

[8] 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.

[9] 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.

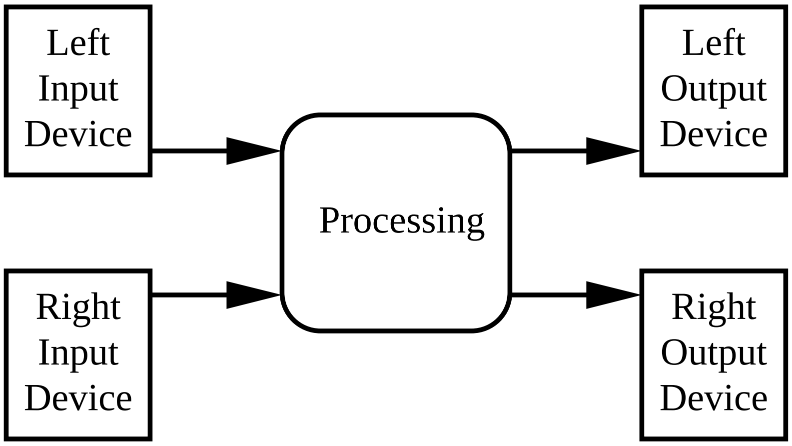

Figure 4: The general flow of data through a Stereocam instance.

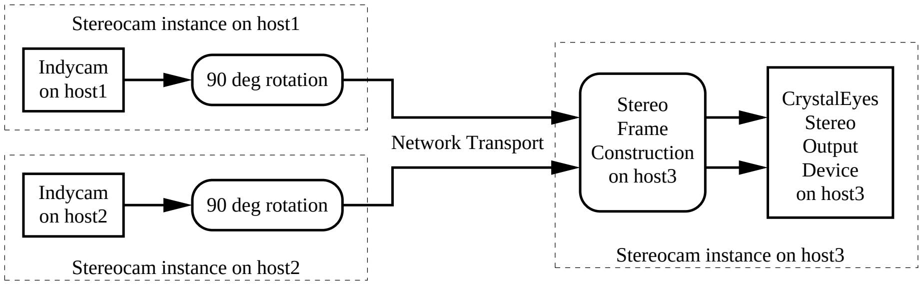

Figure 5: An example of several Stereocam instances using a network to transport images.

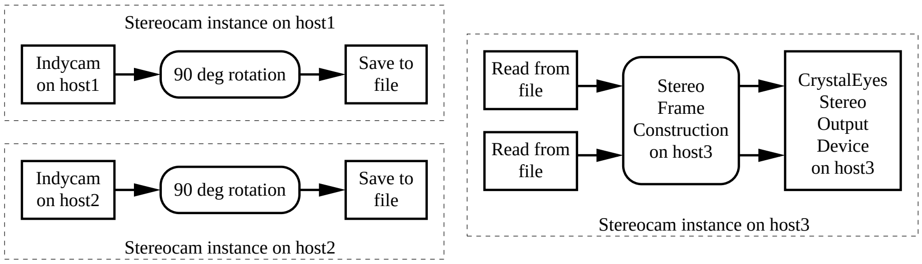

Figure 6: An example of several Stereocam instances using files to transport images.

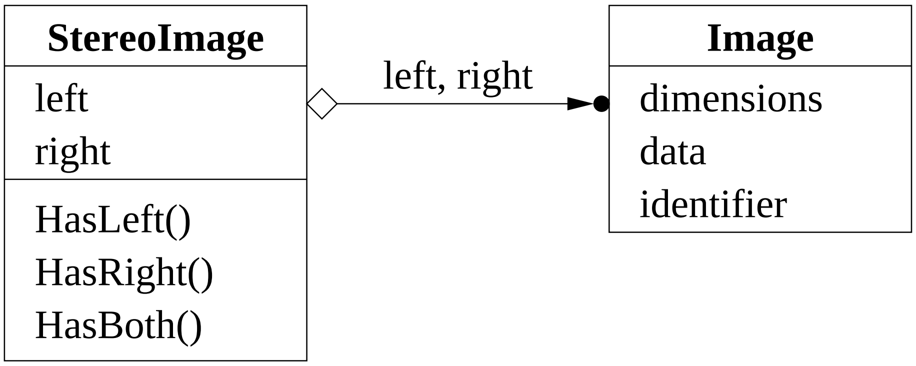

Figure 7: An OMT class diagram showing the relationships of the image related classes.

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 [Gamma et al, 1995, Appendix B]. 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 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 [Gamma et al, 1995] 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.

4.2.1 Image related classes

Figure 7 shows the two image related classes, Image and StereoImage. The Image class is a conceptual image as defined in Section 1.4. It records the image's

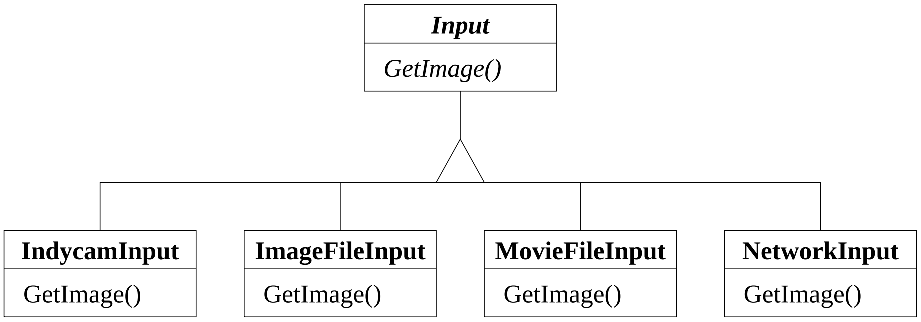

Figure 8: An OMT class diagram showing the relationships of the input related classes.

dimensions (x x y and the colour depth), the raw image data and possibly an identifier, used to tag images for correlation (as described in Section 2.3.1).

The Image class really exists to serve StereoImage, which is basically comprised of two Image objects. A StereoImage may have a left Image only, a right Image only or both a left and right Image. Exactly which Images it has at any point depends on the context, and can be tested with the HasLeft(), HasRight() and HasBoth() boolean-valued methods.

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 a single abstract method, GetImage(), which when called will return either a StereoImage object that is the next (possibly stereo) image from this input source, or NULL if there are no more images available.

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.

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.

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.

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 net-

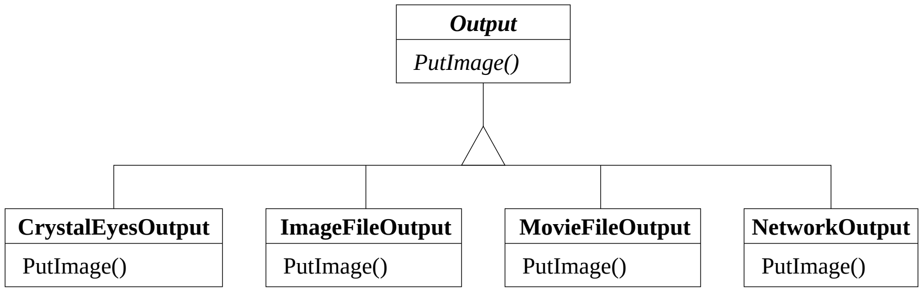

Figure 9: An OMT class diagram showing the relationships of the output related classes.

work connection is broken or severed).

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

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).

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.

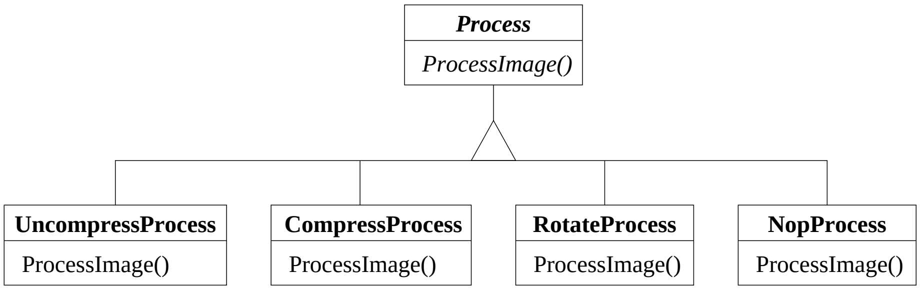

Figure 10: An OMT class diagram showing the relationships of the processing related classes.

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 a single abstract method, ProcessImage(), which is called with a StereoImage object as sole parameter. The process will be performed on the passed StereoImage and returned. 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 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.[10]

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)[11]. This will be used for Stereocams fashioned out of two Indycams strapped together as in Section 2.2.

The NopProcess class is the canonical identity operation, which simply does nothing to the input image[12].

[10] The compression of stereo images as a whole (that is, CompressedStereoImage) rather than the compression of the component left and right images (that is, two instances of CompressedImage or two instances of CompressedStereoImage, each with a single component image) may allow significant improvements in compression to be obtained. This results from the observation that the component images of 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.

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

[12] The term Nop is derived from No OPeration, commonly used in assembly language and other areas of computing.

4.3 Main program

Following is a 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;

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

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

5 Proposed Development Schedule

As mentioned in Section 3.1, the modular design of the system allows it to be built in an incremental fashion. What follows is an outline of the way in which development of the reference implementation (on the SGI Irix platform) is expected to proceed.

It is believed that steps 1 to 6 (inclusive) could be completed in the time corresponding to a 4 credit point semester course. However, due to optimistic estimating (and other factors outlined in [Brooks, 1975]), this may not be possible. In this instance, steps 1 to 5 should be completed, providing functionality similar to the Preliminary Investigation, Section 2.

1. The basic framework. This would include a main program similar in spirit (though more sophisticated in its details) to that outlined in Section 4.3, with facilities for selecting the input, output and processing modules. It would also include the abstract classes Input, Output and Process and most likely some trivial implementations of these[13].

2. The image file input and output modules. This would allow stereo images (for example, those used in the Preliminary Investigation, Section 2) to be input and subsequently output to other files. This step is non-trivial

because reading and writing images files is an important, yet difficult step. It is envisaged that some kind of image processing programming library would be sourced and used for this step. Further, it may be necessary to have a Strategy style pattern for selecting the algorithm required to input and output a given image file format.

3. The rotation processing module, that is, the RotateProcess class. This should be relatively easy, as it's a simple matrix transpose, and will allow the so far manual intermediate step of rotating the images to be performed automatically.

4. The CrystalEyes output module. This would allow stereo images read from image files to be displayed by the system, effectively providing the functionality of the Preliminary Investigation, Section 2.

5. The Indycam input module (and possibly also the O_2cam module, depending on the system interface differences). This would allow images and movies to be captured directly from the cameras. However, at this stage, only still images could really be used (outputting to image files) as it's not possible to have more than one Indycam per computer, and the network support isn't yet present.

6. The network input and output modules. This would allow still images to be taken on two remote computers with the Indycam input module, and then displayed in stereo with the CrystalEyes output module. Further, it should allow stereo movies to be displayed in this fashion, though without special optimisation work it is unlikely that the performance would be adequate.

7. The movie file input and output modules. This would allow movies to be recorded and played back without the potential performance degradation imposed by the network in the previous point. As with the image file input and output modules, it is likely that movie processing programming libraries would be sourced to aid this step.

8. Tune and optimise areas of the system which are degrading performance, especially where this prevents stereo movies from being satisfactorily displayed. This would involve finding these areas (possibly profiling the system) and then implementing the required extensions and enhancements required. For example, this may require image compression and decompression when transmitting across the network.

[13] NopProcess is an obvious choice for the trivial processing module; suitable input and output modules would be something akin to a WhiteHoleInput and BlackHoleOutput classes, which creates blank images (by analogy to so called "White holes", and the UNIX special file /dev/zero) and destroys images (by analogy to "Black holes", and the UNIX special file /dev/null), respectively.

References

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

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

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

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