

Completion of Occluded Surfaces
M.Sc. Dissertation

Freek Stulp

Supervisors:
Dr. J. Hallam
Dr. H.A.K Mastebroek
Dr. R.B. Fisher

August 10, 2001

Acknowledgements

First of all, I would like to thank Craig Robertson, colleague and friend, for all his help and comment. His advice has been invaluable to this dissertation.

Also a variety of thanks to all my colleagues in the Vision Lab: to Petko for letting me use his code, to Helmut for the many walks through Scotland, and to Neil for correcting my English.

Many thanks to Bob Fisher for hiring me as a Research Assistant at the Vision Lab. There was a risk in hiring an under-grad for the CAMERA project, and I would like to thank him for taking this risk, as well as Esther, Hans and Rineke for writing the letters that convinced him to do so! The fellowship has allowed me to stay in Edinburgh twice as long as expected, and has given me a good start in academic experience. In this context I am much obliged to the EU and the CAMERA project for the financial support and many (work related) trips abroad.

I am also grateful for the advice given to me by my supervisors, H.A.K. Mastebroek of the Rijksuniversiteit Groningen, and J. Hallam of the University of Edinburgh. Working together at a distance, or at altitudes of 28.000 feet is not always easy, but it has worked out fine.

Last, but certainly not least, all my family & friends (a set of people certainly not mutually exclusive with people previously mentioned) for all the support and fun, in and outside of Scotland. The support of my parents throughout a variety of courses in Groningen has been essential to the completion of this one. Thank you Sandra for the support and many diversions in and around Edinburgh.

Thank you all!

Abstract

The CAMERA-project is an EC-funded international network that seeks to acquire three dimensional models of industrial and historical buildings through analysis of digital images; the main focus being on range images.

When taking digital images of scenes, it is hard to capture all the information contained in the real-world scene into the image. Often objects are blocking other objects from the sensors' view. These occluded objects are not fully visible in the image, and data needed for a complete 3D reconstruction is missing.

In this dissertation, a method is presented for using information from the surroundings of an occlusion to hypothesize what would have been seen, had there not been an occlusion. This process is called *completion*. Cognitive aspects of this completion are discussed. The main part of this dissertation is devoted to a method of completing surfaces in range images, and finding a correct way of projecting texture onto these surfaces.

Abstract (Dutch)

Het CAMERA-project is een door de EG gefinancierd internationaal netwerk dat onderzoek doet naar het maken van drie dimensionale modellen van industriële en historische gebouwen. Hiertoe worden *digital images*, en met name de zogenaamde *range images*, geanalyseerd.

Het is moeilijk om alle informatie die besloten ligt in een bepaalde scene vast te leggen in slechts enkele *digital images*. Vaak blokeren bepaalde objecten voor de sensor het zicht op andere objecten. Omdat de verborgen objecten niet geheel zichtbaar zijn, is het erg moeilijk om er een volledig drie dimensionaal model van te maken.

In dit afstudeerverslag wordt een methode gepresenteerd waarbij informatie uit de omgeving van het verborgen object wordt gebruikt om een hypothese op te stellen over wat er te zien zou zijn geweest als het verborgen object niet verborgen was geweest; een proces dat *completie* heet. Een literatuurstudie plaatst dit onderzoek in een cognitief perspectief. Het grootste gedeelte van dit verslag wordt gewijd aan het completeren van vlakken in een *range image*, alsmede het projecteren van licht-intensiteits patronen in de *intensity image* op deze gereconstrueerde vlakken.

Contents

1	Introduction	6
1.1	The CAMERA network	6
1.2	The problem: incomplete information	8
1.3	Cognitive science	9
1.4	... and engineering	9
1.5	An overview	10
2	Perceptual Completion	11
2.1	Introduction	11
2.2	Amodal perceptual completion	12
2.3	Image representation	13
2.4	Filling in	14
2.5	... versus finding out	16
2.6	The debate	17
2.7	Conclusion	19
3	Image Acquisition	21
3.1	Introduction	21
3.2	Digital images	21
3.3	Intensity images	22
3.4	Range images	23
3.5	Summary	28
4	Image Segmentation	29
4.1	Introduction	29
4.2	Noise reduction	30
4.3	Surface characteristics	32
4.4	Region growing	38
4.5	Surface fitting	40
4.6	Summary	45
5	Surface Completion	46
5.1	Introduction	46
5.2	Locating possible occlusions	47
5.3	Making the surface hypothesis	54
5.4	Niche or real occlusion?	57

5.5	Results	58
5.6	Summary	59
6	Texture Completion	61
6.1	Introduction	61
6.2	Extracting a regularly sampled texture image	62
6.3	Hypothesizing the unobserved texture	65
6.4	Warping the completion onto the original image	68
6.5	Results	69
6.6	Proposed improvements	69
6.7	Summary	70
7	Conclusions and Future Work	71
7.1	Conclusive summary	71
7.2	Future work	72
A	Derivative estimation	75
B	Direct curvature computation	77
C	The extended Euclidean distance transform	78
D	Publications, presentations, applications	80
E	Further reading	82
	Bibliography	84

List of Figures

1.1	Merging data from the Bornholm church	7
1.2	The problem of missing data	9
2.1	The Kanizsa triangle	11
2.2	Completion of gray rings	12
2.3	Incomplete people.	13
2.4	Representing an image	14
3.1	Interpreting digital images	22
3.2	A time-of-flight laser scanner	24
3.3	Orthogonal and spherical scanning	25
3.4	2.5D and 3D images	26
3.5	Orthogonal and spherical scanning geometry	27
4.1	The processing chain	29
4.2	Noise caused by cylindrical laser	31
4.3	Locating neighbours	33
4.4	Points without neighbours	34
4.5	Surface derivatives	35
4.6	The principal curvatures	36
4.7	The <i>HK</i> surface classification	37
4.8	Detecting surface discontinuities	39
4.9	A result of the region growing algorithm	39
4.10	Two examples of Euclidean and algebraic distance	42
4.11	A result of the region growing algorithm (2.5D)	44
4.12	A result of the region growing algorithm (3D)	44
5.1	The processing chain of completion	46
5.2	A classification of occlusions of single surfaces	48
5.3	Zero-transection occlusions	49
5.4	Locating zero-transection occlusions	50
5.5	Single-transection occlusions	51
5.6	Multi-transection occlusions	51
5.7	Determining the area between two surface patches	53
5.8	Computing intersections and interpolating between them	55
5.9	The area in which the hypothesis is allowed	57
5.10	A <i>true</i> occlusion and a niche	58

5.11	Some results of the surface completion algorithm	59
6.1	An overview: One step further.	61
6.2	Warping surfaces onto a 2D continuous texture space	62
6.3	Finding an appropriate bounding box	63
6.4	Rotating and translating the data in the bounding box	64
6.5	The result of warping	64
6.6	Determining a likely value for unknown pixels	66
6.7	The completed digital image	67
6.8	A result of both completions	69
7.1	Next-best viewplanning	73
C.1	Distance transform	78

List of Tables

2.5D range image and 3D range image	27
(Geometrical) Surface, Region (or patch), and Surface patch	32
(Geometrical) Surface, Region (or patch), and Surface patch	45
Hypothesized point/surface and Completed point/surface	47
3D Distance and 2D Distance	49

Chapter 1

Introduction

1.1 The CAMERA network

This research is part of, and has been funded by the CAMERA network. CAMERA, or CAD Modelling of Built Environments from Range Analysis, is a European research network whose seven participating institutions are located in six EU countries. The coordinator of the project is Bob Fisher. The CAMERA-project provides post-doctoral researchers with experience in a large international project, and aims to develop cross disciplinary skills in computer vision, computer graphics and computer-aided design (CAD). I have done nine months of research for the CAMERA group at the University of Edinburgh in Scotland, as part of the MSc. course Cognitive Sciences and Engineering (now Artificial Intelligence), at the Rijksuniversiteit Groningen in the Netherlands.

The research objectives of the network are described as follows: *“The project will undertake research into the technology for constructing CAD models of existing industrial and historical buildings [12].”* In this section, I will briefly explain exactly what this research is, in which way the research is done, and what the practical use of the results of this research may be.

The goal is to construct models of existing buildings, varying from industrial plants to ancient churches. This process is often called *reverse engineering*. It is like deriving the original blueprint of a building from the building itself, instead of constructing the building from the blueprint. For this reason, reverse engineering is also known as *reconstruction*. Because buildings are objects in three-dimensional (3D) space, an accurate model should also be reconstructed in 3D. A good reconstruction will allow the model to be viewed from any angle and any position.

A simple model can be seen on the right-hand side of figure 1.1. Unfortunately, paper, through its 2D nature, does not lend itself well for rendering 3D images. The CAMERA partner at the Joint Research Centre in Ispra, Italy, is doing research on visualisation of, and navigation through such 3D models. To get a better idea of the 3D models discussed here, visit their web-site at <http://mortimer.jrc.it/sba/>. There are some beautiful demos in the 3D reconstruction section.

The way these 3D models are constructed is by collecting and analysing data describing the building. These are usually in the form of intensity images (digital black-and-white pictures), range images (which will be discussed later on) or video-streams. To acquire a

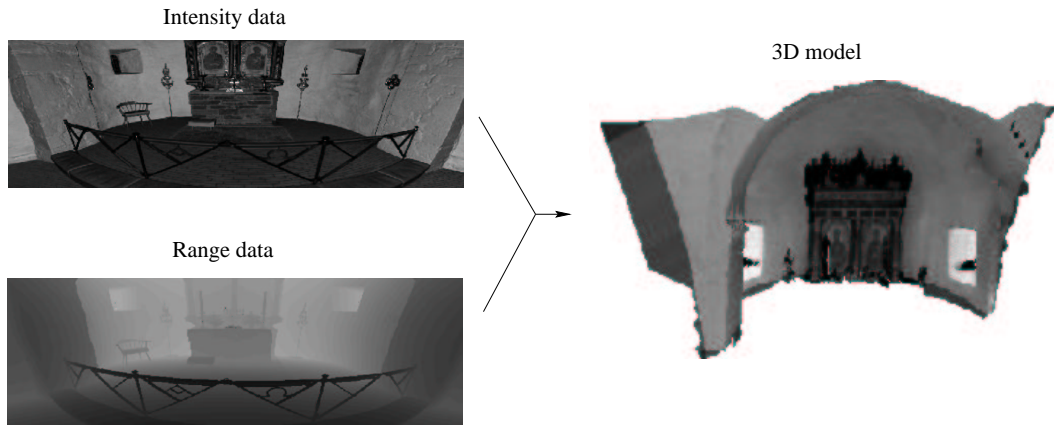


Figure 1.1: Merging different datasets to acquire a 3D model. This is part of a 12th century church on Bornholm Island in Denmark. The model was reconstructed with use of intensity images (texture) and range images (3D geometry).

good model, this data has to be interpreted, and often different data sources have to be merged. Some issues and research topics are:

- Registration: Merging different datasets. Figure 1.1 shows how two types of data can be merged to acquire a textured model in three dimensions.
- Extracting 3D information from 2D data: Video-streams and intensity images do not directly encode 3D information. It is possible to indirectly extract 3D information from them through various methods. One of them is that constantly used by humans: stereopsis.
- Coping with missing data: Often the measured data does not fully describe the real-world scene: information is lacking. This poses serious problems when reconstructing. Methods have to be found to fill in the missing data. This dissertation addresses this specific problem.

The industrial sector has many uses for these 3D models. A good example is modification planning. Often industrial sites have a multitude of pipes, wires and switches running through and alongside walls. Keeping a good record of the locations and dimensions of all these structures is often difficult. Extensive re-measuring and many man-hours are needed to figure out where to fit new pipes or wires. With a 3D model of the site the computer could compute the most efficient way in which the new structure could be inserted. Of course this is not limited to tubes and wires only. Expanding a building or adding a new room could be treated in much the same way. Note that a 2D-model would be of little use for this task, because the 3D lay-out (continuation of tubes through walls for instance) can not be represented in 2 dimensions.

The tourism industry could also enhance its tours and web-sites to a great extent if they had virtual models of some of their main features. Historic Scotland could add a model of Edinburgh Castle to its web-site for instance, enabling you to *walk* through a virtual model of the castle on-line. Some might comment that this negates the need to go to the castle itself, but then again, most people seem to feel that actually seeing Rembrandts' painting

the Nachtwacht is far more impressive than seeing its 2D-models (pictures) rendered on postcards.

Archaeology might also benefit from these techniques. Those investigating archaeological excavation sites could make regular scans and construct 3D models of the site to keep track of progress and make a 3D map of the locations of certain findings. When the excavation is finished a final enhanced 3D model could be reconstructed to give an impression of what the site might have looked like in the past when it was still functional. This would give the public (and the funders) a better feel for the work done, and could be used to enliven archaeological displays in museums.

1.2 The problem: incomplete information

One of the issues in trying to achieve the goals mentioned in the previous section is that real-world objects or scenes can only be realistically modelled if there is enough information about the object or scene. The reason this is an issue, is that information is often not complete.

Let me explain the problem of incomplete data using the next example. Suppose I were to take 50 photographs of the Vision Lab in Edinburgh, roughly describing most of the room. From these pictures someone could get a good idea of what the Vision Lab looks like, even if they had never been there. A CAD-designer, someone specialised in reconstruction, could probably make a pretty realistic model from these pictures as well.

Suppose I were then to throw away 25 of the pictures describing the southern part of the room. On the basis of these 25 pictures, another subject will get a good idea of what the northern part of the Vision lab looks like. A good guess could be made as to what the southern part looks like, based on the northern part: more wall, more lights, maybe more computers. Important information will always be lacking though. It would be unlikely that the subject guesses that there is a REVERSA laser-scanner, a bookshelf, a stereo, and a television. A CAD-designer would have the same problem. How could the CAD-designer make a model of something that is unknown? Building a complete and accurate model¹ is impossible if the data on which the model is based is not complete.

Losing pictures is not the main cause of incomplete datasets². Another more important reason is that objects can occlude each other. Object A is occluded by object B for a certain sensor, if, in this particular configuration of objects and sensor, object B is blocking object A, or a part of it, from the sensor's view. In other words: if I (object B) stand in front of the TV (object A), you (the sensor) will not be able to see what is on it.

How does occlusion lead to missing data, and how does this affect reconstruction? An example can be seen in figure 1.2. Here, a chair is standing in front of a wall. This means that part of the wall is blocked from the sensors' view, so it is occluded. When the image is shown from another angle, a part of the wall is clearly missing. For the moment it does not really matter what a range image is, or how it can be rotated in three dimensions; what does matter is that it is apparent that occlusions are a serious problem for reconstruction. If a 3D model were to be reconstructed of this small scene, there would be a chair-shaped hole in

¹By modelled *complete and accurate model* I mean complete and accurate enough to satisfy the human eye. Describing the Vision Lab as a box is not accurate enough to fool the human eye. Describing it on a molecular level would create a very accurate model, but this level of detail is not necessary for our purposes.

²"Helmut, where are the K2T factory scenes?"

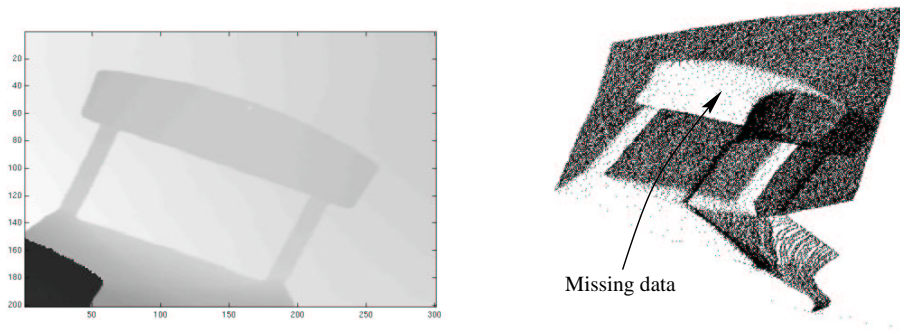


Figure 1.2: A range image (left) and the same image seen from another angle (right).

the wall. The human eye would spot this immediately: the 3D model would be unrealistic. There is an obvious need to fill in the missing data.

1.3 Cognitive science ...

Of course humans suffer from the problem of incomplete information as well. The wall is just as hidden from us as from the sensor. Humans however seem to know intuitively that the wall most likely continues behind the chair. In a literature study, I have tried to discover what this intuition is based on, and if it can be reworked into a functional application.

The term cognitive scientists use for the intuition mentioned above is *perceptual completion*, and it can be demonstrated beautifully using some well-known illusory images shown in chapter 2. In chapter 2 I will elaborate on perceptual completion, and discuss some different explanations for how it emerges. Whether these cognitive processes can be used to create a model on which the program can be based, will be discussed in the conclusion.

1.4 ... and engineering

An obvious way of filling in the missing data is simply collecting more data. If the sensor were placed behind the chair, the wall could certainly be scanned, allowing the missing data to be filled in by merging the two images. This is a *simple* registration problem. This is not always feasible, as the number of scans needed for a complete description of even very simple scenes can exceed hundreds [43].

If collecting more data is not an option, a guess has to be made as to what the missing data would have looked like if it *had* been visible. Preferably it will not be a guess, but a well contemplated and useful hypothesis. To do this, information from the known data surrounding the occluded region in the range image will be used. Summarising:

**The main goal of this dissertation will be:
to hypothesise data that is not visible due to occlusion;
the hypothesis being based on the surroundings of the occluded data.**

This process can be divided into two main subprocesses. First of all, the shape of the occluded area will have to be recovered. In figure 1.2 the occluded object is a planar wall. In

some way or the other, a planar surface will have to be completed behind the chair, keeping in mind that this completed surface should be consistent with the existing wall.

Secondly, a hypothesis will have to be formulated as to what the texture on the reconstructed surface would have looked like. Was it a white or a black wall? Was it homogeneous, or did it have lines, flowers, or Marilyn Monroes on it? A model or template of the known texture will be made, and projected onto the completed, but texture-less surface.

1.5 An overview

In this first chapter a brief overview of the CAMERA project and its goals has been given. The specific research goal of this dissertation has been presented, as well as a short description of possible solutions. The rest of this dissertation will be devoted to elaborate on these solutions, explain the implementation, and show some results of this implementation.

Chapter 2 (**Perceptual Completion**) will describe some of the research done on perceptual completion. Humans use perceptual completion to cope with partially occluded objects. It will be considered which parts, if any, of cognitive perceptual completion can be used to aid the engineering of an algorithm for 3D completion in range and intensity images.

Chapter 3 (**Image Acquisition**) describes what intensity and range images (the two types of data used) are, as well as some methods for acquiring them.

Chapter 4 (**Image Segmentation**) will discuss how the range and intensity images are segmented. The segmentation of these images is very important, because the quality of completion depends on the quality of the segmentation. The segmentation algorithm is kept distinct from the completion algorithm in both theory and implementation. Image segmentation is a relatively mature field in computer vision, whereas the completion algorithm is unprecedented and will contain many new ideas. This chapter therefore allows us to give a more theoretical background of some interesting problems that are encountered in image segmentation and computer vision in general.

Chapter 5 (**Surface Completion**) will show how the occluded data is completed in three dimensional space, using a description of the scene in terms of simple geometric surfaces. The completion is based on taking the surroundings of the occlusion as a representative of the occluded data itself. After a short introduction, the processing chain of the implementation is presented, and some results are shown.

Chapter 6 (**Texture Completion**) discusses the process of projecting texture on the completed surfaces. Again, the method used is presented, followed by some results, similarly to chapter 5.

Chapter 7 (**Conclusions and Future Work**) will draw some conclusions concerning the methods and results of this work. The research has also raised some new questions, and opened possibilities for new areas of research. These will also be discussed in this chapter.

Chapter 2

Perceptual Completion

2.1 Introduction

The cognitive term for *reconstructive* completion is *perceptual completion*. Perceptual completion implies that something seems to be present in a particular region of visual space when it is actually absent from that region, but present in the surrounding area: You see it, even though it's not there. Illusory figures are a good illustration of this phenomenon. A famous example is the *Kanizsa triangle* shown in figure 2.1. In the Kanizsa triangle, illusory contours are perceived of a triangle located between the three black circles. Furthermore, this triangle seems brighter than its surroundings. Both perceptions are illusory, because the paper is homogeneous in luminance.

We are seeing things that are not apparent in the retinal image. This gives a hint as to why research on illusory figures is relevant to our research: we require a program that can 'see' things that, due to occlusion, aren't apparent in the image. It will have to 'see' the wall behind the chair. The relevance will become more apparent in the next section, in which I will also abandon the somewhat undefined terminology of this paragraph.

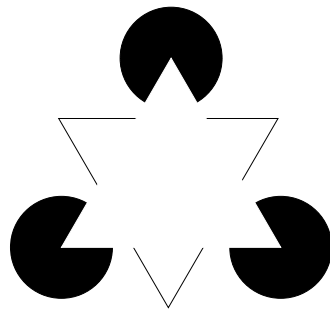


Figure 2.1: The Kanizsa triangle. A bright triangle is perceived, even though it is not present in the stimulus.

2.2 Amodal perceptual completion

For this research it is useful to make the distinction between modal and amodal perceptual completion. In modal perceptual completion, the completed parts display the same types of attributes (or *modes*) as the rest of the figure. Perceptual completion in the Kanizsa illusion is therefore modal. It is caused and expressed as a difference in the same modality: brightness.

Amodal completion is completion of a spatial structure that is not associated with any one modality. To quote Kanizsa himself [29]:

“Amodal completion refers to the completion of an object that is not entirely visible because it is covered or occluded by something else.”

Amodal perceptual reconstruction can best be explained using the example shown in diagram 2.2, taken from Kanizsa and Gerbino [29]. Figure A shows a complete gray ring. In figure B, two halves of two circular rings are shown; or are they two parts of the same gray ring? In experimental studies, subjects report the former: there are two distinct objects in the scene. However, one could imagine that the two rings belong to one and the ring lacking the two central regions. These regions can mentally be interpolated between the two semi-circles. This requires active mental effort, and the interpolation is only a representation, it has no perceptual clarity. One doesn't *see* something that isn't there, one *thinks* it.

Figure C is quite different from figure B. Most subjects do not even consider that there are two gray semi-circles. They report a gray ring occluded by a white rectangle. Joining the two semi-circles into one gray ring no longer requires an active mental process. Their presence possesses *a reality independent of the observer*. Kanizsa and Gerbino call the presence of the central parts in A *modally present*, those in B *represented*, and those in C *encountered*. Amodal completion is reserved for the encountered presence of parts not directly visible.

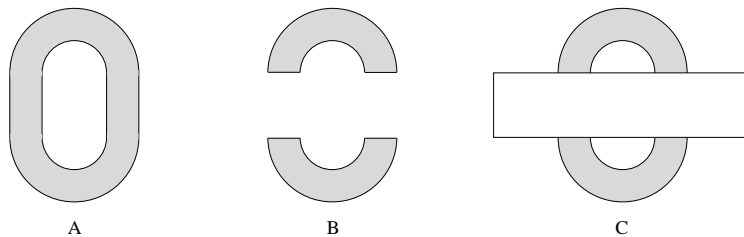


Figure 2.2: The gray rings are: modally present (A), represented (B), or encountered (C).

To clarify what is meant by encountered presence I will give another example, shown in diagram 2.3. In figure A a *piece* of a person can be seen. It is not hard to make a mental representation of the lower body and legs of this person, but it is not apparent perceptually: the missing part is represented, not encountered. Figure B shows the exact same partial person sitting behind a desk. This time, the upper body is not seen as a separate piece, but as the visible part of a whole, but partially occluded, person. The lower body does not need representation, because it is encountered. The difference becomes apparent when the following question is asked: “Is the person wearing trousers?” Given figure A, the answer to

this question would likely be something like: “What a strange question! The person has no legs.”. Given figure B a more probable answer would be: “I don’t know, because I cannot see the legs.” The presence of the legs is not evident in A, but assumed (encountered) in B.

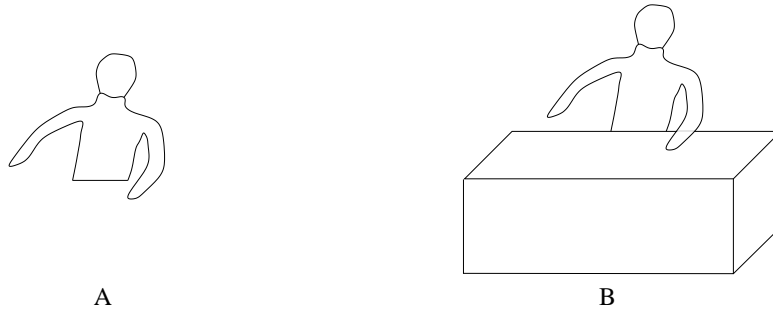


Figure 2.3: A piece of a person (A) and the visible part of a whole person (B).

The relevance of this phenomenon to our research is apparent. Reconstructive completion of data such as seen in figure 1.2 (chair in front of wall) resembles amodal perceptual completion: we see the wall, and just *know* it’s behind the chair as well. It would be convenient if the cognitive process that causes the brain to *encounter* objects, such as the gray ring or the person behind the desk, could be used to *encounter* the wall behind the chair (as seen in figure 1.2) and complete it accordingly.

In the next three sections, two different theories concerning perceptual completion will be discussed. In section 2.6, several arguments for and against both theories will be presented. Section 2.7 will draw some conclusions: can these theories be used to construct a model on which our program can be based? Before I elaborate on the two theories, a brief section on image representation follows. The concepts introduced here will help to explain the theories in sections to come.

2.3 Image representation

Suppose you have drawn figure A of diagram 2.4 in some graphics program on your computer. This picture is taken directly from Dennett’s wonderful book *Consciousness explained* [16]. You are so convinced by its beauty (as apparently Dennett once was) that you want to store it on some medium.

The first way you could do this is to simply print the figure (note that figure A is exactly this printed version). Dennett calls this way of storing color-by-color, because color on the paper is used to represent color on the screen¹.

Another way is to save it as a bit-map (a type of digital image), in which each pixel is encoded by a number. This bit-map representation, which is sensibly called color-by-bit-map, can be seen in figure B. Although it is a roughly continuous representation of the image, interpretation of this data from bit to colour is needed before the original image can be visualised on for instance the screen.

Yet another way of storing the image is by using a compression algorithm, which divides the image into regions of the same color, and stores the bounds and color numbers of all

¹My apologies to those of you who do not own a color printer.

the regions in an archive file. This color-by-number representation could be visualized by an image such as figure C.

It is important to note that color-by-color is the only directly visible image. The other two will need further interpretation by a computer before a true color image can be shown on for instance a screen. The concepts described in this section will be used throughout the following chapters on the different theories explaining perceptual completion.

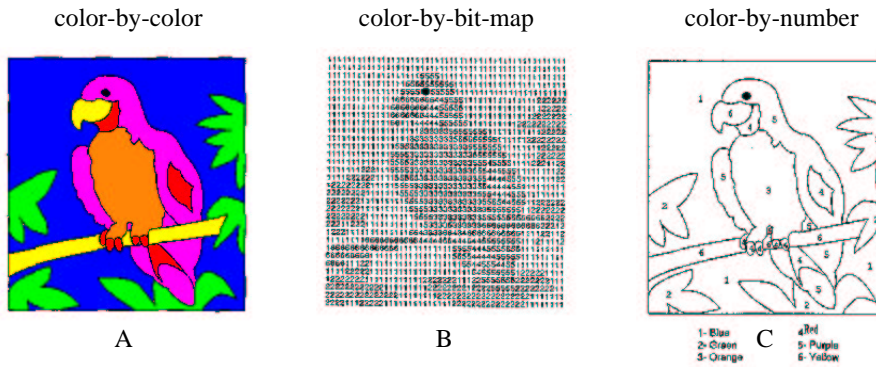


Figure 2.4: Representing an image

2.4 Filling in ...

There are two main theories that try to explain perceptual completion: the brain either *fills in* or *finds out* the missing data². In the next three sections I will explain the two theories.

Dennett has tried to brand ‘filling in’ the ‘F-word’ in cognitive science according to Pessoa, Thompson and Noë [32]. Apart from some strong arguments, Dennett makes the simple but convincing observation that even those who defend the theory dare not use the word ‘filling in’ without scare-brackets. Therefore I will abandon the catch-phrase and call the theory by its correct name: perceptual isomorphism or, more general, *Gestalt isomorphism*. A brief overview of Gestalt theory will be given to place this last term in the right context.

Gestalt theory is founded on the philosophy of epistemological dualism. Theories of visual perception can be separated into two classes: epistemological monism (naive realism), or epistemological dualism (two-worlds hypothesis). Naive realism assumes that the world we see is the objective external world itself. This is the intuitive understanding of vision we accept from the earliest days of childhood. The problem with this view is that the only visual input we have of the objective world is through our eyes. If the brain is the organ of consciousness, then it can never see the world directly, but only indirectly through the two-dimensional images sent to the brain by the eyes. Seeing the world indirectly implies that we see a subjective version of the world instead of the objective world itself.

This is exactly the reasoning of the epistemological dualists such as Kant [30]. Their theory states that we can not experience the world as it actually is, but only an internal perceptual replica of the world. There are therefore two worlds: the external objective

²The slogan “Finding out versus filling in” was suggested by Marcel Kinsbourne at the ZiF conference on the Phenomenal mind, May, 1990, Bielefeld.

world (the nominal world) and the internal perceptual world (the phenomenal world). The phenomenal world is a virtual copy of the external world, constructed in our brain on the basis of images received from the retina. These images are of course caused by light emitting from objects in the nominal world. Therefore, there is certainly a causal relationship between objects in the nominal and phenomenal world, allowing us to manipulate objects in the nominal world through the phenomenal world.

Back to Gestalt theory. The German word ‘Gestalt’ literally means ‘shape’. ‘Gestalt annehmen’ means ‘to take shape’. Gestalt theory states that organisms instinctively perceive whole patterns and not bits and pieces. Whole patterns have characteristics that cannot be perceived by analysing parts. Furthermore, this Gestalt may be perceived before the parts comprising it. The *encountering* in section 2.2 is a good example of this. The entire gray ring (or more accurately: its Gestalt) is encountered, instead of seeing the two semi-circles: the parts comprising it.

The most important concept of Gestalt theory to our research is Gestalt isomorphism, a subtle variation on Müller’s psychophysical axiom [37] which states that the subjective experience of perception can never be of a higher dimensionality than the neurophysiological state by which that experience is encoded. More generally, this concept expresses the materialistic view that consciousness is dependent upon electro-chemical interactions within the physical brain. Gestalt isomorphism is a subtle variation on this axiom, in that it demands that the organisation of experience (the neural configuration) and the underlying psychological facts (the perception) have the same structure [31]; the neural and the perceptual state must have the same Gestalt.

Section 2.3 on image representations can be used to illustrate this. The image projected on the retina is somehow represented in the brain as color-bit-map. Every pixel value³ must be represented in the brain. Gestalt isomorphism does not specify exactly how this should be done though. It could be through neural firing frequencies, specific locations in a neural structure, or any other physical variation possible in the brain. An added constraint is that the structure must also be similar. Somehow the relative spatial locations of the pixels in the image must be encoded in an equivalent structure in the brain. Neighbouring pixels in the image (retina) should therefore also be relatively close to each other in the neural structure.

How can this theory be applied to the perceptual illusions observed in images such as the Kanizsa triangle or the gray ring? According to Gestalt isomorphism, the relationship between the perceptual experience of the image and its representation in the neural substrate must be structurally equivalent: the image is represented as color-by-bit-map. A pixel in the bit-map that is not represented in the neural substrate cannot be perceptually experienced. Reversing the argument: if a pixel is perceived, it must be encoded in the neural substrate. Therefore, if a bright triangle is perceived, the bright pixels must be represented in the neural matrix representing the image, by for instance firing frequencies. This is what is meant with ‘filling in’: the brain actively fills in (through neural activation) the bright triangle that is not present in the stimulus. According to the isomorphists this simply is the only means by which the brain can influence the perception.

A quick summary. According to Gestalt isomorphism, a perceptual experience (which is phenomenal) must be encoded in the neural substrate (which is physical) of equivalent

³The word pixel is used for convenience, but *receptor cell in the retina* would be more accurate

structure and dimension. This is similar to the color-by-bit-map representation discussed in the previous section. This is always true, independent of whether the information in the percept was actually registered by the retina or not.

2.5 ... versus finding out

Dennett gives a different explanation for perceptual completion [17]. He tries to find analogies between all the different ways in which an image can be stored in the computer and in the brain, evaluating each on its plausibility.

The color-by-color (image A in diagram 2.4) is the most contrived, as it assumes that the brain can somehow use real color to represent the image. Is there some location in the brain that produces red, green, blue, or any other type of pigment? If so, where would this pigment be used? Would it be sent to the retina to fill in certain parts of the retina? Dennett does not think this is likely, and I think most of us will agree with him. A slightly less graphic version could also be proposed. Maybe the brain doesn't make true colored pigment, but a not yet discovered substance (some chemical) that can cause color sensation in the brain. Dennett brilliantly calls this *figment*. Figment doesn't necessarily have to be blue, although it does cause the experience of blue. Bright figment could be used to fill in the brighter triangle in the Kanizsa triangle causing the illusion. Dennett also discards this not very plausible figment as a figment of his imagination: "Down with figment!"

Color-by-bit-map seems to be a better contender to be the brain representation of a color image. Like color-by-color, it still requires that the image is represented completely, but does not specify how. Remember that this is exactly how the filling-in proponents see it. The problem with this, Dennett says, is that all these encodings have to be 'decoded' somewhere, converting the representation back to 'color'. And once the bits are decoded, who would *look* at the actual color image? For whom are all the firing frequencies (to name a possible encoding mechanism) decoded into actual color perception?

Over the last decade Dennett has become very good at answering these questions [16]. His response would certainly be: "*There is nobody!*" The data does not have to be neurally represented in a continuous form, structurally equivalent to the image itself. Why would the data have to be represented as a whole? There is no audience in the head that requires a filled in and well edited movie. He calls this metaphor the *Cartesian theatre*, referring to Descartes [19], who thought consciousness (the audience) and the physical brain (the theatre) to be two different entities, living in different worlds altogether. The problem of Descartes' dualism is that, to stick with the metaphor, it is impossible for an audience to watch a movie when it is playing in another world.

Therefore Dennett thinks that color-by-number is the most likely method by which the brain represents images. He gives Marilyn-Monroe-covered wallpaper as an example. Once you've seen one Marilyn, you simply know the other Marilyn-like blobs that aren't as clear must be Marylins as well. You do not have represent them all in focus to perceive them all. Dennett explains perceptual completion with this theory as follows: There is no need to actively 'fill in' the missing data, since it is already labeled (blue, brighter, feathery, Marilyn Monroe). The missing data simply acquires the same label as the surroundings. Color-by-label might be a more accurate description. Once it has received this label there is no need to make it explicit for the audience in the Cartesian theatre.

Dennett's main argument against filling in is that it suggests that the brain is providing something to the Self, whereas Dennett poses that the brain is not providing, but ignoring the missing data; simply labeling it as its surroundings. In his own words: "Filling in theory mistakes *the omission of a representation of absence* for *the representation of presence*."

2.6 The debate

The two theories have been presented. It is obvious that they are quite different. One assumes the brain represents retinal percepts as color-by-bit-map, the other as color-by-label. Completion either takes place through actively changing bits in the bit-map, or ignoring missing data and simply labeling it. Of course proponents of both theories have read their opponents' work and commented on it. In this section I will discuss some of the arguments for and against both theories, including my own opinions throughout the debate.

Verification and falsification

In Pessoa and Neumanns' article, *Why does the brain fill-in?* [38], analysis of three cases of filling-in are presented (brightness-, texture- and signal-filling-in). They discuss three clinical investigations into the matter, all of which conclude that filling in takes place at an early stage in vision. This of course is bad news for Dennett, as he supports a theory which places perceptual completion on a higher level of vision (symbolic labeling). In a response to this article three of the researchers whose research Pessoa and Neuman quote (Maddess, Srinivasan and Davey) put some serious question marks behind this conclusion [46]. They mention other more 'Dennettian' theories which could also explain the phenomena. More importantly though, they think that Pessoa and Neuman try to explain all the three different completion phenomena with the same filling-in theory: "We feel that more experimentation is necessary before one can be sure that the same process underpins all of them."

The previous paragraph is not really intended to prove who is wrong and who is right. However, it does show that the filling-in theory is a theory which can be tested. Actual neural research can be done to verify or falsify the theory explaining the phenomena. Maybe the intricacies of the different types of completion are not yet known, but at least they are falsifiable within the filling in theory paradigm. Dennett seems to have posed a theory that is hard to overthrow, but this is mainly because it does not make any positive or falsifiable statements. Most of the conclusions Dennett draws are negative: There is NO outside viewer, Marilyn Monroe does NOT have to be made explicit, the missing data is NOT filled in. Even if this is true, it is hard to prove. Dennett does not provide us with a theory that is firmly based on neural or cognitive research, as is the case with Pessoa and Neuman.

Single cell recordings

As stated, the filling in theory is a theory that can be tested; and tested it is! A variety of researchers is doing a variety of researches that focus on trying to find neural proof for the filling in theory. There is research on "*Illusory contours and cortical cell responses*", on "*The blind spot and receptive field dynamics*", on "*Scotomata and receptive field dynamics*", on "*Texture filling-in and cortical cell responses*", and many more. Summaries of and references to them all can be found in [32].

The main methodology of the research discussed above is based on single cell recordings. Brains of subjects (usually macaque monkeys) are probed such that the activity of one or several neurons can be displayed on an oscilloscope. Researchers can then compare the activation of these neurons under different circumstances. The interesting test for perceptual completion is to compare differences and similarities between neural activity in certain areas of the visual cortex when the subjects are presented with either real edges, or illusory edges. If the neural substrate is the same for both, this would support the filling-in theory. The researchers do these tests and draw conclusions such as: *“The cells respond as if the illusory contours were formed by real edges or lines, and they respond to variations in the figure in a way that resembles human psychophysical responses to the same variations.”* This conclusion is drawn by Pessoa et al. in [32], basing it on research done on single cell recordings in the V2 area of monkeys. The V2 area is located in the extrastriate visual cortex and it processes all submodalities of vision (motion, orientation, color and depth). I think this conclusion is permissible, as it is an objective account of the observations. They continue with: *“Although making a link between single cell activities and perceptual phenomena is problematic, the evidence here seems to suggest that the perceptual completion of boundaries involves the neural completion of a presence, rather than ignoring an absence.”* In my opinion this link is too problematic. Single cell recordings can surely be used to explain low-level phenomena such as retinal activity. Deriving a complex mechanism such as perceptual completion from single cell recordings however is not valid, especially since the experience of perceptual completion by macaque monkeys can not easily be determined objectively.

Even if it were possible to measure a perfect correlation between neural activity and perceptual completion, it can still not be verified that the neural activity is the cause of the completion. A similar case occurs in color perception. Although certain areas in V1 (another area in the visual cortex) respond to different wavelengths of light, it is often assumed that the actual perception of color happens in V4. To quote Ben Best [4]: *“If V1 neurons respond to wavelength and V4 neurons respond to color, V4 neurons would seem more likely to be ‘experiential’, since we experience color rather than wavelength.”* Suppose that measuring the edge detections of real and illusory edges in V2 was like measuring the wavelength of light in V1, in which a response, but not a percept is being measured. The measurement in V2 can certainly be interpreted as edge detection, but it is by no means certain if the edges registered here are the actual perceptual content. The real perception of these edges, like the perception of actual color, might very well be in V4 (although V3 is more likely in this case). Without further investigation, it cannot be known whether the perceptual completion in V4 is based on filling in or finding out. Therefore I think the correlation between neural activity in V1 and the experience of perceptual completion (which, as mentioned, cannot be determined that well) does not necessarily imply that the activity in V1 *is* the perceptual completion.

The researchers themselves appear to acknowledge these problems, because, as Dennett had noticed, scare-brackets are often used, and phrases as ‘seems to suggest’, ‘suggestive evidence’, and ‘seems to depend’ are in abundance. Again, filling in theory can be considered a good theory, because it has been frequently tested, yet not been falsified. On the other hand, the researchers claim verification of the theory beyond the actual ‘verifiable potential’ of the research done.

Incorrect unification of modal and amodal completion

There is also one aspect of this research that I feel is overlooked by both parties: amodal completion is very different from modal completion. This has also been noticed by Pessoa et al. [32]. Whereas modal completion seems to really fool the perceiver (“I know the paper isn’t brighter. But it sure does look like it. How strange!”), amodal completion seems to have a more symbolic, representational character. Even though the gray ring and the legs of the person are encountered, it still requires active mental processes to imagine the entire ring or the trousers of the person. This brings us back to active representation, not perceptual completion. The bright triangle does not require active mental processes. Even if we set our mind to *not* seeing the triangle, we just can’t ignore it. This difference is so distinct that I do not think it likely that the same type of filling-in or finding-out theory will explain both in the same way. This is a shame, as all research seems to be aimed at modal completion; amodal completion is simply neglected, maybe because it does not lend itself well to single cell research.

Collaboration

I would like to end this section with a quote with another conclusion drawn by Pessoa et al. [32], in which they justly state that collaboration between the different disciplines of research is necessary to solve the problem.

“Without such collaboration, visual scientists run the risk of producing isolated facts that do not contribute to an integrated understanding of perceptual completion, while philosophers run the risk of ignoring important experimental and theoretical studies that bear on the fundamental conceptual issues.”

2.7 Conclusion

The relevance of the cognitive research on completion, and more specifically amodal completion is clear. Let’s compare Kanizsa’s description of amodal completion with our research goal, both repeated below.

Research goal (computer vision) versus Amodal completion (cognitive science)

The main goal of this dissertation will be:

Amodal completion refers

to hypothesize data

to the completion of an object

that is not visible due to occlusion;

that is not entirely visible because it is covered or occluded by something else.

the hypothesis being based on the surroundings of the occluded data.

based on filling in? finding out? other?

In this conclusion it will be argued whether the cognitive research can be useful in implementing a completion algorithm.

It is apparent that the cognitive scientists and philosophers have not yet reached an agreement as to how perceptual completion should be explained, although in my opinion

the fillers-in are on the winning hand. Unfortunately there is much left unexplained even within their paradigm. Many fillers-in disagree on how the filling in process should be explained. One theory for all phenomena? A different theory for different phenomena? Active debate is interesting, but poses a problem for our research: it is hard to model a system if it is not fully understood how the system works itself. Claiming that such a model uses the underlying principles of the system would be futile. How could one prove that the model is correct?

For the sake of argument let us suppose that Dennett is right. This would require symbolic modelling of the images. Our program would have to be able to recognize walls, chairs, houses etc. Even more knowledge about the world would be needed to give the program a sense of continuity of objects: it should know that walls do usually not contain chair-shaped holes. This level of symbolic reasoning in computer vision is still an idyll, making the implementation of this symbolic reconstruction infeasible.

The main reason however, for not being to able to implement Dennett's theory is due to the requirements of the program. We are not looking for a brilliant computer that uses symbolic reasoning to come to the conclusion that: "Yes, occlusion is taking place! There's probably a wall behind that chair." We need a program that can explicitly visualize this wall. Even if 'finding out' is the cognitive solution to occlusion, we will have to make the missing data explicit in the image; we will have to fill in.

This brings us to the isomorphisists. They claim that encountered objects are actually structurally encoded in the neural substrate. This sounds more promising, as it must be theoretically possible to write a neural network of some sort that models our vision, including the structural encoding of encountered objects. Then all we have to do is take the neural encoding, decode it, and make it explicit in the image. I think *theoretically* is the key-word here. Although the cognitive and neuro-scientists have some interesting theories, not many explain the phenomenon on a neural level. Those that do are too basic to be applicable to the reconstruction of an indoor scene in which occlusion is taking place. If it were known the question still remains as to how we would implement this artificial neural network, let alone decode its neurons into pixels in the image.

We can conclude that neither cognitive science, nor computer vision is ready for the intricate task of application based modelling of perceptual completion. Some low-level modeling might be possible, but certainly not at a higher level that makes it usable for a functional program that fits within the goals of the CAMERA project. Not all hope is lost though; we will only need to shift our priorities to the results of the program, not the underlying cognitive theory. This means that the program neither tries to model the cognitive processes causing perceptual amodal completion, nor does it try to explain it. To acquire a program that can hypothesize missing data I will therefore use more traditional methods in computer vision. However, I will allow myself to use the cognitive aspects of the problem as a source of inspiration. As we shall see, one of the main ideas in our research is formulated very well by a philosopher (Lehar), not a computer scientist [34]: "This spatial completion mechanism can be formulated on the assumption that the visible portion is taken as a representative sample of the object as a whole. Non-visible portions (occluded areas) are encountered by the principle of good continuation."

Chapter 3

Image Acquisition

3.1 Introduction

As explained in section 2.7, the program will not be based on a cognitive model. In the next three chapters the methods that *have* been used for completing occluded surfaces will be presented. Before this can be done, a little more needs to be known about the types of data that are being used, as well as the ways in which they are acquired.

First the general format of a digital image will be explained. Then the two types of digital images used in this research will be discussed: range images and intensity images. One of the benefits of working in an international research network is that different partners can specialise in different areas. One of the partners, UK Robotics in Manchester, has some high-tech sensors for measuring range and intensity data. All the images of the Bornholm church have been provided this company. They use a K2T laser range scanner. Other images have been acquired using a REVERSA laser range scanner at the Vision Lab at the University of Edinburgh. Both scanners will be discussed in greater detail in section 3.4.

3.2 Digital images

Computer vision methods use two- or three-dimensional signals to interpret the spatial configuration of the environment in which the signals were acquired. This multidimensionality needs a multidimensional way of representing the data. 2D visual information is usually encoded in digital images. Digital images are 2D matrices in which every matrix-element (also called a *cell* or a *pixel*, which is short for *picture element*), has a row and column orientation. The location of a pixel can be specified by two integer coordinates i and j indicating row and column. This is called the ij -domain. Every pixel has a certain value which represents a sensed physical quantity. A large matrix of this type is called a *digital surface*, because it is the quantised version (in x,y and z direction) of an analogue surface $z = f(x, y)$. In machine vision digital surfaces are more commonly known as digital images. In the following section two types of digital images will be described: intensity images and range images.

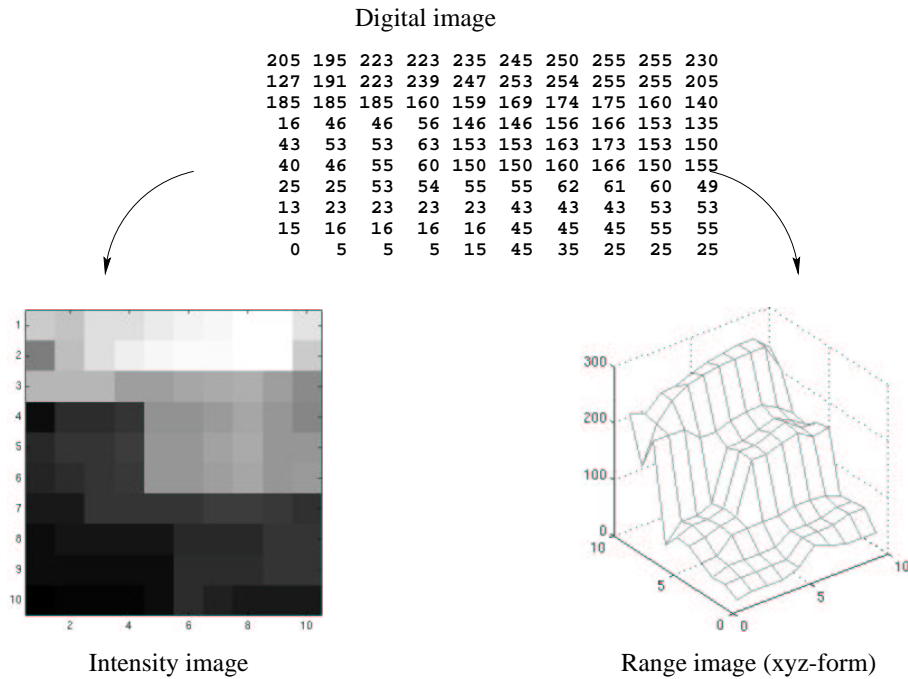


Figure 3.1: A very small digital image, interpreted as an intensity image or as a range image.

3.3 Intensity images

In intensity images, the z -value of the digital image is interpreted as light intensity. Intensity images are typically acquired by cameras, although other photosensitive devices can be used. The incoming light, after being directed through an arrangement of lenses, hits the image plane, which is a rectangular grid of photo-sensors, each measuring the light intensity. All these intensities are stored, resulting in the intensity image.

Intensity images look like, and basically are, black-and-white photographs, the big difference being the digital format. Indeed, modern cameras are often digital, and the process explained above applies to them, with the additional feature that they record the intensities of the light at different wavelengths and so acquire colour images.

The great benefit of intensity images is the low price at which they can be made. Digital cameras can be bought in most hardware stores at affordable and still rapidly decreasing prices. Although extracting 3D information from intensity images is not an easy task, their cheap price certainly makes the research worth while.

3.3.1 3D information in intensity images

In computer vision, knowledge about the three-dimensional arrangements of objects in the scene is often vital. Intensity images are not very good at encoding 3D information. Each pixel represents a light intensity. Although the direction in which this intensity was measured is known, the distance at which it was measured isn't. For a single pixel it is therefore impossible to determine *where* in 3D space a certain intensity was measured.

Although 3D information is not encoded directly in the intensity image, it is possible to derive it using some computer vision methods. One method to infer 3D configuration from

intensity images is to analyse texture on surfaces, as done in [13]. Texture is distorted in 2D representations of 3D scenes. It compresses towards the vanishing point. For instance, if a chess-board is tilted, the squares at the front appear larger than those at the back. This can be used to determine the original 3D configuration. The benefit of this method is that it requires only one intensity image. Unfortunately, the accuracy is quite low. This is because the original pattern is often unknown, and assumed to be repetitive. These assumptions might not always be justified.

More often, multiple images are used. Stereo vision takes two or more images of the same object or scene, but taken from a different angle. After spatial correspondence between pixels is established, differences between their location in the images provide information about their location in 3D space [47]. Another method is based on depth from motion, using an image sequence. This technique uses the principle that closer objects appear to move faster than objects that are further away. This technique has been used in for instance [10].

Although the hardware of these methods is easily configured (1 or 2 cameras is basically enough), they invariably entail solving the complex problem of establishing correspondence of pixels or features between images. This makes applied visual processing more difficult and also less accurate, because pixel and feature correspondence is sometimes hard to determine. Inaccuracies in determining this correspondence also leads to inaccuracies in determining the 3D configuration.

3.4 Range images

Range images are much better at encoding three-dimensional properties than intensity images. In range images, each pixel encodes a measurement of the distance from the sensor to the object measured at that pixel. For this reason they are also known as depth images or surface profiles. Because the exact direction in which the measurement was made is known, as well as the distance to the object in this direction, the location of the object (at that pixel) in 3D space can easily be computed. Because range images are 2D digital images containing information about positions of points in 3D space, they are also referred to as *2.5-dimensional* images.

Range images are extensively used by the CAMERA group (and computer vision in general) because of their efficiency in encoding 3D properties. In this research the accurate range images will be used to analyse 3D configurations of shapes in the image, whereas the intensity images will be used to analyse the texture on these surfaces.

In the next three paragraphs some issues concerning range images are discussed. In section 3.4.1, two different classes of sensor are discussed. Section 3.4.2 discusses two different methods of scanning a scene. The last issue, the way in which the range data is represented, is discussed in section 3.4.3. The goal of these sections is to provide the reader with a better understanding of our data, as well as motivating the choice for writing a new segmentation algorithm.

3.4.1 Methods of acquisition: triangulation or time-of-flight

Acquiring range images is possible using a whole array of different sensors, for instance ultrasound (e.g. fetal images) or laser projection. The most common and reliable methods

of acquiring depth images with laser projection are using time-of-flight or triangulation. The images in this research have been acquired by these methods, so both will be presented.

The laser-scanner in the Vision Lab in Edinburgh uses structured light and triangulation to calculate the distance measures. In the structured light method patterns (grids, stripes, elliptical patterns etc.) are projected onto an object. The image of the pattern on the surface is recorded at a different angle by a camera. Surface shapes can then be deduced from the distortions of the patterns that are produced on an object's surface. Knowing relevant camera and projector geometry, depth can be inferred by triangulation. Triangulation uses the fact that the displacement of the projected line from the mid position is linearly proportional to the object depth [49].

The images provided by UK Robotics were acquired with a laser scanner using the time-of-flight method. The laser scanner consists of three functional components: a transmitter, a scanning mirror and a receiver, shown in figure 3.2. The basic idea is to transmit a laser, aim it at an object with the mirror, and measure how long it takes for the laser-beam reflect back to a receiver. The time this takes is converted to a measurement for the distance from the object to the sensor.

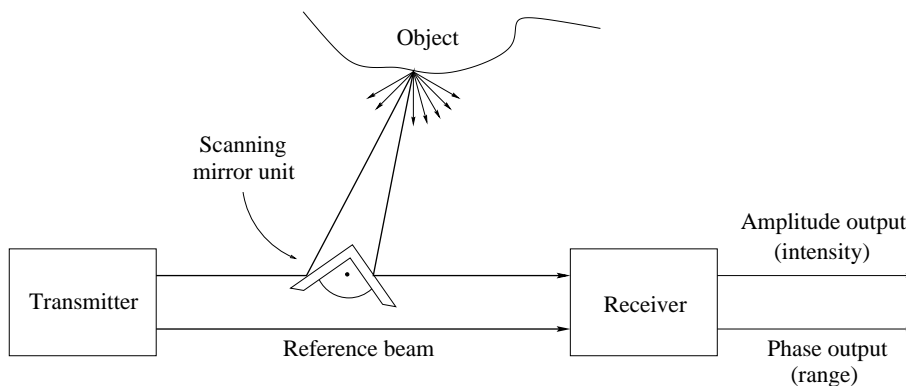


Figure 3.2: The main components of a time-of-flight laser scanner.

The transmitter emits two laser beams to the receiver. One directly as a reference, and another towards the scanning mirror. To aim the beam at different locations in the scene, the sensor is either mobile, or uses mirrors that can rotate, reflecting the beam in different directions. In the latter case there are often two mirrors, one for horizontal and one for the vertical orientation. The UK Robotics scanner uses the latter technique.

The laser-beam emitted by the sensor will reflect off some object in the scene. Through diffraction the laser beam reflects in many directions, one of them being back towards the mirror. This beam is projected towards the receiver. There are two ways of measuring the difference in time-of-flight between the reference and the reflected laser beam. One is to compare two very short pulses, the other is to use a continuous laser beam whose amplitude or frequency is modulated and to measure the phase difference or beat frequency. The phase difference between the received beam and the reference beam gives a direct measurement of the time-of-flight. Because the speed of light ($3 * 10^8 m/s$) is known, the distance to the object can be computed once the time-of-flight is known. The use of a phase-sensitive detector can give a more accurate indication of time-of-flight, and hence distance, than a

pulsed system, in which a direct time measurement is made on an intermittent burst of laser energy. The accuracy of these systems is of the order of 1mm. An extra benefit is that the amplitude of the received waveform is proportional to the intensity of the received light. This means that this laser scanner can make range and intensity images at the same time, with perfect spatial correspondence between range and intensity at each pixel!

Scanner specifications

Scanner	sensor method	scanning method	accuracy
K2T	time-of-flight	spherical	1mm
REVERSA	structured light and triangulation	orthogonal	0.01mm

3.4.2 Methods of scanning: orthogonal or spherical

A feature of laser range sensors is that they are usually fixed structures. They can only measure the distance in one direction. This is because the actual transmitter can emit the light (the laser) only in one direction, giving us one measurement in 0D space. In order to acquire the desired 2D digital image the sensor will have to move around in the x and y directions in order to be able to measure multiple distances in a 2D image plane. There are two ways of doing this.

The first is by using a flat-bed scanner. This type of scanner is fixed on a structure that can move the sensor to any point in a given xy -plane. The direction in which the distance (z) is measured is orthogonal to this plane. This ensures that all distances are measured along parallel beams. If these beams are placed at consistent intervals in the x and y direction (sampling at positions (i, j)), a digital range image is acquired. This is shown on the left-hand-side of figure 3.3. Note that the sensor movement is only shown in the x -direction for clarity.

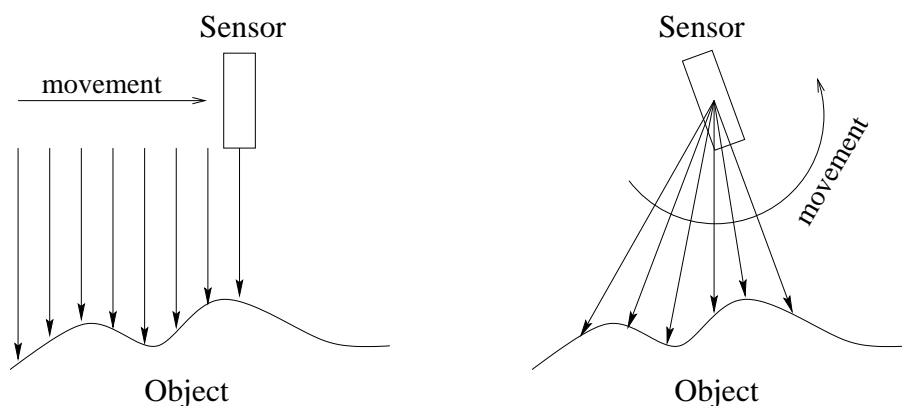


Figure 3.3: Orthogonal (left) and spherical (right) scanning. Only view from top is shown for clarity.

As mentioned earlier, the hardware involved in acquiring intensity images can be very simple. The flat-bed scanner on the other hand is a large machine that needs meticulous calibration and fine-tuning before the increments in x and y direction can be assumed to

be constant. Furthermore, its cost is over 100 times that of normal cameras. Although it gives range images of very high quality, the laborious nature and cost of the technique is a draw-back.

The other scanning method is spherical scanning. The i and j positions of the range image do not correspond to increments in the x and y plane, but to angular increments in the vertical and horizontal directions. This means that the sensor stands at a fixed point in the scene. It now changes its pan and tilt to scan the environment (see figure 3.3). The UK Robotics scanner scans a full 360° in horizontal direction (pan), and 63° in vertical direction (tilt), with increments of 0.045 degrees in both directions. This results digital range images of 1400 rows \times 8000 columns.

Because orthogonal scanning can only scan in a predetermined plane they are not well equipped to scan rooms in buildings. Spherical scanners allow a full turn of the camera, giving the entire 360° view. For this reason UK Robotics has used spherical scanning to scan the inside of buildings.

3.4.3 Methods of representation: 2.5D or 3D images

Range images can be represented in two formats, the r_{ij} -form which is 2.5D, or the xyz -form which is in 3D. The r_{ij} -form is the digital image version of the distance measures. It is the 2D matrix which has been discussed in the previous section.

Because the sensor location is known (or a relative position in space is specified), the range information can easily be warped into 3D space (see the next section for more information). The x and y coordinates have a relationship with the i and j position in the matrix, while the z coordinate is related to the value of the pixel at position (i, j) in the range image. The 3D-form (or xyz -form) of the data is a list of 3D points. Every point has a x, y and z coordinate indicating its position in a 3D coordinate space. This form is more general, since the list does not have to be ordered, as is the case with the r_{ij} form. For this reason, any range image in r_{ij} -form can be converted directly into the xyz -form (given the sensors' location and parameters), while the opposite is not necessarily true [2]. Given this description and the definitions below, the difference between the left and right image in figure 1.2, repeated here in figure 3.4, can be fully understood.

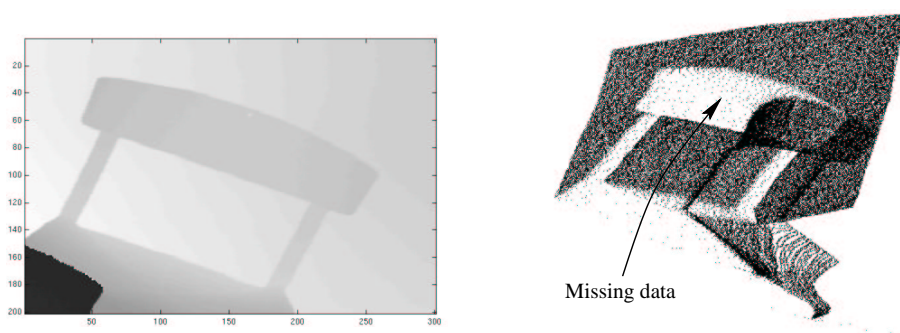


Figure 3.4: A range image in 2.5D (left) and the same image in 3D (right).

Terminology

2.5D range image: A range image represented by an ordered 2D matrix containing unscaled range values. Also called r_{ij} range image, or range image in r_{ij} -form

3D range image: A transformed 2.5D range image represented by a cloud of unordered points in real 3D space. Also called xyz range image, or range image in xyz -form

The way in which the r_{ij} image is converted into a cloud of points in 3D space depends on whether the scene has been scanned orthogonally or spherically. The formulae for both cases are given.

Because the sensor of an orthogonal scanner moves in the xy -plane (also see figure 3.5), and measures the distances with orthogonal beams, the position of the sensor in the plane is an exact measure for the x and y coordinates of every point in the 3D. Because x and y are sampled at consistent intervals (at positions (i, j)), they are linearly dependent on i and j . The z coordinate will only depend on the distance measure. This allows us to use very simple linear formulae for converting the r_{ij} -form into the xyz -form [2]:

$$\begin{aligned} x &= a_x + s_x j \\ y &= a_y + s_y i \\ z &= a_z + s_z r_{ij} \end{aligned} \quad (3.1)$$

Here, the a -values are offsets, and the s -values distance increments (scale factors); both are scanner parameters. Without these parameters a correct conversion from 2.5D to 3D cannot take place.

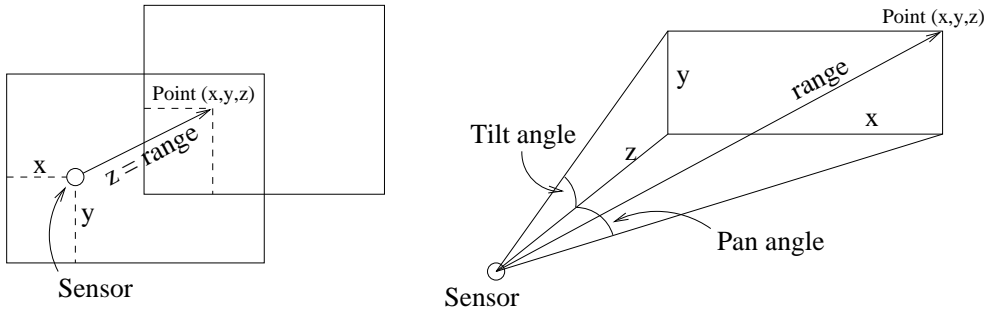


Figure 3.5: Orthogonal (left) and spherical (right) scanning geometry

In the case of a spherical scanner, the relationship between (i, j) and (x, y) is not linear. For this reason, the spherical sampling method is known as 'non-uniform' [7]. Using figure 3.5 as an illustration, formulae 3.2 can be derived, referring to [2] for a complete explanation of this derivation. For the conversion of a spherically scanned range image from r_{ij} - to xyz -form the formulae in equation 3.2 are used. Again the a -values are distance offsets. s_r is a scale factor, and s_θ , and s_ϕ are respectively elevation (tilt) and azimuth (pan) increments.

$$\begin{aligned}
x &= a_x + s_r r_{ij} \tan(js_\theta) / \sqrt{1 + \tan^2(is_\theta) + \tan^2(js_\phi)} \\
y &= a_y + s_r r_{ij} \tan(is_\phi) / \sqrt{1 + \tan^2(is_\theta) + \tan^2(js_\phi)} \\
z &= a_z + s_r r_{ij} / \sqrt{\tan^2(is_\theta) + \tan^2(js_\phi)}
\end{aligned} \tag{3.2}$$

3.5 Summary

In this chapter, some basic image formats have been presented. A digital image is a 2D matrix in which all the cells (also called pixels) have a certain value. Intensity images are digital images in which this value represents a certain light intensity. They are basically black-and-white digital photographs, which can be acquired using a standard digital camera. In range images, the values in the digital image represent a certain depth measurement. Range images are therefore also known as depth images. They can be acquired by triangulation or time-of-flight sensors. The method of scanning, which in this research has been either orthogonal or spherical, influences the geometrical properties of the range image, especially on the visualisation of a range image in 3D. Because the direction in which a measurement was made is known, as well as the distance from the sensor to the object measured along this direction, every pixel in a 2D range image can be represented as a point in 3D space. In general, this method of representation is preferred, as it eliminates the geometrical distortions that are found in its 2D counterpart. The 2D range image is often referred to as a 2.5D range image, as it encodes 3D information in a 2D format.

Chapter 4

Image Segmentation

4.1 Introduction

An overview of the completion algorithm

The (incomplete) data has been acquired. It is time for processing! This section will give a brief overview of the processing chain which will be discussed in more detail next three chapters.

Diagram 4.1 shows the three steps that have been implemented to complete missing data in images in which occlusion is taking place. The up most image in the diagram shows a cloud of 3D points, seen from above. The sensor measured the data from the top of the image. Although this data has (obviously) been made up, it will help to think of it again as a chair standing in front of a wall, seen from above. The part of the wall that is missing is labeled ‘occlusion’.

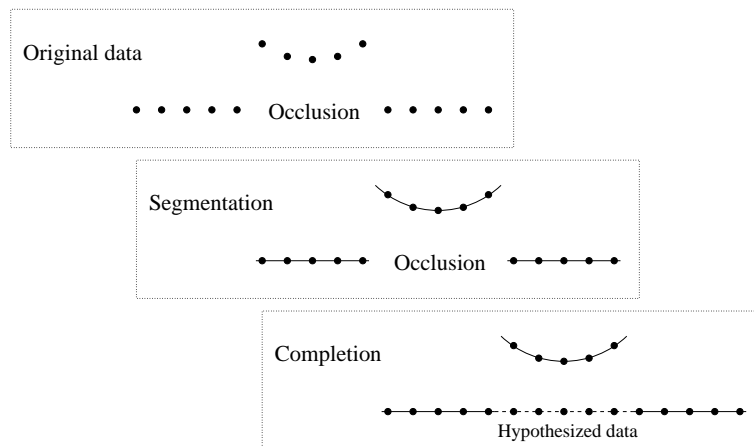


Figure 4.1: The processing chain

The first step towards hypothesizing the entire wall is to segment the image: find simple geometric surfaces that describe the data well. The reason this is done this is that occlusion is an unknown part surrounded by known parts. The goal is to extrapolate the known parts into the unknown parts. Segmentation will provide the completion algorithm with the

generalizations of the data points, needed to extrapolate into the unknown occluded area. For the data in the diagram this leads to a segmentation of the data into two planes and a cylinder (all seen from above).

The next step is to complete the 3D configuration of the occluded surfaces. This is discussed in chapter 5. The algorithm locates coinciding surfaces (e.g. coplanar planes, as the two parts of the wall are). The main idea is that these coinciding surfaces might be instances of the same surface: the two visible parts of the wall actually belong to the same wall! Using the known parts of the wall, the occluded part of the wall is then completed by extrapolating the plane into the occlusion, and placing data points on this plane, as can be seen in the bottom image in diagram 4.1.

The last step is to complete the intensity texture on the texture-less completed surface, which is discussed in chapter 6, but has not been depicted in diagram 4.1 for reasons of clarity.

4.2 Noise reduction

The first step in image segmentation is to reduce the level of noise in the image. In digital images many types of noise can arise, and the best type of noise removal depends on the type of noise. Therefore we will first discuss the characteristics of the noise in the images used. The appropriate method for removing the noise will be presented afterwards.

4.2.1 Noise characteristics

First of all our images suffer from noise that is inherent to digital sampling of an analogue signal: quantisation noise. Because the continuous signal is stored as one of a finite number of digital values, information is lost. The 16 bit/pixel format in which the image is stored is more than sufficient to store the ranges and accuracy of our data, so the quantisation noise is negligible.

Of much greater influence is noise that arises through imperfections in the sensor's calibration and measurements. The measured value $\hat{I}(i, j)$ at pixel (i, j) is assumed to be a combination of the true pixel value $I(i, j)$ and a certain noise signal $n(i, j)$.

$$\hat{I}(i, j) = I(i, j) + n(i, j) \quad (4.1)$$

One major cause of noise in our range images is that the laser beam is not a perfect line, but a cylinder with a small, but noticeable diameter. At depth discontinuities, part of the laser beam might hit the further object, while the other part hits a closer object. This side-effect is shown in figure 4.2. It is called the "mixed point problem" [50].

Experience has shown that the measurement will usually be somewhere between the two distances, but sometimes the values are completely unpredictable. This causes impulsive noise, but only around depth discontinuities. The impulsive noise can be modelled by a salt-and-pepper model, shown in equation 4.2, in which l is a parameter controlling how much of the image is corrupted, and n_{min} and n_{max} determine how severe the noise is. x and y are random variables, ranging from 0 to 1. This model predicts $l\%$ of the measured data is equal to the actual value, whereas $1 - l\%$ is distorted by a noise signal. The strength of this noise signal ranges between n_{min} and n_{max} . The actual value of the noise is determined by

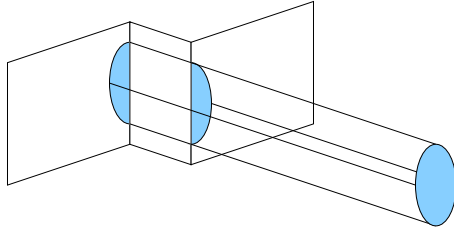


Figure 4.2: Noise can be caused by the fact that the laser is not a perfect line, but a cylinder.

the random variable x . The fact that the specific location of this impulsive noise is known (around depth discontinuities) will be very useful in its removal.

$$\hat{I}_{sp}(i, j) \begin{cases} I(i, j) & x < l \\ n_{min} + y(n_{max} - n_{min}) & x \geq l \end{cases} \quad x, y \in [0, 1] \quad (4.2)$$

In areas where there are no depth discontinuities the noise is assumed to be Gaussian [49]. This means that $n(i, j)$ is modelled by a white Gaussian zero-mean stochastic process. This model is shown in equation 4.3, in which $N(0, \sigma)$ is a normal Gaussian distribution with mean 0 and standard deviation σ . This time, every pixel is distorted with a noise signal. The values of the noise signal are distributed symmetrically around zero with standard deviation σ . Closer inspection of equation 4.3 shows that the actual pixel values will then be distributed symmetrically around their real values. This is what can be expected from high-quality sensors.

$$\hat{I}_{gs}(i, j) = I(i, j) + N(0, \sigma) \quad (4.3)$$

4.2.2 Choice of filter

After analysing the features for the noise, the right methods have to be found for removing it. There are a multitude of ways for removing noise. The trick is to find the one that maximises noise reduction, while still retaining the useful content of the original image. This is hard, because noise-removal often leads to image blurring, and thus information loss. The more you smooth, the more information gets lost.

Because the method for removing the impulsive noise depends on techniques that have not been discussed yet, it is postponed until section 4.3.1. It must however be mentioned that this noise removal only takes place around depth discontinuities. This is perfect, as this is the only place where impulsive noise is to be expected!

As to the Gaussian noise: In [27], Hurt and Rosenfeld compare five methods for removing Gaussian noise in three-dimensional images. They find that, although computationally attractive, median and mean filtering do not retain the image content very well due to image blurring. Selective averaging performs better, but nearest neighbour smoothing and maximum likelihood smoothing provide the best noise removal/information retention trade-off. Because the images can be quite large the most efficient one is chosen: nearest neighbour smoothing.

Nearest neighbour smoothing is a combination of median and mean filtering. In both cases a window is used. A window is an area of a certain size (often square, e.g. 3x3 pixels) in

the image. It is centred around the pixel of interest. The intensity or range at all the pixels in the window (the central pixel's neighbours) is measured. In the case of mean filtering the central pixel's intensity or range is set to the mean value of all its neighbours. Median filtering uses the median value instead of the mean. Nearest neighbour smoothing combines these two by taking the k neighbours whose values are closest to that of the central pixel. It then sets the value of the central pixel to the mean of those k pixels.

Of course k should be a fixed number throughout the smoothing procedure. A good choice is to predetermine k as the number of neighbours who would be on the same side of the neighbourhood as the central pixel if a straight edge ran through the neighbourhood [27]. For our window of 3×3 pixels this yields a value of $k=5$.

4.3 Surface characteristics

The final goal of the next few processing steps is to obtain surface patches: clusters of points (pixels) that can all be described by the same concise function (see the terminology box). Fitting surfaces to arbitrary patches of points would not be efficient or accurate, so we will have to find ways to describe each individual point, and cluster points that have approximately the same characteristics. These are called surface characteristics, because they give us an idea to what kind of surface the point might belong. Sections 4.3 and 4.4 are dedicated to clustering pixels with similar shape characteristics. All the information gained in this process will be compressed into a simple surface description in section 4.5.

I think Besl summarises it very well in Chapter 3 of [3]: *Computing surface characteristics for pixel grouping purposes is the first step toward the final goals of object recognition and image understanding.*

Terminology

(Geometrical) Surface: A parametric description of a continuous function in 3D space. This research is limited to simple special cases of implicit second order polynomials (see section 4.5) such as planes, cylinders and spheres.

Region (or patch): A group of pixels in the r_{ij} image. In our algorithm, these groups are acquired through region growing (section 4.4) and surface growing (section 4.5.3).

Surface patch: A region in the r_{ij} image of which all the pixels are represented by the same surface.

4.3.1 Neighbours

A very useful concept in pixel grouping is *being neighbours*. Usually, two pixels are considered to be neighbours if they are close to each other in the ij -domain. Often a window of certain size is specified (3×3 , 5×5 , 7×7 , etc.); all the pixels within the window are defined to be neighbours of the pixel in the centre of this window. Hence the odd number \times odd number window-size.

When working with orthogonal scanned data, this is a justified assumption since there is a linear relationship between i, j and x, y . This means that if point C is twice as far from point A in the ij -domain as B is, the same relationship will hold in the xy -domain. Neighbours could be defined as being n cells apart from each other in the ij -domain. The same definition could be obtained in the xy -domain by scaling n with the scaling factors mentioned in 3.4.2.

Unfortunately, in spherical scanned data there is no linear relationship between i, j and x, y . Defining neighbourhood relationships in the ij -domain is not justified anymore, because a distance of n cells in the ij -domain represents different distances in the xy -domain, not linearly dependent of n . Also see formulae 3.2, they are clearly non-linear. Not being able to define *being neighbours* in the ij -domain is an inconvenience, but does not introduce any serious theoretical problems.

In this research, the neighbourhood relationship is determined using only the xyz -form of the range data. Doing this alleviates having to deal with any geometric distortions the r_{ij} -image might introduce. Surprisingly, most research seems to ignore this problem completely, probably because working in the xyz -domain is conceptually and computationally harder. One of the motivations for writing this segmentation algorithm was the lack of any available *pure* 3D (instead of 2.5D) segmentation algorithms. Another advantage is that the z component is also included in our distance measure, allowing us to determine the *real* geometric distance between two points, not only in the xy -domain.

The algorithm takes one parameter: the mean number of neighbours per point. 8.0 has proven to be a good value (and also nicely resembles the number of neighbours in a standard 3x3 window). This means that, although the number of neighbours may vary per point, the overall number will average out to 8.0 neighbours per point.

Given this parameter, a distance threshold is computed. Points that are less than this distance threshold apart from each other are classified as neighbours. See figure 4.3 for clarification. This figure shows a 2D case, in which the circle is the distance threshold. This idea can easily be extrapolated into 3D, using a sphere in stead of a circle. The threshold is computed such that the total number of neighbours divided by the total number of points is approximately 8.0. Every point keeps a list of all its neighbours, so they have to be collected only once.

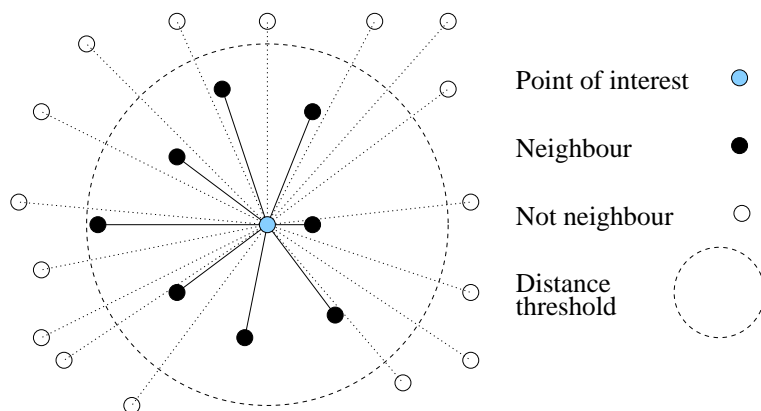


Figure 4.3: Determining neighbours using the distance threshold

Section 4.2 on noise removal is now briefly revisited. Points that have no neighbours are not close to any other point and are obviously floating around somewhere in mid-air. This is typically the kind of point that arises from measuring the range at a depth discontinuity with a cylindrical laser-beam. For this reason these points are discarded as noise. They are not used in any further computation. Image 4.4 shows exactly which points in the image have no neighbour, and are therefore discarded as noise. This takes care of almost all the impulsive noise! Note that this technique can only be applied in the xyz -domain (instead of ij or xy), because the z -component is causing the trouble.

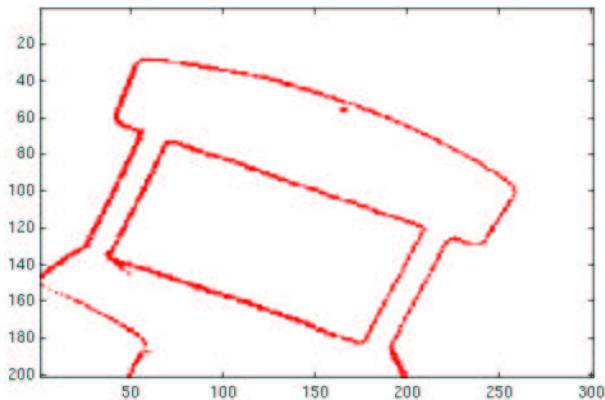


Figure 4.4: Points without neighbours

4.3.2 Surface derivatives

The neighbourhood relationship is useful for defining proximity, but it gives us no indication of the shape the area around a certain point may have. Because surfaces should only be fitted to sets of points of the same shape, shape descriptors such as surface normal and surface curvature will also be used. These will be discussed in sections 4.3.3 and 4.3.4 respectively. Before they can be derived the surface derivatives will need to be computed, a process explained in this section.

To give an explanation of surface derivatives the description of a digital surface is rewritten into the parametric form.

$$S = \left\{ \vec{x}(u, v) \in \mathbb{R}^3 : \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} d(u, v) \\ e(u, v) \\ f(u, v) \end{bmatrix}, (u, v) \in D \subseteq \mathbb{R}^2 \right\} \quad (4.4)$$

This formula is basically a formal rewrite of formulae 3.1 and 3.2. This formula states that the \vec{x} function is a point in continuous 3D space, consisting of components x , y and z (the xyz -domain). These are in turn described by three continuous functions $d(u, v)$, $e(u, v)$ and $f(u, v)$. These functions are exactly those given in formulae 3.1 and 3.2. u and v are elements of a certain continuous domain in 2D space. This has been called the xy -domain up till now.

If S is a smooth surface, and the functions $d(u, v)$, $e(u, v)$ and $f(u, v)$ all have continuous first and second order derivatives the following can be defined:

$$\vec{x}_u(u, v) = \frac{\partial \vec{x}}{\partial u} \quad \vec{x}_v(u, v) = \frac{\partial \vec{x}}{\partial v} \quad (4.5)$$

$$\vec{x}_{uu}(u, v) = \frac{\partial^2 \vec{x}}{\partial u^2} \quad \vec{x}_{uv}(u, v) = \vec{x}_{vu}(u, v) = \frac{\partial^2 \vec{x}}{\partial u \partial v} \quad \vec{x}_{vv}(u, v) = \frac{\partial^2 \vec{x}}{\partial v^2} \quad (4.6)$$

Think of the first order derivatives as representing the rate of change of the surface in either the x and y direction. In figure 4.5 a smooth surface in 3D space is shown. The mapping on a 2D plane (x, y or u, v domain) shows us in which directions the partial derivatives are taken. In figure 4.5, \vec{x}_u is zero, and \vec{x}_v is some negative vector. Along the v -axis, the surface approaches the ground plane. In other words, the function of the surface is decreasing in the direction of v . This is why tangent x_v in the direction of v at point x is negative. Because the function in the direction of u at point x is not changing, derivative x_u is zero. The second order derivatives are harder to visualise, but they are the rate of change of the rate of change of the surface: how fast is the surface changing. In the figure, \vec{x}_{uu} is zero (there is no change in the derivative in the u direction), and \vec{x}_{vv} is some negative value (the derivative is becoming even more negative in the v direction).

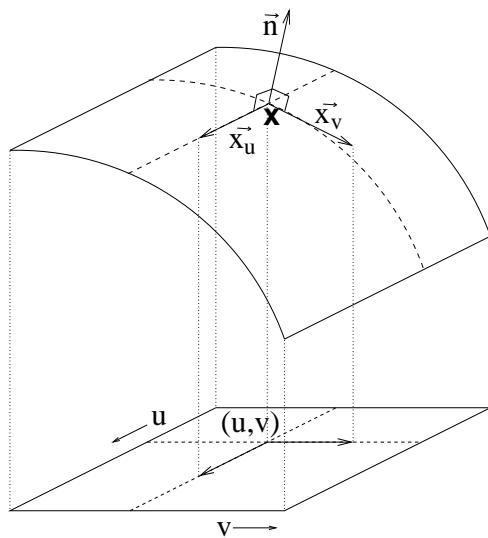


Figure 4.5: Surface derivatives

Computing these derivatives in real digital range images is tricky business, because they are very sensitive to noise. A small deviation in measuring the range can lead to very large deviations in the derivatives. Therefore a technique has been used that smoothes the data to estimate the derivatives at each of the points.

The basic idea is to first compute a continuous differentiable function that represents the given points well. This function is then differentiated and its derivative at the point of interest is determined and considered to be a representative of the actual derivative at that point. The method that has been used for this is based on a local least squares surface model using discrete orthogonal polynomials. It follows exactly the process of fitting (by least-squares) a continuous surface (consisting of discrete orthogonal polynomials) to the points and computing the derivatives of this surface as a representative of the actual derivative. A mathematical summary of this method is given in Appendix A.

Interesting practical features of this method include the possibility of implementing it using convolution. Convolution is very efficient: it has no real impact on the overall computation time of the program. The method also smoothes the data a little to cope with the large influence noise has on the derivatives.

4.3.3 Surface normals

The surface normal (or Unit Normal Vector) is a unit vector that is orthogonal to both first partial derivatives (the tangents to the surface in x and y direction), and can therefore simply be computed by normalising the cross-product between the two derivatives. As depicted in equation 4.7. The surface normal is also depicted in figure 4.5.

$$\vec{n} = \frac{\vec{x}_u \times \vec{x}_v}{|\vec{x}_u \times \vec{x}_v|} \quad (4.7)$$

4.3.4 Gaussian and Mean curvature

To cluster points that are of the same shape, methods will have to be found with which the local shape of the surface at that point can be determined. That is, given a point P on the surface, determine how the surface normal vector \vec{n} changes as on some tangent curve passing through P.

The normal curvature κ_n is the directional derivative of \vec{n} . Think of it as rotating a plane around the surface normal. A continuous surface will intersect the plane at each angle. κ_n can then be computed by taking the derivative of the function in this plane (on the intersection). Note that κ_n can have infinitely many values, depending on the orientation of the plane. κ_1 and κ_2 are the maximum and minimum of all these values. They are called the principal curvatures, and are always perpendicular (except for umbilici when $\kappa_1 = \kappa_2$).

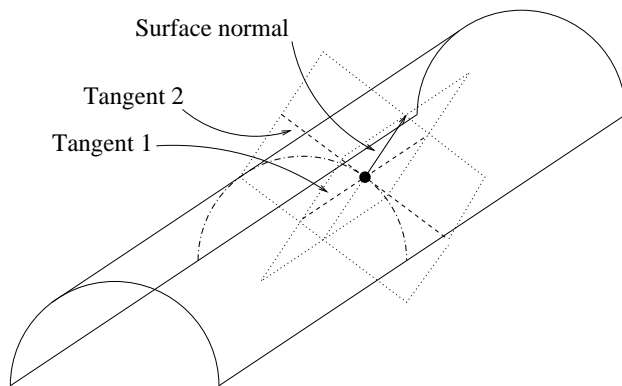


Figure 4.6: The principal curvatures

A small example is given in figure 4.6. Here a cylinder can be seen. At a certain point, the surface normal is drawn, as well as two tangents to the cylinder at that point. Here the tangents projected onto a plane that is rotated around the surface normal can be seen. The normal curvature κ_n can be computed by taking the rate of change of the tangent. The two tangents in the figure have not been chosen randomly: Tangent 1 represents the direction in which the rate of change of the normal has the highest value (zero), whereas Tangent 2

represents the lowest rate of change (some negative value). Their values are κ_1 and κ_2 , the principal curvatures.

Although the principal curvatures are useful descriptors of surfaces, they lack one important feature: view-point invariance. The values of the descriptor change when the image is viewed from another location and angle. Gaussian and Mean curvature do incorporate this feature, and can be computed as follows¹:

$$\text{Gaussian curvature: } K = \kappa_1 * \kappa_2 \quad \text{Mean curvature: } H = (\kappa_1 + \kappa_2)/2 \quad (4.8)$$

It is customary to threshold and combine K and H such that different classes of surface shapes arise [11]. Two thresholds are defined, τ_K and τ_H . Usually, τ_H is predefined; a sensible threshold for K can be computed from τ_H using formula 4.9, discussed in [9]. The signed values of K and H are then computed using the two formulae in 4.10.

$$\tau_K = \tau_H * (\tau_H + 2 * \max\|H(x)\|) \quad x \in \text{Image} \quad (4.9)$$

$$\text{sign}(K) = \begin{cases} > 0 & \text{if } K > \tau_K \\ = 0 & \text{if } \tau_K \leq K \leq \tau_K \\ < 0 & \text{if } K < \tau_K \end{cases} \quad \text{sign}(H) = \begin{cases} > 0 & \text{if } H > \tau_H \\ = 0 & \text{if } \tau_H \leq H \leq \tau_H \\ < 0 & \text{if } H < \tau_H \end{cases} \quad (4.10)$$

The signed curvatures are then combined to define the nine surface classes shown below. If a pixel has a negative signed mean curvature, and a signed Gaussian curvature of zero for instance, it is likely that this pixel belongs to a convex cylinder.

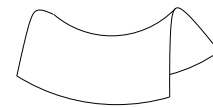



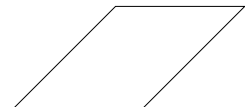

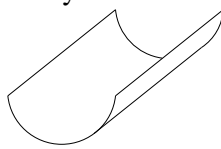
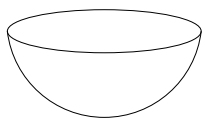
		K		
		-	0	+
H	-	Saddle ridge 	Ridge 	Peak 
	0	Minimal surface 	Flat 	None
	+	Saddle valley 	Valley 	Pit 

Figure 4.7: The HK surface classification

¹Although this computation gives a better feel for the Gaussian and Mean curvature, they are actually computed in a slightly different way, bypassing the principal curvatures. Computing the curvatures directly from the derivatives is less intuitive but more efficient. A mathematical summary is given in Appendix B

Summarising the whole method of surface classification.

1. Compute derivatives, section 4.3.2.
2. Compute principal curvatures κ_1 and κ_2 , see Appendix B.
3. Compute Gaussian (K) and Mean (H) curvature, equation 4.8.
4. Define threshold τ_H and compute τ_K accordingly, equation 4.9.
5. Compute the signed curvatures using τ_H and τ_K , equation 4.10.
6. Combine the signed curvatures to define the 9 surface classes, table 4.7.

4.4 Region growing

Now that some of the features of each of the points in the image have been determined, it is time to merge them into patches of points that all have the same characteristics. This will hopefully provide us with large homogeneous patches to which surface fitting can be applied well.

The region growing algorithm is based on a very simple breadth-first-search queue. It starts off with a certain point and pushes it on the stack. All its neighbours are tested for equality based on requirements that will be discussed in the next three sections. If a neighbour meets all these requirements, it is considered to belong to the same region and pushed on the stack as well. This means that when it is popped from the stack, its neighbours will be tested for equality as well. Of course a record is kept of all the points that have been in the stack, because they all belong to the same homogeneous region. When the stack is empty the region growing (for this region) is finished and a new point (that does not already belong to a certain other region) is pushed on the stack. This goes on until all the points belong to a certain region. After the entire region growing process is finished, regions that are considered too small (≤ 50 pixels) are merged into surrounding larger regions.

4.4.1 Neighbours

As has just been explained, points will only be added to a region if they are close enough to this region, that is, one of their neighbours must be part of this region. This allows us to spot depth discontinuities, a process usually done by edge-detection algorithms. This method is far superior, because it works in the 3D domain, not in the geometrically distorted 2.5D range image. A simple example of detecting a depth discontinuity can be seen in figure 4.8

4.4.2 Difference in normals

Another feature that is useful is the similarity in normal. If the angle between the surface normals of two neighbouring points is large, it is not likely that the two points belong to the same continuous surface. This happens at corners and roofs, which are called fold-edges. Therefore they are not considered to belong to the same region. A reasonable threshold is 7.5 degrees, which is larger than the threshold used in similar methods such as described in [40]. This large threshold had to be chosen to cope with the amount of noise in the Bornholm church dataset. A simple fold-edge detection can be seen in figure 4.8. In this image, the normals are depicted by arrows.

4.4.3 Curvature

A very basic demand is that the surfaces are roughly of the same shape. Cylinders might smoothly meet a plane with tangent continuity (their normals are the same at the intersection of the two surfaces). Neither the segmentation on neighbours or difference in normals will detect this. Therefore they must also have the same curvature sign. Because the cylinder is of a different shape (and thus curvature sign) than the plane the points will be appointed to different regions. The simple cylinder/plane example can be seen in figure 4.8.

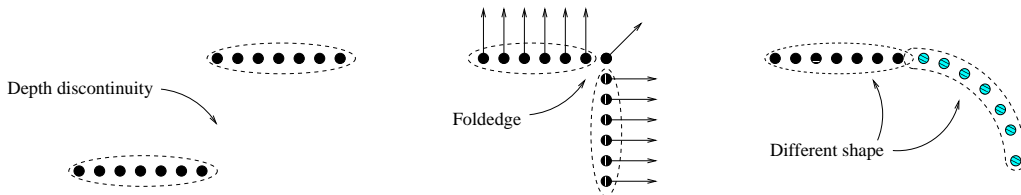


Figure 4.8: Locating depth discontinuities, fold-edges and differences in shape. The ellipses and curves enclose points belonging to the same region.

4.4.4 Intermediate results of the region growing algorithm

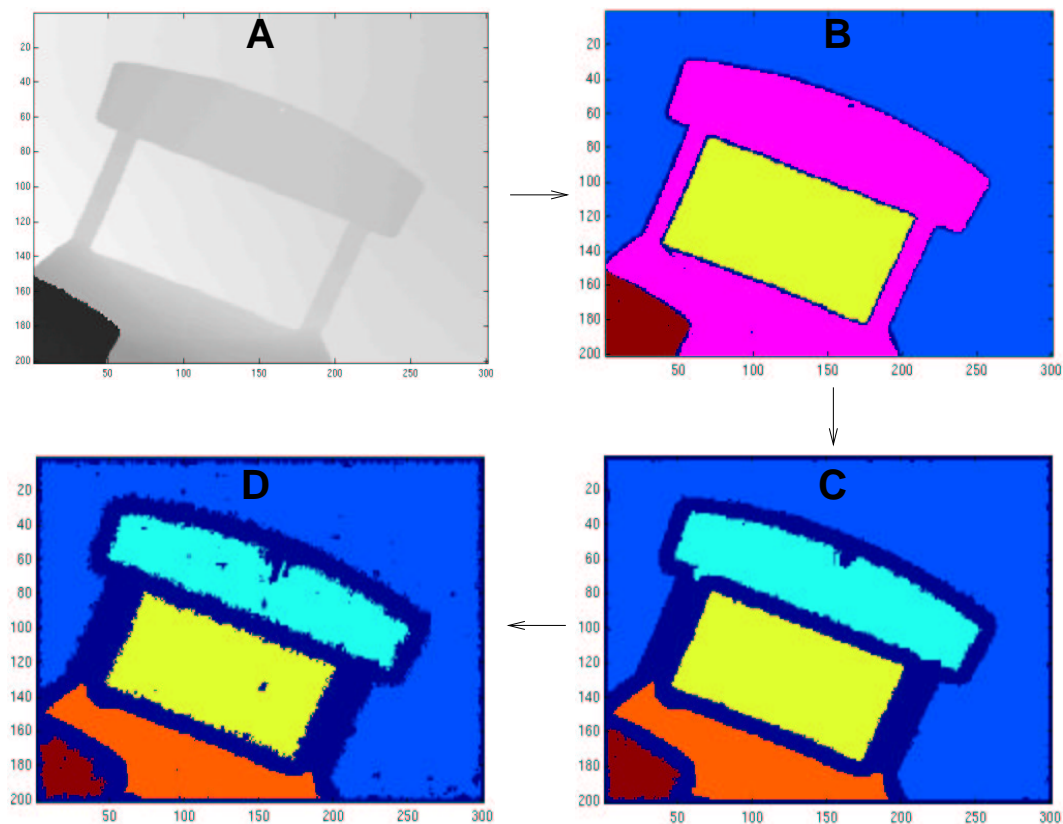


Figure 4.9: Result of the region growing algorithm

Diagram 4.9 shows the result of applying the segmentation algorithm on a range image. Image A is the original range image. Image B shows the different regions that are acquired after region growing on the basis of neighbours (section 4.4.1). The edges between two regions represent a depth discontinuity. Image C shows the regions after region growing on the basis of the difference in normals, as discussed in section 4.4.2. Computing normals around edges is prone to noise. Many points around the edges is therefore lost. Most of these pixels will be regained in the surface growing process discussed in section 4.5.3. Image D is the final segmented image through region growing. These regions are acquired by region growing on the basis of equality in curvature, as discussed in section 4.4.3.

4.5 Surface fitting

As mentioned earlier, clusters of points in the image will be modeled by a simple geometrical description. This representation should make comparison between the different clusters much easier. The goal of the segmentation algorithm so far has been to find continuous surface regions of roughly the same shape. Finding a good representative or generalisation (of low complexity) of a continuous surface region is much easier than finding one for an arbitrary gathering. Because comparing and computing models of low complexity is much easier, our fitter is very lazy, and only tries to fit simple models. This laziness means that the fitter has to be provided with convenient pre-segmented data. The curvature sign also allows us to give the fitter hints as to what kind of model it should fit (planar, cylindrical, spherical). Although the segmentation has been a laborious and intricate process, it has merely been an aid to the surface fitter.

Implicit polynomial curves and surfaces have proven to be good descriptors of 2D and 3D data [22]. They are very compact descriptions and they are good at smoothing noisy data and interpolating through sparse or missing data [33], which is exactly what the completion algorithm needs. Equation 4.11 depicts a general implicit polynomial of degree d in 3 dimensions. Because the power of interpolation through missing data diminishes with the height of the degree, it is restricted to surfaces of degree 2, which are called *quadrics*. In section 4.5.1 examples will be given to clarify matters.

$$f_d(x) = \sum_{\substack{(i+j+k) \leq d \\ \{i,j,k\} \geq 0}} a_{ijk} \cdot x^i \cdot y^j \cdot z^k = 0 \quad (4.11)$$

The goal in fitting a surface to a set of points is to choose the parameters of the surface such that the surface is a good representation of the points. A common criterion for this *goodness of fit* is the total of the (squared) distances of all the points to the surface. Minimising this total will yield the best possible fit.

Computing the real distance between a smooth function f and a certain point \vec{x}_i can be described formally as computing the distance between point \vec{x}_i and the closest point \vec{x}_t for which $f(\vec{x}_t) = 0$ holds. Since $f(\vec{x}_t) = 0$, \vec{x}_t lies on the function. The set of points for which $f = 0$ (also called the *zero-set* for f) is defined as $\mathcal{Z}(f)$. The point \vec{x}_t in this set that is closest to \vec{x}_i will have to be found. The distance between \vec{x}_i and f is then equivalent to the distance between \vec{x}_i and \vec{x}_t . Equation 4.12 summarises this reasoning.

$$dist(\vec{x}_i, \mathcal{Z}(f)) = \min\{\|\vec{x}_i - \vec{x}_t\| : f(\vec{x}_t) = 0\} \quad (4.12)$$

Fitting a function f to a finite set of data points $\{x_1..x_n\}$ is usually done by minimising the mean square distance between the data points and the function:

$$\frac{1}{n} \sum_{i=1}^n \text{dist}(\bar{x}_i, \mathcal{Z}(f))^2 \rightarrow \text{Minimum} \quad (4.13)$$

4.5.1 Euclidean vs. Algebraic distance

A method of fitting a general function to a set of data points has been defined. What has been neglected is an exact explanation of how the minimum distance in equation 4.12 is supposed to be computed. The reason for this is that this is a complicated process.

A first simplification that can be made is to limit the domain of f . Since only implicit polynomials of degree 2 are fitted, only methods for computing the distance between a point and these specific functions will have to be determined.

The matter is still far from trivial though. There is no closed form expression for computing the real or *Euclidean* distance from a point to a *general* quadric. Computationally expensive iterative methods are always needed to solve this problem. For this reason the efficient, but inaccurate, *algebraic* distance is often used. It can simply be computed with:

$$\text{dist}(\bar{x}_i, \mathcal{Z}(f)) = f(x_i) \quad (4.14)$$

To show its inaccuracy two examples will be given. To simplify matters the examples will be given in two dimensions, but the idea is easily extrapolated to three dimensions. In the following equations a quadric in 2D is defined, as well as two special cases of these quadrics: a line and a circle.

$$f_2(x) = \sum_{\substack{(i+j) \leq 2 \\ \{i,j\} \geq 0}} a_{ij} \cdot x^i \cdot y^j = 0 \quad (4.15)$$

$$\text{Line: } a_{10}x + a_{01}y + a_{00} = 0 \quad (4.16)$$

$$\text{Circle: } a_{20}x^2 + a_{02}y^2 + a_{10}x + a_{01}y + a_{00} = 0 \quad (4.17)$$

Figure 4.10 shows a line and a (unit) circle in 2D graphs. The values for a_{ij} are also depicted in the figure. The Euclidean distances are shown in both graphs. For the line, the closest point on the line (x_t) to point (3, 2) is (1, 3), so the real distance between the line and point (3, 2) is $\sqrt{5} = 2.236$. Computing the algebraic distance to the line yields:

$$\begin{aligned} &\text{Algebraic distance from line } 2x - y + 1 = 0 \text{ to point } (3, 3) \\ & \text{dist}([3 \ 3], \mathcal{Z}(f)) = f_2([3 \ 2]) = 2 * 3 - 2 + 1 = 5 \quad (\text{real value is } 2.236) \end{aligned} \quad (4.18)$$

It is also depicted in the graph. In other words, the algebraic distance is more than *twice* the real Euclidean distance. For the circle, the distance of point (2, 1) to the circle is (without further proof) $-1 + \sqrt{5} = 1.236$. Again, computing the algebraic distance yields a completely different value.

$$\begin{aligned} &\text{Algebraic distance from circle } x^2 + y^2 - 1 = 0 \text{ to point } (2, 1) \\ & \text{dist}([2 \ 1], \mathcal{Z}(f)) = f_2([2 \ 1]) = 2^2 + 1^2 + -1 = 4 \quad (\text{real value is } 1.236) \end{aligned} \quad (4.19)$$

This value is especially absurd, as there is not one point x_t on the circle that has a distance of 4 to point $(2, 1)$. It can not sensibly be depicted in the graph.

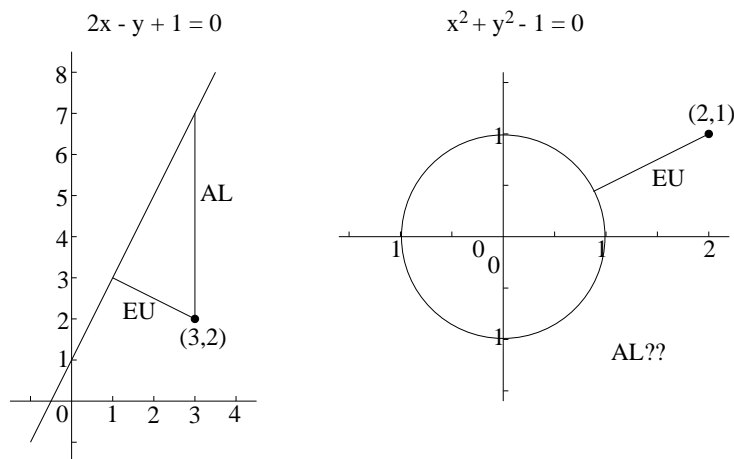


Figure 4.10: Two examples of Euclidean and algebraic distance

It must be said that the difference between algebraic and Euclidean distance can sometimes easily be resolved. Equation 4.16 that describes a line contains an arbitrary scale factor for instance. If the coefficients are chosen such that $a_{01}^2 + a_{10}^2 = 1$, the algebraic and Euclidean distance are the same. The example of a line was chosen because it is a simple introduction. The polynomial for a circle also contains an arbitrary scale factor, but it can never be set to a value such that the algebraic and Euclidean distances are always equivalent. Although the line might have been a bad example, the problem is already unavoidable at other simple functions such as circles. These examples are not the worst case scenario, and it does not take long to find other misjudgements in distance. When using the algebraic distance for surface fitting, the results are often unsatisfactory. This is actually one of the motivations for writing a new segmentation algorithm. All the segmentation algorithms available used the algebraic distance. The fitting was too poor to use for the completion algorithm.

Then why not simply use the real Euclidean distance in surface fitting? The reason for this has been mentioned briefly before: there is no straight-forward method for computing the Euclidean distance from a point to a general quadric. Fortunately, research is never done alone. Petko Faber, also a Research Fellow with the Vision Lab in Edinburgh, has done extensive research on computing the distance between a point and a multitude of shapes and surfaces. He has allowed me to use his code in my program, for which I am very grateful. The rest of this section (as well as section 4.5.2) gives a brief overview of the theoretical background of Petko Faber's algorithm, which is fully described in [22].

The most convenient way of computing the geometric distance is using the closed-form solution. This means that the function $dist(\vec{x}_i, \mathcal{Z}(f))$ can be solved simple and directly, without iteration. It will return the *exact* Euclidean distance. In cases when this was theoretically possible, Petko has derived a closed-form solution for the shape or surface.

Most shapes and surface can not be described in such simple terms, and a closed-form solution is not readily available. In these cases, a first estimate is computed and then

iteratively improved to yield an accurate approximation of the real geometric distance. Although it is always an approximation, it is a far better estimate than the algebraic distance.

The first estimation of the distance is computed using a method developed by Taubin [48]. The idea is to approximate the distance from the point to the surface by approximating the surface by a first order approximation and computing the distance between it and the point. The general idea is much like the computation of the derivatives discussed in 4.3.2.

The Taubin estimation is biased to points close to the surface, and is not very accurate in these cases. For this reason this first estimate will have to be improved. The estimate is updated using the Levenberg-Marquardt algorithm [35, 36] and the resulting new estimation is evaluated. This goes on until the new estimation is not an improvement over the previous one. This does not yield the exact Euclidean distance, but a very good approximation of it.

This approximation of the distance has been used in formula 4.13 to fit the different quadrics to the different sets of points acquired in region growing.

4.5.2 Choosing the right surface

The goal in surface fitting is to find a simple description of the data, that represents the data well. This means there will always be a trade-off between the complexity of the function that has been chosen to represent the data and the goodness-of-fit of this function.

Man-made objects such as buildings often consist of simple surfaces such as planes, cylinders and spheres. This is simply because these shapes are much easy to manufacture. Therefore, only these simple surfaces are fitted to the data. The surface with the best fit is the surface chosen. Because any plane can be described by a cylinder or sphere with infinite radius, a restriction is put on the size of the radius. It is then more likely that a plane is fitted to flat structures.

4.5.3 Surface growing

A surface has been fitted to the original segment acquired through the region growing algorithm. It is often the case that points that are close to the surface patch, and have not been included in the original segment might very well fit into the surface. Although these points have not been involved in determining the parameters of the surface, the surface might very well be a good representative of these points as well.

Therefore the surface is expanded, allowing it to incorporate neighbouring points that fit the surface well (a process some call *slurping*). Points that are adjacent to the original segment, and fall within the noise-level of the surface are simply incorporated into the region. The information content of the surface patch is growing, as it incorporates more and more points, while the *goodness of fit* of the surface does not deteriorate.

4.5.4 Results

In figure 4.11, the segmented image from figure 4.9 is repeated on the left hand side. The right hand image (the same segmentation after surface fitting and growing) clearly shows that many more points have been incorporated into the surface patches. The 3D points that can be seen in figure 4.12 are a visualisation of the surfaces in 3D. The colors match those of image 4.9. In this image, the missing data is again apparent.

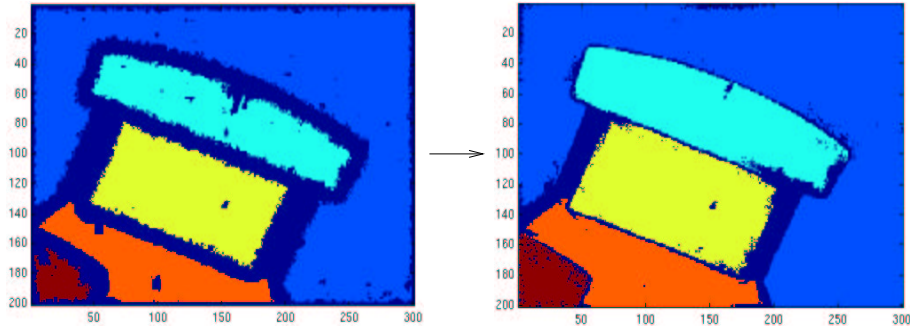


Figure 4.11: A result of the region growing algorithm (left) and the same segmentation after surface fitting and surface growing (right)

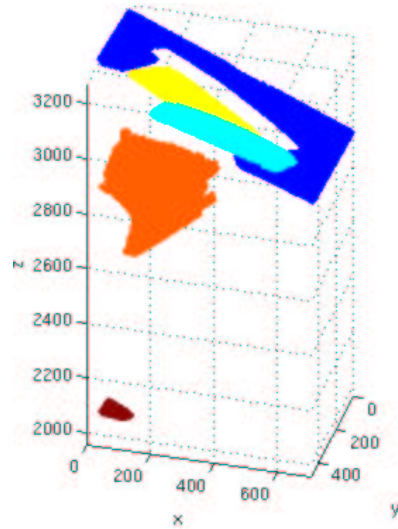


Figure 4.12: A result of the region growing algorithm (3D)

4.5.5 Output format of the segmentation algorithm

As has been mentioned at the beginning of this chapter, the goal of segmentation was to cluster related points in the image, and fit a surface to them. This goal is reflected in the simple output format of the segmentation algorithm, which are surface patches. The surface completion algorithm that will be discussed in the next chapter will only get these surface patches as input. The definition of surface patches is repeated in the box below.

It is important to note that it is very easy to use completely different segmentation algorithms, as long as the data format is the same. The motivation for writing a new segmentation algorithm has been given in sections 4.3.1 and 4.5.1.

Terminology

(Geometrical) Surface: A parametric description of a continuous function in 3D space. This research is limited to simple special cases of implicit second order polynomials (see section 4.5) such as planes, cylinders and spheres.

Region (or patch): A group of pixels in the r_{ij} image. In our algorithm, these groups are acquired through region growing (section 4.4) and surface growing (section 4.5.3).

Surface patch: A region in the r_{ij} image of which all the pixels are represented by the same surface.

4.6 Summary

In this chapter the methods for segmenting a range image were presented. First of all the image is smoothed using nearest neighbourhood filtering. The neighbours of all the points are computed, as well as some surface characteristics: the surface normal and local curvature. Neighbouring points with the same surface characteristics are then merged into regions using a region growing algorithm. Planes, cylinders and spheres are then fitted to the points in these regions, and the surfaces are grown to incorporate all the points that are represented well by this surface. The final result is a group of surface patches: regions in the image described by a simple geometrical surface.

Chapter 5

Surface Completion

5.1 Introduction

This chapter will discuss the method with which a hypothesis for the entire 3D configuration of partially occluded objects is constructed. The algorithm takes the output of the segmentation algorithm (surface patches) as its input. The main idea is that most objects, and certainly those encountered in buildings, usually do not contain surfaces that stop abruptly in mid-air. This is why it is not expected that walls contain chair-shaped holes for instance. This is Lehar’s previously mentioned “principle of good continuation” [34]. The goal now is to actively fill in data in areas where, on the basis of good continuation, it is more likely that there is a non-visible surface than no surface at all.

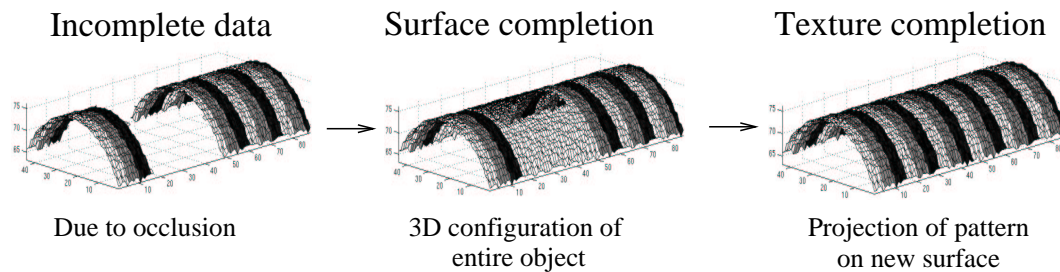


Figure 5.1: The processing chain of completion

An overview of the completion algorithm is shown in diagram 5.1. The input is segmented data as shown in the first image. The two parts of the cylinder are two different surface patches acquired in the segmentation algorithm discussed previously. The two parts of the cylinder appear to be two parts of the same cylinder: intermediate data is missing. The surface completion algorithm first hypothesizes the configuration of the entire cylinder, shown in the second image. The texture completion algorithm is then applied to acquire the final completed cylinder shown in the last image of diagram 5.1.

The rest of this chapter is devoted to the surface completion algorithm, and is divided into the following sections.

Locating possible occlusion (section 5.2): This section discusses how occluded areas are located for different classes of occlusions.

Making the surface hypothesis (section 5.3) This section presents how the 3D configuration of the entire surface is hypothesized in the areas that might contain occlusions.

Niche or real occlusion? (section 5.4): The surface has been hypothesized. One last check has to be made before the data can be added to the original image, yielding the completed surface. This check is whether the possibly occluded area represents a niche or a real occlusion. A niche is a recess in a smooth surface. They are often found in churches, in which a statue is placed in a recess in the wall. Hypothesized data should not be added to niches, as this would cause the recess to be sealed shut; a statue would no longer be visible.

Terminology

Hypothesized point/surface: Once the possibly occluded region is determined, a hypothesis is constructed for each point in the region; what would have been seen at that point if occlusion had not taken place? All these points together make a hypothesized surface.

Completed point/surface: If the possible occlusion turns out to be a niche, the hypothesized points are simply discarded. If it is a real occlusion, the hypothesized points are added to the image. These points complete the image, and so are called completed points. Similar terminology holds for surfaces.

The difference between a hypothesized and a completed point/surface is purely temporal: before the niche-check it is hypothesized, after the niche-check it is either discarded or added to the image, which is called completed.

5.2 Locating possible occlusions

This section is distinct from the other sections discussing segmentation and completion for two reasons. The first distinction is that occlusions are classified, and different methods are applied to different classes. This is distinct, because it is the only section in which generalization is lost. After this section, the classification is abandoned, all occlusions are treated the same, and generalization is regained. The second distinction is that the methods are applied to the 2.5D image, not the 3D image that has been used so far. After this section all computations will be made in the 3D domain again. These choices will briefly be motivated, after which the different methods used for locating possible occlusion in each class of occlusion are described.

5.2.1 Classification of occlusions

In the design of the program I have tried to abide by Ockham's razor: The simplest explanation is usually the best. This means the program should be adaptive, and should be able to cope with a diverse set of examples in one uniform way. During the research it became apparent that there are many types of occlusion. Determining where these occlusions are

located has proven to be too hard to generalize over all these different types. A classification was needed.

One useful classification is shown in figure 5.2. It is based on how many times the occluding object transects a boundary of the occluded surface in the 2.5D range image: 0, 1 or >1 . The description of the three classes below will clarify matters.

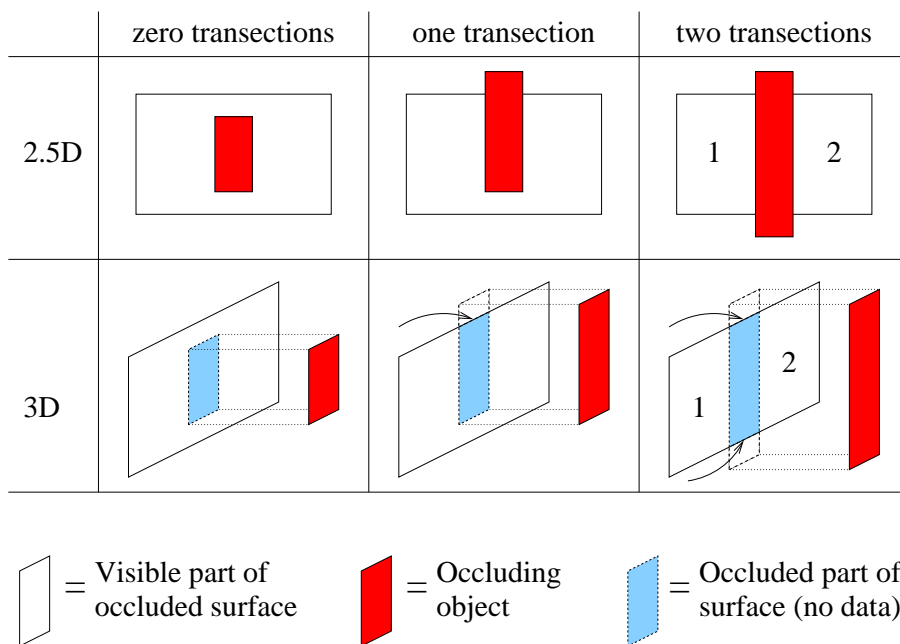


Figure 5.2: A classification of occlusions of single surfaces

In the first class, the occluding object is completely contained within the occluding surface patch, as seen in the 2.5D range image. The occluding object does not transect the boundary of the occluded surface once. An example of this type of occlusion is the top view of a book lying in the middle of a table: in the range image the region describing the (planar) table surface completely surrounds the region describing the (planar) book surface.

The second class describes objects that transect the boundary of the occluded surface once. Suppose the book were pushed to the edge of the table, so that part of the book was hanging over the table's edge (one transection of the boundary). The single-transection occlusion is shown in the center column of the table in diagram 5.2. In the 3D image, the arrow points out the transection.

The last class describes occlusion in which the occluding object transects the boundary of the surface patch at two or more locations. In this case, multiple instances of the same surface can be seen in the range image, depicted in the right column of diagram 5.2 with 1 and 2. An example of this is a long ruler lying on the table. If the ruler is longer than the table, the ruler could be placed such that both ends are hanging over the tables' edge. Because the ruler transects the table into two parts, the range image now contains two regions, both instances of the same table.

5.2.2 Back to 2.5 dimensions

This section will briefly motivate why possible occlusions are located in the 2.5D-form of the range image, and not in the 3D-form as has been the case so far. The main reason is that it ensures that points are only hypothesized at locations that could actually have been registered by the sensor. This seems like an intuitive domain to restrict the methods to.

Because the sensors' movements are not incremented continuous but by evenly spaced increments (digital), there is a limited number of possible locations where the measurement takes place. These measurements are of course exactly those registered in the pixels of the 2.5D range image. It is therefore valid to locate the occluded 3D points pixel by pixel in the 2.5D range image.

Usually, whole areas are occluded, instead of single pixels. This occluded area will manifest itself as one continuous region in the 2.5D range image. The two great benefits of applying the methods to the 2.5D domain are efficiency (because the points are ordered) and the multitude of well known 2D methods (computing the distance transform, flood-filling) that can be used on this form of the data. Briefly returning to the 2.5D will change the concept of distance that has been used so far. This change is explained below.

Terminology

3D Distance: The *real* or Euclidean distance between two points in 3D space.

2D Distance: The distance (in pixels!) between two pixels in a 2D digital image.

Up till now the word *distance* has been used to indicate the 3D distance. From now on it will refer to the 2D distance; reference to the 3D distance will be made explicitly.

Now that the classification has been made, and motivation has been given why computations are made in the 2.5D, the next three sections (5.2.3 through 5.2.5) will discuss how possibly occluded regions are located for zero-, single-, and multi-transection occlusions.

5.2.3 Locating zero-transection occlusions

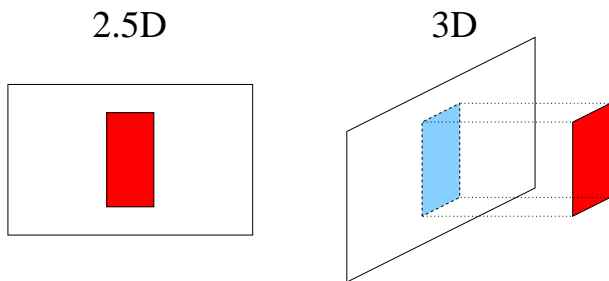


Figure 5.3: Zero-transection occlusions

This type of occlusion is the 'book on the table' example as shown in diagram 5.3. Because the region of the occluding object is completely contained within the region of the

occluded surface, only one region in the image represents the surface. This means that only one surface is needed to complete the hidden data.

The goal now is to find regions (possibly representing occluding objects) completely contained within other regions (possibly representing occluded surfaces). Because these occlusions can take place in any kind of surface patch, they all have to be checked for possible occlusions. The method used employs a simple flood-fill algorithm.

Image 5.4 describes the method. In this image, a plate and a book can be seen lying on a table, the image being taken from above. Note that both objects do not transect the boundary of the table, and are therefore zero transection occlusions. The way to extract these two regions from the image is done as follows. First of all, the area of the table, as well the area around the table is excluded by flood-filling the image from the boundary with a *NOT-AN-OCCLUSION* label. The result of this process can be seen in image B of diagram 5.4. The two remaining regions are those that need to be extracted. This is done by flood-filling both separately, for instance with labels *OCCLUSION-1*, *OCCLUSION-2*, etc. The final result is shown in image C.

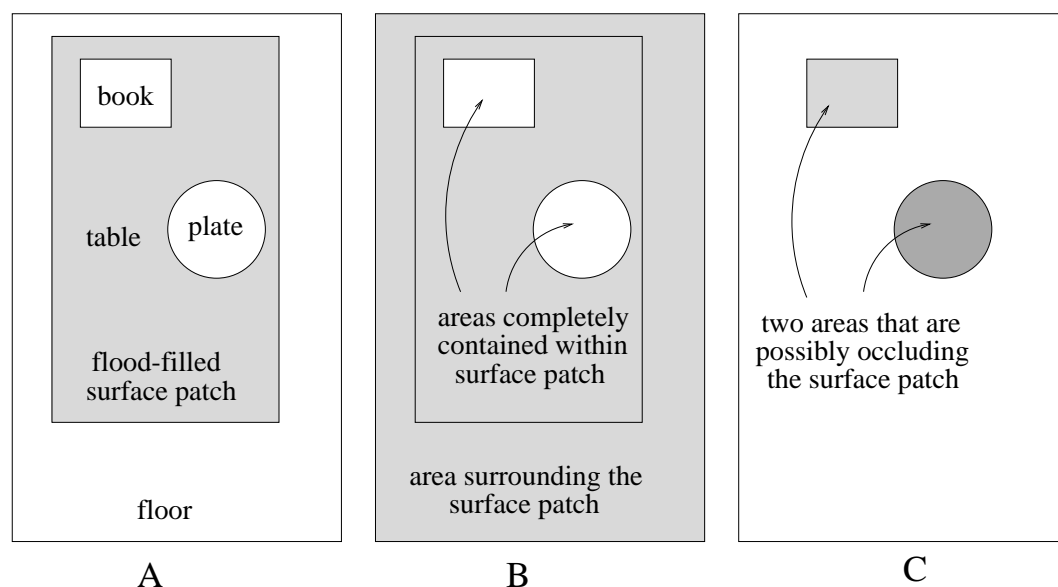


Figure 5.4: Locating zero-transection occlusions

5.2.4 Locating single-transection occlusions

This class of occlusion (an example is repeated in diagram 5.5) is by far the most difficult class to locate. This is because the occluding object partially overlaps the boundary of the occluded surface. Because the boundary of the occluded surface cannot be seen, it is not clear within which bounds the surface should be completed. Simply hypothesizing data behind the entire occluding object might lead to strange artifacts, as completion might be taking place in areas where there is no occlusion.

An example. Suppose a book is placed on a table, such that part of the book extends beyond the table. An image from above would show that the book is occluding the table, so completion is necessary. This is shown in the left image of diagram 5.5. Note that

completion is only necessary behind part of the book (the blue region in diagram 5.5), not all of it. Because the book is covering the boundary of the table (arrow in diagram 5.5), it is not immediately clear where the boundary of the table lies, so it is also not clear behind which part of the book completion should take place. Completing behind the entire book is not valid, as more table would be completed than is actually present in the real world. Imagine the protruding part above the blue region in the right image of diagram 5.5 to be completed as well. This would lead to a book-shaped piece of table protruding from the table; an undesirable artifact.

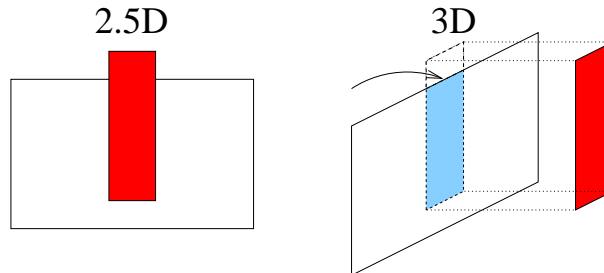


Figure 5.5: Single-transection occlusions

The boundary of the table is apparent to humans, because they are good at interpolating between the two visible parts of the boundary of the table. Computer vision techniques would depend on fitting lines and other functions to the boundaries, and check if interpolation between them is valid. Although such techniques have been used before in for instance [23], they are quite intricate, and cause many difficulties. Unfortunately, implementing the methods needed to locate these occlusions (line and curve fitting and matching and interpolating between them) was not feasible given the duration of the research. In the future, the problem is sure to be solved however, as discussed in section 7.2.1.

5.2.5 Locating multi-transection occlusions

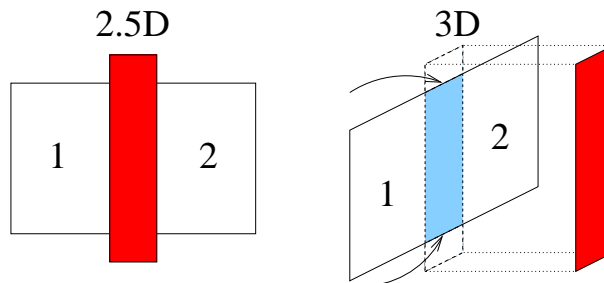


Figure 5.6: Multi-transection occlusions

The main issue in multi-transection occlusions is that the occluding object divides the occluded surface into one or more regions (see diagram 5.6). The goal is therefore to find groups of surfaces patches that are instances of the same (occluded) surface, e.g. find two parts of a wall that actually belong to the same wall. The areas between these regions are

then determined.

Coinciding surface patches

The way this is done is by comparing the geometrical surfaces of the surface patches to locate coinciding surface patches. Remember that a surface patch is the combination of a region in the image and a geometrical surface representing the points in this region. Here we see the strength of the surface fitting: comparing the individual points in the region of the surface patch would be infeasible. The generalization of the geometrical surface allows a meaningful comparison. Coinciding surface patches are defined as surface patches that roughly have the same geometrical surface describing the points they contain. To see if two surface patches coincide, their geometrical surfaces are compared. First of all, two surface patches can only coincide if they are represented by the same class of geometrical surface (plane, sphere or cylinder). If this is the case, they must also meet the following requirements for each class.

Planes: The first requirement for planes to be coinciding (also called coplanar) is that the angle between their surface normals is less than 5 degrees. Furthermore, the displacement between the two planes may not be too large. The planes may not lie too far apart from each other. The issue now is where, and in which direction the distance between the two planes should be measured. The location will be the center of mass of all the points in the two planes. The direction will be the summation of the two surface normals. The line through this location and with that directional vector has an intersection with both planes. The distance between these two intersections is taken as measure for the distance between the two planes.

Spheres: First of all, the radii of the spheres must not differ by more than 10%. Furthermore, the centers of the two spheres may not be too far apart from each other in 3D space. The threshold used for this is 10% of the mean of the radii.

Cylinders: A first requirement is again that the radii of the cylinders must not differ by more than 10%. Furthermore, the angle between the vectors representing the direction of the axis must not be more than 5 degrees. The last requirement is that the axes of the cylinder must be collinear. This is done by determining the point on each axis nearest to the center of mass of both cylinders. The vector through these two points must also meet the 5 degrees requirement with both vectors describing the axes of the cylinders.

This last requirement has a bias towards accepting long cylinders. This is because the center of masses of two long cylinders will be further apart than those of short cylinders. The vector between these two points is then more likely to be collinear with the two axes. A new method that removes this bias could easily be implemented. Instead of taking the points roughly in the middle of the cylinders, the points on the most extreme sides of the cylinders (on the axis and within the boundaries of the cylinder), closest to the other cylinder could be used. This introduces a bias towards cylinders that are far apart. This bias is not as influential, as distance constraints are placed elsewhere (section 5.3.3). Furthermore, interpolation between cylinders that are far apart will cause less apparent discontinuities than between cylinders that are close to one another.

After the coinciding pairs of surfaces have determined, an algorithm that clusters these pairs into groups is applied. Groups of coinciding surfaces are defined such that every surface in the group must coincide with every other surface in that group. Coincidence relationships within a group are therefore always transitive. The reason this is done is that when the surface hypothesis is made, it is necessary to take into account all the instances of the same occluded surface.

Area between coinciding surface patches

The area between a set of surface patches is simply the summation of all the areas between every pair of surface patches in the set. The idea behind computing the area between a pair of surface patches is to find perimeter points of one area that ‘face’ perimeter points of the other area. ‘Facing’ perimeter points are points that can be connected with a line that does not cross any part of any of the two areas of the surface patches itself. A consequence of this requirement is that this line usually only crosses pixels that lie between the two surface patches. Because completing between two surface patches that are very far apart is undesirable, another requirement is that the distance between two facing perimeter points may not be too large. How the threshold for this measure is computed is described in section 5.3.3. In image A of diagram 5.7, two areas and their perimeter points are shown. In image B, the perimeter points facing each other are drawn in black. Apparently, it is not possible to connect the gray (non-facing) perimeter points in image B with any of the perimeter points of the other area without either being too far apart, or the line between them crossing one of the two areas¹.

All the points that are facing each other according to the definitions given above are connected by lines. The pixels on these lines are then considered to lie between the two surface patches. Which pixels are on the line is determined using the Bresenham algorithm [8]. In image B, the gray area represents the pixels that have labeled using the this algorithm. Connecting facing perimeter points is not sufficient though. As can be seen in image B, some holes between the two patches remain. These are simply filled in using a flood-fill algorithm. The final result can be seen in image C.

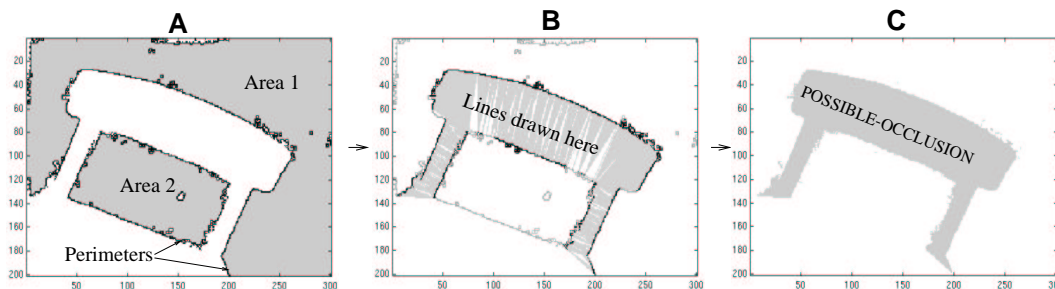


Figure 5.7: Determining the area between two surface patches

¹The requirement of the line between two perimeter points not crossing any of the surface patches was not as strict as described above. In reality, it was specified that no more than 10% of the pixels in the image that are crossed by the line were allowed to be pixels that belonged to one of the surface patches.

5.3 Making the surface hypothesis

Now that some areas that are possibly occluded have been found, it is time to hypothesize what would have been seen in these areas had occlusion not taken place. Note that, although locating the possibly occluded area had to be done with different methods for each class of occlusion, the next method will be the same for all of them. This is reflected in the universal data-type, which contains:

The possibly occluded area This is the area that has been located in the previous chapter. It will be completed using information from the surrounding surface patches.

A list of surface patches These are the surfaces and the regions in the image they represent. The list can be of length one (zero-transection occlusions) or larger (multi-transection occlusions).

5.3.1 Interpolation between intersections

As mentioned before, points should only be hypothesized at locations that could actually have been measured by the sensor. Because the points in the possibly occluded area have been actually measured, the replacing completed point must lie on a line through the sensor and the measured point. As this line overlaps the optical ray of the sensor, the hypothesized pixel is placed in a position that could actually have been sensed by the sensor.

Now it needs to be hypothesized what would have been seen, had the surface, and not the point actually measured been seen. This is simply done by intersecting the line through the sensor and the measured point (optical ray of the sensor) with the surface of the surface patch. This ensures that the acquired 3D point could have actually been measured, and lies on the surface. These intersections are computed for all points in the possibly occluded area.

This method alone would work for occlusions causing only one occluded surface patch, but when more surface patches are involved it fails. Due to noise, the surface fitting is never perfect, and the surfaces fitted to the points in the regions will never coincide perfectly. When extrapolating the surfaces into the occluded area, they will never perfectly coincide, so computing the intersection with one of them is not enough, as this can lead to strange depth discontinuities when going from one surface patch to the other. For this reason the ray is intersected with all of the surfaces of the surface patches. Interpolation between the different intersections yields the final point.

The whole process of intersecting the ray of the sensor with the surfaces followed by the interpolation can be seen in figure 5.8. It must be noted that this is not an actual case; the dimensions have been distorted for clarity.

The chosen interpolation is a weighted averaging between all the intersections. It is weighted, because the influence of certain surface patches should be more than that of others. This all depends on the distance between the point to be hypothesized and the different surface patches. If a pixel is very close to one surface patch but very far from another, the hypothesized pixel should be more like the closer surface patch. This ensures continuity around the edges between the completed area and the patches itself. Therefore the weights of the averaging depend on the proximity of the point to the different surface patches. In other words, given a pixel to be hypothesized, two or more intersections are computed as

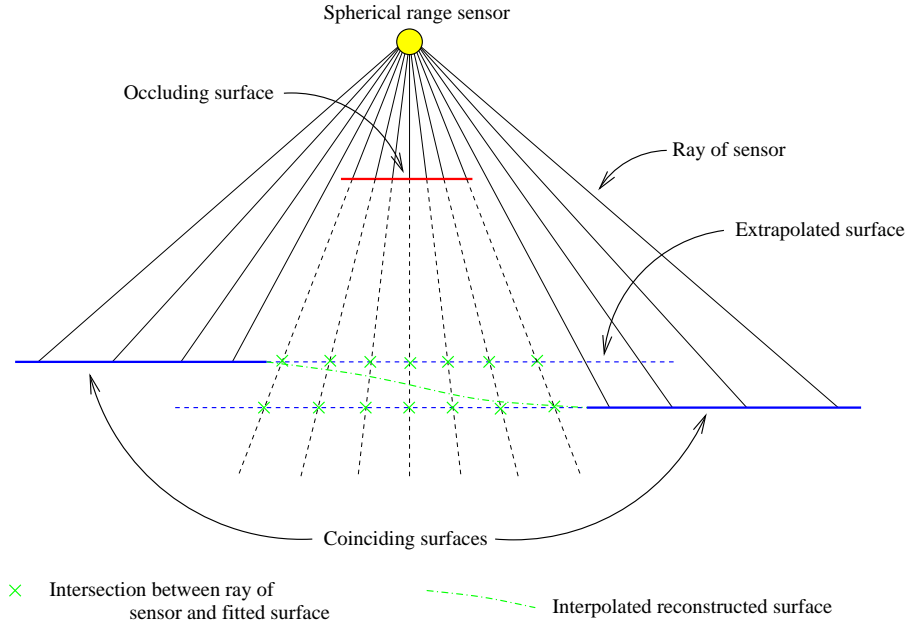


Figure 5.8: Computing intersections and interpolating between them

explained above, one per each candidate surface. The position of the hypothesized pixel is computed as follows:

$$\vec{x}_{inter} = \frac{\sum (f(d_s) \cdot \vec{x}_s)}{\sum f(d_s)} \quad (5.1)$$

$$f(d) = (d_{max} - d)^{\frac{3}{2}} \quad (5.2)$$

The summation in equation 5.1 (which is a simple weighted averaging function) takes place over all the surfaces patches involved in the hypothesis; that is, all the transitively matching surfaces combined in a group. \vec{x}_s is the intersection with the s -th surface, d_s is the distance to the closest actually observed point in surface s , and $f(d_s)$ a weighting function. d_{max} is the maximum of all the distances of all the pixels to be hypothesized to all the surface patches. Note that d_s will therefore never exceed d_{max} . It is also important to note that $f(d_s)$ expresses exactly that the influence of a surface patch on the hypothesized point is greater if it is closer.

Some explanation needs to be given on how d_s is computed. Computing the smallest distance from a point (the pixel at \vec{x}_s) to a set of points (the points in the surface patch) is no trivial task, as might be recalled from section 4.5.1 on Euclidean fitting. To avoid serious computational problems, the distance transform of all the surface patches involved is computed. The distance transform contains information about the shortest distance of every pixel in the image to a certain set of object pixels, in this case the points of the surface patch. More information about distance transforms can be found in Appendix C. Another useful interactive source on distance transforms is [39]. The distance transform of the surface patch therefore specifies for each pixel what its distance is to the surface patch (in pixels). Determining d_s is therefore a simple table-lookup: It is the value of pixel at \vec{x}_s in the distance transform.

5.3.2 Incorporating observed data for continuity

Interpolating between the intersections with the different surfaces has proven to be insufficient, because discontinuities around the edges arise when the surface is not a perfect fit of the points in the region at these edges. The problem is again solved by interpolating, but this time only between the closest perimeter point of the closest surface patch and the point found through interpolation of the intersections. Remember that every pixel in the extended distance transform not only contains information about the distance to the closest point in the surface patch, but also exactly which point in the surface patch this is (Appendix C). This allows us to easily find the perimeter point that should be used for interpolation.

This interpolation is based on a logarithmic decay function. The influence of the perimeter point on the hypothesized point decays as the point to be hypothesized lies further away from the known data. The decay function can be seen in equation 5.4. At the transition from measured data to hypothesized data the perimeter point completely overrides the influence of the intersection, guaranteeing a smooth transition into the measured data. Given equations 5.3 and 5.4 the influence of the perimeter point at distances 0, 5, 10 and 20 pixels is 100%, 53%, 29% and 8% respectively.

$$\vec{x}_{hyp} = g(d_s) \vec{x}_{perim} + (1 - g(d_s)) \vec{x}_{inter} \quad (5.3)$$

$$g(d_s) = e^{(-x/8)} \quad (5.4)$$

\vec{x}_{hyp} is the final hypothesised point. \vec{x}_{perim} is the closest perimeter point of the closest surface. \vec{x}_{inter} is the point found in equation 5.1. $g(d_s)$ is the weighting function discussed in the previous paragraph.

5.3.3 Limiting the area in which the hypothesis is allowed

The further the distance to a surface patch, the less certain the surface patch can be extrapolated “on the basis of good continuation”. Therefore, the hypothesis is only allowed in a limited area surrounding the surface patch. Suppose the light gray area in the left figure in diagram 5.9 represents the surface patch. An area could be defined in which the hypothesis is allowed: the darker gray area in the same figure. If a surface patch contains many points, completion is allowed further away from the patch than would be the case with smaller patches. There is simply more evidence for this surface being correct, so the surface can be extrapolated further on the basis of good continuation.

The area in which the hypothesis is allowed is computed as in three steps, using diagram 5.9 as a visualisation:

1. **Compute relative area of surface patch.** First of all, the area of the surface patch relative to the total area of the image is computed. Simply $N/(rows * columns)$, in which N is number of pixels in the surface patch.
2. **Use function to compute relative area in which the hypothesis is allowed.** Using the function shown in diagram 5.9 (right) the desired (relative) area in which the hypothesis is allowed is then computed. An exponential function has been chosen because it puts a limit on the area in which completion can take place. In the function in figure 5.9 this limit is 100% of the image, which seems sensible.

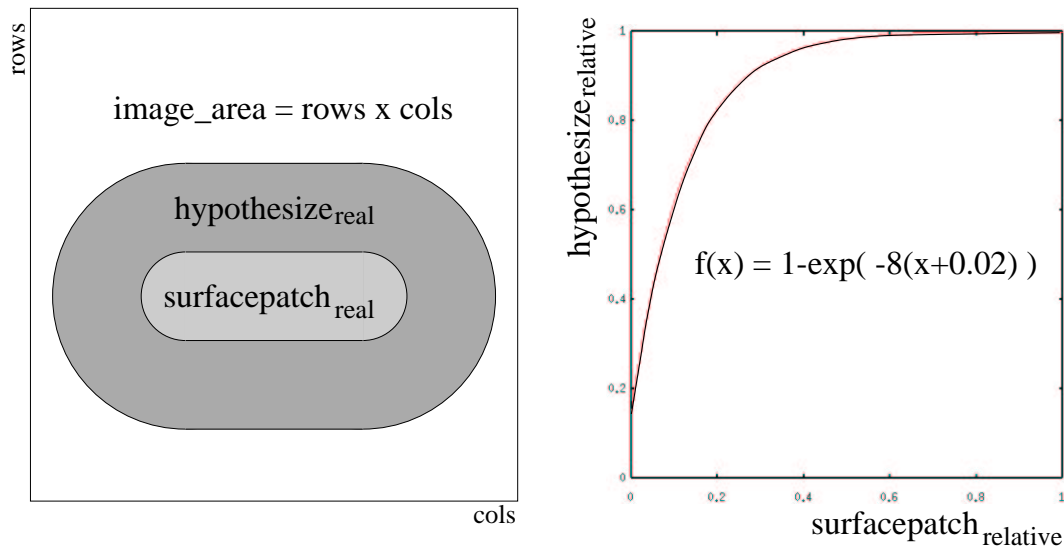


Figure 5.9: The area in which the hypothesis is allowed

3. **Compute real area in which the hypothesis is allowed.** The relative area acquired from this function is then multiplied with the number of pixels in the image to compute the real area N (in pixels) in which completion is allowed. The N pixels nearest to the observed patch are then completed.

5.4 Niche or real occlusion?

A surface hypothesis has been made! Before this hypothesized data can be added to the original image to fill in the missing data, one final test is needed. Adding the data is a severe change to the image, so we want to be very careful in applying it.

One issue that has not been dealt with so far is that the possibly occluded area for which the hypothesis has been made is not a real occlusion, but a niche. The difference between them can be seen in image 5.10. Real-life examples of niches in buildings are windows and doors. Adding the hypothesized data to these would certainly not make the 3D model look more realistic: it would cause all doors and windows to be removed, or cemented shut, so to speak.

The intuitive way to determine if a possible occlusion is actually a niche is to compare the measured data with the hypothesized data. As can be seen in image 5.10, a niche causes the hypothesized data to lie in front of the measured data, whereas in a true occlusion the hypothesized data lies behind the measured data.

The data are compared pixel-by-pixel. For each pixel, it is determined if it is further away from the sensor than the original range measured at that pixel. If this is the case it is worth completing; if it is not the case, it belongs to a niche and should not be completed. Considering completion of pixels individually is of course not very robust. For this reason a voting system is introduced.

Basically, a pixel votes for or against completion, depending on the issues described above. The area between the surfaces itself might contain several other surfaces, which do

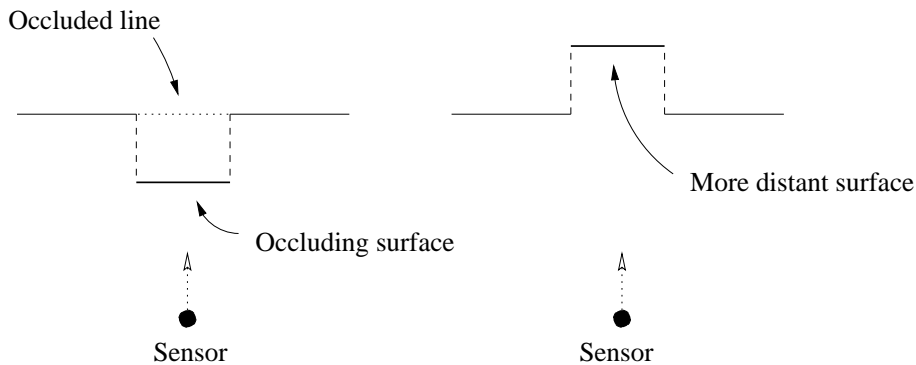


Figure 5.10: A *true* occlusion (left) and a niche (right)

not belong to the group of surfaces taking part in its completion. The votes of the pixels are polled per surface. If enough pixels are in favour of this completion, it is completed, otherwise it isn't. A high percentage of 90% was chosen to ensure that completion was justified.

5.5 Results

The algorithm has been tested on a database of 22 images. 17 of these were acquired using the K2T time-of-flight laser-scanner. Another 5 images were acquired using the REVERSA laser-striper in the Vision Lab. The images contained a total of 35 occlusions, taking human judgement as ground truth.

The first key result is that no incorrect completion occurs. Four regions that had been deemed possibly occluded were rejected for completion as they were correctly classified as niches. This is an important result, as completing data when this is not valid can introduce strange artifacts into the image. Adding data where it shouldn't be added will make the image look even more unrealistic than the original with its missing data!

Furthermore, 31 *true* occlusions were completed correctly, of which three are shown in the diagram below. In this diagram, the original *xyz*-images are in the left column, whereas the right column shows the same data with the hypothesized data added. More complicated images were also completed correctly, but these are not shown, as the results is not clearly visible on paper.

The first row shows a chair in front of a wall, the sensor having measured the image from the right. The surface completion algorithm fills in the missing wall, and yields the data shown on the second columns. In the next rows an image acquired by the REVERSA scanner (scanning from top of image) can be seen. The two cylinders are completed correctly. In the last row, a disc and a cylinder lying on top of a plane, can be seen from above. Completion takes place under both the disk and the cylinder. These are two examples of single region surface completion.

Four occlusions were completely missed. This was largely due to two connected planes being incorrectly classified as not coplanar (three cases). The segmentation algorithm did not segment the data well enough to allow a good plane-fit, so a better segmentation should solve this problem. One case involved two small coinciding surface patches that were quite

Original image

Completed image

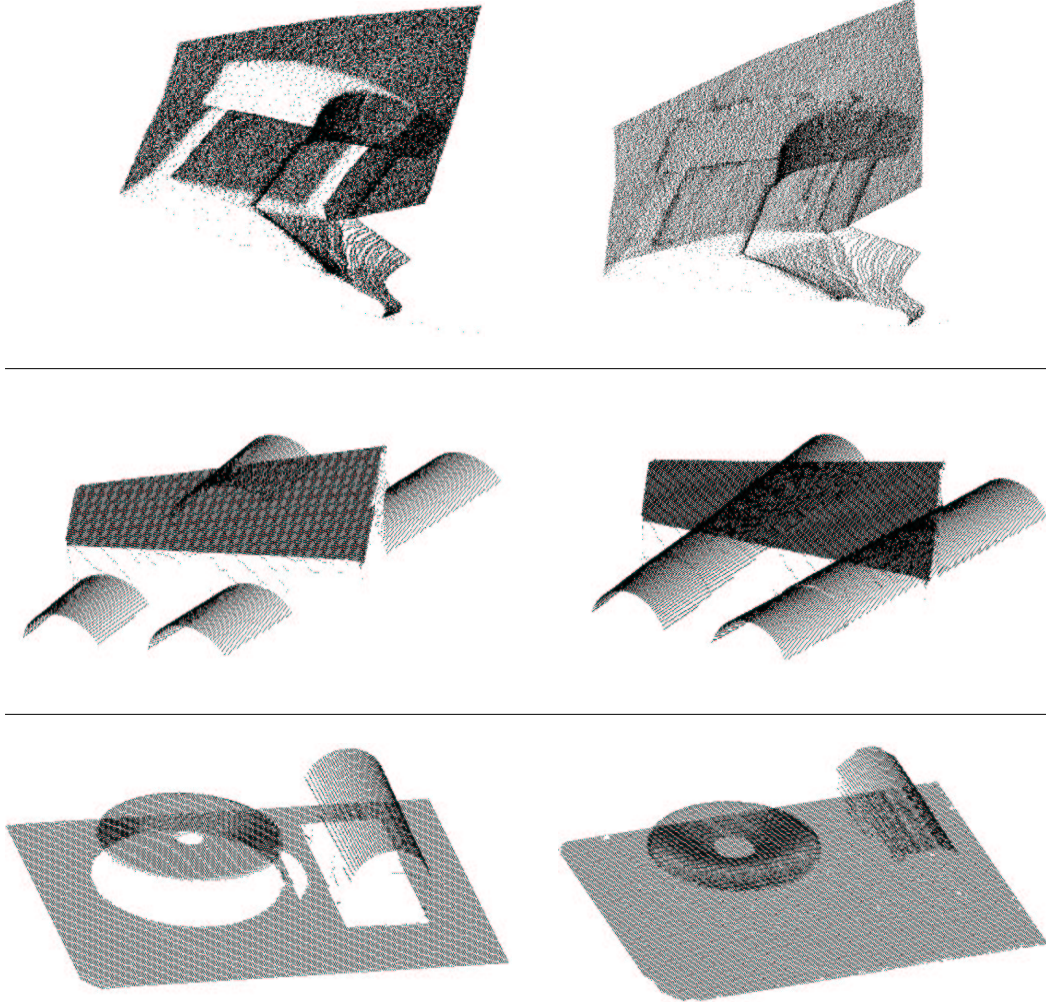


Figure 5.11: Some results of the surface completion algorithm

far apart, between which occlusion was taking place. Because the patches were small, the area in which the hypothesis was allowed was not very large. Because the regions were far apart, these small areas could not bridge the gap.

The completion took between 1 and 14 seconds on a 400MHz Sun workstation. The images were between 20000 and 250000 pixels in size. The time is roughly linear with image size.

5.6 Summary

The output of the segmentation algorithm (surface patches) is used by the surface completion algorithm to complete occluded surfaces. The first step was to locate areas in which occlusion might be taking place. Because there are different classes of occlusion, different methods were used to locate them. For each area that is deemed to be possibly occluded, a hypothesis

as to what would have been seen if occlusion had not been taken place is made. The method is based on intersecting the optical ray of the sensor with the geometrical surface of the surface patch. If multiple surface patches are involved, interpolation between the intersections is necessary. The hypothesized data is compared with the observed data. If the possible occlusion turns out to be a real occlusion instead of a niche, the data is added to the original incomplete data, yielding a surface-completed image.

Chapter 6

Texture Completion

6.1 Introduction

Although completing the occluded surfaces makes the 3D models more realistic (no more chair-shaped holes as in figure 1.2), they still lack intensity texture. For the moment the completed surfaces are merely points in 3D space without any information about the light intensity or color at that point as seen in the center image of diagram 6.1. This section presents a method to hypothesize what the intensity on the surface would have been if it had been visible, the last step in the diagram below.

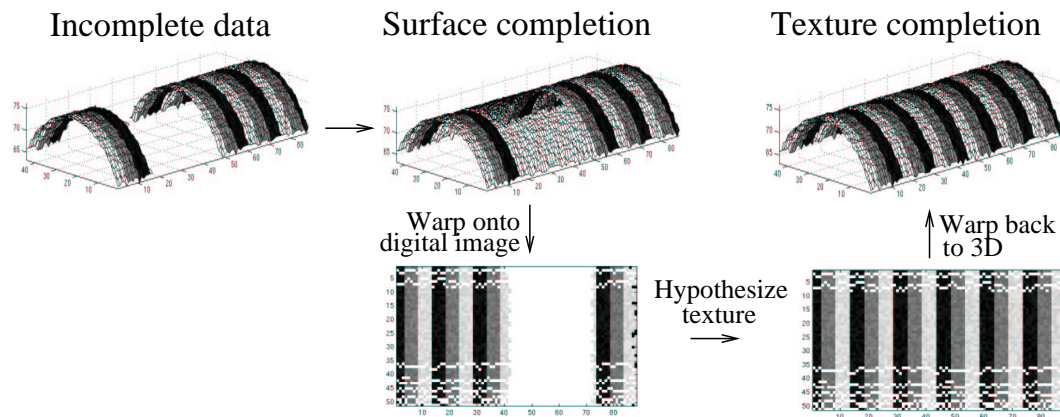


Figure 6.1: An overview: One step further.

The main idea is to warp the observed and completed points of the surface from 3D continuous space onto a 2D digital texture space (section 6.2). In this incomplete digital intensity image the intensity of the surface-completed points with unknown texture is hypothesized (section 6.3). The hypothesized intensity is then warped from the 2D digital image back onto the original 3D surface, as discussed in section 6.4.

6.2 Extracting a regularly sampled texture image

In this section the process of warping the 3D continuous surface onto a 2D digital image is presented. The reason this is done is that textures on planes, cylinders and spheres are usually flat textures projected onto the surface. Warping them back onto a 2D plane will recover this original structure. The digital image also allows for easier and more efficient processing than the unordered points in 3D space do. First the points of the original and completed surface patch are warped onto a 2D continuous texture space as described in section 6.2.1. Then the points in this space are aligned using a genetic algorithm (section 6.2.2). The last step is to sample the points on the 2D continuous space to acquire a 2D digital intensity image (section 6.2.3).

6.2.1 Warping the 3D surfaces onto the 2D texture space

Because the number of classes of surfaces is restricted to planes, cylinders and spheres, there will only be a need to warp these specific surfaces on the 2D texture space.

For planes this is relatively easy. The plane in 3D space can be transformed and rotated such that it lies in a flat 2D plane on the x and y axis. The points in the plane will then also approximately lie in this x,y plane. Simply removing the z component from the transformed and rotated points truly converts them into a 2-dimensional space. The first image of diagram 6.2 depicts this homographic warping.

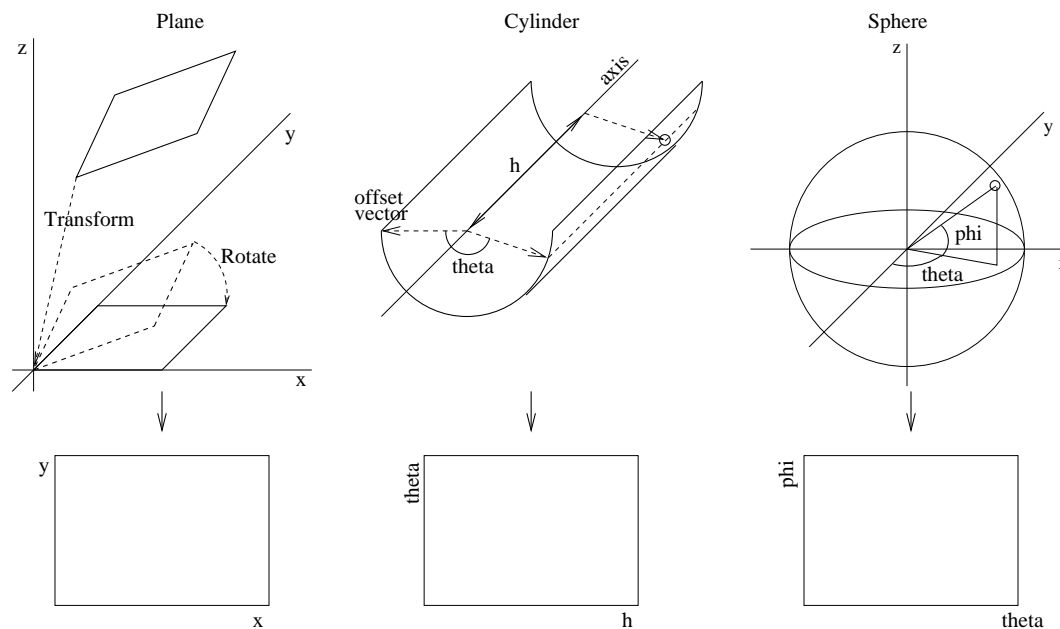


Figure 6.2: Warping surfaces onto a 2D continuous texture space

For the cylinders a h - θ surface parameterization is applied. A certain vector whose offset lies in a point on the axis of the cylinder is set as a reference. See the second image of diagram 6.2 for clarification. For each point on the cylinder, a vector through this point to a point on the axis can be constructed, with the requirement that the vector must be perpendicular to the axis. The distance between the reference point on the axis and the

current point on the axis is h , the angle between the reference vector and the current vector along the axis is θ .

For spheres, a similar θ - ϕ parameterization can be made. This time h is replaced by another angle. Both θ and ϕ are the angles between the vector from the center of the sphere to a point on the sphere, and a certain reference vector. The last image in diagram 6.2 shows this parameterization.

6.2.2 Aligning the 2D texture space using a bounding box

The warping from 3D to the 2D continuous texture space causes some practical problems: as the sampled 3D data points do not have a regular mapping to 2D, the data is usually not well aligned or evenly spaced in the continuous texture space. To recover a regularly sampled texture image, we first find a bounding box that incorporates most of the points. Points inside this box will quantized and incorporated in the regularly sampled texture image.

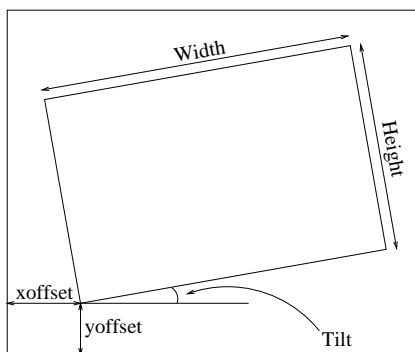


Figure 6.3: Finding an appropriate bounding box using a genetic algorithm

The search space of the problem of finding a bounding box that incorporates a representative number of points is sufficiently complex that a closed form solution is not readily available. For this reason a genetic algorithm was used. For those with no experience in genetic algorithms, tutorial [28] provides a lucid explanation.

The genetic algorithm optimises five parameters, all shown in diagram 6.3. Every gene is a continuous variable which represents one of these five parameters. A chromosome therefore has five genes. The number of chromosomes in each population is 10, and the maximum number of generations is 200. The mutation rate is 15%, and the crossing-over rate is 30%. During each generation the chromosome that is the fittest is automatically placed in the next generation.

A penalty function determines the fitness of a chromosome by computing the distance from each point to the circumference of the bounding box. It is also determined if this point is inside the box or outside of it. The penalty function is then computed as shown in equation (6.1). In this equation, $\sum dist(p_{in})$ is the total of all the distances from points inside the box. Similar for points outside of the box p_{out} . The goal is to minimize the value of this penalty function.

$$penalty = 1.0 * \sum dist(p_{in}) + 5.0 * \sum dist(p_{out}) \quad (6.1)$$

This penalty function causes the bounding box to be configured in such a way that the total sum of the distances from the points to the box is minimized, with a strong bias against points lying outside the box. The box will try to incorporate many points, without becoming too large. This bounding box is then taken to be the perimeter of the regularly sampled texture image. All the points are translated and rotated accordingly into this new box, as shown in diagram 6.4. This gives a rectangular, axis aligned, surface texture space, although the observed intensity points are not uniformly spaced.

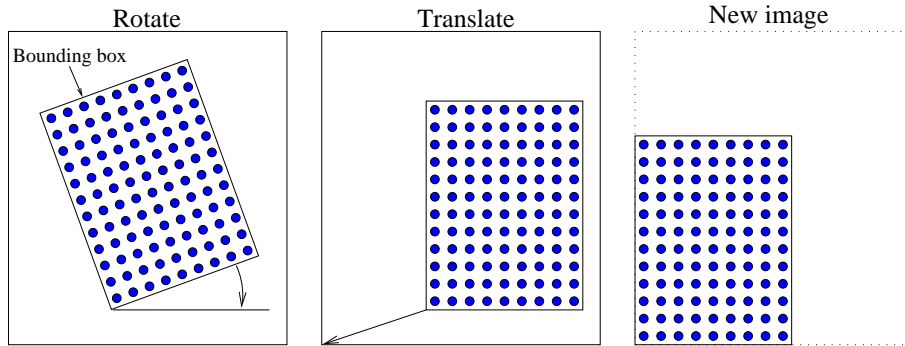


Figure 6.4: Rotating and translating the data in the bounding box

6.2.3 Sampling the 2D texture space

The texture completion algorithm used below requires a uniform grid, so we estimate the regular grid points based on the observed points. The points in the regularly sampled texture image are estimated using the following algorithm. In the x and y direction, different sampling rates are tried (ranging from 10 to 200, with 0.5% increments). For all the intermediate sampling rates, the points are histogrammed into the different rows and columns. The sampling rates in x and y direction with the lowest standard deviation for points per row/column is chosen to be the sampling rate that will be used. This ensures an even distribution of the points over the rows and columns.

The final result of warping the 3D continuous space onto the digital intensity image can be seen in diagram 6.5.

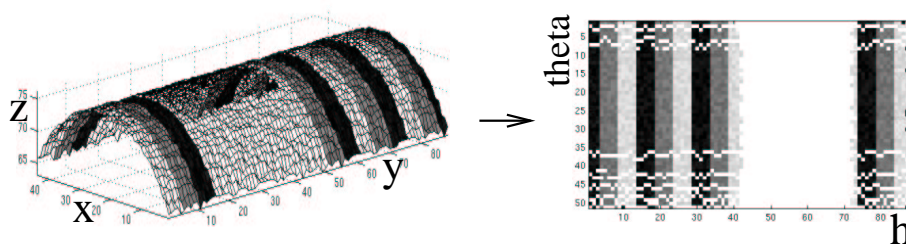


Figure 6.5: The result of warping

6.3 Hypothesizing the unobserved texture

The intensity of the unobserved surface-completed pixels is hypothesised in the digital intensity image using a template matching method¹. The texture in the image is modeled as a Markov Random Field, which assumes that the probability distribution of brightness values for a pixel given the brightness values of its spatial neighbourhood is independent of the rest of the image [21]. The neighbourhood of the pixel is specified by an $M \times N$ window.

6.3.1 Completing a single pixel

The main idea is to compare the surroundings of a pixel with unknown intensity (one that has been surface-completed) with those of pixels who are known (actually observed pixels). It can be expected that an unknown pixel will have approximately the same value as a known pixel with the same surroundings, given the independence assumption of the Markov Random Field.

Suppose one pixel p_u in image I is in need of completion. To achieve completion, first of all the $M \times N$ window surrounding this pixel is determined. This window will be called $w(p_u)$. The main question now is what the most likely value for pixel p_u is, given its surroundings $w(p_u)$. In other words

$$\text{maximize } P(p_u | w(p_u))$$

In order to do this, a conditional probability distribution for $P(p_u | w(p_u))$ is approximated. If this distribution is available, the maximum value can easily be found by computing the mode. To acquire this distribution, all likely values of p_u are accumulated. This is done by comparing the surroundings of the unknown pixel $w(p_u)$ with those of observed pixels $w(p_k)$ in the set of $M \times N$ windows surrounding the known pixels. The set of windows of which the center pixel is known is defined by

$$W_k = \{w(p_k) \in I, \text{ with value of } p_k \text{ known}\}$$

The next step is to compare the unknown window $w(p_u)$ with all known windows in set W_k , which of course are of similar $M \times N$ size. If the difference between $w(p_u)$ and $w(p_k)$ (from W_k) is zero ($d(w(p_u), w(p_k)) = 0$), p_k is a likely value for p_u . Therefore, the value of p_k is added to the set of likely values of p_u , called $L(p_u)$. The set $L(p_u)$ is therefore defined as

$$L(p_u) = \{p_k, \text{ with } d(w(p_u), w(p_k)) = 0, \text{ and } w(p_k) \in W_k\}$$

Because it rarely occurs that the difference between all the pixels in two windows is zero, a threshold is implemented, allowing all values for which $d(w(p_u), w(p_k)) \leq \epsilon$ holds to be included in the set of likely values of p_u . The difference measure between two windows $d(w_1, w_2)$ is simply the normalized sum of squared differences between all the values of the pixels which are known in both windows.

L_u , the set of all likely values for the unknown pixel, has been computed. The conditional probability distribution of p_u can now be estimated with a histogram of all values in set $L(p_u)$. In this estimated probability distribution the x-axis represents the value of p_u ,

¹After having implemented the method in Edinburgh and presenting this work in Lisbon, Josè Santos-Victor pointed out an article which uses a very similar method [21]. Although the implementation was my own, I have used the theory discussed in the article to give it a better theoretical foundation.

and the y-axis represents $P(p_u|w(p_u))$. Determining the mode of this histogram yields the maximum value for $P(p_u|w(p_u))$. So, given surroundings $w(p_u)$, p_u is most likely to be the value of this mode. Therefore, the value of this unknown pixel is simply set to the value of the mode. In the example graph shown in diagram 6.6 this value would be 155.

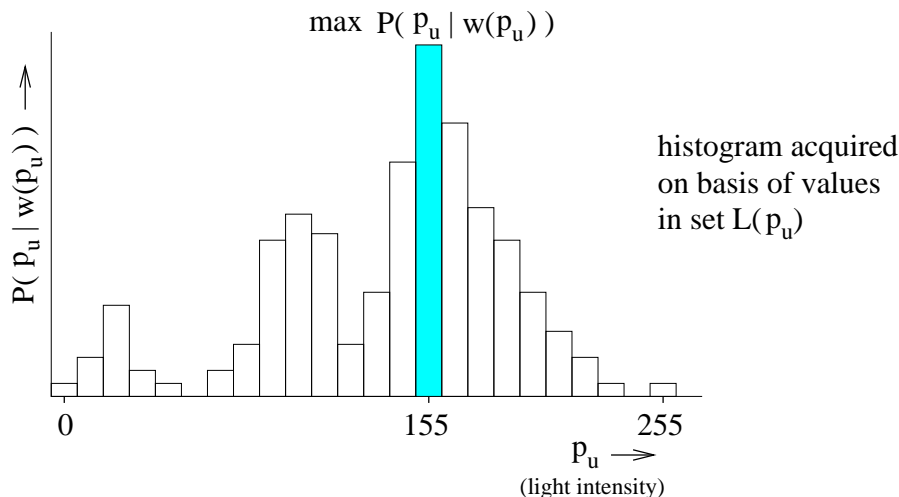


Figure 6.6: Determining $\max(P(p_u|w(p_u)))$, the most likely value for p_u , given $w(p_u)$

6.3.2 Completing all pixels in a region

Being able to complete one pixel is not sufficient to complete entire regions in an image. If a region is large, most pixels will have no known pixels in their surrounding window, so the method does not apply as the difference measure cannot be evaluated. To cope with this problem the order in which unknown pixels are completed will have to be specified: first those that have sufficient information in their surroundings, the others at a later time.

Because there is not one unknown pixel, but a whole set, a new set W_u is defined, which contains $M \times N$ windows in which the center pixel is unknown

$$W_u = \{w(p_u) \in I, \text{ with value of } p_u \text{ unknown}\}$$

Initially, this set will be the complement of set W_k , which contains all windows whose center pixel is known. Of course, the goal is to somehow convert all unknown pixels into known pixels, essentially removing all the windows from W_u . This will be done pixel by pixel, using the method to complete a single pixel described in the previous section.

The question now is which unknown center pixels of windows in the unknown set W_u should be completed first. First of all, it is obvious that pixels that have no known pixels in their surrounding window ($w(p_u) = \emptyset$) cannot be completed. How should their surroundings be compared with the known windows in set W_k ? It makes more sense to first complete unknown pixels p_u that have a lot of information in their surroundings: many pixels in $w(p_u)$ should be known. Suppose a threshold of 75% is chosen. This means that an unknown pixel can only be completed if 75% of the pixels in its window are known.

This threshold will determine the order in which the unknown pixels are completed. First of all, all windows in the unknown set W_u that meet the 75% requirement are removed from

W_u and completed immediately. The center pixels of these windows are simply completed using the single-pixel-completion algorithm discussed in the previous section. Note that the completion of these pixels yields new information, because the value of some previously unknown pixels has been completed. All the windows in W_u are updated with this new information. Pixels that were previously unknown in the windows of W_u might now have been completed. This means that other windows in W_u might now meet the 75% requirement, so the process is repeated. This loop continues until even the last pixel has enough information in its surrounding window to be completed. The process proceeds until the set of windows with unknown center pixels is empty.

This algorithm basically starts completing the texture at the perimeter, where it is likely that pixels will have many neighbouring pixels that are known. The perimeter of the unknown area is ever changing because more information becomes available. It becomes smaller and smaller as the algorithm progresses. Essentially, the texture grows into the unknown region until it is entirely filled with completed texture.

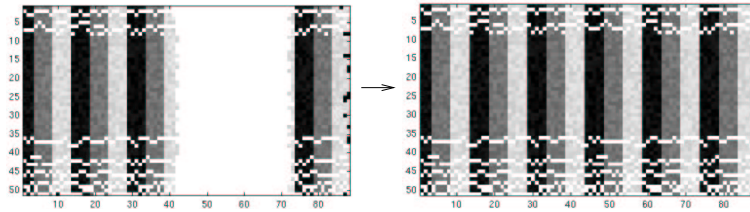


Figure 6.7: The completed digital image

6.3.3 Dynamic thresholding

In the previous section, the threshold on how many pixels must be known in a pixels' surroundings before it can be completed was fixed at 75%. The actual implementation is more sophisticated, as it varies in time. Initially, the threshold is set such that all the pixels in the surrounding window must be known. This is $M \times N - 1$, which is a very high threshold! If none of the unknown pixels have enough information in their surrounding window to meet the threshold, the loop will never terminate. Therefore, if this happens, the threshold is simply lowered by one pixel. Some windows in W_u will surely lie above the new threshold (or at least after the threshold has been lowered several times consecutively), so more pixels are completed, again yielding more information for the completion of other pixels: The loop continues!

6.3.4 Pseudo-code

To show all the procedures described in this section, as well as to demonstrate the interactions between them, the entire algorithm is presented in pseudo-code on the next page.

```

function complete_intensity(image I, int M, int N)
    // Initialize sets of windows with known and unknown center pixels.
    for all pixels  $p$  in image  $I$ 
        if ( $p$  is unknown) add  $w(p)$  to  $W_u$ 
        if ( $p$  is known) add  $w(p)$  to  $W_k$ 
    end
    // Initialize threshold and start the completion loop.
    threshold =  $M \times N - 1$ 
    while ( $W_u \neq \emptyset$ )
        for all windows  $w_u$  in  $W_u$ 
            if (#pixels known in  $w_u \geq$  threshold)
                // Enough information in surroundings. Complete one pixel.
                determine  $p_u$  such that  $\max(P(p_u|w(p_u)))$  given  $W_k$ 
                update  $W_u$  with  $p_u$ 
            end
        end
        if (no new pixels completed)
            // No window met the requirement, so make it more lenient.
            threshold = threshold - 1
        end
    end
end
end

```

6.4 Warping the completion onto the original image

The points that had been hypothesized in the surface completion algorithm have now received an intensity value. Unfortunately, this value is stored in the 2D digital texture space, not in the original continuous 3D space. Warping from one to the other is trivial in this case, as the point data-structure (see pseudo-code below) which has been used throughout processing contains information about the location in all the different spaces the point has been warped to and from. The corresponding intensity can therefore easily be warped to any of the spaces, including the original 3D space.

```

// Multi-point datatype
structure multi_point {
    // Locations in different spaces
    double x3,y3,z3; // Original location of point in 3D space
    double x2,y2;    // Location of point in 2D plane
    double i,j;      // Pixel location in digital image

    // Observed or hypothesized intensity value
    double intensity;
}

```

6.5 Results

In diagram 6.8 the result of applying both the surface and texture completion algorithms is shown. The left image in this diagram shows the original data (measured from above): a plane, occluded by two cylinders. The center image shows the (texture-less) result of surface completion (the original cylinders have been removed for clarity). Applying the texture completion yields the final results shown on the right.

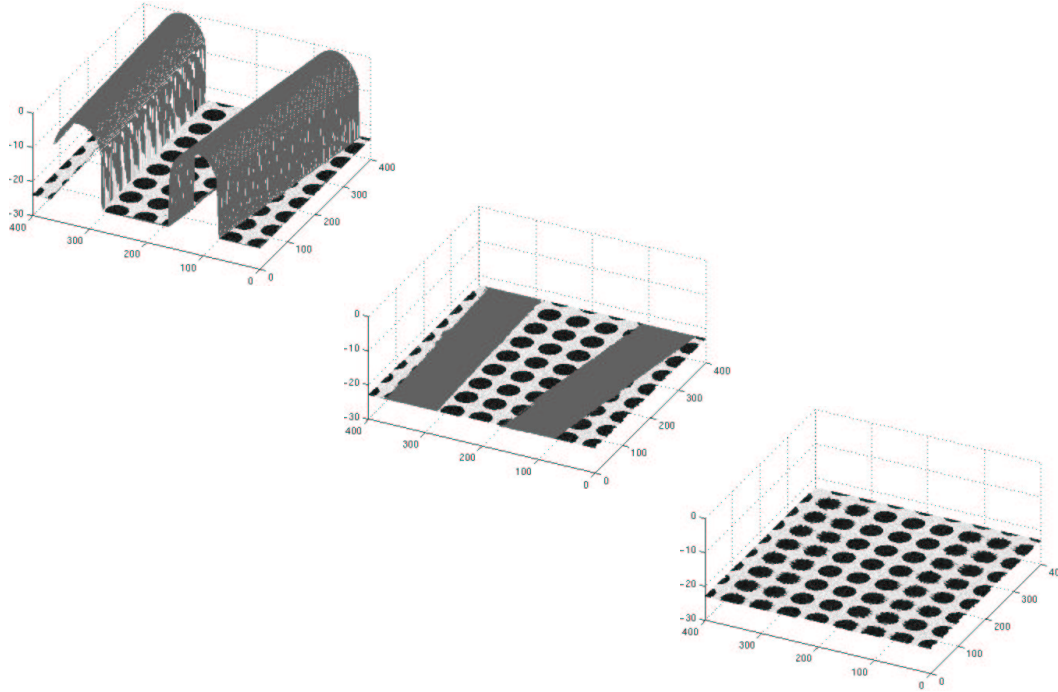


Figure 6.8: A result of both completions

6.6 Proposed improvements

The surface completion algorithm has gone through a number of revisions, guided by advice and comment from colleagues, reviewers and fellow students. It appears to be quite robust and efficient, and not much will be changed in the future.

The texture completion algorithm is far from perfect though. The idea of warping the 3D continuous data onto a digital image seems to be a good approach, but the actual algorithm for completing the texture in this digital image must be improved. The biggest objection is that human intervention is needed to specify the size of the window. If not the results can be poor for repetitive textures. In this section some new methods are proposed to improve the algorithm. Future work will be aimed at implementing and evaluating these methods to make the program usable for the CAMERA-group.

6.6.1 Automatic window selection

A problem that was also encountered by Efros and Leung [21] is that the results are very realistic if the size of the $M \times N$ window is chosen correctly. If not, the results are reasonable or poor, at least for images with repetitive patterns. The unit of the repeated texture element (or texel) seems to be an important factor. If the image of Marilyn Monroe is the unit texel, the best results are acquired if the $M \times N$ window incorporates exactly one Marilyn Monroe texel. The window can now be set manually to allow this, but automatic window selection would of course be much better: an autonomous algorithm is required.

One method of doing this is by computing the autocorrelation of the image in both x and y directions. This might yield the period of the a repetitive pattern. The values of this periods (in x and y) are likely to be good estimators for the size of the texels. M and N can therefore be set to the period in x and y direction respectively to yield the same results as are now acquired manually. This method will be implemented and evaluated.

6.6.2 Proximity weighting on the set of known windows

The algorithm described above puts equal weights on all windows in the set W_k . Results might be better if a positive bias is placed on windows in W_k that are close to the pixel that is currently being completed. If there is a overall change in the texture across the digital image (e.g. due to shading), it is better to use samples that are close to the area of interest. This method is easily implemented, and will be soon.

6.6.3 Proximity weighting in the comparison of windows

The same principles described in section 6.6.2 also apply on a more local scale. When comparing two $M \times N$ windows, a positive bias could be placed on pixels in the window that are close to the center pixel, the one currently being completed. A straight-forward method of doing this, as is done in [21], is by multiplying the sum of squared differences with a two-dimensional Gaussian kernel.

6.7 Summary

In the previous chapter methods with which occluded surfaces could be completed was presented. Because texture is still lacking on these newly completed surfaces, the texture completion algorithm described in this chapter is employed. The continuous 3D data on the surface patch is warped onto a 2D plane using geometrical translations. A genetic algorithm is then employed to find a suitable bounding box. After acquiring the new plane in the bounding box, the point in the plane are sampled to acquire a digital image. The texture in this digital image is completed by comparing the surroundings of known pixels with those of unknown pixels, and completing accordingly. The texture-completed digital image is then warped onto the original 3D data to acquire the final surface-and-texture-completed image.

Chapter 7

Conclusions and Future Work

7.1 Conclusive summary

Reconstructing 3D models of buildings from 2D and 2.5D data, such as the CAMERA-project does, is problematic for several reasons, one of them being occlusion. Occlusion can lead to missing data, which make a 3D model look unrealistic. There is a need to fill in the missing data.

Humans do not have these problems, as they employ perceptual completion. There is much debate amongst the cognitive scientists, visual scientists, and philosophers as to how perceptual completion should be explained. Visual scientists assume the missing data is actively filled in through neural activation, whereas the philosophers believe in a more symbolic principle in which missing data is ignored and simply labeled. How exactly perceptual completion works is not yet well enough understood to be used as a model for our program, unfortunately.

Therefore more traditional computer vision methods are used. Range and intensity images are acquired by range sensors, after which the data is segmented. Points with similar characteristics are clusters in a region growing algorithm. Surfaces are fitted to the points in these clusters, after which the surface is grown. The results is a group of surface patches: continuous regions in the image represented by a geometrical surface.

The segmented data is used by the surface completion algorithm, which locates different classes of possible occlusions and hypothesizes data at these locations: What would have been seen if occlusion had not been taken place? If the possible occlusion turns out to be a real occlusion (instead of a niche), the data is added to the original incomplete data, yielding a surface-completed image.

Because texture is still lacking on our newly completed surfaces, a texture completion algorithm is employed. The data on the surface patch is warped onto a digital image using geometrical translations and a genetic algorithm. The texture in this digital image is completed by comparing the surroundings of known pixels with those of unknown pixels, and completing accordingly. The texture-completed digital image is then warped onto the original 3D data to acquire the final surface-and-texture-completed image.

7.2 Future work

This section will discuss some of the future work that shall be done on this research topic. Section 6.6 has already discussed some improvements on the texture algorithm. These will not be repeated here. Finally, appendix D gives a list of all the publications, presentations and applications that have resulted from this research.

7.2.1 The single-transection occlusion

As mentioned in section 5.2.4, the single-transection class of occlusions can not yet be detected. This was due to lack of time. Research on this issue will continue, as Bob Fisher has employed a PhD student, Umberto Castellini, to investigate single-transection occlusions for four months.

7.2.2 More general surfaces

A first improvement over the current segmentation algorithm would be to fit more surfaces than just planes, cylinders, and spheres. Whereas this simplification has caused no real problems in the data-set used, it can be expected that for instance modern buildings, with modern design and construction techniques, will incorporate more exotic surfaces, maybe even free-form quadrics. Cones and elliptical cylinders are also encountered in industrial sites, so incorporating them would be a first useful step.

Incorporating new surfaces is no problem for the segmentation algorithm. Petko's code actually already allows the fitting of general quadrics. A slightly larger technical problem arises in the surface completion algorithm. For each type of surface, it should be specified when they coincide, which is very difficult for quadrics for instance. Computing the intersection between a ray and the new surface should also be provided for the surface hypothesis method. This should not pose any theoretical problems. A more serious problem occurs in texture completion. It can be expected that mapping all these different surfaces onto a 2D plane might not lead to very regularly spaced data, necessary for a dense digital image. The texture completion should be altered to cope with this problem.

As mentioned however, the three surfaces used cover most of the possible surfaces used in past and present architecture. Only extreme and very modern designs will pose a problem. A cynic might respond that it is to be expected that buildings of the future will have a virtual 3D model made before the construction starts anyway.

7.2.3 Knowledge of the world

The whole completion program would, like most computer vision programs, probably improve greatly if more knowledge of the world could be incorporated. Hypothesizing a wall behind a chair is much easier if one known what a chair is, what a wall is, and that walls do not usually contain chair-shaped holes. Writing a program that can recognize chairs, walls, floors, and can distinguish between parts of people and partially visible people is hard work though. It is more a goal of the entire vision community than that of an MSc. research.

7.2.4 Symbiosis with view-planning algorithms

As discussed briefly in section 1.4, a simple solution for retrieving occluded data is to make more scans of the environment. This solution brings us into an area of research called *view planning*. When acquiring 3D data it is unlikely that a sensor will collect all the necessary¹ information contained in the real-world scene by one single scan, so usually more scans are needed. View planning tries to find the position in space from which an imaging sensor can inspect a scene or object optimally. The ultimate goal is to acquire as much data as possible with a minimum number of scans. That is, to maximise the information/cost trade-off.

Miguel Sanchiz has done some excellent work on this topic [43]. His next-best-view algorithm computes at which following location the information gain can be expected to be highest, given what is already known and unknown. A simple example is shown in image 7.1. From the original position, the camera with next-best view-planner registers what it knows, and estimates what it doesn't know. Given this known and unknown area, the camera then tries to maximize the area that can become known in the next view. In the diagram, the camera thinks it can gain the most information in the other corner, so it is placed there. This goes on until the camera is satisfied it has sufficient knowledge about the scene.

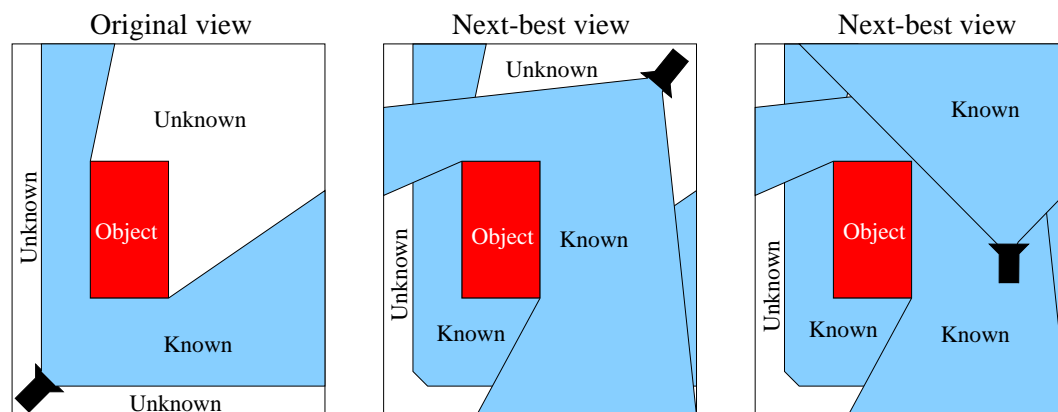


Figure 7.1: Next-best viewplanning

For a simple scene as in diagram 7.1, 4 or 5 views might be enough to have viewed the entire scene. When Miguel's program was tested on more life-like scenes, hundreds of scans were needed to acquire sufficient knowledge about the scene. I have personally spoken to the brave men who have taken the Bornholm church images (CAMERA researchers in Stockholm). The weather was very cold in Denmark at the time, the sensor was only just working, and needed to be shipped to England for a demonstration within 4 hours. Other CAMERA researchers at the Joint Research Centre in Ispra, Italy, had better weather when scanning the *Sala dello Scrutinio* at Venice Doges' Palace. The scanner worked as well, but they were only allowed to scan this site for a very short time (I believe about 2 hours), before it had to be opened to the public again. This goes to show that hundreds of scans are not practical in daily research. Furthermore, today's scanners are high-tech and heavy equipment, not designed for constant relocation even within one room. This has been one

¹Again, by necessary we mean: necessary for making a complete and accurate (also see footnote to section 1.2) 3D model

of the motivations in writing the completion algorithm: if more scanning is too much work, a good guess is a good alternative.

The guess is sometimes too hard to make though. If only one side of the Bornholm church were scanned, the completion program cannot be expected to complete the other side. There is simply not enough data to go on, so another scan would be needed. Miguel's program could compute from which location this next scan could best be taken.

This example shows how beautifully the two programs complement each other. The completion algorithm could fill in bits and pieces, or maybe even larger areas if enough information is available in the surroundings. This would relieve the scanner from having to scan behind every chair in the room: a good guess has already been made as to what is behind the chair. If the completion algorithm does not have enough data to go on, the next-best-view algorithm could take over, and acquire some more scans.

A symbiosis between the two would probably result in an algorithm that works as follows.

```
function investigate_scene {
  while (not enough known to make complete 3D model of scene) {
    make scan with sensor                // scanners' task
    complete as many occlusions as possible // Freek's task
    merge old data with new data          // registration
    compute next-best-view and place sensor there // Miguel's task
  }
  return data (observed and completed alike)
}
```

With this algorithm, it might occur that the next-best-view algorithm places the sensor in a position such that it measures data that has already been completed. In this case, the completed data, which is merely a hypothesis, should of course be replaced by the known, observed data. An interesting thought is that the completion algorithm could then compare the completed data with the newly observed data. If a learning algorithm was implemented, the completion algorithm could then learn from its mistakes; but this taking it very far into the future indeed.

Appendix A

Derivative estimation

The goal is to fit a function to a set of points. The functions will be discrete orthogonal polynomials, and the set of points is defined by a $N \times N$ window (N is odd). Each point in the window is associated with a position (u, v) from the set $U \times U$ in which:

$$U = \{-M, \dots, -1, 0, 1, \dots, M\} \quad \text{with} \quad M = \frac{N-1}{2} \quad (\text{A.1})$$

The following discrete orthogonal polynomials provide local biquadratic surface fitting capability:

$$\phi_0(u) = 1, \quad \phi_1(u) = u, \quad \phi_2(u) = (u^2 - M(M+1)/3) \quad (\text{A.2})$$

A corresponding set of functions $b_i(u)$ are the normalised versions of the orthogonal polynomials $\phi_i(u)$.

$$b_i(u) = \frac{\phi_i(u)}{P_i(M)} \quad \text{with} \quad P_i(M) = \sum_u \phi_i^2(u) \quad (\text{A.3})$$

The three normalisation constants are given by:

$$P_0(M) = N$$

$$P_1(M) = \frac{2}{3}M^3 + M^2 + \frac{1}{3}M \quad (\text{A.4})$$

$$P_2(M) = \frac{8}{45}M^5 + \frac{4}{9}M^4 + \frac{2}{9}M^3 - \frac{1}{9}M^2 + \frac{1}{15}M$$

A surface function estimate $\hat{f}(u, v)$ is obtained in the form

$$\hat{f}(u, v) = \sum_{i+j \leq 2} a_{ij} \phi_i(u) \phi_j(v) \quad (\text{A.5})$$

that minimises the total square error term

$$\epsilon^2 = \sum_{(u,v) \in U^2} (f(u, v) - \hat{f}(u, v))^2 \quad (\text{A.6})$$

The solution for the unknown coefficients is given by

$$a_{ij} = \sum_{(u,v) \in U^2} f(u, v) b_i(u) b_j(v) \quad (\text{A.7})$$

The first and second order partial derivative estimates for the centre point in the window are then given by:

$$f_u = a_{10} \quad f_v = a_{01} \quad f_{uv} = a_{11} \quad f_{uu} = 2a_{20} \quad f_{vv} = 2a_{02} \quad (\text{A.8})$$

These estimates are plugged directly into the equations described in Appendix B to obtain the mean and Gaussian curvature

Note: This appendix is almost a literal transcription of pages 88-89 of Besl's *Surfaces in Range Image Understanding* [3]. Interested reader might want to look at [5] or [25] for a more extensive explanation.

Appendix B

Direct curvature computation

Instead of computing the principal curvatures to derive the mean and Gaussian curvatures, the latter have been computed directly using the first and second order derivatives. This appendix explains exactly how.

First of all, the surface is described by:

$$\vec{x}(u, v) = [u \ v \ f(u, v)] \quad (\text{B.1})$$

The derivatives can be defined as:

$$\begin{aligned} \vec{x}_u &= [1 \ 0 \ f_u] \\ \vec{x}_v &= [0 \ 1 \ f_v] \\ \vec{x}_{uu} &= [0 \ 0 \ f_{uu}] \\ \vec{x}_{vv} &= [0 \ 0 \ f_{vv}] \\ \vec{x}_{uv} &= [0 \ 0 \ f_{uv}] \end{aligned} \quad (\text{B.2})$$

Note that computing the derivatives (Appendix A) yields exactly the f -terms in definitions B.2; the vector forms are never used. The mean and Gaussian curvature can be computed directly from the f -terms using the following formulae:

$$K = \frac{f_{uu}f_{vv} - f_{uv}^2}{(1 + f_u^2 + f_v^2)^2} \quad (\text{B.3})$$

$$H = \frac{1}{2} \frac{(1 + f_v^2)f_{uu} + (1 + f_u^2)f_{vv} - 2f_u f_v f_{uv}}{(1 + f_u^2 + f_v^2)^{3/2}} \quad (\text{B.4})$$

Appendix C

The extended Euclidean distance transform

The distance transform (DT) is an image in which the value of each pixel is the distance to the nearest pixel from a set of objects. Which pixels belong to the set of objects is defined by a binary image, in which the 'ones' represent pixels belonging to the set of objects.

There are many methods of representing this distance. Common and easily computed measures are for instance the city-block (Manhattan) [42] and Chamfer metrics [6]. For our purposes, these approximate distances are not accurate enough. The real or Euclidean distance is needed. In figure C.1 a binary image and its Euclidean distance transform are shown.

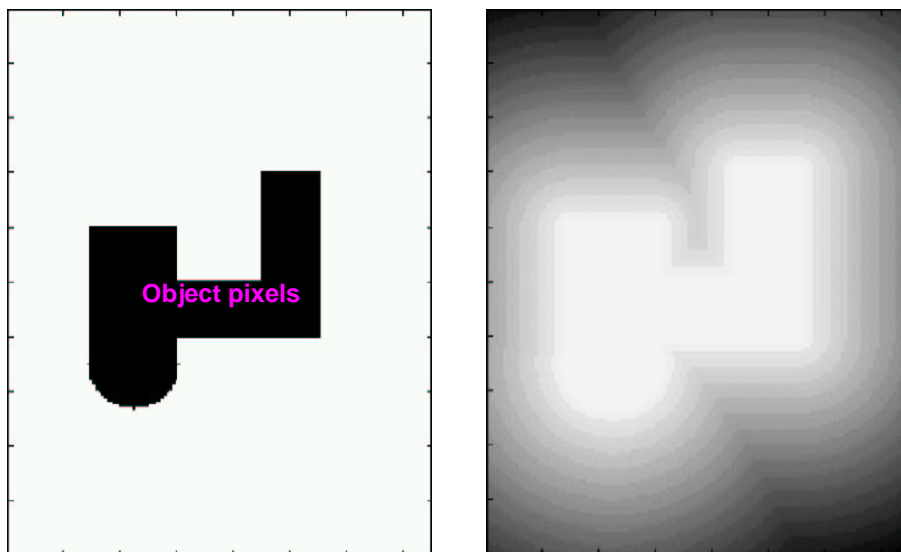


Figure C.1: A binary image and its distance transform

There are many methods for computing the Euclidean distance transform (EDT). An older version is [51] while a very recent algorithm described in [14] is so efficient that it approximates the theoretical optimum for such algorithms.

One of the more simple algorithms for computing distance transforms has been implemented two reasons. First of all, our distance transform will not have to be computed for all the pixels in the image (for reasons explained in section 5.3.3), so efficiency is not of the highest importance. Because this dissertation is about reconstruction and not the intricacies of computing distance transforms, the simple but slower algorithm was the sensible option.

Furthermore, current algorithms encode information about the distance to the nearest object pixel (NOP), but not about the actual 2D location of this pixel. This information is needed in later computations. For these reasons the algorithm also stores information about the closest pixel in the area. The distance to the area can easily be computed using this pixel, but for sake of efficiency it is computed once and stored as well. Because this version of the distance transform contains more information than a standard EDT it will be referred to as an *extended* Euclidean distance transform (EEDT).

The EEDT is computed using methods similar to those described in [51], and is based on a simple growing algorithm. The main interest of the EEDT is what it is and how it can be used, not how it is computed. For this reason (as well as not wanting to lose the bigger picture) the exact computational methods are not discussed here. Interested readers are advised to read [51], as the methods described there are very similar.

Appendix D

Publications, presentations, applications

It might be interesting and relevant to mention that this MSc. research has led to some publications, presentations and applications.

First of all the research has led to the submission and publication of some articles written together with Bob Fisher. Hopefully, these will not be the last, as I am currently continuing this research for CAMERA with José Santos-Victor at the *Instituto de Sistemas e Robótica* in Lisbon, Portugal.

- **Published:** *Reconstruction of surfaces behind occlusions in range images* [45]
Conference on 3D digital imaging and modeling, 3DIM01, Quebec, Canada
The article has been published in the proceedings of the conference.
- **Submitted:** *Shape and texture hypothesis of 3D occluded surfaces* [44]
Conference on Computer Vision and Pattern Recognition, CVPR2001, Hawaii, US
If this paper has been accepted by the reviewers will be made known 10 August 2001.
- **To submit:** *Completion of occluded surfaces*
Transactions on Pattern Analysis and Machine Intelligence, PAMI, journal.
Hopefully, the new texture completion methods that have been proposed in section 6.6 can be incorporated in this paper. Maybe some material on perceptual completion can also be added, as computer vision often seems to neglect the interesting cognitive issues involved.

Furthermore, a list of all the presentations of this work is given:

- **25 January 2001** A presentation at a CAMERA meeting at the Joint Research Center in Ispra, Italy.
- **2 June 2001** A poster presentation at the 3DIM01 conference in Quebec, Canada. My colleague Craig Robertson has been kind enough to represent me there.
- **4 June 2001** As part of the course Artificial Intelligence, I had to present and defend my work at a so-called colloquium. This was done at the Rijksuniversiteit Groningen in Groningen, the Netherlands.

- **25 June 2001** A presentation at a CAMERA meeting at Trinity College in Dublin, Ireland.
- **4 July 2001** A seminar in the Vislab Seminar series at the Instituto de Sistemas e Robótica in Lisbon, Portugal.
- **12 December 2001** If the article mentioned in the previous paragraph is accepted, I will present it at the CVPR conference in Hawaii. One can imagine that my hope for acceptance of the article is not purely professional.

Last, but not least: UK Robotics (now RTS Advanced Robotics) has shown interest in using my code for their Light Form Modeller (LFM). RTS is a company in Manchester, and the LFM is an application that provides a user interface for navigation through 3D models. RTS mainly sells this LFM to industrial companies. One of the problems they are facing is that industrial scenes contain many pipes running alongside walls, as well as each other. This causes many occlusions. Pipes and walls appear to be *split* in several pieces due to the missing data. To the average user, who cannot be expected to know about the problem of occlusion, this looks quite unrealistic. This is exactly the problem I faced about a year ago! It is satisfying to know that the research might eventually lead to a commercial product, and not just publications, which are useful on a more theoretical level.

Appendix E

Further reading

For those interested in further reading some references to some interesting articles are given.

After finishing this research, many new texture synthesis algorithms were published: three at SIGGRAPH2001 conference alone! This field of research is growing rapidly due to it's usefulness for the graphics industry. Anyone interested in texture synthesis should certainly read these articles; some of the results are very impressive!

- A.A. Efros and W.T. Freeman. Image quilting for texture synthesis. In *Conference Proceedings of SIGGRAPH01*, 2001. [20]
Efros presents a new method called *image quilting*. Basically the technique synthesizes the texture patch by patch (as one would make a quilt), not pixel by pixel. Because the patches never match perfectly, smoothing around the boundaries is necessary. Although the technique is very fast, the results do not look as good as those based on pixel-by-pixel, and for the moment can not be used to fill in missing textures. Also, there is no real theoretical foundation. In my opinion, the algorithm is far less elegant than their previous one [21]. The speed is simply necessary for the graphics community, who care about speed of rendering, not elegance.
- L.Y. Wei and M. Levoy. Texture synthesis over arbitrary manifold surfaces. In *Conference Proceedings of SIGGRAPH01*, 2001. [52]
This work is basically an extension of Efros' earlier pixel-by-pixel work (described in [21], not to be confused with the work directly above), but a very good one at that! They present a multi-level image representation. The texture is synthesized from the coarsest level to the finest. This causes large-scale textures to be synthesized at higher levels, whereas fine-scale details are filled in at lower levels. The algorithm is also deterministic, making it far quicker. Further acceleration is achieved by ordering the templates in a tree-structure, making the search for the closest matching template of order $O(\log(n))$, not $O(n)$. Since there are usually many pixels (n is very large), the benefit is huge.
- P. Harrison. A non-hierarchical procedure for re-synthesis of complex textures. In *Conference proceedings of WSCG2001*, pages 190–197, 2001. [26]
Harrison takes a completely different approach to the problem. Although using the pixel-by-pixel method, his method uses information theory to determine the order in which pixels are to be synthesized. By ingeniously ordering the pixels, high-level

structures in the image are maintained, without having to rely on large windows. The small window-size allows the technique to be much faster than the technique described by Efros in 1999, but slower than that of Wei's solution described above. The algorithm can be downloaded as a GIMP plug-in at:

<http://www.csse.monash.edu.au/~pfh/resynthesizer/>.

- M. Ashikhmin. Synthesizing natural textures. In *Symposium on Interactive 3D graphics*, 2001. [1]
A. Hertzmann, C.E. Jacobs, N. Oliver, B. Curless, and D.H. Salesin. Image analogies. In *Conference Proceedings of SIGGRAPH01*, 2001. [15]
Both are variations on Efros' work of 1999 [21].

Some articles recommended to me by Henk Mastebroek on the subject of simulations of filling in phenomena with neural networks:

- S. Grossberg. Cortical dynamics of three-dimensional form, colour and brightness perception. In S. Grossberg, editor, *Neural Networks and Natural Intelligence*. The MIT Press, Cambridge, 1988. [24]
- F. Heitger, R. Von der Heydt, E. Peterhans, L. Rosenthaler, and O. Kuebler. Simulation of neural contour mechanisms: representing anomalous contour. In *Image and Vision Computing*, volume 16, pages 407–421, 1998. [18]
- R. Ritz. Pattern segmentation in an associative network of spiking neurons. In H A K Mastebroek and J E Vos, editors, *Plausible neural networks for biological modelling*. Kluwer Academic Press, 2001. [41]

Bibliography

- [1] M. Ashikhmin. Synthesizing natural textures. In *Symposium on Interactive 3D graphics*, 2001.
- [2] Paul J. Besl. Range imaging sensors. Technical report, General Motors, 1988.
- [3] Paul J. Besl. *Surfaces in range image understanding*. Springer-Verlag, 1988.
- [4] Ben Best. Basic cerebral cortex function with emphasis on vision.
<http://www.benbest.com/science/anatmind/anatmd5.html>
- [5] R.M. Bolle and D.B. Cooper. Bayesian recognition of local 3D shape by approximating image intensity functions with quadric polynomials. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(4):418–429, 1984.
- [6] G. Borgefors. Distance transformations in arbitrary dimensions. In *Computer Vision, Graphics and Image Processing*, pages 321–345, 1984.
- [7] Pierre Boulanger. Knowledge representation and analysis of range data. In *Second International Conference on Recent Advances in 3D Digital Imaging and Modeling*, 1999.
- [8] J. Bresenham. Incremental line compaction. *The Computer Journal*, 1(25):116–120, 1982.
- [9] L.D. Cai. *Scale-based surface understanding using diffusion smoothing*. PhD, University of Edinburgh, January 1990.
- [10] A. Calway. Estimating the structure of textured surfaces using local affine flow. In B. Thomas and M. Mirmehdi, editors, *British Machine Vision Conference 2000*, pages 92–101, 2000.
- [11] H. Cantzler and R. Fisher. Comparison of HK and SC curvature description methods. In *Conference on 3D Digital Imaging and Modeling*, 2001.
- [12] European Commission. *Research Training Networks, 1997-2001*. Office for Official Publications of the European Community, 2000.
- [13] A. Criminisi and A. Zisserman. Shape from texture: homogeneity revisited. In B. Thomas and M. Mirmehdi, editors, *British Machine Vision Conference 2000*, pages 82–91, 2000.

- [14] O. Cuisenaire and B. Macq. Fast and exact signed Euclidean distance transformation with linear complexity. In *IEEE Intl Conference on Acoustics, Speech and Signal Processing*, volume 6, pages 3293–3296, 1999.
- [15] A. Hertzmann, C.E. Jacobs, N. Oliver, B. Curless, and D.H. Salesin. Image analogies. In *Conference Proceedings of SIGGRAPH01*, 2001. To be published after the conference (12-17 August 2001).
- [16] D.C. Dennett. *Consciousness Explained*. Boston: Little Brown, 1991.
- [17] D.C. Dennett. Filling in versus finding out: A ubiquitous confusion in cognitive science. *Cognition, Conceptual, and Methodological Issues*, 1992.
- [18] F. Heitger, R. Von der Heydt, E. Peterhans, L. Rosenthaler, and O. Kuebler. Simulation of neural contour mechanisms: representing anomalous contour. In *Image and Vision Computing*, volume 16, pages 407–421, 1998.
- [19] R. Descartes. *Principia Philosophiae*. Ludovicus Elzevirius, Amsterdam, 1644.
- [20] A.A. Efros and W.T. Freeman. Image quilting for texture synthesis. In *Conference Proceedings of SIGGRAPH01*, 2001. To be published after the conference (12-17 August) 2001.
- [21] A.A. Efros and K. Leung. Texture synthesis by non-parametric sampling. In B. Werner, editor, *International Conference on Computer Vision*, pages 1033–1038, 1999.
- [22] P. Faber and R.B. Fisher. Euclidian fitting revisited. In *Proceedings of the 4th International Workshop on Visual Form*, 2001.
- [23] R.B. Fisher. *From surfaces to objects*. Wiley and Sons, 1987.
- [24] S. Grossberg. Cortical dynamics of three-dimensional form, colour and brightness perception. In S. Grossberg, editor, *Neural Networks and Natural Intelligence*. The MIT Press, Cambridge, 1988.
- [25] R.M. Haralick. Digital step edges from zero-crossings of second directional derivatives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(1):58–68, 1984.
- [26] P. Harrison. A non-hierarchical procedure for re-synthesis of complex textures. In *Conference proceedings of WSCG2001*, pages 190–197, 2001.
Also see <http://www.csse.monash.edu.au/~pjh/resynthesizer/>
- [27] S.L. Hurt and A. Rosenfeld. Noise reduction in three-dimensional digital images. *Pattern Recognition*, 17(4):407–421, 1984.
- [28] N. Johnson. Nick’s ai tutorial. <http://members.aol.com/DaemonBBS/ai/tut/>
- [29] G. Kanizsa and W. Gerbino. Amodal completion: Seeing or thinking? In J. Beck, editor, *Organization and representation in perception*, pages 167–190, 1982.
- [30] I. Kant. *Critique of pure reason*. 1781.
- [31] W. Köhler. *Gestalt Psychology: An introduction to new concepts in modern psychology*. New York: Liveright Publishing Corporation, 1947.

- [32] E. Thompson, L. Pessoa, and A. Noë. Finding out about filling-in: A guide to perceptual completion for visual science and the philosophy of perception. *Behavioral and Brain Sciences*, Volume 21, December 1998.
- [33] Peter Lancaster and Kęstutis Šalkauskas. *Curve and surface fitting*. Office for Official Publications of the European Community, 2000.
- [34] Steven Lehar. Gestalt isomorphism and the quantification of spatial perception. *Gestalt Theory*, 21(2):122–139, 1999.
- [35] K. Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics*, 2:164–168, 1944.
- [36] D.W. Marquardt. An algorithm for least squares estimation of non-linear parameters. *Journal of the Society of Industrial and Applied Mathematics*, 11:431–441, 1963.
- [37] G.E. Müller. Concerning the psychophysics of visual sensations. *Zeitschrift für die Psychologie*, 17:1–82, 1896.
- [38] L. Pessoa and H. Neumann. Why does the brain fill-in? *Trends in Cognitive Sciences*, 2(11):422–424, 1998.
- [39] A. Walker, R. Fisher, S. Perkins, and E. Wolfart. HIPR2: Distance transform. <http://www.dai.ed.ac.uk/HIPR2/distance.htm>.
- [40] A.W. Fitzgibbon, R.B. Fisher, and D.W. Eggert. Extracting surface patches from complete range descriptions. In *3DIM97*, pages 6 – Geometric Processing, 1997.
- [41] R. Ritz. Pattern segmentation in an associative network of spiking neurons. In H.A.K. Mastebroek and J.E. Vos, editors, *Plausible neural networks for biological modelling*. Kluwer Academic Press, 2001.
- [42] A. Rosenfeld and J.L. Pfaltz. Distance functions on digital images. *Pattern Recognition*, 1(1):33–61, 1968.
- [43] M. Sanchiz and R.B. Fisher. A next-best-view algorithm for 3D scene recovery with 5 degrees of freedom. In *Proc. British Machine Vision Conference (BMVC99)*, pages 163–172, September 1999.
- [44] F. Stulp and R.B. Fisher. Hypothesized reconstruction of occluded surfaces. Submitted to the conference on Computer Vision and Pattern Recognition (CVPR2001) Hawaii, December 2001.
- [45] F. Stulp, F. Dell’Acqua, and R.B. Fisher. Reconstruction of surfaces behind occlusions in range images. In *Proc. of Int. Conf. on 3-D Digital Imaging and Modeling (3DIM01)*, pages 232–239, 2001.
- [46] M.V. Srinivasa, T. Maddess, and M.P. Davey. Response to pessoa and neumann: Why does the brain fill in? http://people.mydesk.net.au/~mdavey/papers/why_fill.html

- [47] L. van Gool and T. Tuytelaars. Wide baseline stereo matching based on local, affinely invariant regions. In B. Thomas and M. Mirmehdi, editors, *British Machine Vision Conference 2000*, pages 82–91, 2000.
- [48] G. Taubin. An improved algorithm for algebraic curve and surface fitting. In *4th International Conference on Computer Vision*, pages 658–665, 1997.
- [49] E. Trucco and A. Verri. *Introductory techniques for 3D computer vision*. Prentice Hall, 1998.
- [50] P. Dias, V. Sequeira, J.G.M. Gonçalves, and F. Vaz. Combining intensity and range images for 3D architectural modelling. In C. O’Sullivan, B. Fisher, and K. Dawson-Howe, editors, *Virtual and Augmented Architecture (VAA ’01)*, pages 139–145, 2001.
- [51] L. Vincent. Exact Euclidian distance function by chain propagations. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 520–525, 1991.
- [52] L.Y. Wei and M. Levoy. Texture synthesis over arbitrary manifold surfaces. In *Conference Proceedings of SIGGRAPH01*, 2001. To be published after the conference (12-17 August 2001).