Using surface curvature and topological features from shape-from-shading to improve shape-from-stereo

William Thimbleby

March 13, 2003

Abstract

This projects investigates the use of shape-from-shading to improve shape-from-stereo. Using of topological and curvature information provided by the 2.5D surface information provided from shape-from-shading, to provide better information to create an improved point-to-point correspondence between stereo images. The techniques will be compared with a simple stereo algorithm and the performance of different shape-from-shading information will be compared to direct pixel intensities.

Word count: 17847 using command line program `'wc'` on LaTeX sources

Contents

List of Figures

1 Introduction

1.1 Outline

Shading and stereo disparity are both important visual clues that the human brain uses to extract 3D data from the 2D projections projected onto the left and right eyes' retinas. The aim of this project is to try and combine these two visual depth clues, replicating part of the human brain's ability to see in a computer algorithm.

Stereo algorithms are mostly based on creating a point-to-point correspondence between two stereo images from two different view points. However these algorithms are especially bad when it comes to smooth texture less surfaces such as skin, due to the lack of any significant features. Shape-from-shading seems like the ideal solution, not only does it extract topological and curvature information from smooth texture less surfaces but it can provides rotationally invariant information, a quality necessary to create a correspondence between two different view points, looking at the same scene where the rotation of a surface is different from each point of view; an ideal combination.

Scharstein first suggested using shading information to aid stereo algorithms in [18]. Where he used the actual image gradient to create a point-to-point correspondence between images. Worthington [26] adapted Scharsteins work to use normal fields generated from shape-from-shading, instead of the gradient of the image. Using Worthington and Hancocks robust shape from shading algorithm that was outlined in [28]. The shape-from-shading algorithm was improved in [29] by applying topological constraints to it.

The outline of this project is as follows. In this section, Section-1, I look at Vision and how humans see. In Section-2 I take a general look at visual information, then specifically at stereo disparity and shading information. In Section-3 I look at shape-from-stereo and shape-from-shading algorithms in detail, and the combination of the two. In Section-4 I look at the actual methods I use for evaluating the effectiveness of topological and curvature information to improve stereo matching. In Section-5 I briefly overview the actual implementation. Creating a suitable data set for testing is then covered in Section-6. In Section-7 I cover the analysis of the results. Section-8 is a review of all the results, and an evaluation of the effectiveness of the different curvature information. Finally in Section-9 I offer some conclusions and a few suggestions for further work.

1.2 Introduction

"vision is the process of discovering from images what is present in the world, and where it is." Marr [12]

Vision is one of the strongest of our senses. In any instance we process an enormous amount of information, and we process this information with apparent effortlessness. For instance there is little conscious effort that goes into reading this page, unlike the mental effort that goes into playing a game of chess.

We rely on sight for almost everything we do; consequently the world around us has been shaped according to our ability to see. From signposts to books and videos, we have shaped our world to our strongest sense. Although it is possible to live successfully without sight, life becomes much harder. Therefore for a computer or robot to successfully function well in the world we have created, it has to be able to see. Not surprisingly then it has long been a dream of computer scientists to make computers "see".

In 1966, artificial-intelligence guru Marvin Minsky famously assigned a student a little summer project, to simulate human vision. With vision completed by the end of summer, Minsky figured, it would be onward and upward from there. Locomotion, speech, face recognition, language translation, manual dexterity, spatial orientation – all would fall to the might of symbolic logic. In short, real AI was supposedly just a few years clicks away.

However contrary to what was originally thought, when solving computer vision was famously set for that final year project by Marvin Minsky in 1966. It turned out to be one of the more hubristic predictions in the history of science. The seemingly hard problems for humans have turned out to be easy for computers and vice-versa the things we do with little effort, such as understanding languages and seeing, have turned out to be virtually impossible for computers to do. Whereas computer programs can now almost consistently beat the best human players at the brain taxing problem of playing chess. Whether this is inherent in the differing structure of our brain to that of a microchip is a question for psychologists. However what is clear is that we have had little success in replicating the ability of human vision after 30 or so years of trying.

Perhaps one of the reasons we have failed to do this so far is because our vision is so important. Perhaps our standard of what we mean by 'see' is too high. Do computer need to see as well as we do, or even in the same way? We know from robotics (and frogs) you can perform well in the world without being able to see as well as (knowledge-based) humans do. And we know from perceptual psychology, our vision is pretty idiosyncratic, we suffer from lots of illusions; do we want computer vision to have the same illusions? Is it possible to avoid them? Unfortunately all these interesting questions are far beyond the scope of this project.

Even seeing for us although we don't notice is extraordinarily complex in fact almost half our brain is thought to be used in enabling us to see, it is only that it is sub-conscious that means we don't notice. Yet it only takes about 200 msecs from seeing to the occurrence of depth perception. A duration that is very close to the time needed for the information on the retinae to reach to the visual cortex via the visual pathway. Do we even have the computer power available to us yet?

2 Vision

2.1 Visual information

In seeing we use many different bits of visual information to extract what we are actually looking at. Taking the essentially 2D information projected onto the back of our retinas, we pick out different bits of information that allows piece together the 3D world we live in. In this section I summarise something of what vision involves and then briefly cover both shape-from-stereo and shape-from-shading processes.

We call the different bits of visual information, from which 3D data can be recovered, depth cues. Below I list the most important depth cues, which we use as humans and computer scientists have tried to exploit.

- Perspective: The further away something is the smaller it looks. Parallel lines join at vanishing points and textures grow finer as distance increases.
- Parallax: Closer objects appear to move faster than those further away, as the viewer either moves their head or the objects move. This happens all the time as our eyes move around.
- Binocular Disparity: The same point in two stereo images seen from two separate view points, appear to be in different places. This disparity is directly related to the distance to the point.
- Interposition: Light travels in a straight line and objects closer to the viewer exclude those further away.
- Shadowing: An object lit from a singular direction casts a shadow along the direction from the light source.
- Occlusion: Light travels in a straight line, an object closer to the viewer than another blocks the object further away.
- Hazing: Objects distant in the real world are hazed (increasingly filtered to blue) because of dust and water droplets.
- Accommodation: As the eyes lenses adjust to look at things further away, those close up become blurred, and vice-versa.
- Shading: The angle of a surface relative to the viewer and light source, and the surface properties dictate the surface's colour and shading.
- High Level Information: Actual knowledge about the shape and structure common to objects before they are seen. This is the one visual clue that will probably not

be realised using computers for a long time. For instance the general makeup of a face, nose, eyes, ears, and mouth, is high level information, we expect to see a nose in between two eyes.

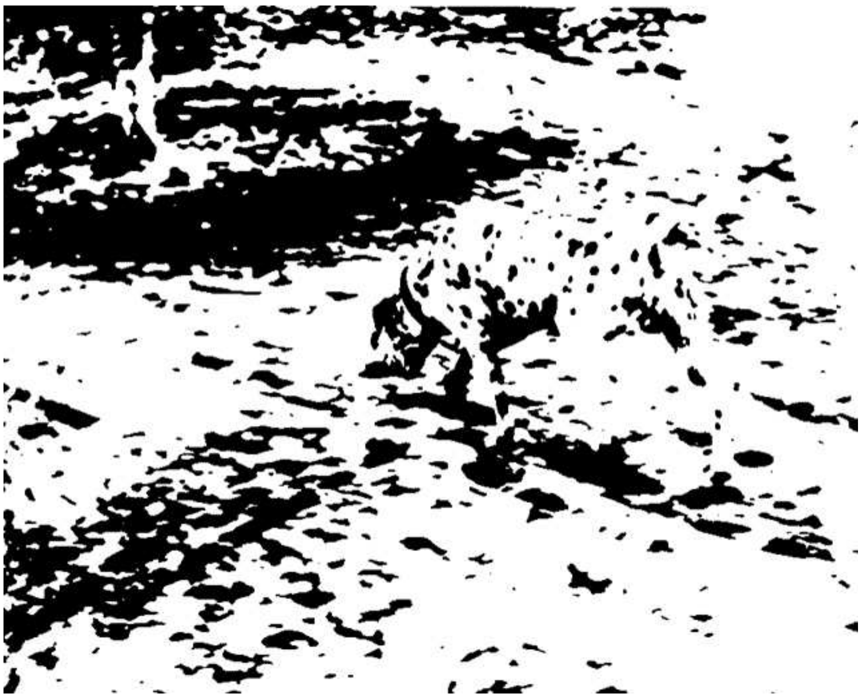


Figure 1: Dalmatian by R.C. James

For example this image, Figure-1, of a Dalmatian by R.C. James, is hard for a computer to process but a human who knows what a Dalmatian is has a much easier job.

Each of these different sources provides some information that helps create the internal model of the external world that we are actually seeing. We are surprisingly adept at determining what we are looking at from even very limited information, a few clues are all we need. For instance people can function almost as equally well with one eye as with two. One of the challenges of vision, although not addressed in this project is combining all these different sources of visual information. Most current techniques use only a single depth cue, such as stereo disparity or shading. This is partially because of the enormous complications of combining the separate and probably inconsistent depth cues. This general aim of this project is to combine both shading and binocular disparity in the hope that this will lead to an improved image extraction, specifically for texture-less objects where current stereo algorithms struggle.

2.2 Assumptions

While the above list of depth clues is by no means complete, it demonstrates a wealth of information from which 3D shape can be computed. However, scene reconstruction is an inverse problem and generally does not have a unique solution, i.e., it is ill-posed [21]. Consequently, additional assumptions and heuristics are generally needed to make the problem tractable. Some of the common assumptions are listed below.

Ideal reflectance: Surfaces in the scene are often assumed to satisfy ideal reflectance models. For instance, stereo and shape-from-shading techniques generally assume a perfect Lambertian (isotropic) reflection model with no transparency. Consequently these techniques perform poorly in the presence of specularities and other deviations from the model.

Smoothness: Imposing smoothness, or regularisation functions is also a very common method for making the problem tractable. Choosing the reconstruction that is smoothest yields a better-conditioned problem but has it's own disadvantages, for example it has the tendency to over smooth, removing sharp edges or missing thin structures in the scene.

Ideal projection: Simplified projection models like orthographic projection and the ideal pinhole projection are used to make the reconstruction equations more tractable. Consequently, techniques that use these approximations pay a penalty in terms of accuracy and are not well-suited for applications that demand high-accuracy surface measurements. This is can however be an accurate assumption, if care is taken over the production of the images.

2.3 Shape-from-shading

Shape from shading is probably one of the simplest problems to state in vision and one of the most complicated to solve. Given a single intensity image of a smooth curved object, how can the shape of the object be recovered? The shape-from-shading problem and its solution was pioneered in the vision community by Berthold Horn in his 1970 doctoral dissertation [5]. Additional constraints are needed to make the problem well-posed. It is generally solved by assuming a simplified ideal reflectance, Lambertian, and surface smoothness.

Shading on its own is a powerful depth and surface clue that leads to an understanding of what you are seeing. For instance below in Figure-2 is an artificially created image of a sphere and cube that contains only shading information. Even though only shading information is available, the human brain has no problem in determining what the objects are.

The cube and sphere above in Figure-2 are lit by a single light source at the same location as the viewer, so there is no shadowing of any part of the objects.

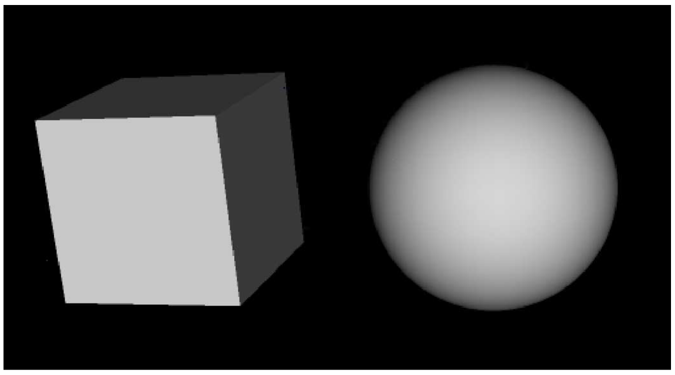


Figure 2: A cube and sphere with only shading information.

2.3.1 What is shading?

To solve the shape-from-shading problem, it makes sense as the first step to study the image formation process. Shading is the result of illumination, light energy is either emitted or reflected from the surface. Most objects do not emit light but reflect light energy from a light source such as the sun or a light bulb. The perception of an object's colour depends on the spectrum of energy in various wavelengths illuminating from the objects surface, the reflectance function of the objects surface and the spatial location and orientation of the surface and the viewer. What is seen also depends on the spectral sensitivity of the sensor. For instance eyes have different wavelength responses to an infra-red camera.

Shading is the variation in illumination across the object, which primarily occurs because the surface reflects differing amounts of light from the light source back to the viewer, depending on what angle the surface is at relative to both the light source and the viewer. For instance, the greater the angle between the surface normal and the light incident vector the darker the surface will usually appear. The actual amount of reflected light in a particular direction depends on the microstructure of the material, which can be described by a bi-directional reflectance distribution function (BRDF). That for each slant and tilt angle of the incident ray and each wavelength the ratio of the reflected intensity for each tilt and slant angle of the reflected light is computed. The reflected light can be measured by a reflection goniometer and approximated by a continuous function, these are called reflectance functions. However most of these are usually too complicated for algorithms to handle well. Luckily there are a couple of simpler models, most notably the ideal-diffuser or Lambertian surface. Which provide a good approximation for matt surfaces, and for most surfaces including skin. For a Lambertian surface the incident light is reflected equally in all directions, and the brightness is proportional to the cosine of the angle between the surface normal and the incident angle of the ray of light.

The assumption of a matt material for the right hand sphere above in Figure-3 will reproduce the wrong surface when shape-from-shading is applied. This is the disadvantage of

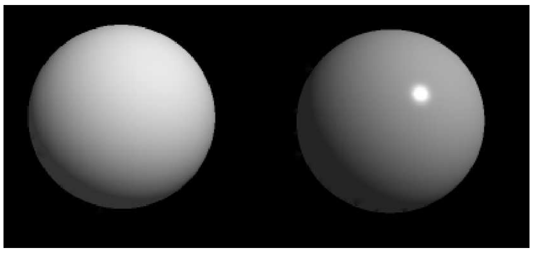


Figure 3: A matt and specula sphere.

assuming a constant surface reflectance function. Using simpler models for shading schemes will inaccurately reproduce the surface, for any but the surfaces that the assumptions are accurate. Some effort has been put into using shading information from textured objects, however there has not been a great deal of success.

Variations in brightness are not only due to surface orientation, but also to variations in surface properties. For instance the picture of a cube and sphere, Figure-2, is actually only two-dimensional and the apparent shading changes due to apparent surface orientation, are in fact due to variations in surface properties on the printed page. The human brain is remarkably good at distinguishing shading from texture; computers are not however. And even we are fooled by gradual changes in surface properties as opposed to sharp changes that are easier to distinguish. Which accounts somewhat for the success of makeup, which relies on gradual changes in surface properties to make features appear to have a different shape, hopefully a more aesthetic one.

And not only do surface properties affect the shading of a surface. In the real world, especially indoors, illumination is often extremely complex. This comes from multiple light sources and secondary illumination, light bouncing off one surface then another before reaching our eyes. And mutual illumination where the light from one surface may illuminate another, which in turn can re-illuminate the first. For instance when a room is lit from a single source it is still possible to read a book held between you and the light. The complexity introduced by secondary and mutual illumination, means that these effects are almost impossible to treat analytically. It is not surprising then that, little real progress has been made with the shape from shading problem except under the assumption of very simple lighting conditions. Horn [6] was one of the first to give careful analysis of the shape from shading problem and a single distant point light source was one of his assumptions.

Shadowing also plays a part, as it removes all shading detail in a singular point light source environment. It is however useless alone as a depth cue as it provides very limited information over small parts of an image. Compared to shading which provides information all over the image. But the edges of the shadow can provide contour information, and shading information when the shadow is from a volume light source. From a distance fine structure, such as ridges, can provide a different reflectance function because the shadowing of the ridges cannot be resolved.

It is possible to uniquely generate an image from a representation of an object under given lighting conditions and surface properties. Such as the image in Figure-2. The shading-from-shape problem is easily solvable and is used for all sorts of applications, from games to CAD. The reverse problem of shape-from-shading is much more complex.

2.3.2 Recovering shape

How does the human visual system recover shape from shading and contours? We find that some shapes are perceived very easily, while others take longer. High level information is an important part of our shape-from-shading process. Barrow and Tenenbaum showed that the line drawing of the shading pattern appears to play a central role in interpreting shaded patterns. Mingolla and Todd's study of human visual system [13] has shown that the traditional assumptions of piece-wise smoothness, a Lambertian surface, and known light source direction are invalid from a psychology perspective. In fact it is most likely that the majority of shape-from-shading algorithms, with their underlying mathematics, integration and iteration do not come close to the method the human visual system uses. Even then there are impossible surfaces for uniform lighting and reflectance function, that computers will have no hope recovering.

Shape-from-shading is usually only initially concerned with recovering the local surface orientation from the local variations in measured brightness. The actual surface, rather than the surface orientation, can be obtained but only through further processing. The problem is complicated because the brightness of a point on a surface does not depend alone on the surface orientation, but also on the spatial position and the reflectance function of the surface.

2.4 Shape-from-stereo

We use stereo vision all the time, in order to play games like tennis or football. We use it when reaching to pick up an object. It is one of the most powerful depth cues, often overriding many of the others. Predators use it to catch their prey, and the military use it to detect camouflaged objects, the camouflage only conceals a monocular image; in stereo you've still got the tell-tale bumps.

We easily manage to recover the 3D shape from 2D photographs and sketches. Shading and contours are some of depth cues we use there to recover the 3D shape. But there are also many cues that our visual system uses for recovering 3D shape from the 3D world. Stereo disparity is one of these cues, there is no stereo disparity in a 2D photograph, it exists from looking at the 3D world itself.

Stereo disparity is a powerful visual depth clue that humans use to extract 3D information from the world around them. Stereo vision emerges from the differences between what the left and right eyes see. Due to the distance between the eyes (interocular distance), the projections of a certain point in the world on the two retinae are at different positions in each eye. The difference in these two positions in each eye is called disparity and is directly related to the distance to the point in the real world. In simple situations is inversely proportional to distance and may therefore be used to compute 3D geometry. It is this baseline difference in the positions of the eyes that produces the stereo disparity. Some birds move their heads in order to increase their baseline (to increase the disparity). And radio telescopes use baselines, distance between the two sensors, the diameter of the earth's orbit, taking a picture every half year, one on either side of the sun. This is because the increased baseline provides an increased disparity and thus a greater accuracy.

2.4.1 What is stereo?

An different image is created on each of the eyes' retinas. These two images are different from each other because of the distance between the eyes, this is the very basis of any vision system in nature. The eyes and the brain attempt to overlap the two images as much as possible, by rotating the eyeballs towards a mutual focal point, thus creating an angle between the two eyes, called their parallax. Computer stereo algorithms often do not have to deal with rotation, because the input is constrained so that the algorithm can be simplified. Below in Figure-4 is a synthetic stereo pair of heads, the different image each eye would see if looking straight at a three dimensional head is positioned side by side. You can trick the brain to see the three dimensional head by looking through the page until the two heads merge, it takes practice.

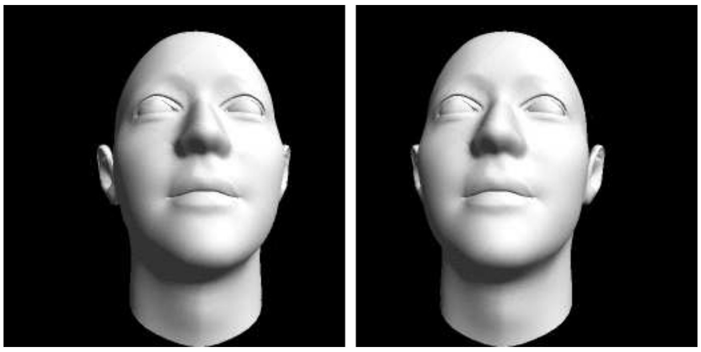


Figure 4: A stereo pair of heads.

In the auto-stereogram below in Figure-5, the only information available to the brain is that of stereo disparity. Yet when the brain is tricked into looking for greater disparity than there actually is a clearly defined 3D object can be seen. Just like the stereo pair above you can trick the brain to see a three-dimensional shape by looking though the paper until the triangles at the bottom of the auto-stereogram merge. You should see a square floating above the background.

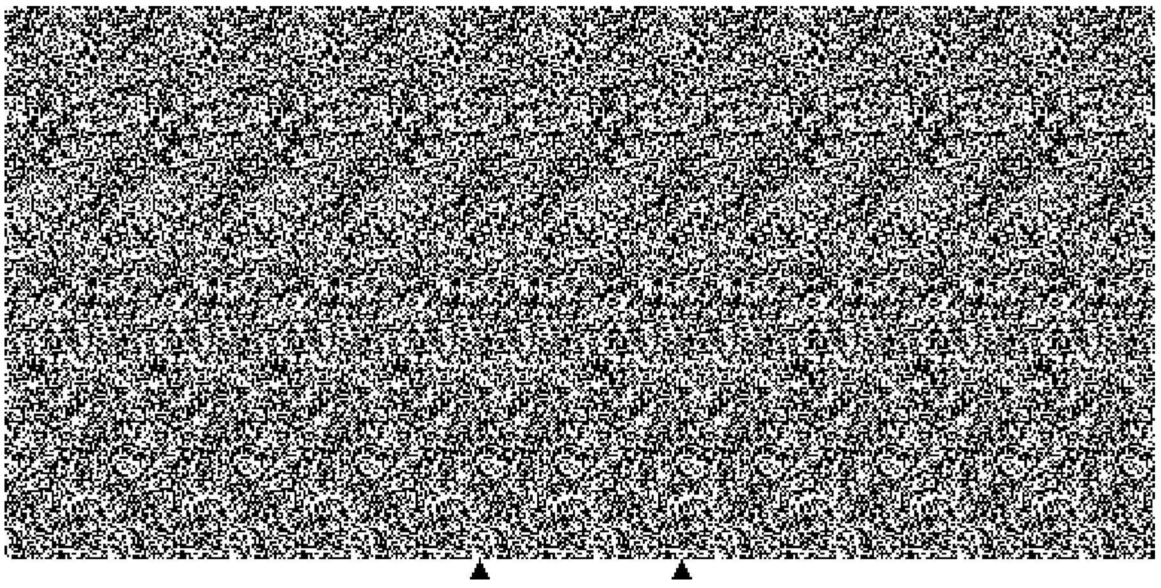


Figure 5: An auto-stereogram.

The auto-stereogram above also provides a useful example showing that stereopsis for humans is a low-level process and does not require any abstract or high-level understanding of the image. Although that is not to say a high-level understanding wouldn't improve the process. Creating a stereo cohesion between what the two eyes see therefore does not need any recognisable features in either eyes' image.

2.4.2 Recovering shape

To extract a three dimensional surface from a stereo pair, first you correspond each point in the left image with exactly the same point in the right image. The disparity is the distance between the two points and directly gives the distance to that point, from which the surface can be constructed. Easy? No, actually finding the same point in both images is very hard. You could shine a laser pointer at every point seen from one eye to find the same point from the other eye, but it would negate the point of stereo vision, a range finder would be much more use.

It is the accuracy of the point-to-point correspondence created between the left and right images that determines the accuracy of the three dimensional surface that you recover. If the correspondence is done without error then the 3D surface is without error. However there are many reasons why determining corresponding points is made more difficult. One reason is the non-Lambertian reflectance properties of most non-matt surfaces, the light reflected from the surface depends on the viewing angle and hence a specific point on the surface has a different appearance, intensity or colour, for each eye. Another is that noise that appears in any sensor either biologic or electronic, which creates differences in intensity values individual to each eye. Areas with no significant texture and areas with repetitive texture like a chess board increase the ambiguity as similar matches could be made to many different points. Occluded areas, areas that are seen in one image but in the other are another source of ambiguity because we do not know which areas are occluded before calculating the correspondence. The same effect can be seen looking through your hand close to your face, each eye sees different bits of the world behind your hand, and it is impossible to create a point-to-point correspondence for those areas, thus lacking depth information.

The term disparity was first introduced in the human vision literature to describe the difference in location of corresponding features seen by the left and right eyes by Marr [12]. Horizontal disparity is the simplest and most commonly studied phenomenon, but vertical disparity is possible if the eyes don't lie in a horizontal plane. And although in general disparity is not confined to a single horizontal axis, and has been generalized to a three-dimensional projective transformation by [2, 20]. I assume that it is confined to a horizontal axis, so as to simplify the analysis of the algorithms and also because the assumption is accurate for the images I will use. All of the images I use are taken on a linear path with the optical axis perpendicular to the camera, i.e no parallax. Therefore the original inverse-depth interpretation of disparity suffices. Thus the correspondence between a pixel (x, y) and a pixel (x, y) in the matching stereo image, is given by.

$$\begin{aligned} x' &= x - d(x, y) \\ y' &= y \end{aligned} \qquad (1)$$

Where $d(x, y)$ is the disparity for the pixel (x, y) in the left eye, and (x', y') is the corresponding pixel in the right eye. The disparity is always positive, and the corresponding point in the right image is always to the left of the same point in the left image.

2.5 Photometric Stereo

The difficulties with shape from shading may be avoided by a technique that cannot possibly have any biological twin. It works by acquiring two or more images of the object under different illuminations. Introduced by Woodham [24]. Each image provides one constraint on the normal, and therefore two images are sufficient to recover the normal up to a small number of possible solutions, and three images almost always yield a unique solution for each image pixel. Photometric stereo enables the relaxing of the strong smoothness conditions imposed by classical shape from shading approaches, and therefore yields much more reliable shape estimates.

Although not directly related to shape-from-stereo, this process has aspects of both shape-from-shading and stereo. Whether it is possible to apply a similar sort of technique to stereo images of the same object is unlikely, and if possible very complicated, but it would be very neat.

2.6 Combining shape-from-shading and shape-from-stereo

Shape-from-stereo relies on creating a point-to-point correspondence between two images. However when there is little detail or large texture-less areas finding the same point in both images can be made harder, since most stereo algorithms rely on defined texture, which is easy to match. This deficiency in shape from stereo is ideally complemented by shape-from-shadings ability to generate a surface from texture-less surfaces.

We can use the surface information from the shape calculated from the shading on a surface to create the point-to point correspondence that stereopsis requires. This can be achieved either by directly matching the needle maps like Worthington [26] or by using higher level curvature and topological information about the surface. The benefit of using topological and curvature information is that it can be created rotationally invariant and hence the same point on a surface has exactly the same value when viewed from different view points. Which is one of the necessary requirements to create an accurate point-to point correspondence.

What curvature information and how to weight the information, to achieve an improved match between stereo images, is what this project aims to discover.

3 Algorithms in detail

In this section I will cover the theory in more depth and the main ways of achieving both shape-from-shading and shape-from-stereo.

3.1 Shape from shading

3.1.1 Theory

Shape from shading is a very complex problem. This comes from the fact that there are an infinite number of surfaces and orientations, not to mention an infinite number of lighting conditions, for every pixel in the image that would give the correct appearance. Even with constraints on lighting, such as a single point light source at an infinite distance, and a constraints on the surface reflectance function, uniformly being a Lambertian surface there are still an infinite number of orientations for each point on the surface that achieves the correct appearance.

Therefore to make the problem tractable, simplifying assumptions are made. Different shape-from-shading schemes may make different assumptions. Although almost universally the assumption that brightness is independent of spacial position is used. In other words it is assumed that the viewer and the light source at far enough away from the object so that only the orientation of a surface affects its brightness. In other word the shading of an object is directly related to the surface orientation at that point independent of its spatial position. Thus most shape-from-shading algorithms do not concern themselves with extracting depth or true 3D information from the image but only the local surface orientation, this is often called a 2.5D sketch. A 3D representation of the image can then be calculated from the 2.5D sketch, or the sketch can be used to calculate curvature or topological features.

A surface is described by its height at any particular location, however the 2.5D sketch recovered from shape-from-shading is only partial, describing only the orientation. Which can be most straightforward represented by a set of surface normals. Otherwise known as a needle-map, since the normals resemble a collection of needles.

The surface orientation at any singular point on the surface can then be characterised by the surface gradient in the x and y direction, or the slant and tilt angles. $p = \partial z/\partial x$ and $q = \partial z/\partial y$. which gives the local unit surface normal as $\hat{\mathbf{n}} = (-p, -q, 1)^T$.

Since the reflecting properties of a point on the surface determine the brightness of the corresponding portion of the image, a method specifying the surface orientation from the brightness is required. For a unit surface normal, the brightness as a function of the orientation is written as $R(\hat{\mathbf{n}})$. This function is known as the reflectance map. The general shape-from-shading solution can therefore be expressed as:

$$E(x, y) \propto R(\hat{n}(x, y)) \tag{2}$$

Where $E(x, y)$ is the brightness of the image at the point (x, y) and $\hat{n}(x, y)$ is the unit surface normal at the point on the surface that corresponds to the point (x, y) in the image. Usually it is assumed that some calibration process has taken place either on the brightness or the surface needle map with respect to the other. Therefore the proportionality factor is usually ignored and the above solution is written as:

$$E(x, y) = R(\hat{n}(x, y)) \tag{3}$$

This is known as the image irradiance equation (IIR).

The function R is directly related to the BRDF of the material. Which can be determined experimentally of by a standard equation. R can be very complicated for any real surface, which is why we use simpler models that provide approximations to the real BRDFs. The most common of these solutions is the Lambertian surface, where the brightness of a point on a surface is proportional to the cosine of the angle between the surface normal and the light incident vector. The IIR of a Lambertian surface is therefore:

$$E(x, y) = R(\cos\theta) \tag{4}$$

Where θ is the incident angle. Since $\mathbf{s}$ is the global unit light source direction and $\hat{\mathbf{n}}$ is the local unit surface normal, then the reflectance function for a Lambertain surface is given by $R(\hat{\mathbf{n}}) = \hat{\mathbf{n}} \cdot \mathbf{s}$.

Shading plays an important role in human perception of surface shape. Researchers in human vision have attempted to understand and simulate the mechanisms by which our eyes and brains actually use the shading information to recover the 3-D shapes. The extraction of shape-from-shading by our visual system is strongly affected by stereoscopic processing. Barrow and Tenenbaum discovered that it is the line drawing of the shading pattern that seems to play a central role in the interpretation of shaded patterns [1]. Mingolla and Todd's study of human visual system based on the perception of solid shape [13] indicated that the traditional assumptions in shape-from-shading (Lambertian reflectance, known light source direction, and local shape recovery) are not valid from psychology point of view. One can observe that human visual system uses shape-from-shading differently than computer vision normally does.

3.1.2 Approaches

Summarised the shape-from-shading problem is to recover a piece-wise continuous surface, whose height function is $z(x, y)$ from the image brightness $E(x, y)$. The IIR unfortunately still defines an infinite number of possibilities for the surface orientation for any given brightness. Even after all the many simplifications that have been made, the shape-from-shading problem is still under constrained. Thus using the IIR alone to recover a surface would be impossible. This has lead to many different approaches being tried, each using different constraints to make the problem tractable.

Given the brightness at each point in the picture, any of the infinite choices of surface orientation could be picked at random that satisfies the IIR. However this will almost certainly not result in a continuous surface. Since the orientations correspond to some underlying surface, it is possible to show that neighbouring orientations cannot be chosen at random. The most common approach to constraining the shape-from-shading problem thus is to enforce some sort of smoothness measure on the surface. This is the constraint Horn used in his original work [6]. The shape-from-shading problem is then solved in an iterative manner. An initialisation function is used to choose an initial needle map directly from the image brightness. The needle map is then adjusted according to the constraints and smoothed iteratively. The most common approach to enforcing a smoothness measure is to use a regularisation function that penalises departure from the IIR while imposing smoothness on the needle map, this is known as an energy minimisation approach.

Not all shape-from-shading algorithms however are energy minimisation approach algorithms. The majority of shape-from-shading algorithms can be classified based on the conceptual differences in the algorithms in one of three ways. 1. Minimisation approaches 2. Propagation approaches, and 3. Local approaches. The following subsections briefly review these approaches. This is covered more fully in Zhang's survey of shape-from-shading [16].

Minimisation approaches Minimisation approaches obtain the solution by minimizing an energy function. The shape-from-shading function is formulated as a function of surface gradients. Brightness and smoothness constraints are usually added to overcome the inherent under-constrained nature of the IIR.

The brightness constraint requires that the reconstructed shape produce the same brightness as the input image at each surface point, while the smoothness constraint ensures a smooth surface reconstruction. The shape is then computed by minimizing an energy function that consists of the above constraints.

We consider the original variational approach, Horn and Brooks [7], which is expressed in terms of unit surface normals. It is a minimisation approach that uses the following error function as its energy minimisation function.

$$I = \iint \underbrace{(E(x,y) - (\mathbf{n}\cdot\mathbf{s}))^2}_{BrightnessError} + \underbrace{\lambda \left(\left\| \frac{\partial \mathbf{n}}{\partial x} \right\|^2 + \left\| \frac{\partial \mathbf{n}}{\partial x} \right\|^2 \right)}_{RegularisingTerm} + \underbrace{\mu(\|\|n\|\|^2 - 1)}_{NormalisingTerm} \, dx\, dy \qquad (5)$$

Where $\mathbf{s}$ is the light source direction and $\mathbf{n}$ is the surface normal, $\mathbf{n}\cdot\mathbf{s}$ is then the cosine of the incident angle for unit vectors $\mathbf{s}$ and $\mathbf{n}$. $(E(x,y) - (\mathbf{n}\cdot\mathbf{s}))^2$ is then the squared difference in measured brightness form the predicted value for a Lambertian surface. This first term is known as the brightness error and encourages satisfaction of the IIR, or data-closeness between the measured image intensity and the reflectance function, in this case the Lambertian IIR Equation-4. The middle term or regularising term imposes a smoothness constraint on the recovered surface normals. The final term imposes normalisation constraints on the recovered normals. μ is a Lagrangian multiplier.

The principle criticism of Horn and Brooks' method, and of similar approaches, is the tendency to over smooth. The smoothness constraint that penalises large surface orientation changes. using directional derivatives is trivially minimised by a flat surface. Therefore there is a conflict between the data and the model. The λ Lagrange multiplier typically also needs to be large for numeric stability and therefore tends to dominate the brightness error, which typically leads to over smoothing of the needle map and the lose of fine detail. Also since the approach introduces parameters in the form of the Lagrange multipliers, it requires parameter tuning.

Global minimisation approaches have many advantages, over the local and global propagation techniques since they can recover a surface from an image and a light source direction without extra information. They are also more robust and generally applicable.

Propagation approaches Propagation approaches propagate the shape information from a set of surface points (e.g., singular points) to the whole image.

In this approach there is a characteristic strip along which one can compute the surface height and gradient, provided these quantities are known at the starting point of the strip. Singular points where the intensity is either maximum or minimum are usually the starting points. At singular points the shape of the surface is either known or can uniquely be determined. The first attempt for solving SFS problems using this method seems to be mentioned in Horn [16]. In this technique, shape information is propagated outward along the characteristic strips. The direction of characteristic strips is toward the intensity gradient

The major problem with these techniques is that they require initialising with surface information, and hence require prior calculations or methods to work.

Local approaches Local approaches derive shape based on the assumption of surface type. The standard surface assumption in this approach is that the surface is locally spherical at each pixel point. Intensity and its first and second derivatives or intensity and only its first derivative have also been used to estimates the shape type. All local methods suffer from incorrect assumptions of the local surface type.

Pentland's local approach [14] recovers shape information from the intensity, and its first and second derivatives. He uses the assumption that the surface is locally spherical at each point. Pentland's approach is a linear algorithm and solves for shape by linearizing the reflectance map.

Another local approach uses linearised approximations of reflectance maps to solve the shape-from-shading problem. Pentland [15] used the linear approximation of the reflectance function in terms of the surface gradient, and then applied Fourier transform to the linear function to get a closed form solution for the depth at each point.

Yet another method makes use of computing the discrete approximation of the gradient [23]. Then the linear approximation of reflectance map as a function of height (depth) is employed. Using a Jacobi iterative scheme, this technique can estimate the height (depth) at each point. The main problem of the two latter approaches is with the linear approximation of the reflectance map. If the nonlinear terms are large, it will diverge.

Algorithmic steps For global algorithms almost all shape-from-shading schemes are iterative algorithms that follow a certain pattern.

1. initialise the needle map with general assumption
2. smooth the needle map
3. make the needle map fit the data better
4. normalise the needle map
5. go to 2

The majority of shape-from-shading techniques will produce a surface normal map that will almost certainly not represent a real surface. There are two options. 1. Try to recover the surface itself, or to try and recover a surface as similar as possible to that of the real worlds surface. Bichsel and Pentland [7] developed a propagation algorithm that is not under-constrained, but that requires prior knowledge of heights of singular points on the surface. 2. Ignore the fact that the surface normal map does not describe a real surface and simply use this information on its own to describe the object.

To ensure the surface normal map does represent a real surface, intergrability must be enforced. This means that the surface $z(x, y)$, can be recovered by evaluating the line integrals of $(pdx + qdy)$ along arbitrary contours. In fact Frankot and Chellapa [4] describe a method that enforced the integrability of the recovered surface, by mapping each estimate in step-3 to the nearest integrable needle map.

3.2 Shape-from-stereo

3.2.1 Theory

There is no standard method or algorithm for shape-from-stereo and there are many different ways people have come up with to match points in two images. And as many different ways to represent the corresponding output data.

Stereo algorithms roughly fall into two categories, dense stereo algorithms and sparse algorithms. The majority of stereo algorithms are dense stereo algorithms, i.e. they produce a univalued disparity function $d(x, y)$, or in other words dense stereo algorithms provide a disparity at each pixel. Whereas sparse stereo algorithms might match features such as edges. I will concentrate on dense stereo algorithms since the curvature and topological information from shape-from-shading is dense itself and would be useless for sparse feature based matching.

The output from a shape-from-stereo algorithm can also take many forms. The majority of stereo correspondence methods compute a function $d(x, y)$, which is the disparity for a the pixel (x, y), with respect to a reference image, which could be one of the input images, or a "cyclopean" view in between some of the images. This will be the representation I will be using. However there are other approaches; in particular multi-view stereo methods, where n different input images are used, possibly from all round the object, use multi-valued [20], voxel-based [10], or layer-based [8] representations, these are especially useful when dealing with multi-view stereo methods where each line of sight can have multiple depths. And others use full 3D models such as deformable models [22] or level-set methods [3].

I am assuming a singular horizontal disparity, which given the simple dense stereo algorithms means the distance to an point is inversely proportional to the disparity between the images of where the point appears $d(x, y)$. The mathematics is more complicate when dealing with stereo images that are rotated, for instance human eyes rotate to look at objects. As you look at an object to the side the displacement between the eyes changes and the distance to a specific point is different for each eye. Using pin-hole cameras that always face forward perpendicular to the horizontal axis joining the cameras, keeps the mathematics simple. The way disparity works for pin hole cameras is shown below:

Where d is the distance to the object, f_d is the focal distance, e_d is the distance between the eyes, x the distance from the left eye to the object, a_x and b_x are the different distances from the center of the image for each eye.

$$\begin{aligned} a_x &= \frac{x_d + x}{d} \times f_d, b_x = \frac{x}{d} \times f_d \\ \Rightarrow a_x - b_x &= \frac{e_d + x - x}{d} \times f_d = \frac{e_d}{d} \times f_d \end{aligned}$$

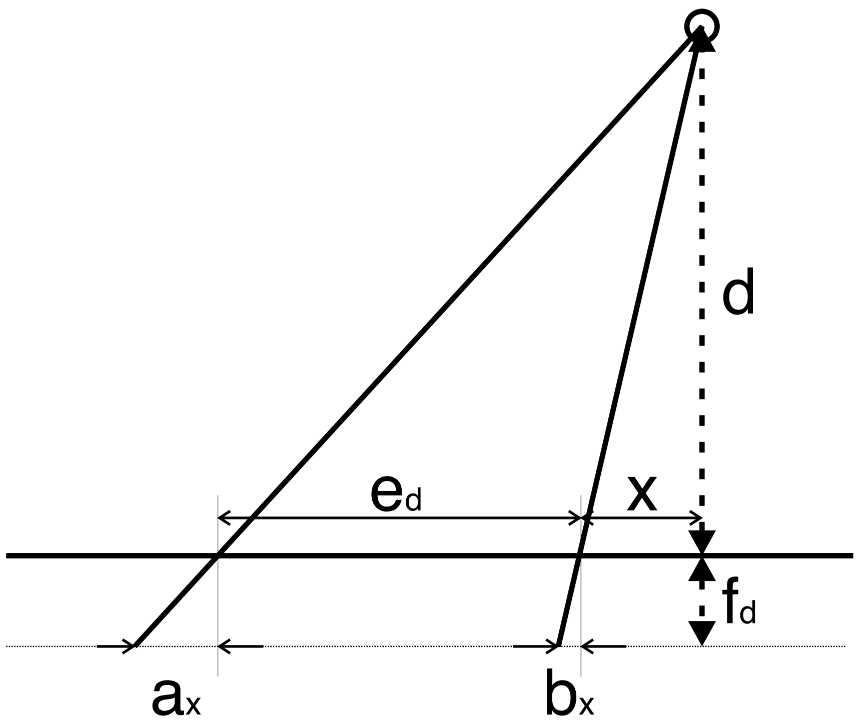


Figure 6: Stereo disparity.

$$\begin{aligned}\Rightarrow d &= \frac{f_d \times e_d}{a_x - b_x} \\ \Rightarrow d &\propto \frac{1}{a_x - b_x} \qquad (6)\end{aligned}$$

Where $a_x - b_x$ is the disparity. For a cyclopeain view half way between the two eyes, the same point would be viewed at $(a_x + b_x)/2$.

The goal of a stereo correspondence algorithm is then to produce univalued function in disparity space $d(x, y)$ that best describes the shape of the surfaces in the real world. this can be seen as finding the surface in the disparity space that has some optimal property, such as lowest cost and best piecewise smoothness. For simpler algorithms, i.e. most local algorithms this involves a simple "winner-take-all" over the confidence disparity space image (DSI). In general, a disparity space image (DSI) is any image or function defined over a continuous or discretized version of disparity space (x, y, d). In practice, the DSI usually represents the confidence or cost of a particular match implied by d(x, y).

3.2.2 Stereo algorithms

The majority of dense stereo algorithms can be broken into a subset of four generalised steps. These are fairly generally accurate for a majority of stereo algorithms, although the actual steps will depend on the algorithm.

1. matching cost computation
2. cost (support) aggregation
3. disparity computation / optimisation
4. disparity refinement.

This general breakdown of a dense stereo algorithm was suggested by Scharstein [18].

The matching cost computation is the initial comparison of individual points in each image. Cost (support) aggregation refines the initial matching cost, usually by making an implicit smoothness assumption, fairly similar to the way smoothness is used as a regularising term for shape-from-shading. Disparity computation and refinement compute the disparity function $d(x, y)$ from the matching/aggregated cost.

Dense stereo algorithms can be divide into two broad classes, local and global. Local (window-based) algorithms compute the disparity at a given point based on only the intensity values within a finite window, usually making implicit smoothness assumptions by aggregating support. On the other hand global algorithms make explicit smoothness assumptions and then solve an optimisation problem. Such algorithms typically do not perform any aggregation step, but try to seek a disparity computation that combines some sort of smoothness function with minimising a global cost function. There is also a handful of iterative algorithms that fall in-between the two broad classes, whose operation more closely resembles that of iterative optimisation algorithms.

As an example the traditional sum-of-squared-differences (SSD) is a local method. The SSD algorithm can be cleanly split into the first three steps, as it has no disparity refinement. Thus the SSD algorithm can be described as these three steps:

1. the matching cost is the squared difference between intensity values at a specific disparity
2. aggregation is done by summing the cost over square windows with constant disparity
3. disparities are then calculated by choosing the minimum aggregated cost at each pixel

3.2.3 Matching cost computation

The most common and traditional pixel based matching methods include the squared intensity differences (SD) and absolute intensity differences (AD). Which are simple intensity comparisons. Other traditional methods include binary matching costs (i.e. match / no match) [11] which is based on binary features such as edges. These are fairly limited for dense stereo output but are fairly successful in decoding random dot stereograms. And normalised cross-correlation which is fairly similar to sum-of-squared-differences (SSD). The matching cost values for each pixel and all disparities is $C_0(x, y, d)$. From the initial disparity space image.

3.2.4 Aggregation of cost

Global methods usually skip this step as it is combined with minimising a global cost function to calculate the disparities. Local and window-based methods sum or average the matching costs over a small local window or support region in the DSI $C(x, y, d)$. The support region can either be two-dimensional, or three-dimensional in x-y-d space. A two-dimensional support region is in a disparity plane, encouraging disparities to be equal or for planes to be generated facing the cameras. Or three-dimensional where the aggregation encourages slanted surfaces as-well. Two-dimensional cost aggregation can be implemented in many different ways. Gaussian convolution at a constant disparity. Summing over square windows. Shift-able windows, windows that can be anchored at different points. Adaptive window sizes. Or even just windows based on connected components.

Aggregation with a fixed support region can be computed with a singular weight function w. This works for both three-dimensional and two-dimensional regions.

$$C(x, y, d) = w(x, y, d) \times C_0(x, y, d) \tag{7}$$

A different method all together is to use iterative diffusion where aggregation is performed by some iterative function, such as iteratively adding the weighted neighbours costs to each pixel.

3.2.5 Disparity computation

Local Methods: In local methods most of the work is done in the cost computation and the matching cost aggregation. Computing the final disparities is usually trivial. Winner takes all (WTA) optimisation is the usual method, where at each pixel the 'best' disparity is chosen, i.e. the one with the minimum cost. However this only enforces a uniqueness match in only one image. The image from which pixels you choose the winning disparities, the reference image, will be created a many-to-one mapped, instead of the ideal one-to-one mapping. More than one pixel in the first image can be mapped to the same pixel in the second image. Breaking one of Marr's stereo rules of computational stereopsis. This is the major limitation of this approach and many other correspondence algorithms. However solutions do exist.

Global methods: In contrast global methods often don't bother with the aggregation step and perform almost all the work in the disparity computation step. Most global methods minimise a global energy function E for a disparity function d. Finding the best disparity that minimises E which is a combination of how well the disparity function d agrees with the input images, $E_{data}(d)$, and how smooth it is $\lambda E_{smooth}(d)$.

$$E(d) = E_{data}(d) + \lambda E_{smooth}(d) \tag{8}$$

Compare this equation Equation-8 with Equation-5 which is the global minimisation function for shape-from-shading. Both have the same structure of a data closeness term and a regularising smoothness term.

$\lambda E_{data}(d)$ can be written as:

$$E_{data}(d) = \sum_{(x,y)} C(x, y, d(x, y)) \tag{9}$$

where C is the initial matching cost or aggregated matching cost. $E_{data}(d)$ is the sum of the costs of the disparity function d at each pixel.

It is the last term, the smoothness or regularising term $\lambda E_{smooth}(d)$, that is enforces smoothness assumptions that the algorithm makes, a substitute for the aggregation in local methods. Which to make the optimisation possible is often restricted to only the pixel's immediate neighbours.

$$E_{smooth}(d) = \sum_{(x,y)} \rho(d(x, y) - d(x + 1, y)) + \rho(d(x, y) - d(x, y + 1)) \tag{10}$$

Where ρ is a monotonically increasing function of disparity difference, that can take many forms depending on what sort of surface you want to encourage. Just like the 2D and 3D cost aggregation support regions.

3.2.6 Disparity refinement

Most algorithms compute a set of disparity results that are not very smooth, or better resemble a staircase than a smooth surface. Some algorithms don't, such as algorithms that use splines to model disparity, which usually have an entirely different idea of what disparity refinement is. For some applications such as view synthesis, these 'staircase' disparities can lead to poor results, where there many tears are created between each discrete level of disparity. For most applications however it does not matter, such as robot navigation.

Still to solve this problem many algorithms apply a final stage of disparity refinement that usually involves sub-pixel smoothing. This can be calculated in numerous ways such as curve fitting, iterative gradient descent or even linear smoothing. However, none of these work accurately over surface boundaries and have to be confined to the same surface.

As well as sub-pixel smoothing there are also other refinements that you can apply to the disparity map. For instance occluded areas can be cross-checked, comparing left-to-right and right-to-left disparity maps, which is useful for algorithms that do not account for occluded areas such as local based shape-from-stereo algorithms.

3.3 Topological and curvature information

The local shape at each surface point is characterized by several scalar and vector quantities that are described below. All of which are could be used instead of pixel intensities to improve a matching algorithm.

Derived from the needle-map that shape-from-shading provides there are numerous different values that can be calculated. The simplest shape measure that you can get is the vector measure of the needle map itself. However the needle map is not itself rotationally invariant and so is possibly prone to difficulties where the objects are close to the stereo cameras and so have a large rotational difference between left and right images. It is however possible to produce a matrix capturing the differential structure of a surface that is rotationally invariant.

The differential structure of a surface is captured by the Hessian matrix:

$$\mathscr{H} = \begin{pmatrix} -\frac{\partial^2 z}{\partial x^2} & -\frac{\partial^2 z}{\partial x \partial y} \\ -\frac{\partial^2 z}{\partial y \partial x} & -\frac{\partial^2 z}{\partial y^2} \end{pmatrix} \tag{11}$$

Where the set of surface normals, $\mathbf{n} = (-p, -q, 1)^T = (-\partial z/\partial x, -\partial z/\partial y, 1)^T$ can be substituted into Equation-11. This gives the local Hessian matrix in terms of the needle map as:

$$\mathscr{H} = \begin{pmatrix} \left(\frac{\partial \mathbf{n}}{\partial x}\right)_x & \left(\frac{\partial \mathbf{n}}{\partial x}\right)_y \\ \left(\frac{\partial \mathbf{n}}{\partial y}\right)_x & \left(\frac{\partial \mathbf{n}}{\partial y}\right)_y \end{pmatrix} \tag{12}$$

where $(\cdots)_x$ and $(\cdots)_y$ are the x and y components of the parenthesised vector, respectively. However $\mathscr{H}$ is still a viewer-centred representation of the surface curvature and will provide viewer-centred curvature and topological information. Which would make any matching functions between stereo pairs more inaccurate. There is fortunately a way to create a viewpoint-invariant representation of curvature $\mathscr{C}$, developed by Woodham [25].

$$\mathscr{C} = \frac{1}{\sqrt{(1+p^2+q^2)^3}} \begin{bmatrix} q^2+1 & -pq \\ -pq & p^2+1 \end{bmatrix} \mathscr{H} \tag{13}$$

The extra terms involving the gradient can be viewed as the conditions necessary to account for the geometric foreshortening associated with viewing a surface element obliquely.

Principle curvatures The eigenvalues of the viewpoint compensated Hessian matrix $\mathscr{C}$, found by solving the eigenvector equation $|\mathscr{C} - \kappa\mathbf{I}| = 0$, are the principle curvatures of the surface. In terms of surface normals, these are given by:

$$\begin{aligned} \lambda^k_M &= -\frac{1}{2}(c_{11} + c_{22} + S) \\ \lambda^k_m &= -\frac{1}{2}(c_{11} + c_{22} - S) \\ S^2 &= (c_{11} - c_{22})^2 + 4(c_{12}c_{21}) \end{aligned} \tag{14}$$

These are the maximum and minimum curvatures on a surface at a specific point, λ^k_M and λ^k_m respectively. The integrability condition ($c_{12} = c_{21}$) is frequently used as an additional constraint for smoothness, to make the surface recovery tractable. However the Hessian in Equation-13 as an estimate does not in general result in equal values of c_{12} and c_{21}. Therefore we do not enforce intregrability at every point on the surface. However if S^2 is negative, i.e. S is undefined, then surface intregrability is enforced using $c_{12} = c_{21} = (c_{12} + c_{21})/2$ which results in a positive value for S^2 and a real value for S.

From the principle curvatures calculated from the Hessian matrix we can calculate further curvature values that might provide a better match for stereo algorithms.

Mean curvature: The mean curvature K is half the sum of the principal curvature.

$$K = \frac{1}{2}(\lambda^k_M + \lambda^k_m) \tag{15}$$

Gaussian curvature: The Gaussian curvature H is the multiple of the principal curvature.

$$H = \lambda_M^k \lambda_m^k \tag{16}$$

Shape Index: The shape index was created by Koenderink and van Doorn, is a continuous measure that encodes the curvature information of the surface. Koenderink and van Doorn [9] defined the shape index ϕ as:

$$\phi = \frac{2}{\pi} \arctan \frac{\lambda_m^k + \lambda_M^k}{\lambda_m^k - \lambda_M^k}, \lambda_M^k \geq \lambda_m^k \tag{17}$$

The shape index ranges from -1 to 1, is an angular physical measure. A convex surface point with equal principal curvatures has a shape index of 1. A concave surface point with equal principal curvatures has a shape index of -1. A saddle surface point with principal curvatures of equal magnitude and opposite sign has a shape index of 0. A "ridge-like" surface point has a shape index of about 0.5 while a "valley-like" surface point has a shape index of about -0.5. Table-1 shows the different classes and the respective bounds of the shape index.

Table 1: Topographic classes

Class	Region-type	Shape index
Dome	Elliptic	$[\frac{5}{6}, 1)$
Ridge	Parabolic	$[\frac{3}{8}, \frac{5}{8})$
Saddle ridge	Hyperbolic	$[\frac{1}{8}, \frac{3}{8})$
Plane	Hyperbolic	Undefined
Saddle point	Hyperbolic	$[-\frac{1}{8}, \frac{1}{8})$
Saddle rut	Hyperbolic	$[-\frac{3}{8}, -\frac{1}{8})$
Rut	Parabolic	$[-\frac{5}{8}, -\frac{3}{8})$
Cup	Elliptic	$[-\frac{5}{8}, -1$

Curvedness: The curvedness, R, is another measure that is derived from the principal curvatures:

$$R = \sqrt{(\lambda_M^k)^2 + (\lambda_m^k)^2} \tag{18}$$

The curvedness describes the scale of the surface independent of its shape. Planar points have a curvedness of 0 while highly curved regions have large curvedness.

Direction of maximum curvature: Lastly two more vector measures. The direction of maximum curvature as a 2D vector on the surface can be calculated from the Hessian matrix.

$$\mathbf{M}_2^k = \left[\begin{array}{ll} (c_{12}, -\frac{1}{2}(c_{11} - c_{22} + S))^T & c_{11} \geq c_{22} \\ (\frac{1}{2}(c_{11} - c_{22} - S), c_{21})^T & c_{11} < c_{22} \end{array} \right. \tag{19}$$

And from the surface normal and the 2D maximum curvature direction on the surface we can compute the 3 component unit-vector $\mathbf{M}^k$ in the image co-ordinate system. Which is $(-\mathbf{n} \times \mathbf{y}, \mathbf{n} \times \mathbf{x}, \mathbf{n})$, where $\mathbf{n}$ is the unit surface normal, $\mathbf{x}$ and $\mathbf{y}$ are the unit direction vectors in the image co-ordinate system.

4 Techniques used

In this section I cover the actual techniques used for this project.

4.1 Shape-from-shading

The actual implementation I will use was programmed by Will Smith [19]. It is still important to understand method and the mathematics behind the shape-from-shading scheme used.

A novel geometric shape-from-shading framework was reported in [28, 29, 27] by Worthington and Hancock. The framework, which rather than just penalising departure from the IIR , ensured the IIR was satisfied as a hard constraint during every iteration of the algorithm. It has been shown to recover needle-maps of higher fidelity and with greater speed than many existing methods.

The approach is a geometric one, the IIR is viewed as a cone of ambiguity on which the surface normal must fall. The axis of the cone is the light source direction and the opening angle is determined by the measured brightness at the corresponding image location. At each iteration the updated normal is free to move away from the cone. However all normals are then mapped back to the closest normal lying on the cone of ambiguity. Thus every intermediate needle map, during iteration satisfies the IIR as a hard constraint. Which makes sense seeing as the IIR is the only information available to us, this approach therefore ensures the fullest use of data possible. A side benefit by treating the minimisation function as a separate constraint to the IIR, the need for a weighting parameter, and thus parameter tuning is removed entirely.

The infinite set of surface normals that satisfy the IIR for a given point can be considered to define a cone of ambiguity around the light source direction. That is every surface normal that satisfies the IIR for a point on the surface falls on this "cone of ambiguity". This only works if the surface brightness is irrespective of the angle of viewing.

The set of surface normals that satisfy the image irradiance equation, for a Lambertian surface, i.e. $E - \mathbf{n} \cdot \mathbf{s} = 0$ a lie on the cone of ambiguity. Where $\mathbf{n}$ is the surface normal, $\mathbf{s}$ is the light direction.

This shape-from-shading scheme is a global minimisation algorithm and can be broken into to steps similar to:

1. initialise the needle map with general assumption
2. smooth the needle map

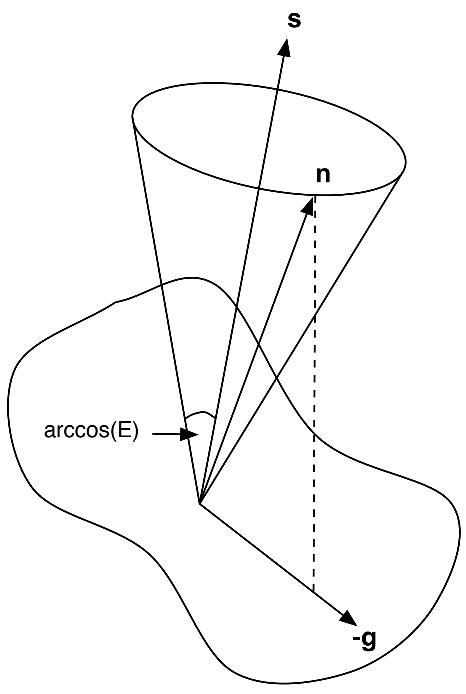


Figure 7: Cone of ambiguity

3. enforcing the IIR
4. normalise the needle map
5. go to 2

The algorithm is an iterative one, where the needle map is repeatedly refined. It takes a needle map smoothes it then reinforces the IIR as a hard constraint.

4.1.1 Initialising

The needle map has to start somewhere that satisfies the IIR as a hard constraint. This is different to the Horn and Brooks algorithm, which is usually initialised by estimating the occluding boundary normals and pointing all other normals in the direction of the light source. To satisfy the IIR we could choose any normal that lies on the cone of ambiguity.

Worthington and Hancock's method initialises each normal so that its projection onto the image plane lies in the opposite direction to the image gradient **g**. The normal points away from the direction of the brightest gradient. This results in an initialisation with an implicit bias towards convex surfaces. Bright regions are assumed to be peaks and dark region cups, the image gradient therefore points towards the peaks. For surfaces where this assumption is invalid the shape-from-shading algorithm performs more poorly.

4.1.2 Smoothing

Choosing a constraint function: Worthington and Hancock's method requires the constraint function:

$$I_C = \iint \psi(\mathbf{n}(x, y), N(x, y))\, dx\, dy \tag{20}$$

be minimised. $N(x, y)$ is a set of local neighbourhood vectors about location (x, y) and $\psi(\mathbf{n}(x, y), N(x, y))$ is a localised function of the current surface normal estimates. The size of the neighbourhood may vary according to the nature of ψ. For example for a grid of coordinates i, j the 4-neighbourhood of $\mathbf{n}_{i,j}$ is defined as:

$$N = \{\mathbf{n}_{i+1,j}, \mathbf{n}_{i-1,j}, \mathbf{n}_{i,j+1}, \mathbf{n}_{i,j-1}\} \tag{21}$$

One of the simplest choices for a constraint function is smoothness that can be written as:

$$\psi(\mathbf{n}, N) = \left\| \frac{\partial \mathbf{n}}{\partial x} \right\|^2 + \left\| \frac{\partial \mathbf{n}}{\partial y} \right\|^2 \tag{22}$$

The terms $\frac{\partial \mathbf{n}}{\partial x}$ and $\frac{\partial \mathbf{n}}{\partial y}$ are the directional derivative of $\mathbf{n}$ or the needle map in the x and y directions respectively. The magnitudes of these quantities are used to measure the smoothness of the surface, large values indicating a highly curved region. This function imposes the smoothness constraint by penalising large local changes in surface direction. In [29] Worthington and Hancock use an adaptive robust regulariser that uses curvature consistency to improve the smoothing. By using the variance in the shape index to control the width of the robust error kernel applied to the needle map to improve the algorithms performance. This is however beyond the scope of this project.

The update equation for the needle map at iteration $k + 1$ using the estimate at iteration k can be derived from ψ to give:

$$\mathbf{n}_{i,j}^{k+1} = \overline{\mathbf{n}}_{i,j}^{k} \tag{23}$$

$\overline{\mathbf{n}}_{i,j}$ is the surfacenormal that satisfies the constraint function I_C. Using the simple constraint function in Equation-22, and if the local neighbourhood is the same as that described in Equation-21, then we can simply make $\overline{\mathbf{n}}^k_{i,j}$ the local 4-neighbourhood mean of the surface normals around the pixel position i, j. The local 4-neighbourhood mean of the surface normals can be calculated as a simple mean value of the neighbour hood $\overline{\mathbf{n}}^k_{i,j} = \frac{1}{4}(\mathbf{n}_{i+1,j}, \mathbf{n}_{i-1,j}, \mathbf{n}_{i,j+1}, \mathbf{n}_{i,j-1})$, however this quickly leads to a flat surface and results in poor surface recovery. There are a few alternatives, most notably ... .These try to create a piece-wise smooth surface by smoothing everywhere except where there is a surface boundary. ... We can smooth by a simple Gaussian distribution. The Gaussian smoothing operator is a 2D convolution, that provides gentler smoothing and better edge preservation than simple local averaging. The Gaussian distribution formula is:

$$G(m, n) = \frac{e^{-\frac{m^2+n^2}{2\sigma^2}}}{2\pi\sigma^2} \tag{24}$$

where σ is the standard deviation and (m, n) is a position in the local neighbourhood relative to the current pixel $(0, 0)$. The Gaussian distribution is a circularly symmetric distribution, or an isotropic distribution. Will Smith [19] uses a simple 3×3 integer approximation to the Gaussian convolution as his smoothing function.

The severity of the smoothing is provided by the standard deviation, the larger the standard deviation the heavier the smoothing. Since the distribution is non-zero at every point, an infinitely large neighbourhood would be required for each pixel. However the Gaussian distribution is effectively zero further than three standard deviations away from the centre, $(0, 0)$, thus the neighbourhood can be truncated at this point. Thus for the most commonly used neighbourhoods, 3×3 or and 5×5 the approximate standard deviations of 0.6 and 1.0 respectively are used,

Thus for a 3×3 neighbourhood the update equation is:

$$\begin{aligned} \overline{\mathbf{n}}_{i,j} &= \sum_{m=-1}^{1} \sum_{m=-1}^{1} G(m, n) \times \mathbf{n}_{i+m,j+n} \\ G(m, n) &= \frac{e^{-\frac{m^2+n^2}{0.72}}}{0.72\pi} \end{aligned} \tag{25}$$

4.1.3 Enforcing the IIR as a hard constraint

The hard constraint imposed by the IIR is:

$$\iint (E - \mathbf{n} \cdot \mathbf{s})\, dx\, dy = 0 \tag{26}$$

which must be satisfied.

Although it is clearly possible to incorporate the hard data closeness constraint required by the IIR directly into ψ, it needlessly complicates the mathematics. The hard constraint of the IIR is therefore imposed after each iteration.

We satisfy the IIR as a hard constraint after each iteration by mapping the smoothed updated normals back to the closest normal that lies on the cone, i.e. the closest normal that satisfies the IIR. The resulting update equation for the normals can therefore be written as:

$$\mathbf{n}_{i,j}^{k+1} = \Theta \overline{\mathbf{n}}_{i,j}^{k} \tag{27}$$

The hard image irradiance constraint is imposed by the rotation matrix Θ, which maps the updated normal to the closest normal on the cone of ambiguity. The axis of the rotation is given by a vector perpendicular to both the updated normal and the light source direction.

$$(u, v, w)^{\mathrm{T}} = \overline{\mathbf{n}}_{i,j}^{k} \times \mathbf{s} \tag{28}$$

The angle of rotation is then given by the difference between the angle subtended by the intermediate update and the light source, and the apex of the cone of ambiguity. The angle rotated through is therefore the angle between the updated normal and the light source direction minus the apex angle of the cone.

$$\theta = -\cos^{-1}\left(\frac{\overline{\mathbf{n}}_{i,j}^{k} \cdot \mathbf{s}}{\|\overline{\mathbf{n}}_{i,j}^{k}\| \|\mathbf{s}\|} \right) + \cos^{-1} E \tag{29}$$

Thus a standard rotation matrix is constructed using the axis, $(u, v, w)^{\mathrm{T}}$, and the angle of rotation, θ.

$$\Theta = \begin{pmatrix} c + u^2 c' & -ws + uvc' & vs + uwc' \\ ws + uvc' & c + v^2 c' & -us + vwc' \\ -vs + uwc' & us + vwc' & c + w^2 c' \end{pmatrix} \tag{30}$$

where $c = \cos\theta, c' = 1 - c, s = \sin\theta$.

4.2 Shape-from-stereo

In comparison to the shape-from-shading technique this technique is simple. In order to achieve a better comparison of what curvature and topological information performs best, I use a simple shape-from-stereo scheme so that hopefully the comparisons will be more general and will extrapolate to other shape-from-stereo schemes.

I use a variation of the sum-of-squared-differences (SSD) shape-from-stereo scheme, the same method as Worthington used in [26]. Using Scharstein's break down of dense stereo correspondence methods [18], we can write the SSD algorithm as a number of steps:

1. the matching cost is the squared difference between intensity values at a specific disparity
2. aggregation is done by summing the cost over square windows with constant disparity
3. disparities are then calculated by choosing the minimum aggregated cost at each pixel

4.2.1 Matching cost computation

The initial matching cost computation, Step-1, varies depending on the data used to create the matching. There are several different topological and curvature measures that can be calculated, and there are even more different ways of creating evidence measures that describe the similarity between the two different points in each stereo image. The evidence measures can also be combined in many different ways, I however will restrict myself to just a few:

Pixel intensity: The actual raw pixel data, using a sum-of-squared-differences method to compare pixel intensities. This is the base evidence measure that I hope some of the surface curvature and topological measures will improve upon. Unfortunately for most of the synthetic data I will use the sum-of-squared-differences of intensities performs unrealistically well.

Normal map: The normal map calculated from shape-from-shading, although it is not rotationally invariant is a useful measure of similarity. To compare two surface normals, the evidence measure between two normals could be expressed as the angle between them or more succinctly as the dot product of the normals. Worthington [26] used this measure whilst also using an exponential to encourage good matches and penalise poor matches, and also scaling the evidence measure to give a maximum of 1 and a minimum of slightly less then -1.

$$E(\mathbf{n}_a, \mathbf{n}_b) = \frac{e^{\mathbf{n}_a \cdot \mathbf{n}_b} - e^{0.5}}{e^1 - e^{0.5}} \tag{31}$$

Principle curvatures We can also use the directions of minimum and maximum curvature, λ_M^k and λ_m^k, on a surface at a specific point for an evidence measure. Either a direct sum-of-squared or sum-of-absolute differences comparison method can be used. The direction of maximum curvature will give the best results, since the direction of minimum curvature is a poor description of a surface.

Mean and Gaussian curvature The same direct sum-of-squared or sum-of-absolute differences can be used for mean and Gaussian curvature evidence measures.

Curvedness The same comparison can be also used for curvedness. Curvedness can be used in evidence measures to provide a weighting for some topological or directional measure that lack curvedness, such as the normal map or the shape index. This weighting improves the matching of highly defined areas, weak gradients such as flat surfaces give a lower confidence than highly defined areas such as ridges.

Shape Index The shape index which ranges from -1 to 1, is an angular physical measure. And has to be compared so that -1 and 1 are similar. So a simple sum of squared differences wont work. However with a little simple modification that allows for short walks over the pole, shape index too can be compared as either an absolute difference or a squared difference. Another way is to think of the shape index as a unit vector in direction ϕ and use the dot product.

Shape index is only a topological measure and ignores the surface curvature, so it is probably a good idea to try and weight the shape index with by some function of curvedness. For instance, a simple multiple of $e^{-\frac{1}{R}}$, where R is the curvedness. This can be compared in a polar like fashion, using ϕ as the angle and the weighting as the length of a vector in 2D space. The comparison can then be made as a dot product.

Direction of maximum curvature The direction of maximum curvature is either a 2D vector in the surface that points in the direction of maximum curvature, or a 3D vector in the image co-ordinate system, either can be used to calculate the evidence for a match. Both forms are vectors and it makes sense to compare both of them with a dot product. They can also be weighted in the same way as the normal map using curvature.

In summary, below is a table of the different evidence measures that I will analyse for the project.

4.2.2 Aggregation

As an aggregation function rather than a window based approach I use a Gaussian convolution over all evidence values at the same disparity. This encourages neighbouring pixels to have a similar disparity. The aggregation can be thought of as taking the disparity space image of evidences and applying a Gaussian convolution to each disparity plane.

Table 2: Evidence measures

Computation	Evidence function
SSD of pixel intensity	$-(I_a - I_b)^2$
Dot product of normals	$\frac{e^{\mathbf{n}_a \cdot \mathbf{n}_b} - e^{0.5}}{e^1 - e^{0.5}}$
Dot product of normals and curvature	$e^{\lvert R_a - R_b \rvert} + \frac{e^{\mathbf{n}_a \cdot \mathbf{n}_b} - e^{0.5}}{e^1 - e^{0.5}}$
Maximum curvature	$(\lambda^k_{Ma} - \lambda^k_{Mb})^2$ or $\lvert \lambda^k_{Ma} - \lambda^k_{Mb} \rvert$
Mean curvature	$(K_a - K_b)^2$ or $\lvert K_a - K_b \rvert$
Gaussian curvature	$(H_a - H_b)^2$ or $\lvert H_a - H_b \rvert$
Curvedness	$(R_a - R_b)^2$ or $\lvert R_a - R_b \rvert$
Shape index	$\Delta\phi$
Shape index weighted by curvature	$e^{\lvert R_a - R_b \rvert} + \Delta\phi$
2D Direction of maximum curvature	$\mathbf{M}^k_{2a} \cdot \mathbf{M}^k_{2b}$
3D Direction of maximum curvature	$\mathbf{M}^k_a \cdot \mathbf{M}^k_b$

4.2.3 Disparity calculation

I will use a simple winner takes all approach, ignoring its flaws, as it is the simplest method. By using a relatively simple method, it allows me to concentrate on the comparison of the evidence measures and not on the matching abilities of the algorithm.

5 Implementation

It does not serve much purpose detailing the exact method of implementation, since the project's aim is to investigate the use of topological and curvature information to improve stereo algorithms not the actual coding. However I will briefly overview of the code, to enable a greater understanding of it, and perhaps to better enable its further use. The code was written in C++ an object orientated language, which produces fast and clean code. Some effort was made to keep the design of the whole system clean and neat, using an object orientated design.

The whole system makes use of Will Smith's shape-from-shading code [19]. Two grayscale pixmaps are both run though his shape-from-shading scheme to produce normal maps, which are output and saved to files. The normal maps are then supplied to my stereo comparison program. My program provided with the two normal maps, uses a sum-of-squared-differences algorithm to create a number of stereo comparisons for each curvature or topological method. It also compares the each calculated disparity map for every evidence measure with a provided ground truth, using the analysis in Section-8. Finally it outputs the results to a tab separated data file for comparison. The methods used for comparisons are described in Section-7.

As a brief overview of the process used in my program, the program can be divided into several steps.

- Loads the normal map files, checking they are in the right format.
- Loads either a ground truth disparity map or calculates a ground truth from two colour surfaces provided using a simple SSD of intensities and a WTA disparity calculation.
- The disparity maps are calculated for every evidence measure by `StereoMatch` and saved as a grey-scale image with 256 levels in portable grey-map (PGM) format.
- The calculated disparity maps are then compared to the ground truth and the results are output to a data file.

The function `StereoMatch` is the main body of the program, implementing the sum-of-squared-differences algorithm which is fully explained in Section-4.2. It takes four parameters:

- `match` The output, the disparity map calculated from the evidence function.
- `left` The left normal map.
- `right` The right normal map.

- `compare` The evidence function to use.

The evidence calculators are separated into a separate file “Evidence.cpp”, these are passed as function pointers to the main algorithm `StereoMatch`, which uses them to compare pixels in the left and right images. Each evidence function is passed the stereo pair of normal maps and the two locations being compared, returning the numerical evidence value between the two locations. For each stereo pair of normals multiple evidence measures are used and multiple output files are created. Most of the functions make use of one of several classes, `SMatrix`, and `SImage` or `SCImage` which inherit from `SMatrix`, these encapsulate the file operations, the data storage and retrieval. These classes make use of the `()` operator to access the data.

All the code can be found in Appendix-B.

6 Test data

To achieve a good comparison of all the methods discussed they have to be applied to many different images. The ability of a shape-from-shading or stereo algorithm is entirely dependant on the actual images used as input. To have an understanding of how the different methods perform, it is important to test under as many different conditions as possible.

To accurately compare the results of two different methods is impossible if there is no ground truth data to which to compare the calculated disparity maps to. Unfortunately ground truth values for disparity or depth are extremely hard to compute or determine from real world images without specialised equipment. Due to this complexity in creating an accurate stereo data with disparity maps from real world images, I will use mostly synthetic computer generated images, where disparity is in some calculable. This is unfortunate since synthetic images are rarely an accurate substitute for real world images.

6.1 Disparity calculation

A simple stereo image pair with no noise and a texture that provides a unique surface colour at each surface point, can be used to calculate disparity using a simple comparison and a winner-takes-all disparity calculation. When creating synthetic stereo images, it is possible to use the same models and camera positions and apply a different texture with uniform ambient lighting that provides images suitable for calculating disparity. Even using this method there are still problems as there are still areas which are occluded in each image for which no disparity can be calculated. Unfortunately calculating disparity in the real world poses problems that makes it too hard to create some test data of real world images and disparity maps for this project. Figure-8 shows the disparity calculation of a head, using a rainbow texture to provide each horizontal position on the surface with a unique colour so that the disparity can be calculated. The occluded area is shown in red

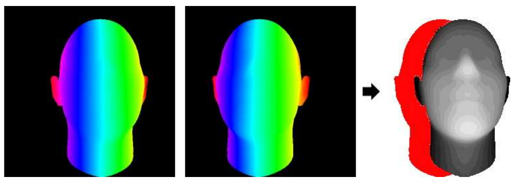


Figure 8: Disparity calculation.

6.2 Synthesis

I use Strata 3D Pro to render the results, which I then convert to the portable greymap (PGM) format. Synthetic images have been used extensively for qualitative evaluations of stereo methods, however there are many problems with using them. Real cameras are seldom modelled properly, I use some noise but there are many other ‘features’ of real cameras that are harder to model, for instance, aliasing, slight misalignment, lens aberrations, and fluctuations in gain and bias. Frequently results on synthetic images therefore usually do not extrapolate to images taken with real cameras.

To create a suitable data set for a comparison of different methods that covers a wide variety of image types. I look at several different image features to try and achieve a range of suitably different images from which to test the different evidence measure functions. The different features or data sets I will look at are:

Frequency: A test data set to look at how the frequency of ridges on a surface affect the different evidence measures abilities. The images range from a smooth curve to lots of ripples. I used the function $z = \cos(x \times s) \times \cos(y \times s)$ to create the surfaces. Which I then rendered in Strata 3D Pro using a Lambertian surface and a single point light source, from two different points of view to create a stereo pair. I then repeated the process using different values for s to create surfaces of differing frequencies.

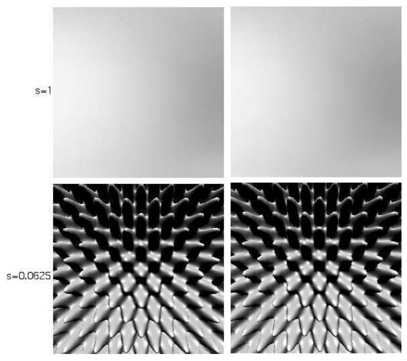


Figure 9: Stereo pairs for different frequencies.

I used a range of 1, 0.5, 0.25, 0.125 and 0.0625 for s. The stereo pairs for $s = 1$ and $s = 0.0625$ are shown in Figure-9. All the other pictures used can be found in Appendix-A. The surface is repetitive and it is likely that some methods will find the wrong correspondences. However even though the surface is unnaturally smooth and repetitive, hopefully it will give some insight into how the different evidence measures compare.

Amplitude: I created this data set to look at how the amplitude of the bumps on a surface affect the surface reconstruction. The images range from flat to “very bumpy”. I used a fractal landscape which I then rendered at several different scales of height. The structure of the landscape is rough and has no repetitions, unlike the surfaces generated for the frequency data set.

Lighting: Looking at how different lighting conditions affect the surface reconstruction. From bright to dark, and different directions. Using the head model, Figure-4, I rendered the head under four different lighting conditions. 1. Single point source from above 2. Single point source from below, (this could create a worse normal map as the shape from shading algorithm assumes an initial lighting direction from above, just as our brain does) 3. Two point light sources from either side 4. Radiosity, area light sources and multiple reflections with lots of ambient light, an attempt to model real world lighting conditions. These four lighting conditions are shown for the left head in the stereo pair in Figure-10

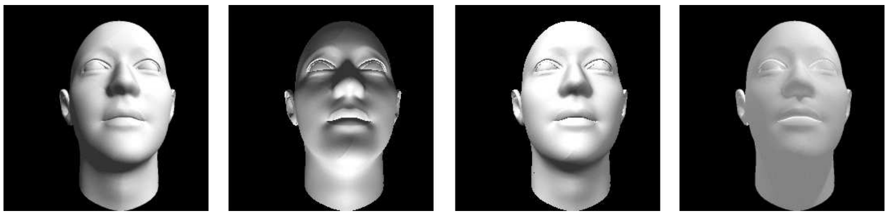


Figure 10: Different lighting conditions.

Noise: Resilience of any method with respect to noise is an important feature, and to test the differing performance of the evidence functions I use these simulated camera noise pictures. The simulated camera noise, aims to reflect real world conditions better. Although camera noise is much more complicated by adding noise I aim to gain some idea of the evidence measures abilities to cope with real world situations. Using the head model again, different amounts of simulated noise are applied to the final rendered model. To create the noise I used Photoshop's noise filter, using a monochromatic noise for an amount range of 25, 50, 100 for both Gaussian and uniform distributions. Below is the left member of the stereo head pair with uniform and Gaussian noise applied of amount 50.

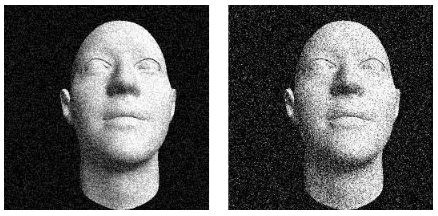


Figure 11: Uniform and Gaussian noise.

The head stereo pair is relatively simple, there are no large depth discontinuities, it is also fairly smooth. I will also make use of the synthetic University of Bonn's "corridor" data set, Figure-12, which is a more complicated scene and is also supplied with different levels of noise. Although it is unlikely that the textured floor or walls will work well for the initial shape-from-shading step.

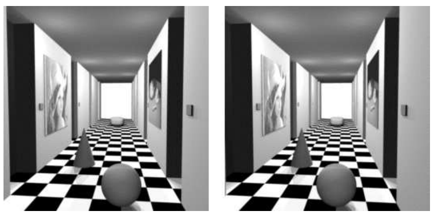


Figure 12: University of Bonn's synthetic corridor.

Real world: Real world images are completely different to synthetic ones, and usually many times harder to compute shape from. They are made up of complex textured non-Lambertian objects under complex lighting I don't expect good results for the more complex images where the shape-from-shading assumptions completely fail.

I use a few actual real world stereo images, to help gauge how the different methods perform with real images. Although for most of these true disparity maps are not calculable or available, so the they have to be compared by eye. The selection of images are a random assortment. The "Head" scene is a courtesy of the Computer Vision Lab at the University of Tsukuba (Japan). The pentagon, mud ruts and Renault car part stereo pairs are from the Carnegie Mellon University Vision and Autonomous Systems Center's Image Database. The stereo image of the eye, is a courtesy of Vision Imaging Systems Inc.

All the stereo images used in this project and the corresponding disparity maps are shown in Appendix-A.

7 Analysis

The aim of analysis is to provide an accurate measure of the comparative performance of each method at creating an accurate disparity map. Although the eventual aim of the shape-from-stereo algorithm is to create an actual depth map or surface from the stereo images, the comparison is just as valid if it is performed on the disparity images.

A fairly accurate analysis of which solution performs best is possible by eye, however it is important to have an accurate numerical comparison. So that we can properly evaluate the performance of each evidence function, we need a qualitative way of measuring its performance.

7.1 Methods

To compare a calculated disparity map with the known ground truth, I use two simple pixel by pixel comparisons, the same methods Scharstein used in [18].

Root-mean-squared error between ground truth $d_T(x, y)$ and the calculated disparity $d_C(x, y)$ is:

$$R = \sqrt{\frac{1}{N} \sum_{(x,y)} (d_T(x, y) - d_C(x, y))^2} \tag{32}$$

where N is the total number of pixels. The root-mean-squared (RMS) error roughly can be seen as a measure of how closely overall the calculated disparity map matches the ground truth. An RMS error of 0 would mean a perfect match. It can however overlook image defects, even if the actual disparity image structure does not resemble that of the ground truth disparity map the RMS error can give good results.

To provide a more structural comparison is why we need a further performance measure, the percentage of bad matching pixels. This roughly measures how well the actual structure of the calculated disparity map conforms to the ground truth, regardless of large errors. The percentage of bad matching pixels between ground truth $d_T(x, y)$ and the calculated disparity $d_C(x, y)$ is:

$$B = \frac{1}{N} \sum_{(x,y)} ((d_T(x, y) - d_C(x, y)) > \delta_d) \tag{33}$$

where δ_d is the disparity error threshold, the disparity error tolerance inside which pixels are accurate. It is the percentage of pixels with a greater difference in disparity to the ground truth than the threshold δ_d. For my results I used $\delta_d = 1$.

8 Results

In the rest of this section I will overview all the results, comparing the evidence measures for all the test data sets. Due to the large number of different results (over 300 disparity maps) I only include results in the project that provide useful information and to clarify comments, a lot of the results are fairly similar and where there is differences or interesting features the results have been provided. If required the test data and the source code have been provided in the appendices, to enable the calculation of further results. I will also make some preliminary conclusions about the results, which are summed up in Section-9.

8.1 Evidence measures

To give an idea of what the evidence measures are using to create their matches, I created actual images from the various different curvature and topological measures extracted from the normal map.

Normal map: The normal map displayed as a needle-map is shown below in Figure-13. The head and all its features can clearly be seen. The surface orientation is shown as “needles” which point in the direction of the surface normal.

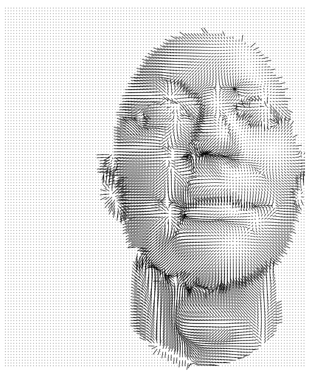


Figure 13: Normal map of the left stereo head

There is a little noise on the left hand side under the chin and at the left side of the nose, this is probably caused by these areas being shadowed in the original images, Figure-4. The shadowing removes all real shading information. However the rest of the surface is clear and the orientation and structure of the surface seem reasonable. A good normal map is essential, it is what all the other curvature and topological measures are calculated from, any noise or inaccuracies in the normal map are passed down to all the different measures calculated from the normal map. If the normal map is bad or inaccurate for any reason then the curvature and topological evidence measures are going to be similarly inaccurate and bad. Unfortunately there is no way around this problem, and the only solution is to use a robust shape-from-shading scheme and stereo images which work well with it.

Maximum curvature: The maximum curvature of the stereo pair is below in Figure-14, the intensity levels of the image have been adjusted to make the detail more clear.

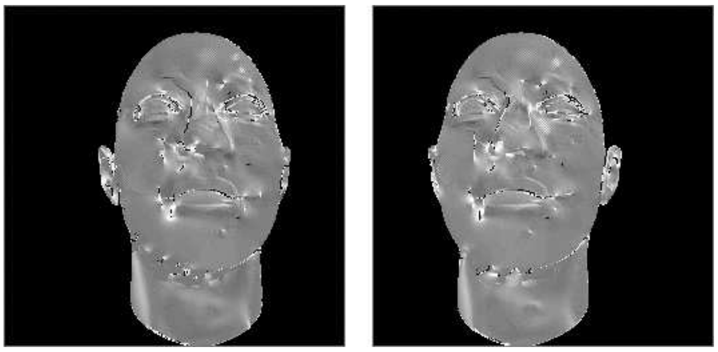


Figure 14: Maximum curvature of the stereo pair of heads

Most of the detail or the normal map is still clear, although there is some noise in areas, particularly the nose and under the left hand side of the chin, which comes directly from the noise in the normal map. Overall the mean curvature seems to convey the surface fairly well. However apart form where there are sharp edges and ridges in the surface the maximum curvature is fairly even and texture-less over smooth areas on the surface, which is a hindrance to applying shape-from-stereo.

Mean curvature: The mean curvature shown in Figure-15 is similar to the maximum curvature. The intensity levels of the image have again been adjusted to make the detail more clear, but the actual values have a similar spread as those of the maximum curvature.

All the details of the face are completely clear, and in general the mean curvature seems more defined than the maximum curvature. In particular the nose is more defined and less noisy than the maximum curvature nose. However the mean curvature also seems to suffer from large texture-less areas of very similar shading. The mean curvature again inherits the noise from the normal map, as all curvature measures will.

Gaussian curvature: Similar to the mean and maximum curvatures, the Gaussian curvature of the stereo heads is shown in Figure-16. The Gaussian curvature is however much more noisy, the range of values are also much smaller than either the maximum or mean curvatures. The nose and the eye sockets are very messy, and particularly noisy.

Curvedness: Pretty similar to all the other curvature measures the curvedness is below in

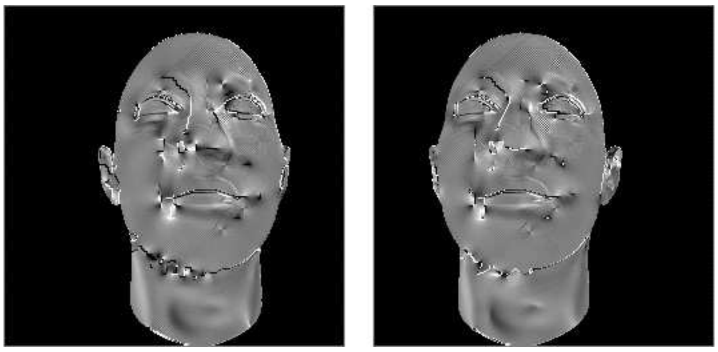


Figure 15: Mean curvature of the stereo pair of heads

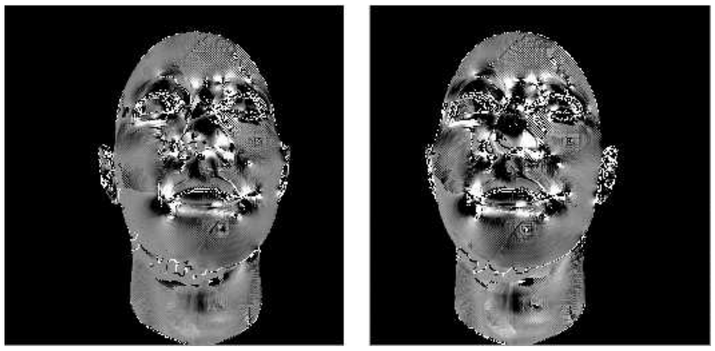


Figure 16: Gaussian curvature of the stereo pair of heads

Figure-17.

Shape index: Shape index shown in Figure-18 is a continuous topological measure representing the curvature information. It is not a curvature but a topological measure, and encodes the type of surface cup, rut or saddle, Table-1. The shape index is more highly defined, than any of the curvature measures. Although the nose is again a little weak. The texture-less areas are also smaller. The better definition and the smaller texture-less areas probably mean that the shape index will produce the best evidence measure.

Conclusions: All the different curvature values are fairly similar in both contrast and shading, they also seem to have similar defined and texture-less areas. These similarities will certainly result in similar performances. The shape index is the most highly defined of the curvature and topological measures and therefore will probably perform best. In contrast the Gaussian curvature is the most noisy and will probably perform worst.

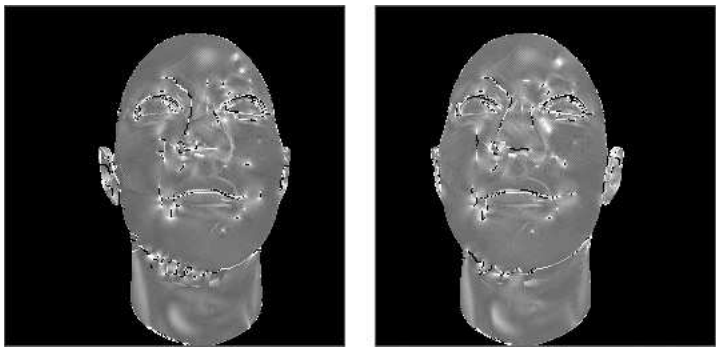


Figure 17: Curvedness of the stereo pair of heads

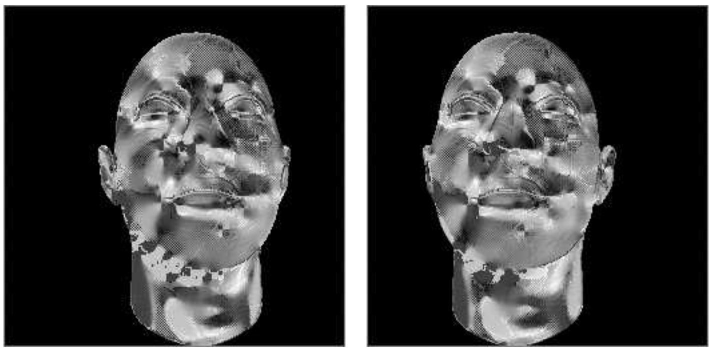


Figure 18: Shape index of the stereo pair of heads

8.2 Frequency

In this experiment I compare the success of different evidence measures for different frequencies of a surface. I used five stereo pairs of the surface $z = \cos(x \times s) \times \cos(y \times s)$, these can be found in Appendix-A. I hope to show that some of the evidence measures prove to be more effective than a simple intensity comparison. A summary of the evidence measures used can be found in Table-4.2.1. The stereo pairs are synthetic, which means that the simple SSD of pixel intensities is almost always going to beat any surface curvature or topological measures. This is because the pixel intensities are exactly the same in either view at any point in the surface since it is a Lambertian surface. However the surface is highly repetitive and inaccurate matches are possible. To properly compare the ability of the surface curvature or topological measures with the simpler pixel comparison we will have to look at the noisy synthetic stereo pairs and the real world images. Figure-19 shows the performance of the three evidence measures of the normal map, the normal map weighted by the curvature, and the simple SSD of pixel intensities.

The initial dip in error both for the normals and the SSD of intensities evidence measures is certainly caused by the lack of structure in the first stereo pair, it is almost flat. Both the normals and the SSD of intensities evidence measures have problems accurately creating a

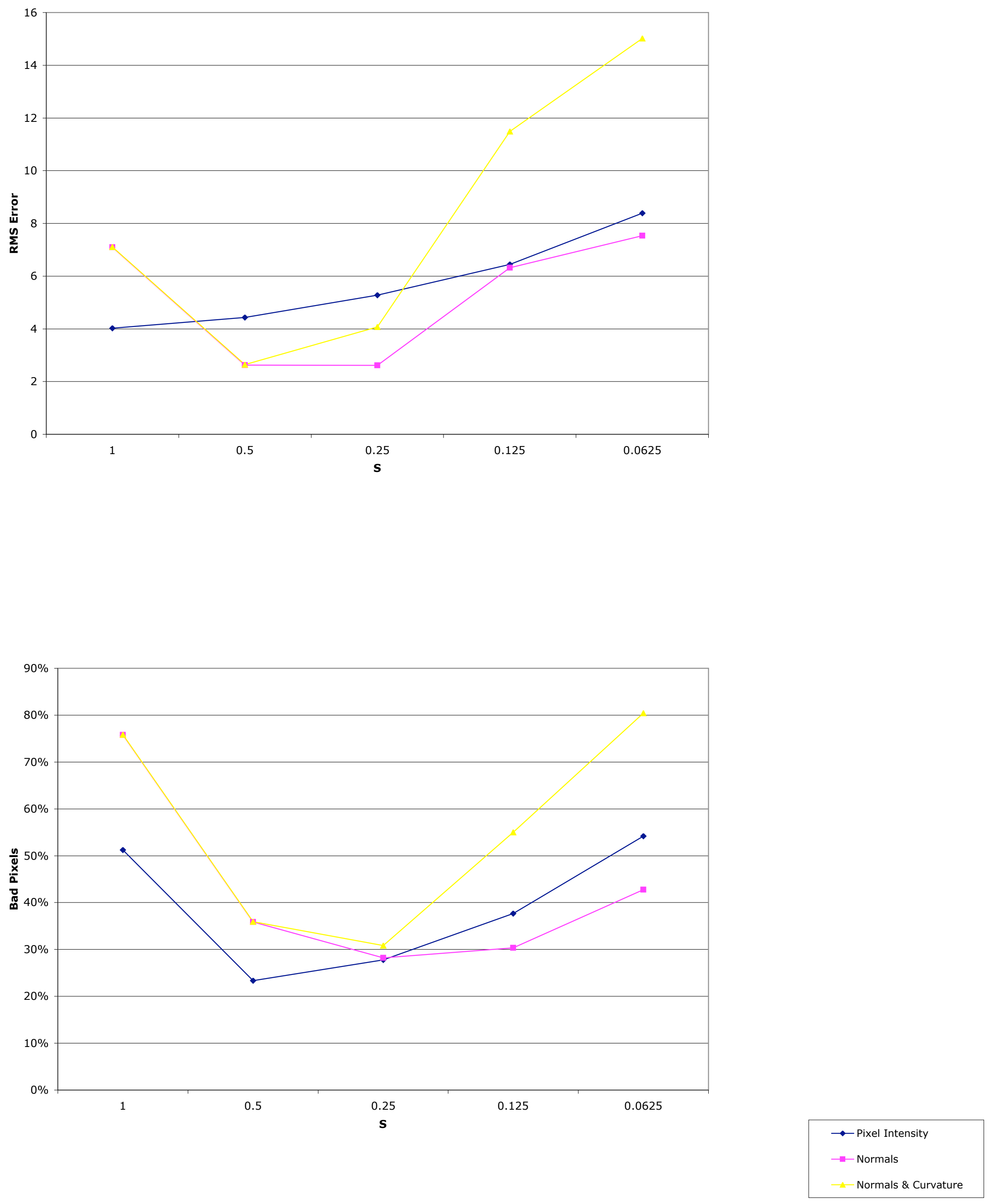


Figure 19: Performance of normal map evidence measures for surface frequencies

point-to-point correspondence in the absence of much information.

For the surfaces $s = 1$ and $s = 0.5$ both the dot product of normals and the curvedness weighted dot product of normals evidence measures have very similar results. In fact the difference is in favour of the unweighted measure by only 0.01 RMS error. However as s continues to decrease and the surface frequency increases the difference between the two evidence measures increases dramatically. The curvedness weighted evidence measure has almost double the error of the non-weighted evidence measure when $s = 0.0625$. As the frequency increases so does the curvedness value, and the error introduced by the curvedness weighting increases, which accounts for the increasing error between the two evidence measures seen as s decreases.

The dot product of normals evidence measure improves upon the simple intensity comparison for the last three surfaces. This probably happens because there is an increasing area that is shadowed as s decreases. The disparity maps for the SSD of pixel intensities and the normal dot product evidence measures are shown below in Figure-21 for a surface of $s = 0.125$, shown in Figure-20.

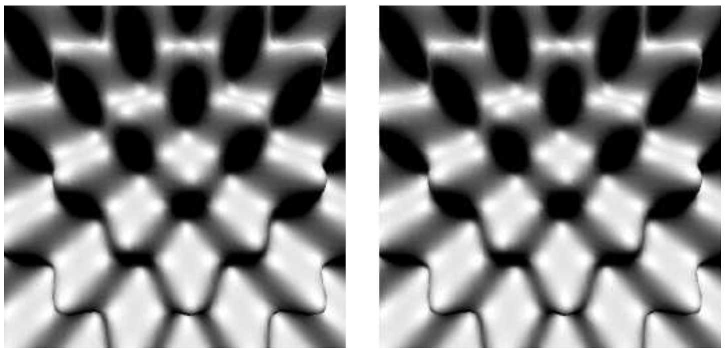


Figure 20: Stereo pair for surface $z = \cos(x \times s) \times \cos(y \times s)$ where $s = 0.125$

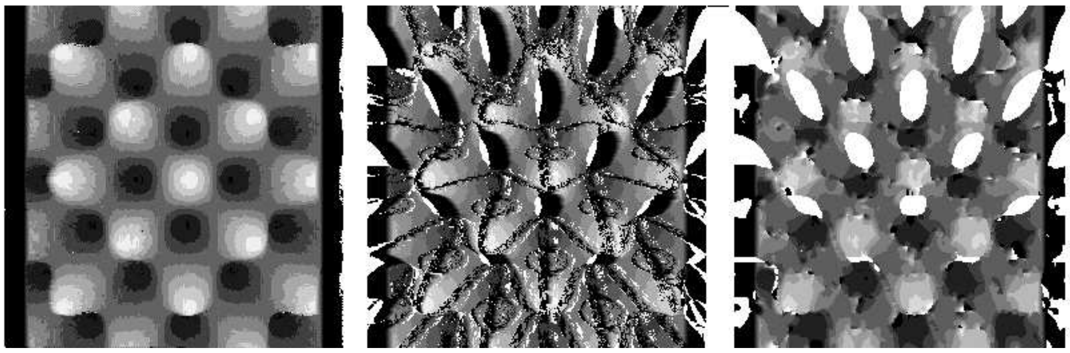


Figure 21: Ground truth and calculated disparity maps for pixel intensities and normals

Both the disparity maps of SSD of intensities and the dot product of normals evidence measures have holes where the surface is shadowed. However the disparity map created by the SSD of pixels intensities is far more noisy, almost certainly due to smaller shadowed edges in the bottom half of the image which are covered over with the normal map and the texture-less surfaces facing upwards. The shape-from-shading scheme tries to recover an accurate integrable surface, and will fill in the smaller shadowed gaps, hopefully with a smooth integrable surface. The SSD of intensities does not smooth over the shadowed gaps and the shadowed areas even the smaller edges will create large errors when doing such simple matching. Behind the noise the disparity map of the SSD of the pixel intensities is however fairly accurate, but the dot product of normals is clearly a better evidence measure in this circumstance. The other disparity maps were very similar to the one created from the normals, most of them were slightly more noisy.

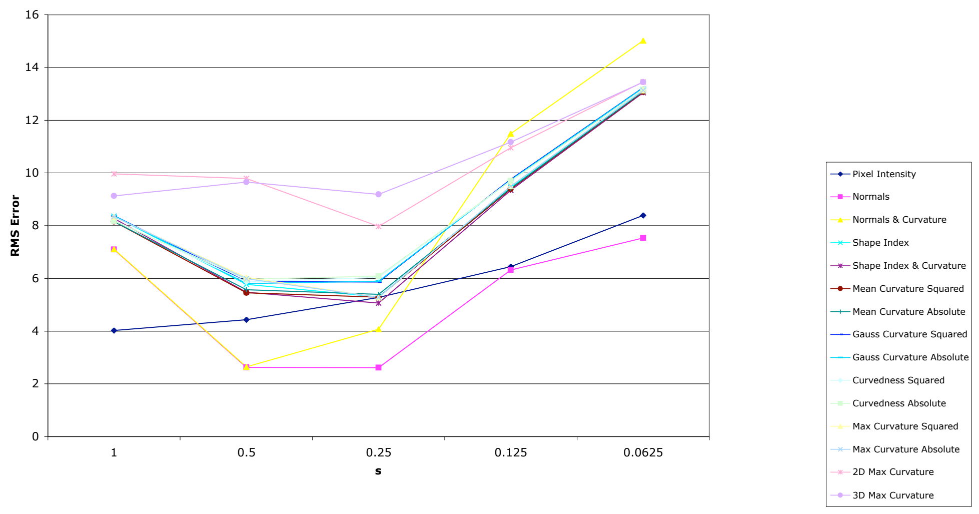


Figure 22: Performance of all evidence measures for surface frequencies

The performance of all the other evidence measures is shown in Figure-22, the graph of bad pixels is essentially the same. The only other evidence measure that improves upon the standard SSD of pixel intensities is the shape index weighted by curvedness, which is a better measure than simply the shape index on its own. Almost all the different topological and curvature measures are bunched together, with a similar initial dip before an increasing error. In fact this 'bunching' of the curvature measures producing very similar results is repeated in most of the test sets. The 2D and 3D maximum curvature directions however seem to be consistently worse than any other evidence measures.

8.3 Amplitude

In this experiment I compare the success of different evidence measures for the same surface scaled to different heights. I used a fractal surface then applied a height scale of 1, 2, 4, 8 and 16, the resulting stereo pictures can be found in Appendix-A.

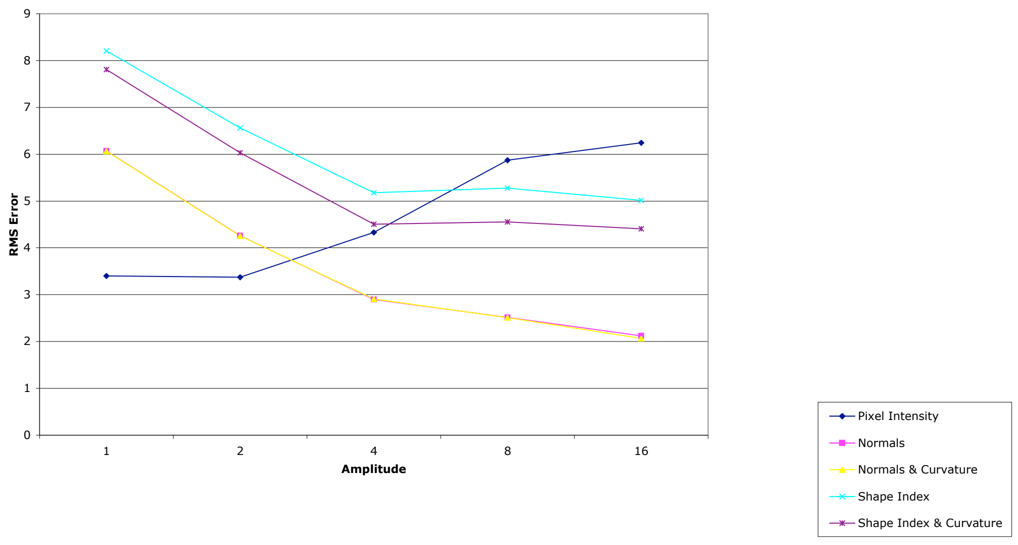


Figure 23: Performance of normal and shape index evidence measures for surface amplitudes

In Figure-23 I show just the performance of the two different normal and shape index evidence measures, compared with the pixel intensity measure. The performance of all the other curvature measures is very similar to the performance of the shape index. The curvedness weighted shape index is the best performing measure, all the other measures follow a similar shape just with slightly worse performance. The 2D and 3D direction of maximum curvature again consistently provide the worst performance of any evidence measure.

Just as with the regular rippled surface used to test performance against frequency, the SSD of intensities beats all surface curvature and topological evidence measures for the smooth flat surfaces of amplitudes of multiples 1 and 2. However as the amplitude continues to increase the error for the different surface curvature and topological evidence measures continues to decrease whilst the error for the SSD of intensities increases. Compared to the surfaces used for frequency the errors decrease as the surface becomes more curved instead of increasing. This might be because of the lack of regular structure make correct matches more likely. The SSD of intensities evidence measure however continues to increase.

The disparity maps, Figure-24 do not show any clear information, in general though the disparity maps of the normals and curvature values are less ‘bitty’. Both of the evidence measures suffer from failing again to calculate the disparity in the shadowed area, top left.

The addition of a curvature weighting seems to make little difference to the normals, and consistently makes an improvement to the shape index. There is no repeat of the increasing error as the surface became more highly curved, as there was in the repetitive surface used

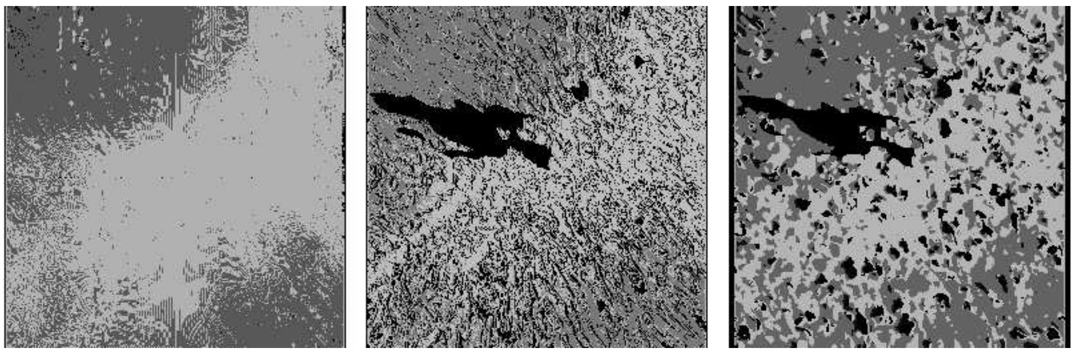


Figure 24: Ground truth and calculated disparity maps for pixel intensities and normals

in the frequency test set. This is probably because the rougher surface does not have areas of similar normals and the correct matches are more likely.

8.4 Lighting

Figure-25 shows the performance of the normal and shape index evidence measures under different lighting conditions. We can see that the unweighted normals provide a consistently better disparity map than the SSD of pixel intensities. Yet again the curvature enhanced normals provides a strangely inaccurate disparity map, the rest of the curvature evidence measures are similar to that of the shape index.

The actual lighting seems to make little difference to the performance of any of the evidence measures. Even when the head is lit from underneath the shape-from-shading scheme still manages to retrieve a good normal map from the heads and provide a good evidence measure.

8.5 Noise

The results are shown in Figure-26, we can clearly see that noise decreases performance as one would expect. It is interesting to note that a Gaussian distribution of noise is worse than a uniform one, this is probably because of the clumps, it is much more speckled and bitty. Have a look at Figure-11.

The other curvature measures all perform similarly well, even the 2D and 3D maximum curvature evidence measures have similar performance. In fact several of them actually perform better than the shape index for once.

The structure of the disparity maps retrieved is shown below in Figure-27. The curvature and topological methods are much better at discovering the surface behind the noise, than

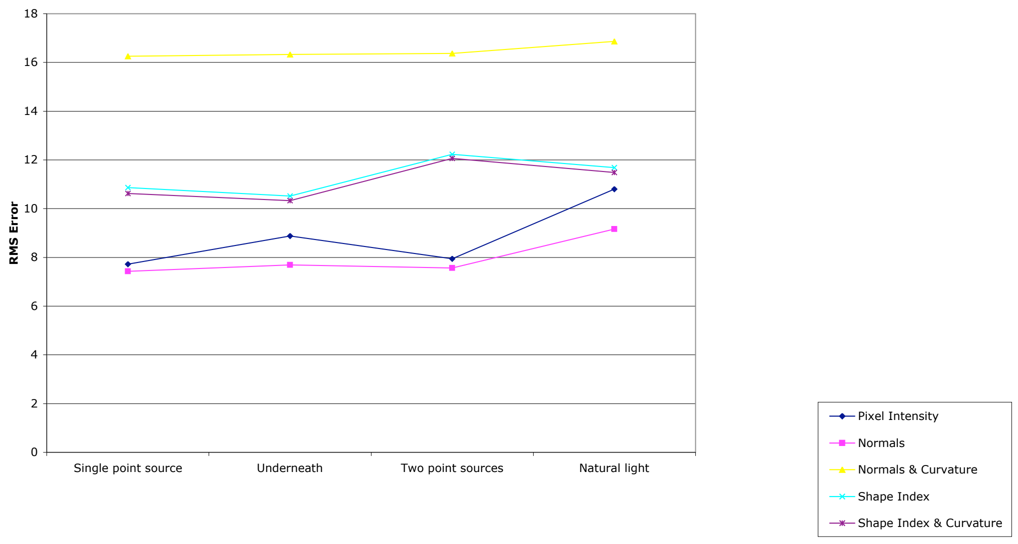


Figure 25: Performance of normal and shape index evidence measures for different lighting conditions

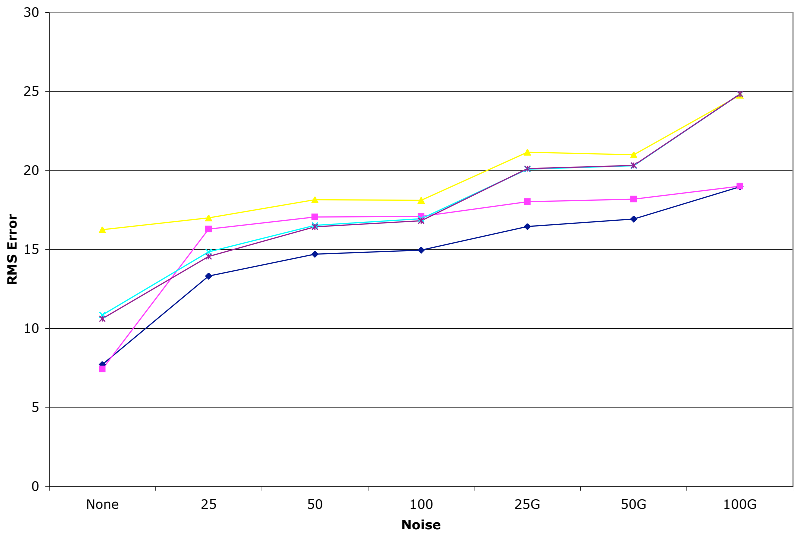
the straight forward pixel comparison. This is evident in the percentage of bad pixels, where the noise makes little difference to the abilities of the evidence measures.

The other images used, the images of the corridor were a failure, just like the real world images they were too textured and did not function well when using shape-from-shading.

8.6 Real world

Unfortunately none of the real world images provided any real comparison between evidence measures. The normal maps and the derived curvature and topological information were extremely noisy and inaccurate. The calculated disparity maps were even worse. Which is unsurprising since none of the images where the sort of images that shape-from-shading requires.

Although the real world stereo pairs did not provide insight into the performance of the evidence measures, they did serve to highlight the short comings of shape-from-shading and hence the short comings of the curvature and topological measures.



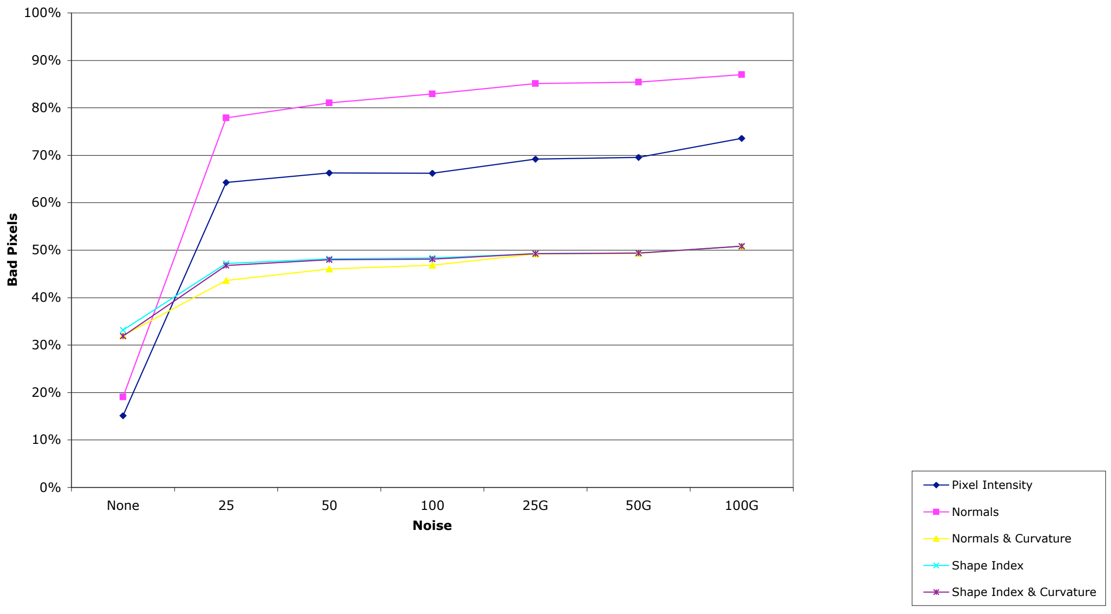


Figure 26: Performance of normal and shape index evidence measures for noisy pictures

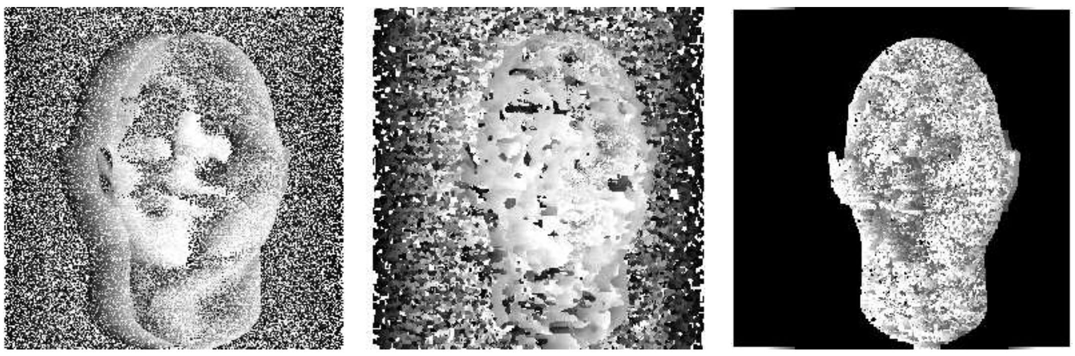


Figure 27: Disparity maps for pixel intensities and normals and mean curvature for uniform noise of 20

9 Conclusion

In general the topological and curvature evidence measures did not perform well enough to be considered a substitute for intensity based evidence measures. Especially when considering the limited set of images shape-from-shading can be applied to and the heavy extra processing needed before shape-from-stereo can be applied. However as the disparity images in Figure-27 showed, the performance of the normals as an evidence measure can produce significantly better results than simply the pixel intensities in some circumstances. And the ability to cope with noise is impressive.

There are several conclusions that can be drawn from the results, some of which have either been mentioned in part previously:

- Out of all the evidence measures tested almost all the curvature and topological measures produced very similar results. With the exception of the 2D and 3D maximum curvatures. All of these measures were derived from the Hessian matrix, the differential structure of the surface. It appears that it is the Hessian matrix itself that is the source of the similarity. Perhaps no matter how the data in the matrix is presented, the differential structure of the surface is still the same data and matches similar ways.
- The poor performance that topological and curvature evidence measures had on the real world images is unsurprisingly a reflection of the lack of ability of shape-from-shading when dealing with real world images. This will always be the case because shape-from-shading will probably never extract shape information properly from highly textured surfaces. It'd could almost be called shape-from-texture if it did.
- However when restricted to non-repetitive highly defined Lambertian surfaces the shape-from-stereo-from-shading scheme did produce a significant improvement over the simple pixel intensities.
- The noise resilience of the curvature and topological evidence measures is perhaps the most impressive of their features. Although the mean curvature did not recover the actual shape of the face well in Figure-27 it did somehow extract it from the background exceedingly well.
- The actual normal map is the best performing evidence measure, although it is sometimes improved by curvature.
- The shape index is the best performing of the scalar evidence measures. This is no surprise as in Figure-18 the shape index out of all the curvature and topological measures had the best definition and most texture.

In conclusion, this project has investigated the use of shading information to improve the matching ability of a simple shape-from-shading algorithm. The results show some promise especially in regard to highly defined Lambertian surfaces and to noisy images. Where the curvature and topological evidence measures have shown that they can provide a significantly better source of matching information for stereo images.

9.1 Future work

More data sets: The lack of any real stereo images for which there existed a ground truth disparity map is a serious lack. There can be little in the way of meaningful comparison without ground truth data. Scharstein [18] used a predicting method that using the calculated disparity map predicted the appearance other views which were then compared to the ground truth. Unfortunately this does not work extremely well with texture-less images, however it would be useful to test the different evidence measures thus. There is also very few good stereo images of smooth Lambertian objects, since most shape-from-stereo algorithms concentrate on textured objects. To do a more detailed study of the ability of shape-from-shading to improve shape-from-stereo, it would be essential to have a larger synthetic and real world data set to work with.

Further uses of curvature measures: To look at the curvature and topological data not as evidence measures in themselves but to look at using the information to guide comparisons based on the pixel intensities. There is no need for the information to be used on its own, and there are many more ways it could be used to weight other measures. Perhaps to help improve shape-from-stereo in especially noisy conditions.

Other shape-from-stereo algorithms: By using a simple shape-from-stereo algorithm I hoped to study the affects of the different evidence measures more clearly. Whether this will extrapolate to more complicated shape-from-shading algorithms requires further investigation.

Other shape-from-shading algorithms: The shape-from-shading algorithm I used although fairly robust, is a simple implementation. It could be improved in several ways, notably by using the variance in the shape index to control the width of the robust error kernel applied to the needle map to improve the algorithms performance, similar to Worthington and Hancock's method [29].

Test with other images: One of the possible advantages of using shape-from-shading that could be explored further is its ability to work on images of different contrasts and gains. This would be a large benefit for use with real cameras, however unfortunately the synthetic images I used did not test the shape-from-shadings ability to do this.

Investigate the ability on noisy images: The major surprise was the superb ability of the shape-from-stereo-from-shading schemes ability to deal with noisy images. It would be useful to further investigate its ability to deal with noise. Perhaps this could a purpose in itself; to recover features from noisy images.

References

[1] H. G. Barrow and J. M. Tenenbaum. Retrospective on interpreting line drawings as three-dimensional surfaces. *Artificial Intelligence*, 59:71–80, 1993.

[2] R.T. Collins. A space-sweep approach to true multi-image matching. *CVPR*, pages 358–363, 1996.

[3] O. Faugeras and R. Keriven. Variational principles, surface evolution, pdes, level set methods, and the stereo problem. *IEEE Transactions on Image Processing*, 7(3):336–344, 1998.

[4] R.T. Frankot and R. Chellappa. *Shape from Shading*, chapter Obtaining Shape from Shading Information. MIT Press, Cambridge , Mass, 1989.

[5] B.K.P. Horn. Shape from shading: A method for obtaining the shape of a smooth opaque object from one view. *PhD thesis, Massachusetts Institute Of Technology, Cambridge, MA*, 1970.

[6] B.K.P Horn. *Obtaining Shape from Shading Information*, pages 115–155. McGraw Hill, NY, 1975.

[7] B.K.P Horn and M.J. Brooks. Shape and source from shading. *International Joint Conference on Artificial Intelligence*, pages 932–936, 1985.

[8] J.Y.A.Wang and E.H. Adelson. Layered representation for motion analysis. *CVPR*, pages 361–366, 1993.

[9] J.J. Koenderink and A.J. van Doorn. Surface shape and curvature scales. *Image Vision Comput. 10*, pages 557–565, 1992.

[10] K.N. Kutulakos and S.M. Seitz. A theory of shape by space carving. *IJCV*, 38(3):199–218, 2000.

[11] D. Marr and T. Poggio. Cooperative computation of stereo disparity. *International Journal of Computer Vision*, 194:283 287, 1976.

[12] D.C. Marr. *Vision*. W. H. Freeman and Company, San Francisco, 1982.

[13] E. Mingolla and J. T. Todd. Perception of solid shape from shading. *Biological Cybernetics*, 53:137–151, 1986.

[14] A.P. Pentland. Local shading analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:170–187, 1984.

[15] A.P. Pentland. Shape information from shading: a theory about human perception. *Proceedings of International Conference on Computer Vision*, page 404 413, 1988.

[16] J.E. Cryer R. Zhang, P. Tsai and M. Shah. Shape from shading: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(8):690–706, 1999.

[17] V.S. Ramachandran. Perceiving shape from shading. *Scientific American*, 159:76–83, 1988.

[18] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47:7–42, 2002.

[19] W. Smith. Face recognition using shape from shading, 2000.

[20] R. Szeliski and P. Golland. Stereo matching with transparency and matting. *IJCV*, 32(1):45–61, 1999.

[21] V. Torre T. Poggio and C. Koch. Computational vision and regularization theory. *Nature*, 317:314–319, 1985.

[22] D. Terzopoulos and K. Fleischer. Deformable models. *The Visual Computer*, 4(6):306–331, 1988.

[23] P.S. Tsai and M. Shah. Shape from shading using linear approximation. *Image and Vision Computing Journal*, 12(8):487 –498, 1994.

[24] R.J. Woodham. Photometric stereo: A reflectance map technique for determining surface orientation from image intensity. *Image Understanding Systems and Industrial Applications, Proc. S.P.I.E*, 155, 1978.

[25] R.J. Woodham. Gradient and curvature from the photometric-stereo method, including local confidence estimation. *Journal of the Optical Society of America*, 11(11):3050–3068, 1994.

[26] P.L Worthington. Novel view synthesis using needle-map correspondence. *?*, pages 718–727, 2002.

[27] P.L Worthington and E.R. Hancock. New constraints on data-closeness and needle map consistency for shape-from-shading. *Pattern Recognition*, 21(12):1250–1267, 1999.

[28] P.L Worthington and E.R. Hancock. Object recognition using shape-from-shading. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 31–40, 2001.

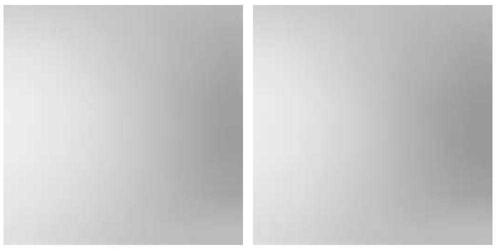
[29] P.L Worthington and E.R. Hancock. Surface topography using shape-from-shading. *Pattern Recognition*, 34:823–840, 2001.

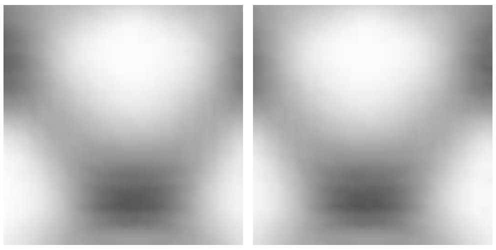
[30] A. Yuille Y. Yang and J. Lu. Local, global, and multilevel stereo matching. *CVPR*, pages 274–279, 1993.

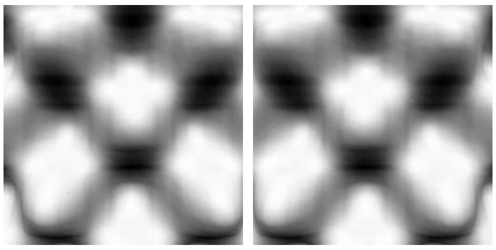
[31] Y. Yeshurun and E.L. Schwartz. Cepstral filtering on a columnar image artchitecture: A fast algorithm for binocular stereo segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):759–767, 1989.

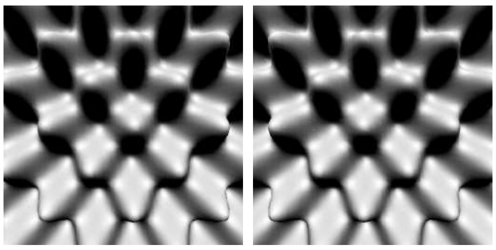
A Test data

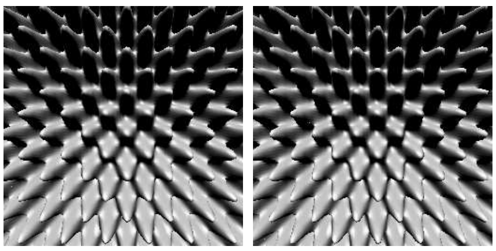
Frequency:



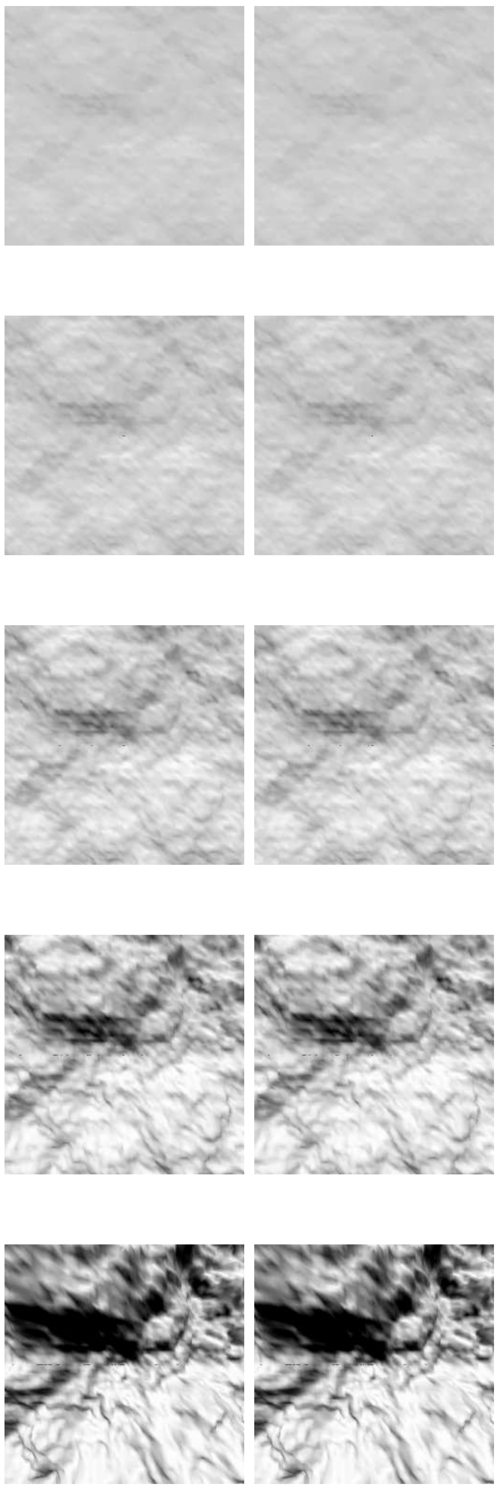




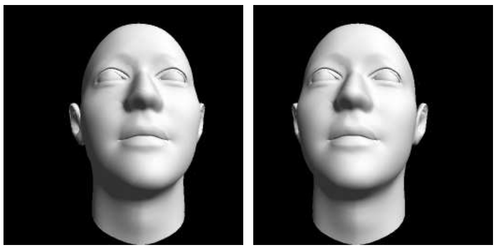


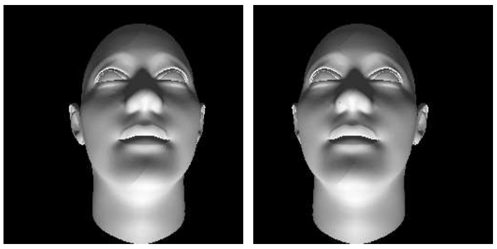


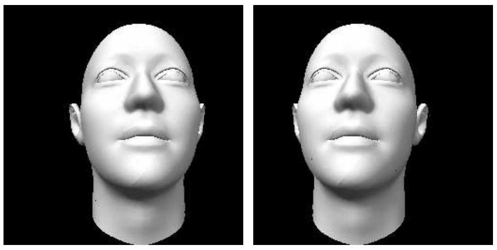
Amplitude:

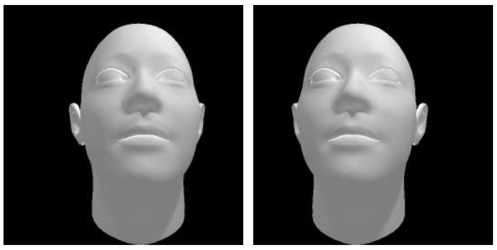


Lighting:

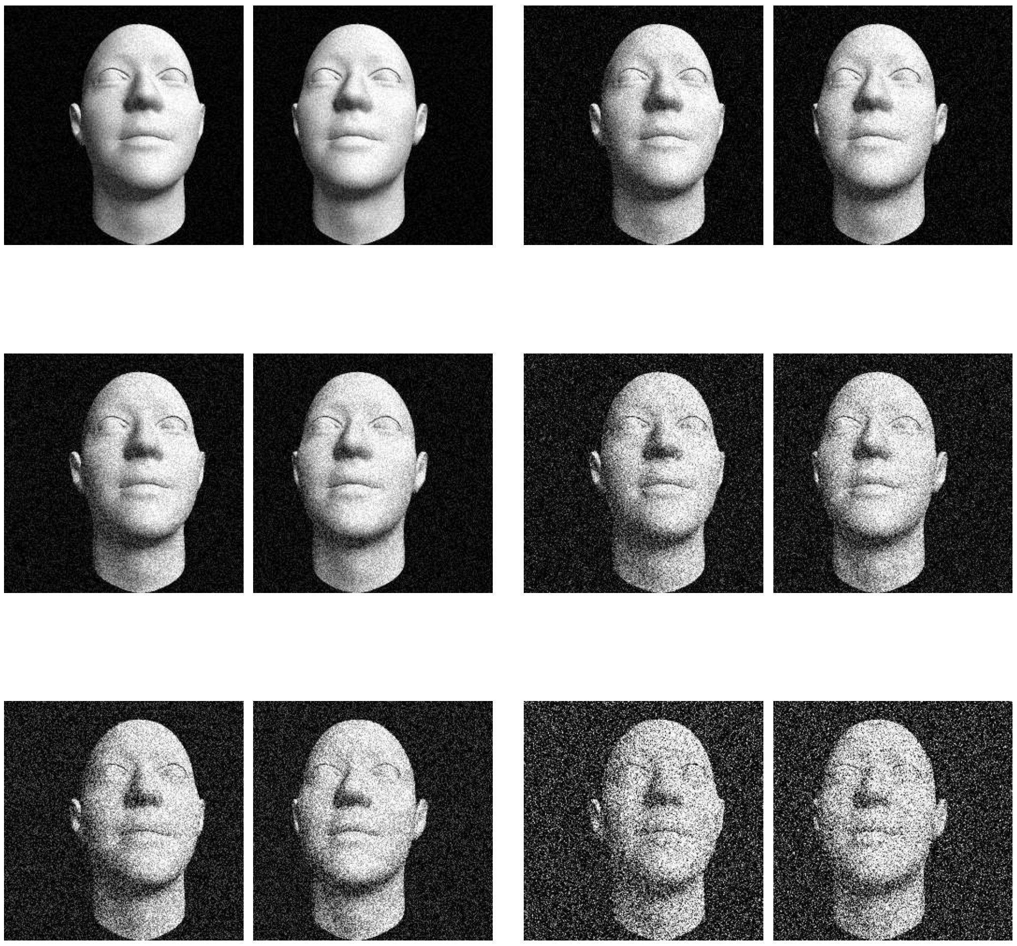




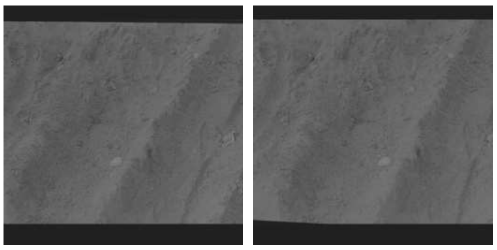


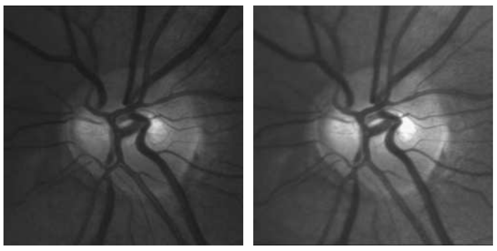


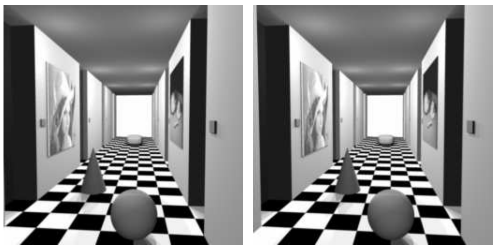
Noise:



Real world:







B Source code

SCImage.h

```
/*
 *  Image.h
 *  StereoMatch
 *
 *  Created by Will Thimbleby on Wed Feb 05 2003.
 *  Copyright (c) 2003 Will Thimbleby. All rights reserved.
 *
 */

#include "SMatrix.h"

class SCImage : public SMatrix
{
public:

        SCImage();
        SCImage(int w, int h);

        void ReadFromFile(char *fname);
        void WriteToFile(char *fname);
};
```

SImage.h

```
/*
 *  Image.h
 *  StereoMatch
 *
 *  Created by Will Thimbleby on Wed Feb 05 2003.
 *  Copyright (c) 2003 Will Thimbleby. All rights reserved.
 *
 */

#include "SMatrix.h"

class SImage : public SMatrix
{
public:

        SImage();
        SImage(int w, int h);

        void Normalise();
        void Linear(double m, double c);

        void ReadFromFile(char *fname);
        void WriteToFile(char *fname);
};
```

SMatrix.h

```
/*
 *  Matrix.h
 *  StereoMatch
 *
 *  Created by Will Thimbleby on Wed Feb 05 2003.
 *  Copyright (c) 2003 Will Thimbleby. All rights reserved.
 *
 */

#ifndef SMATRIX
#define SMATRIX

class SMatrix
{
public:
        double  *data;
        int             width, height, depth;

        SMatrix();
        SMatrix(int w, int h = 1, int d = 1);
        ~SMatrix();
        void SetSize(int w, int h = 1, int d = 1);

        double& operator() (int x, int y = 0, int z = 0);
        double  operator() (int x, int y = 0, int z = 0) const;

        void GaussSmooth(double SMOOTH);
};

#endif
```

SNormals.h

```
/*
 *  Normals.h
 *  StereoMatch
 *
 *  Created by Will Thimbleby on Wed Feb 05 2003.
 *  Copyright (c) 2003 Will Thimbleby. All rights reserved.
 *
 */

#include "SMatrix.h"

class SNormals : public SMatrix
{
public:

        SNormals();
        SNormals(int w, int h);

        void ReadFromFile(char *fname);
        void WriteToFile(char *fname);
};
```

Evidence.cpp

```
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include "SImage.h"
#include "SCImage.h"
#include "SNormals.h"
```

```
#define abs(x) ((x) < 0? -(x) : (x))
#define PI                                      acos(-1)

SMatrix& Hessian(SNormals &normals)
{
        SMatrix                 *hessian;
        double                  Hessian[2][2], TempMat[2][2], C[2][2], S, c, temp;
        double                  k1, k2, Nx, Ny, Nz, p, q;
        int                             i, j;

        hessian = new SMatrix(normals.width, normals.height, 4);

        for(i=1;i<normals.width-1;i++)
        {
                for(j=1;j<normals.height-1;j++)
                {
                        /*      H =     (0,0    0,1)    */
                        /*              (1,0    1,1)    */
                        //printf("%f,%f,%f\n",normals(i,j,0),normals(i,j,1),normals(i,j,2));

                        Hessian[0][0] = (normals(i,j,0)/normals(i,j,2))-(normals(i-1,j,0)/normals(i-1,j,2));
                        Hessian[0][1] = (normals(i,j,1)/normals(i,j,2))-(normals(i-1,j,1)/normals(i-1,j,2));
                        Hessian[1][0] = (normals(i,j,0)/normals(i,j,2))-(normals(i,j-1,0)/normals(i,j-1,2));
                        Hessian[1][1] = (normals(i,j,1)/normals(i,j,2))-(normals(i,j-1,1)/normals(i,j-1,2));

                        // Implementation of Woodhams viewpoint-invariant curvature calculations

                        Nx = normals(i,j,0)/normals(i,j,2);
                        Ny = normals(i,j,1)/normals(i,j,2);
                        Nz = normals(i,j,2)/normals(i,j,2);

                        p = -Nx;
                        q = -Ny;

                        TempMat[0][0] = (q*q) + 1;
                        TempMat[0][1] = -(p*q);
                        TempMat[1][0] = -(p*q);
                        TempMat[1][1] = (p*p) + 1;

                        temp = 1 / sqrt( pow(1 + (p*p) + (q*q), 3) );

                        TempMat[0][0] *= temp;
                        TempMat[0][1] *= temp;
                        TempMat[1][0] *= temp;
                        TempMat[1][1] *= temp;

                        C[0][0] = (TempMat[0][0]*Hessian[0][0])+(TempMat[0][1]*Hessian[1][0]);
                        C[0][1] = (TempMat[0][0]*Hessian[0][1])+(TempMat[0][1]*Hessian[1][1]);
                        C[1][0] = (TempMat[1][0]*Hessian[0][0])+(TempMat[1][1]*Hessian[1][0]);
                        C[1][1] = (TempMat[1][0]*Hessian[0][1])+(TempMat[1][1]*Hessian[1][1]);

                        (*hessian)(i, j, 0) = C[0][0];
                        (*hessian)(i, j, 1) = C[0][1];
                        (*hessian)(i, j, 2) = C[1][0];
                        (*hessian)(i, j, 3) = C[1][1];
                }
        }

        return *hessian;
}

SImage& Curvedness(SMatrix &hessian)
{
        double                  S, c;
        double                  k1, k2;
        SImage                  *curvedness = new SImage(hessian.width, hessian.height);
        int                             i, j;
        for(i=1;i<hessian.width-1;i++)
        {
                for(j=1;j<hessian.height-1;j++)
                {
                        c = pow(hessian(i,j,0) - hessian(i,j,3),2) + 4*(hessian(i,j,2)*hessian(i,j,1));
                        if (c<0)
                        {
                                c = (hessian(i,j,2)+hessian(i,j,1))/2;
                                c = pow(hessian(i,j,0) - hessian(i,j,3),2) + 4*c*c;
                        }
                        S = sqrt(c);

                        k1 = -0.5*(hessian(i,j,0)+hessian(i,j,3)-S);
                        k2 = -0.5*(hessian(i,j,0)+hessian(i,j,3)+S);

                        (*curvedness)(i, j) = sqrt(k2*k2+k1*k1);
                }
        }

        return *curvedness;
}



SImage& MaxCurve(SMatrix &hessian)
{
        double                  S, c;
        double                  k1, k2;
        SImage                  *curve = new SImage(hessian.width, hessian.height);
        int                             i, j;

        for(i=1;i<hessian.width-1;i++)
        {
                for(j=1;j<hessian.height-1;j++)
                {
                        c = pow(hessian(i,j,0) - hessian(i,j,3),2) + 4*(hessian(i,j,2)*hessian(i,j,1));
                        if (c<0)
                        {
                                c = (hessian(i,j,2)+hessian(i,j,1))/2;
                                c = pow(hessian(i,j,0) - hessian(i,j,3),2) + 4*c*c;
                        }
                        S = sqrt(c);

                        k1 = -0.5*(hessian(i,j,0)+hessian(i,j,3)-S);
                        k2 = -0.5*(hessian(i,j,0)+hessian(i,j,3)+S);

                        (*curve)(i, j) = k1;
                }
        }

        return *curve;
}

SImage& ShapeIndex(SMatrix &hessian)
{
        double                  S, c;
        double                  k1, k2;
        SImage                  *shapeIndex = new SImage(hessian.width, hessian.height);
        int                             i, j;

        for(i=1;i<hessian.width-1;i++)
        {
                for(j=1;j<hessian.height-1;j++)
                {
                        c = pow(hessian(i,j,0) - hessian(i,j,3),2) + 4*(hessian(i,j,2)*hessian(i,j,1));
                        if (c<0)
                        {
                                c = (hessian(i,j,2)+hessian(i,j,1))/2;
                                c = pow(hessian(i,j,0) - hessian(i,j,3),2) + 4*c*c;
                        }
                        S = sqrt(c);

                        k1 = -0.5*(hessian(i,j,0)+hessian(i,j,3)-S);
                        k2 = -0.5*(hessian(i,j,0)+hessian(i,j,3)+S);

                        (*shapeIndex)(i, j) = 2/PI*atan((k2+k1)/(k2-k1));
                }
        }

        return *shapeIndex;
}

SImage& Mean(SMatrix &hessian)
{
        double                  S, c;
        double                  k1, k2;
        SImage                  *shapeIndex = new SImage(hessian.width, hessian.height);
        int                             i, j;

        for(i=1;i<hessian.width-1;i++)
        {
                for(j=1;j<hessian.height-1;j++)
                {
                        c = pow(hessian(i,j,0) - hessian(i,j,3),2) + 4*(hessian(i,j,2)*hessian(i,j,1));
                        if (c<0)
                        {
                                c = (hessian(i,j,2)+hessian(i,j,1))/2;
                                c = pow(hessian(i,j,0) - hessian(i,j,3),2) + 4*c*c;
                        }
                        S = sqrt(c);

                        k1 = -0.5*(hessian(i,j,0)+hessian(i,j,3)-S);
                        k2 = -0.5*(hessian(i,j,0)+hessian(i,j,3)+S);

                        (*shapeIndex)(i, j) = 0.5 * (k1 + k2);
                }
        }

        return *shapeIndex;
}
SImage& Gauss(SMatrix &hessian)
{
        double                  Hessian[2][2], S, c;
        double                  k1, k2;
        SImage                  *shapeIndex = new SImage(hessian.width, hessian.height);
        int                             i, j;

        for(i=1;i<hessian.width-1;i++)
        {
                for(j=1;j<hessian.height-1;j++)
                {
                        c = pow(hessian(i,j,0) - hessian(i,j,3),2) + 4*(hessian(i,j,2)*hessian(i,j,1));
                        if (c<0)
                        {
                                c = (hessian(i,j,2)+hessian(i,j,1))/2;
                                c = pow(hessian(i,j,0) - hessian(i,j,3),2) + 4*c*c;
                        }
                        S = sqrt(c);

                        k1 = -0.5*(hessian(i,j,0)+hessian(i,j,3)-S);
                        k2 = -0.5*(hessian(i,j,0)+hessian(i,j,3)+S);
                        //printf("G:%f,%f=%f\n",k1,k2,k1*k2);

                        (*shapeIndex)(i, j) = k1 * k2;
                }
        }

        return *shapeIndex;
}

#pragma mark -

double E_Dot(SNormals &left, SNormals &right, int x, int y, int xx, int yy)
{
        double          xD, yD, zD;

        xD = left(x, y, 0) * right(xx, yy, 0);
        yD = left(x, y, 1) * right(xx, yy, 1);
        zD = left(x, y, 2) * right(xx, yy, 2);

        return (exp(xD+yD+zD)-exp(0.5))/(exp(1)-exp(0.5));
}

double E_DotCurve(SNormals &left, SNormals &right, int x, int y, int xx, int yy)
{
        static SImage           siLeft, siRight;
        static bool                     inited = 0;

        if(!inited)
        {
                siLeft = Curvedness(Hessian(left));
                siRight = Curvedness(Hessian(right));
                inited = 1;
        }

        double          xD, yD, zD;

        xD = left(x, y, 0) * right(xx, yy, 0);
        yD = left(x, y, 1) * right(xx, yy, 1);
        zD = left(x, y, 2) * right(xx, yy, 2);

        return (xD+yD+zD)+exp(-abs(siLeft(x,y)-siRight(x,y)));
}

double E_SI(SNormals &left, SNormals &right, int x, int y, int xx, int yy)
{
        static SImage           siLeft, siRight;
        static bool                     inited = 0;

        if(!inited)
        {
                siLeft = ShapeIndex(Hessian(left));
                siRight = ShapeIndex(Hessian(right));
                inited = 1;
        }

        double angle = siLeft(x, y);
        double angle2 = siRight(xx, yy);

        angle = angle-angle2;
        if(angle<0)angle*=-1;
        if(angle > 1)
                angle = 2-angle;

        return -angle;
}

double E_SICurve(SNormals &left, SNormals &right, int x, int y, int xx, int yy)
{
        static SImage           siLeft, siRight;
        static SImage           cLeft, cRight;
        static bool                     inited = 0;
        if(!inited)
        {
                siLeft = ShapeIndex(Hessian(left));
                siRight = ShapeIndex(Hessian(right));
                cLeft = Curvedness(Hessian(left));
                cRight = Curvedness(Hessian(right));
                inited = 1;
        }

        double angle = siLeft(x, y);
        double angle2 = siRight(xx, yy);
        double c = cLeft(x, y);
        double c2 = cRight(xx, yy);

        angle = angle-angle2;
        if(angle<0)angle*=-1;
        if(angle > 1)
                angle = 2-angle;

        return -angle+exp(-abs(c-c2));
}

double E_MeanSqr(SNormals &left, SNormals &right, int x, int y, int xx, int yy)
{
        static SImage           siLeft, siRight;
        static bool                     inited = 0;

        if(!inited)
        {
                siLeft = Mean(Hessian(left));
                siRight = Mean(Hessian(right));
                inited = 1;
        }

        double m1 = siLeft(x, y);
        double m2 = siRight(xx, yy);

        return -(m1-m2)*(m1-m2);
}

double E_MeanAbs(SNormals &left, SNormals &right, int x, int y, int xx, int yy)
{
        static SImage           siLeft, siRight;
        static bool                     inited = 0;

        if(!inited)
        {
                siLeft = Mean(Hessian(left));
                siRight = Mean(Hessian(right));
                inited = 1;
        }

        double m1 = siLeft(x, y);
        double m2 = siRight(xx, yy);

        return -abs(m1-m2);
}


double E_GaussSqr(SNormals &left, SNormals &right, int x, int y, int xx, int yy)
{
        static SImage           siLeft, siRight;
        static bool                     inited = 0;

        if(!inited)
        {
                siLeft = Gauss(Hessian(left));
                siRight = Gauss(Hessian(right));
                inited = 1;
        }

        double m1 = siLeft(x, y);
        double m2 = siRight(xx, yy);

        return -(m1-m2)*(m1-m2);
}

double E_GaussAbs(SNormals &left, SNormals &right, int x, int y, int xx, int yy)
{
        static SImage           siLeft, siRight;
        static bool                     inited = 0;

        if(!inited)
        {
                siLeft = Gauss(Hessian(left));
                siRight = Gauss(Hessian(right));
                inited = 1;
        }

        double m1 = siLeft(x, y);
        double m2 = siRight(xx, yy);

        return -abs(m1-m2);
}


double E_CurveAbs(SNormals &left, SNormals &right, int x, int y, int xx, int yy)
{
        static SImage           siLeft, siRight;
        static bool                     inited = 0;

        if(!inited)
        {
                siLeft = Curvedness(Hessian(left));
                siRight = Curvedness(Hessian(right));
                inited = 1;
        }

        double m1 = siLeft(x, y);
        double m2 = siRight(xx, yy);

        return -abs(m1-m2);
}


double E_CurveSqr(SNormals &left, SNormals &right, int x, int y, int xx, int yy)
{
        static SImage           siLeft, siRight;
        static bool                     inited = 0;

        if(!inited)
        {
                siLeft = Curvedness(Hessian(left));
                siRight = Curvedness(Hessian(right));
                inited = 1;
        }

        double m1 = siLeft(x, y);
        double m2 = siRight(xx, yy);

        return -(m1-m2)*(m1-m2);
}

double E_MaxCurveSqr(SNormals &left, SNormals &right, int x, int y, int xx, int yy)
{
        static SImage           siLeft, siRight;
        static bool                     inited = 0;

        if(!inited)
        {
                siLeft = MaxCurve(Hessian(left));
                siRight = MaxCurve(Hessian(right));
                inited = 1;
        }

        double m1 = siLeft(x, y);
        double m2 = siRight(xx, yy);

        return -(m1-m2)*(m1-m2);
}


double E_MaxCurveAbs(SNormals &left, SNormals &right, int x, int y, int xx, int yy)
{
        static SImage           siLeft, siRight;
        static bool                     inited = 0;

        if(!inited)
        {
                siLeft = MaxCurve(Hessian(left));
                siRight = MaxCurve(Hessian(right));
                inited = 1;
        }

        double m1 = siLeft(x, y);
        double m2 = siRight(xx, yy);

        return -abs(m1-m2);
}

double E_2DMaxCurve(SNormals &left, SNormals &right, int x, int y, int xx, int yy)
{
        static SMatrix          lh, rh;
        static bool                     inited = 0;
        double                          xl, yl, xr, yr, S, c;

        if(!inited)
        {
                lh = Hessian(left);
                rh = Hessian(right);
                inited = 1;
        }

        c = pow(lh(x,y,0) - lh(x,y,3),2) + 4*(lh(x,y,2)*lh(x,y,1));
        if (c<0)
        {
                c = (lh(x,y,2)+lh(x,y,1))/2;
                c = pow(lh(x,y,0) - lh(x,y,3),2) + 4*c*c;
        }
        S = sqrt(c);

        if(lh(x,y,0) >= lh(x,y,3))
        {
                xl = 0.5*(lh(x,y,0)-lh(x,y,3)+S);
                yl = lh(x,y,2);
        }
        else
        {
                xl = -lh(x,y,1);
                yl = 0.5*(lh(x,y,0)-lh(x,y,3)-S);
        }

        c = pow(rh(xx,yy,0) - rh(xx,yy,3),2) + 4*(rh(xx,yy,2)*rh(xx,yy,1));
        if (c<0)
        {
                c = (rh(xx,yy,2)+rh(xx,yy,1))/2;
                c = pow(rh(xx,yy,0) - rh(xx,yy,3),2) + 4*c*c;
        }
        S = sqrt(c);

        if(rh(xx,yy,0) >= rh(xx,yy,3))
        {
                xr = 0.5*(rh(xx,yy,0)-rh(xx,yy,3)+S);
                yr = rh(xx,yy,2);
        }
        else
        {
                xr = -rh(xx,yy,1);
                yr = 0.5*(rh(xx,yy,0)-rh(xx,yy,3)-S);
        }

        return xl*xr+yl*yr;
}

double E_3DMaxCurve(SNormals &left, SNormals &right, int x, int y, int xx, int yy)
{
        static SMatrix          lh, rh;
        static bool                     inited = 0;
        double                          xl, yl, xr, yr, S, c;
        double                          xl3, xr3, yl3, yr3, zl3, zr3, len;

        if(!inited)
        {
                lh = Hessian(left);
                rh = Hessian(right);
                inited = 1;
        }

        c = pow(lh(x,y,0) - lh(x,y,3),2) + 4*(lh(x,y,2)*lh(x,y,1));
        if (c<0)
        {
                c = (lh(x,y,2)+lh(x,y,1))/2;
                c = pow(lh(x,y,0) - lh(x,y,3),2) + 4*c*c;
        }
        S = sqrt(c);

        if(lh(x,y,0) >= lh(x,y,3))
        {
                xl = 0.5*(lh(x,y,0)-lh(x,y,3)+S);
                yl = lh(x,y,2);
        }
        else
        {
                xl = -lh(x,y,1);
                yl = 0.5*(lh(x,y,0)-lh(x,y,3)-S);
        }

        c = pow(rh(xx,yy,0) - rh(xx,yy,3),2) + 4*(rh(xx,yy,2)*rh(xx,yy,1));
        if (c<0)
        {
                c = (rh(xx,yy,2)+rh(xx,yy,1))/2;
                c = pow(rh(xx,yy,0) - rh(xx,yy,3),2) + 4*c*c;
        }
        S = sqrt(c);

        if(rh(xx,yy,0) >= rh(xx,yy,3))
        {
                xr = 0.5*(rh(xx,yy,0)-rh(xx,yy,3)+S);
                yr = rh(xx,yy,2);
        }
        else
        {
                xr = -rh(xx,yy,1);
                yr = 0.5*(rh(xx,yy,0)-rh(xx,yy,3)-S);
        }

        //convert to 3d

        xl3 = xl*left(x,y,2);
        yl3 = yl*-left(x,y,2);
        zl3 = xl*left(x,y,1)+yl*-left(x,y,0);
        len = sqrt(xl3*xl3+yl3*yl3+zl3*zl3);
        xl3 /= len; yl3 /= 3; zl3 /= 3;

        xr3 = xr*right(xx,yy,2);
        yr3 = yr*-right(xx,yy,2);
        zr3 = xr*right(xx,yy,1)+yr*-right(xx,yy,0);
        len = sqrt(xr3*xr3+yr3*yr3+zr3*zr3);
        xr3 /= len; yr3 /= 3; zr3 /= 3;

        return xl3*xr3+yl3*yr3+zl3*zr3;
}
```

SCImage.cpp

```
/*
 *  SCImage.cpp
 *  StereoMatch
 *
 *  Created by Will Thimbleby on Wed Feb 05 2003.
 *  Copyright (c) 2003 Will Thimbleby. All rights reserved.
 *
 */

#include "SCImage.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAXLINE                 256

SCImage::SCImage() : SMatrix()
{
}

SCImage::SCImage(int w, int h) : SMatrix(w, h, 3)
{
}

void SCImage::ReadFromFile(char *fileName)
{
        FILE            *file;
        char            line[MAXLINE];
        int                     x, y;

        // Open file
        if( !(file = fopen( fileName , "r" )) )
        {
                printf("Error Opening file : %s\n", fileName);
                exit(1);
        }

        //get first line check file is right type
        fgets(line, MAXLINE, file);
        if (strcmp(line,"P5\n") == 0)
        {
                //skip comments
                fgets(line, MAXLINE, file);
                while (line[0] == '#')
                {
                        fgets(line, MAXLINE, file);
                }

                //get size of image
                sscanf(line,"%d %d",&width,&height);

                //get image levels
                fgets(line, MAXLINE, file);
                //sscanf(line,"%d",&width,&height);

                //allocate data
                SetSize(width, height, 3);

                //get the data
                for(y=0; y<height; y++)
                {
                        for(x=0; x<width; x++)
                        {
                                data[x+width*(y+0*height)] =
                                data[x+width*(y+1*height)] =
                                data[x+width*(y+2*height)] = (double)fgetc(file)/255;
                        }
                }

                //close the file
                fclose(file);
                return;
        }
        else if (strcmp(line,"P6\n") != 0)
        {
                printf("Cannot Read This Kind of PGM File\n");
                exit(1);
        }

        //skip comments
        fgets(line, MAXLINE, file);
        while (line[0] == '#')
        {
                fgets(line, MAXLINE, file);
        }

        //get size of image
        sscanf(line,"%d %d",&width,&height);

        //get image levels
        fgets(line, MAXLINE, file);
        //sscanf(line,"%d",&width,&height);

        //allocate data
        SetSize(width, height, 3);

        //get the data
        for(y=0; y<height; y++)
        {
                for(x=0; x<width; x++)
                {
                        data[x+width*(y+0*height)] = (double)fgetc(file)/255;
                        data[x+width*(y+1*height)] = (double)fgetc(file)/255;
                        data[x+width*(y+2*height)] = (double)fgetc(file)/255;
                }
        }

        //close the file
    fclose(file);
}
```

```
void SCImage::WriteToFile(char* fileName)
{
        FILE            *outFile;
        int                     x, y;

        //open the file
        if( !(outFile = fopen( fileName, "w" )) )
        {
                printf("Error Opening file : %s\n", fileName);
                exit(1);
        }

        //write the header data
        fprintf(outFile, "P6\n");
        fprintf(outFile, "#PPM Image Generated by Will Thimbleby\n");
        fprintf(outFile, "%d %d\n%d\n", width, height, 256);

        //write data
        for(y=0; y<height; y++)
        {
                for(x=0; x<width; x++)
                {
                        fprintf(outFile, "%c%c%c", (unsigned char)(data[x+width*(y+0*height)]*255), (unsigned char)(data[x+w
                }
        }

        //close the file
        fclose(outFile);
}
```

SImage.cpp

```
/*
 *  Image.cpp
 *  StereoMatch
 *
 *  Created by Will Thimbleby on Wed Feb 05 2003.
 *  Copyright (c) 2003 Will Thimbleby. All rights reserved.
 *
 */

#include "SImage.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAXLINE                 256

SImage::SImage() : SMatrix()
{
}

SImage::SImage(int w, int h) : SMatrix(w, h, 1)
{
}

void SImage::Linear(double m, double c)
{
        int             x, y;

        for(y=0; y<height; y++)
    {
        for(x=0; x<width; x++)
        {
```

```
                        data[x+width*y] = m*data[x+width*y]+c;
        }
    }
}

void SImage::Normalise()
{
        double  max_disparity, min_disparity, noneofthis;
        int             x, y;

        //normalise levels
        noneofthis = data[0];
        max_disparity = -100000;
        min_disparity = 100000;//data[0];

    for(y=0; y<height; y++)
    {
        for(x=0; x<width; x++)
        {
                        if(data[x+width*y] > max_disparity && !(data[x+width*y] > (noneofthis-1) && data[x+width*y] < (nonec
                                max_disparity = data[x+width*y];
                        if(data[x+width*y] < min_disparity && !(data[x+width*y] > (noneofthis-1) && data[x+width*y] < (nonec
                                min_disparity = data[x+width*y];
        }
    }
        printf("%lf,%lf\n",min_disparity,max_disparity);
    for(y=0; y<height; y++)
    {
        for(x=0; x<width; x++)
        {
                        data[x+width*y] = (data[x+width*y]-min_disparity)/(max_disparity-min_disparity);
        }
    }
}

void SImage::ReadFromFile(char *fileName)
{
        FILE            *file;
        char            line[MAXLINE];
        int                     x, y;

        // Open file
        if( !(file = fopen( fileName , "r" )) )
        {
                printf("Error Opening file : %s\n", fileName);
                exit(1);
        }

        //get first line check file is right type
        fgets(line, MAXLINE, file);
        if (strcmp(line,"P5\n") != 0)
        {
                printf("Cannot Read This Kind of PGM File\n");
                exit(1);
        }

        //skip comments
        fgets(line, MAXLINE, file);
        while (line[0] == '#')
        {
                fgets(line, MAXLINE, file);
        }

        //get size of image
        sscanf(line,"%d %d",&width,&height);

        //allocate data
        SetSize(width, height, 1);
        //get the data
        for(y=0; y<height; y++)
        {
                for(x=0; x<width; x++)
                {
                        data[x+width*y] = (double)fgetc(file)/255;
                }
        }

        //close the file
    fclose(file);
}

void SImage::WriteToFile(char* fileName)
{
        FILE            *outFile;
        int                     x, y;

        //open the file
        if( !(outFile = fopen( fileName, "w" )) )
        {
                printf("Error Opening file : %s\n", fileName);
                exit(1);
        }

        //write the header data
        fprintf(outFile, "P5\n");
        fprintf(outFile, "#PPM Image Generated by Will Thimbleby\n");
        fprintf(outFile, "%d %d\n%d\n", width, height, 256);

        //write data
        for(y=0; y<height; y++)
        {
                for(x=0; x<width; x++)
                {
                        fprintf(outFile, "%c", (unsigned char)(data[x+width*y]*255));
                }
        }

        //close the file
        fclose(outFile);
}
```

SMatrix.cpp

```
/*
 *  Matrix.cpp
 *  StereoMatch
 *
 *  Created by Will Thimbleby on Wed Feb 05 2003.
 *  Copyright (c) 2003 Will Thimbleby. All rights reserved.
 *
 */

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

double  PI=acos(-1);

SMatrix::SMatrix()
{
        width = height = depth = 0;
}
```

```
SMatrix::SMatrix(int w, int h = 1, int d = 1)
{
        SetSize(w, h, d);
}

SMatrix::~SMatrix()
{
        delete data;
}

void SMatrix::SetSize(int w, int h = 1, int d = 1)
{
        width = w;
        height = h;
        depth = d;

        data = new double [width*height*depth];
        if(!data)
                printf("Error allocating matrix");
}

double& SMatrix::operator() (int x, int y = 0, int z = 0)
{
        return data[x+width*(y+height*z)];
}

double SMatrix::operator() (int x, int y = 0, int z = 0) const
{
        return data[x+width*(y+height*z)];
}

void SMatrix::GaussSmooth(double SMOOTH)
{
        double  *tempData = new double [width*height*depth];
        double  pix;
        int             k_sz = (int)(3*SMOOTH), R;
        int             x, y, z, X, Y, Z;
        double  *G = new double [(2+2*k_sz)*(2+2*k_sz)];

        if(SMOOTH > 0)
        {
                //kernel size (3x std)

                //setup gaussian distribution
                for(X=-k_sz; X<=k_sz; X++)
                {
                        for(Y=-k_sz; Y<=k_sz; Y++)
                        {
                                G[(k_sz + X)+(1+2*k_sz)*(k_sz+Y)] = exp(-(X*X+Y*Y)/(double)(2*SMOOTH*SMOOTH))/((double)(SMOO
                        }
                }

                //apply
                for(x=0; x<width; x++)
                        for(y=0; y<height; y++)
                                for(z=0; z<depth; z++)
                                {
                                        pix = 0;
                                        for(X=-k_sz; X<=k_sz; X++)
                                        {
                                                for(Y=-k_sz; Y<=k_sz; Y++)
                                                {
                                                        if((x+X >= 0 && x+X < width) && (y+Y >= 0 && y+Y < height))
                                                                pix = pix + data[(x+X)+width*((y+Y)+height*z)]*G[(k_sz + X)+
                                                }
                                        }
                                        tempData[x+width*(y+height*z)] = pix;
                                }
                //copy into data
                for(x=0; x<width; x++)
                        for(y=0; y<height; y++)
                                for(z=0; z<depth; z++)
                                {
                                        data[x+width*(y+height*z)] = tempData[x+width*(y+height*z)];
                                }
        }

        delete tempData;
        delete G;
}
```

SNormals. cpp

```
/*
 *  Image.cpp
 *  StereoMatch
 *
 *  Created by Will Thimbleby on Wed Feb 05 2003.
 *  Copyright (c) 2003 Will Thimbleby. All rights reserved.
 *
 */

#include "SNormals.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAXLINE                 256

SNormals::SNormals() : SMatrix()
{
}

SNormals::SNormals(int w, int h) : SMatrix(w, h, 3)
{
}

void SNormals::ReadFromFile(char *fileName)
{
        FILE            *file;
        int                     x, y;

        // Open file
        if( !(file = fopen( fileName , "r" )) )
        {
                printf("Error Opening file : %s\n", fileName);
                exit(1);
        }

        //get size of image
        fscanf(file,"%d %d",&width,&height);

        //allocate data
        SetSize(width, height, 3);

        //get the data
        for(y=0; y<height; y++)
        {
                for(x=0; x<width; x++)
                {
                        fscanf(file, "%lf %lf %lf", &(data[x+width*(y+height*0)]), &(data[x+width*(y+height*1)]), &(data[x+w
                }
        }

        //close the file
    fclose(file);
}

void SNormals::WriteToFile(char* fileName)
{
        FILE            *outFile;
        int                     x, y;

        //open the file
        if( !(outFile = fopen( fileName, "w" )) )
        {
                printf("Error Opening file : %s\n", fileName);
                exit(1);
        }

        //write the header data
        fprintf(outFile, "%d %d\n", width, height);

        //write data
        for(y=0; y<height; y++)
        {
                for(x=0; x<width; x++)
                {
                        fprintf(outFile, "%lf %lf %lf", data[x+width*(y+height*0)], data[x+width*(y+height*1)], data[x+widt
                }
        }

        //close the file
        fclose(outFile);
}
```

main.cpp

```
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include "SImage.h"
#include "SCImage.h"
#include "SNormals.h"

#define xDisparity_Max          -50
#define EVID_THRESH                     0.5
#define PI                                      acos(-1)

//crude but works
#define isnan(x) ((x) != (x))
#define abs(x) ((x) < 0? -(x) : (x))

FILE  *outFile;

void StereoMatch(int width, int height, SImage &match, SNormals &left, SNormals &right, double (*compare)(SNormals&, SNorma
void StereoMatch2(int width, int height, SImage &match, SCImage &left, SCImage &right);
void StereoMatch3(int width, int height, SImage &match, SImage &left, SImage &right);

SImage& Curvedness(SMatrix &normals);
SImage& ShapeIndex(SMatrix &normals);
SImage& Mean(SMatrix &normals);
SImage& Gauss(SMatrix &normals);
SImage& MaxCurve(SMatrix &normals);
```

```
SMatrix& Hessian(SNormals &normals);

double E_Dot(SNormals &left, SNormals &right, int x, int y, int xx, int yy);
double E_DotCurve(SNormals &left, SNormals &right, int x, int y, int xx, int yy);
double E_SI(SNormals &left, SNormals &right, int x, int y, int xx, int yy);
double E_SICurve(SNormals &left, SNormals &right, int x, int y, int xx, int yy);
double E_MeanSqr(SNormals &left, SNormals &right, int x, int y, int xx, int yy);
double E_MeanAbs(SNormals &left, SNormals &right, int x, int y, int xx, int yy);
double E_GaussSqr(SNormals &left, SNormals &right, int x, int y, int xx, int yy);
double E_GaussAbs(SNormals &left, SNormals &right, int x, int y, int xx, int yy);
double E_CurveSqr(SNormals &left, SNormals &right, int x, int y, int xx, int yy);
double E_CurveAbs(SNormals &left, SNormals &right, int x, int y, int xx, int yy);
double E_MaxCurveSqr(SNormals &left, SNormals &right, int x, int y, int xx, int yy);
double E_MaxCurveAbs(SNormals &left, SNormals &right, int x, int y, int xx, int yy);
double E_2DMaxCurve(SNormals &left, SNormals &right, int x, int y, int xx, int yy);
double E_3DMaxCurve(SNormals &left, SNormals &right, int x, int y, int xx, int yy);

void CompareDisp(char *data, SImage &truth, SImage &disp)
{
        double  ms, s;
        int             i, j, bad=0;

        ms = 0;
        for(i=0;i<disp.width;i++)
        {
                for(j=0;j<disp.height;j++)
                {
                        s = truth(i,j)-disp(i,j);
                        ms += s*s;

                        if(s*s > 1)
                                bad++;
                }
        }

        fprintf(outFile, "%s\t%f\t%f\n", data, sqrt(ms/(i*j)),(float)bad/(i*j));
}

void appleEv(SImage *truth, SNormals *left, SNormals *right, double (*compare)(SNormals&, SNormals&,int,int,int,int), char *
{
        SImage          *disparity;

        disparity = new SImage(left->width, left->height);

        StereoMatch(*disparity, *left, *right, compare);
        //disparity->Normalise();
    disparity->WriteToFile(file);
        CompareDisp(file, *truth, *disparity);

        delete disparity;
}

int main (int argc, const char * argv[])
//*
{
        outFile = fopen( "results.out", "w" );


    SNormals            *left, *right;
        SImage                  *disparity, *truth;

        //colour image
        //truth = new SImage;
        //truth->ReadFromFile(argv[5]);
    SCImage             *leftIC, *rightIC;

        leftIC = new SCImage;
        rightIC = new SCImage;
```

```
leftIC->ReadFromFile(argv[5]);
rightIC->ReadFromFile(argv[6]);

if(leftIC->width != rightIC->width || leftIC->height != rightIC->height)
{
        printf("Colour Image sizes do not match");
        exit(1);
}
truth = new SImage(leftIC->width, rightIC->height);

StereoMatch2(leftIC->width, leftIC->height, *truth, *leftIC, *rightIC);

//truth->Linear(-1, 255);
//disparity->Normalise();
truth->WriteToFile("colourpixelDisparity.ppm");
//*
//normals
//return;
left = new SNormals;
right = new SNormals;
left->ReadFromFile(argv[1]);
right->ReadFromFile(argv[2]);

if(left->width != right->width || left->height != right->height)
{
        printf("Normal sizes do not match");
        exit(1);
}

//output pretty pics
SMatrix         hl, hr;
SImage          l, r;

hl = Hessian(*left);
hr = Hessian(*right);
l = ShapeIndex(hl);l.Linear(.5, .5);l.WriteToFile("siLeft.ppm");
r = ShapeIndex(hr);r.Linear(.5, .5);r.WriteToFile("siRight.ppm");

l = MaxCurve(hl);
r = MaxCurve(hr);
l.Normalise();
r.Normalise();
l.WriteToFile("maxcurveLeft.ppm");
r.WriteToFile("maxcurveRight.ppm");

l = Curvedness(hl);
r = Curvedness(hr);
l.Normalise();
r.Normalise();
l.WriteToFile("curveLeft.ppm");
r.WriteToFile("curveRight.ppm");

l = Mean(hl);
r = Mean(hr);
l.Normalise();
r.Normalise();
l.WriteToFile("meanLeft.ppm");
r.WriteToFile("meanRight.ppm");

l = Gauss(hl);
r = Gauss(hr);
l.Linear(40,.5);
r.Linear(40,.5);
l.WriteToFile("gaussLeft.ppm");
r.WriteToFile("gaussRight.ppm");
//return;
//calculate disparities
appleEv(truth, left, right, &E_Dot, "dot_normals.ppm");
```

```
        appleEv(truth, left, right, &E_DotCurve, "dotcurve_normals.ppm");
        appleEv(truth, left, right, &E_SI, "shape_index.ppm");
        appleEv(truth, left, right, &E_SICurve, "shapecurve_index.ppm");
        appleEv(truth, left, right, &E_MeanSqr, "mean_sqr.ppm");
        appleEv(truth, left, right, &E_MeanAbs, "mean_abs.ppm");
        appleEv(truth, left, right, &E_GaussSqr, "gauss_sqr.ppm");
        appleEv(truth, left, right, &E_GaussAbs, "gauss_abs.ppm");
        appleEv(truth, left, right, &E_CurveSqr, "curve_sqr.ppm");
        appleEv(truth, left, right, &E_CurveAbs, "curve_abs.ppm");
        appleEv(truth, left, right, &E_MaxCurveSqr, "maxcurve_sqr.ppm");
        appleEv(truth, left, right, &E_MaxCurveAbs, "maxcurve_abs.ppm");
        appleEv(truth, left, right, &E_2DMaxCurve, "2dmaxcurve_abs.ppm");
        appleEv(truth, left, right, &E_3DMaxCurve, "3dmaxcurve_abs.ppm");

        //image
    SCImage             *leftI, *rightI;

        leftI = new SCImage;
        rightI = new SCImage;
        leftI->ReadFromFile(argv[3]);
        rightI->ReadFromFile(argv[4]);

        if(leftI->width != rightI->width || leftI->height != rightI->height)
        {
                printf("Normal sizes do not match");
                exit(1);
        }
        disparity = new SImage(leftI->width, rightI->height);

        StereoMatch2(leftI->width, leftI->height, *disparity, *leftI, *rightI);

        //disparity->Linear(1.0/leftI->width, 0);
        CompareDisp("pixelDisparity.ppm", *truth, *disparity);
        disparity->WriteToFile("pixelDisparity.ppm");

        fclose(outFile);
    return 0;//*/
}

void StereoMatch(SImage &match, SNormals &left, SNormals &right, double (*compare)(SNormals&, SNormals&,int,int,int,int))
{
        int             width, height;

        width = left.width;
        height = left.height;

        int                     x, y;
        int                     x_disparity, y_disparity;
        int                     max_disparity, min_disparity;
        double          xD, yD, zD;
        SMatrix         evidence(width, height);
        SMatrix         confidence(width, height);
        SMatrix         disparity(width, height, 2);

        for(x=0; x<width; x++)
                for(y=0; y<height; y++)
                {
                        confidence(x, y) = -100;
                        match(x, y) = 0;
                        disparity(x, y, 0) = 0;
                        disparity(x, y, 1) = 0;
                }

        for(x_disparity=0; x_disparity>xDisparity_Max; x_disparity--)
        {
                y_disparity = 0;
                //for(y_disparity=-20; y_disparity<=20; y_disparity++)
                {
```

```
          //set all evidences to -100
          for(x=0; x<width; x++)
              for(y=0; y<height; y++)
                  evidence(x, y) = -100;

          for(x=abs(x_disparity)+1; x<width-abs(x_disparity)-1; x++)
          {
              for(y=abs(y_disparity)+1; y<height-abs(y_disparity)-1; y++)
              {
                  evidence(x, y) = compare(left, right, x, y, x+x_disparity, y+y_disparity);
                  /*
                  xD = left(x, y, 0) * right(x+x_disparity, y+y_disparity, 0);
                  yD = left(x, y, 1) * right(x+x_disparity, y+y_disparity, 1);
                  zD = left(x, y, 2) * right(x+x_disparity, y+y_disparity, 2);
                  evidence(x, y) =xD+yD+zD;//*/

                  /*
                  double angle = siLeft(x, y);
                  double angle2 = siRight(x+x_disparity, y+y_disparity);

                  angle = angle-angle2;
                  if(angle<0)angle*=-1;
                  if(angle > .5)
                      angle = 1-angle;
                  evidence(x, y) = 2-angle;//1-angle;//*

                  //*/
                  /*double c = curveLeft(x, y);
                  double c2 = curveRight(x+x_disparity, y+y_disparity);
                  if(isnan(c) || isnan(c2))
                      evidence(x, y) = 0;
                  else
                      evidence(x, y) = -((c-c2)*(c-c2));

                  //evidence(x, y) = (4*angle-(c-c2)*(c-c2));*/

                  //evidence(x, y) = gaussLeft(x, y)-gaussRight(x+x_disparity, y+y_disparity);
                  //evidence(x, y) = 100-evidence(x, y) * evidence(x, y);
              }
          }

          //smooth the evidence function here
          evidence.GaussSmooth(0.7);

          for(y=0; y<height; y++)
          {
              for(x=0; x<width; x++)
              {//printf("%f,%f\n",evidence(x, y) , confidence(x, y));
                  if((/*evidence(x, y) > EVID_THRESH &&*/ /*!isnan(evidence(x, y)) &&*/ evidence(x, y)
                  {
                      confidence(x, y) = evidence(x, y);
                      disparity(x, y, 0) = x_disparity;
                      disparity(x, y, 1) = y_disparity;
                  }
              }
          }
      }
  }

for(y=0; y<height; y++)
{
  for(x=0; x<width; x++)
  {
    if(confidence(x, y) > -100)
          {
              disparity(x, y, 0) = sqrt(disparity(x, y, 0)*disparity(x, y, 0)+disparity(x, y, 1)*disparity
              match(x/*-disparity(x, y, 0)/2*/, y) = disparity(x, y, 0);
          }
```

```
        }
    }
}

#define MAX(a, b) a>b ? a : b
#define MIN(a, b) a>b ? b : a
#define MAX(a, b, c) MAX(a,MAX(b,c)
#define MIN(a, b, c) MIN(a,MIN(b,c)

void RGBtoHSV( double r, double g, double b, double *h, double *s, double *v )
{
        double min, max, delta;

        //min = MIN( r, g, b );
        min = (r>g ? g : r) > b ? b : (r>g ? b : r);
        //max = MAX( r, g, b );
        min = (r>g ? r : g) > b ? (r>g ? b : r) : b;
        *v = max;                               // v

        delta = max - min;

        if( max != 0 )
                *s = delta / max;               // s
        else {
                // r = g = b = 0                // s = 0, v is undefined
                *s = 0;
                *h = -1;
                return;
        }

        if( r == max )
                *h = ( g - b ) / delta;         // between yellow & magenta
        else if( g == max )
                *h = 2 + ( b - r ) / delta;     // between cyan & yellow
        else
                *h = 4 + ( r - g ) / delta;     // between magenta & cyan

        *h *= 60;                               // degrees
        if( *h < 0 )
                *h += 360;

}

void StereoMatch2(int width, int height, SImage &match, SCImage &left, SCImage &right)
{
        int             x, y;
        int             x_disparity, y_disparity;
        int             max_disparity, min_disparity;
        double  xD, yD, zD;
        SMatrix evidence(width, height);
        SMatrix confidence(width, height);
        SMatrix disparity(width, height, 2);

        for(x=0; x<width; x++)
                for(y=0; y<height; y++)
                {
                        confidence(x, y) = 100;
                        match(x, y) = 0;
                }

        for(x_disparity=0; x_disparity>=xDisparity_Max; x_disparity--)
        {
                y_disparity = 0;
                //for(y_disparity=-20; y_disparity<=20; y_disparity++)
                {
                        //set all evidences to -100
                        for(x=0; x<width; x++)
                                for(y=0; y<height; y++)
```

```
                                        evidence(x, y) = 100;

                        for(x=abs(x_disparity); x<width-abs(x_disparity); x++)
                        {
                                for(y=abs(y_disparity); y<height-abs(y_disparity); y++)
                                {
                                        // Original measure used in BMVC02 paper
                                        //*
                                        xD = left(x, y, 0) - right(x+x_disparity, y+y_disparity, 0);
                                        yD = left(x, y, 1) - right(x+x_disparity, y+y_disparity, 1);
                                        zD = left(x, y, 2) - right(x+x_disparity, y+y_disparity, 2);
                                        evidence(x, y) = xD*xD+yD*yD+zD*zD;//*/

                                        /*double lh, ls, lv, rh, rs, rv;
                                        double angle;

                                        RGBtoHSV(left(x, y, 0),left(x, y, 1),left(x, y, 2),&lh,&ls,&lv);
                                        RGBtoHSV(right(x+x_disparity, y+y_disparity, 0),right(x+x_disparity, y+y_disparity,

                                        angle = abs(lh-rh);

                                        if(angle > 180)
                                                angle = 360-angle;

                                        evidence(x, y) = angle+40*(ls-rs)*(ls-rs);//angle+abs(ls-rs)*40;//+abs(lv*rv)*60;
                                */}
                        }

                        //smooth the evidence function here
                        //evidence.GaussSmooth(1);

                        for(x=0; x<width; x++)
                        {
                                for(y=0; y<height; y++)
                                {
                                        if(/*evidence(x, y) > EVID_THRESH &&*/ evidence(x, y) < confidence(x, y))
                                        {
                                                confidence(x, y) = evidence(x, y);
                                                disparity(x, y, 0) = x_disparity;
                                                disparity(x, y, 1) = y_disparity;
                                        }
                                }
                        }
                }
        }

        //normalise levels
    for(y=0; y<height; y++)
    {
        for(x=0; x<width; x++)
        {
                        /*{
                if(disparity(x, y, 0) > max_disparity)
                    max_disparity = disparity(x, y, 0);
                if(disparity(x, y, 0) < min_disparity)
                    min_disparity = disparity(x, y, 0);
            }*/
        }
    }
    for(y=0; y<height; y++)
    {
        for(x=0; x<width; x++)
        {
            disparity(x, y, 0) = sqrt(disparity(x, y, 0)*disparity(x, y, 0)+disparity(x, y, 1)*disparity(x, y, 1));
                        match(x/*-disparity(x, y, 0)/2*/, y) = disparity(x, y, 0);
        }
    }
}
```