University of Edinburgh
Division of Informatics

Evaluation of a range data segmentation program

4th Year Project Report
Computer Science

Alexander Clarke

May 30, 2003

Abstract: This project report explains the steps that were involved in evaluating Edinburgh Universities range segmentation program, RangeSeg, using the automated evaluation framework developed by the University of South Florida. RangeSeg and the framework are described, as well as the necessary adaptation process (data conversion, pre and post processing, among others) needed to run the evaluation program correctly. The final evaluation results are presented, as well as a comparison with performances from other range segmentation programs.

Contents

1. Introduction

Range data segmentation, the process of extracting meaningful three-dimensional shapes and their equations from range data, has been a widely studied subject within the field of computer vision. The vast array of scientific and industrial applications this low-level technique could support, ranging from robot vision (a machine can analyze its environment in terms of mathematical equations) to Computer Assisted Design (creating an ideal, non polygonal shape from a scanned real life object) only serve to comfort further research in this field. Despite this potential, range data segmentation has not yet matured as a commonly used technology. A reason for this is a lack of empirical evaluation methods, which could measure the performance of range segmentation algorithms objectively. The rationale behind this need is the following: proving the accuracy of an algorithm against real world data is much more relevant than providing a theoretical proof based on ideal data; objects in real life are typically uneven, bumpy or eroded. Being able to clearly describe the efficiency of a segmentation algorithm against real data is a key factor in developping applications whose requirements demand a specific level of accuracy or robustness (a CAD application would be required to fit scanned objects with maximum precision, for the pupose of reverse engineering). Finally, if the performance of an algorithm can be measured in terms of clear metrics (such as over and under-segmentation), weaknesses in current algorithms can be identified, providing clear requirements for future research in data segmentation.

Recently, with the realisation of its importance, much attention has been put into developing such an objective evaluation method; the most notable result so far has been the evaluation framework developed by the University of South Florida. This package revolves around real-life test data for which ideal segmentations, so called ground truth (GT) images are specified in fig.1.1. The test data is segmented and the result compared to the GT image, returning performance metrics based on the similarities between both images. This evaluation system has proven to be quite successful and as such is slowly acquiring the status of standard empirical evaluation tool, used in several inter-university segmentation competitons. In this project, we use this evaluation package to measure the performance of RangeSeg, Edinburgh Universitys vision groups range data segmentation algorithm. When last measured 6 years ago[1], RangeSeg performed very well, receiving the best scores in 9 out of 10 accuracy measurements. Nevertheless, this only covered range data including planar scenes; here we evaluate RangeSegs performance on both planar and curved data. Furthermore, RangeSeg has since then been rewritten using the much more precise albeit slower Euclidean distance to fit surfaces[3], and as such has gained in robustness

The goal of this project is to extract meaningful evaluation results so as to

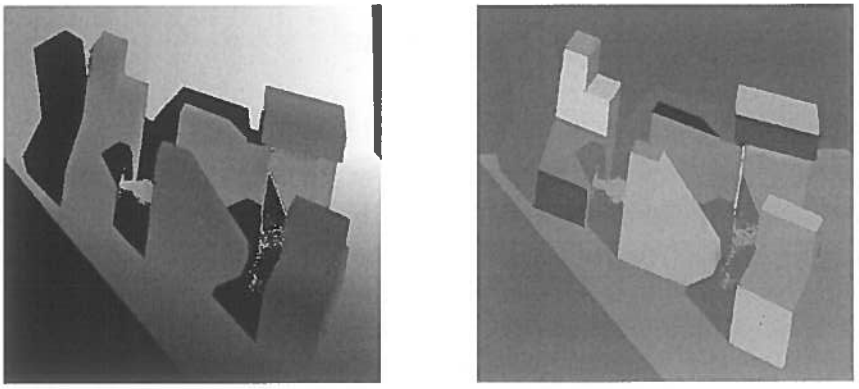

Figure 1.1: A range image and its corresponding ground truth specification

measure RangeSegs current performance and eventually identify its weaknesses. To this end evaluation is performed both on a planar data set and a curved-surface data set. Beyond this, this project covers the steps that were necessary to adapt RangeSeg to the evaluation package.

The scope of this project considers RangeSeg to be a black box; all the adaptation operations needed to run the evaluation framework of Rangeseg are external to the program and performed by a data converting interface. As will be covered in this report, this approach has certain limitations, and in the case of testing RangeSeg on the planar range data, information was lost due to data transformation. As a consequence, some evaluation results were slightly affected, and can therefore only be used as an approximation to the real performance. A further part of this report addresses the usefulness of pre-and post processing techniques such as hole-filling algorithms, to improve evaluation results. Finally, the project also involved choosing correct parameters for evaluation; some of them were trained using the evaluation packages inbuilt training facility, the others set manually.

This report is organised into 7 chapters; Chapter 2 describes the workings of the USF evaluation package, its performance metrics and its training and validation methods. Chapter 3 introduces the RangeSeg application, along with its parameters and its visual and textual output. Chapter 4 builds on the previous chapters to describe the steps taken in adapting Rangeseg to the evaluation package, the test data in particular. Correct parameters choices are also explained for both planar and curved data segmentation evaluation. Chapter 5 discusses the advantages of additional pre-and post processing on the test range data as well as the on the segmented output in order to yield more satisfactory results. The specific algorithms used to these ends are described. Chapter 6 presents the final evaluation results, and compares them to previous RangeSeg results as well as results obtained from other segmentation algorithms. Chapter 7 offers a conclusion to the project.

2. The Automated Evaluation Framework

The initial work made on segmentation evaluation by the University of South Florida was made in 1996 as reported in [1]. In this experiment, 4 different algorithms (including RangeSeg) were evaluated and compared. The evaluation framework used today is essentially an enhancement of this older framework; while the ground truth precept and performance metrics stay the same, it is now able to evaluate segmentation of curved surfaces [4] and includes an automated parameter training device. This chapter introduces the evaluation package. The workings of the framework are covered in the first 3 sections, which describe performance measurement, parameter tuning and validation, respectively. Section 3.4 describes the test data associated with the package used in this project. Finally section 3.5 describes the various tools in the package and how they are used in practice.

2.1 Performance measurement

The main challenge in developing an objective evaluation framework is indefining an adequate measurement for performance. Here, performance of a segmentation algorithm is defined as the lack of discrepancies between the algorithms segmentation of a scene and the equivalent ground truth (GT) segmentation, a manually created, ideal segmentation of a set scene. For clarity, we define segmentation as described in [2]:

A segmentation of an image R into regions $r_1, \cdots, r_n$ is defined by the following properties:

1. $r_1 \cup r_2 \cup \cdots \cup r_n = R$.(Every pixel belongs to a region.)
2. Every region is spatially connected. (Currently defined as four-connectivity.)
3. $\forall r_i, r_j \in R, i \neq j, r_i \bigcap r_j = \emptyset$.(All regions are disjoint.)
4. $\forall r_i \in R, P(r_i) = true$ (All pixels in a region satisfy a specified similarity predicate; in the case of range images, they belong to the same surface)
5. $\forall r_i, r_j \in R, i \neq j, r_i$ and r_i, r_j are four-connected and adjacent, $P(r_i \cup r_j) = false$. (If two regions are four-connected and adjacent, then they represent different surfaces.)

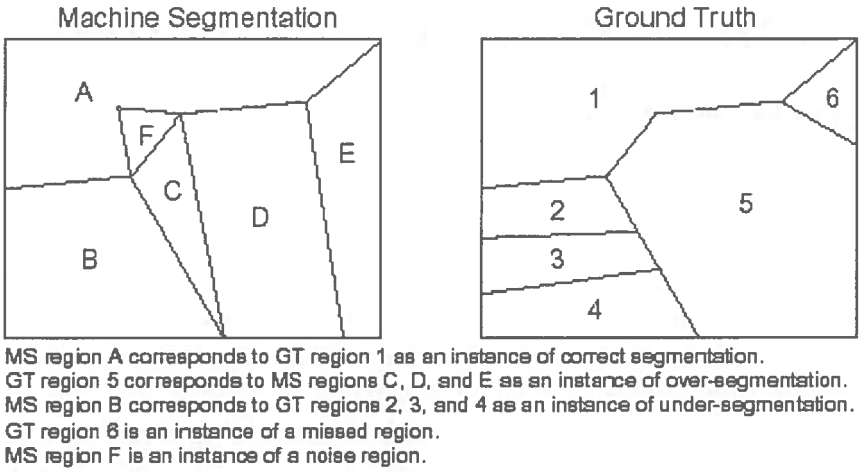

Figure 2.1: A graphical explanation of the scoring metrics

6. There are artifact regions in the image where no valid measurement was possible which all have the same label (violating rule 2) and for which rules 4 and 5 do not apply.

2.1.1 Ground truth

As mentioned above, GT images are the manually elaborated ideal segmentations of a range image. In order to achieve such a perfect segmentation, each pixel from the original range image is considered separately and assigned a surface (or segment) label using an interactive tool. A pixel cannot be assigned a surface label in every case; noise pixels, shadow pixels, and cross edge pixels (pixels located on the edge between 2 adjacent surfaces) are all labeled as nonregion, as defined in rule 6. The task of specifying ground truth is straightforward for scenes comprising only of planes, but more complex when curved surfaces are concerned. Indeed, it has been noted [4] that regions in ground truth can only be defined for as many quadric surface types that can be easily categorized. In the current generation of ground truth images, this only includes planes, spheres, cones, cylinders and torii. (More shape categories can in fact be defined, such as ellipses, elliptic cylinders and cones, but these are not considered here).

2.1.2 Performance metrics

Once a segmentation algorithm has created its machine segmentation (MS) of a scene, it is compared to the GT specification, yielding the following five metrics, based on degrees of mutual overlap: correct segmentation, over segmentation, under segmentation, missed region and noise region. These are represented in fig

2.1 and defined as follows: *correct segmentation* occurs when a sufficient ratio of pixels share the same x and y coordinates in both MS and GT instances of a region. *Over-segmentation* occurs when a region in GT is mapped to more than one region in MS. Conversely, *under-segmentation* occurs when several regions in GT are mapped to a single region in MS. A *missed region* is when a GT region is mapped to a unsegmented (nonregion) patch. Finally, a *noise region* occurs when an MS region appears where only nonregion pixels are present in GT. A more formal description of these 5 measurement can be found in [1]. Each metric can be a useful measurement in itself, depending on the application the algorithm is destined for. Nevertheless, in the current implementation of the evaluation framework, performance is solely defined as a function of correct segmentation instances. As already mentioned , correct segmentation is a function of a certain ratio; for example, a ratio of 0.6 means 60% of MS pixels in a region must be share the same coordinates as the pixels in the corresponding GT region. To calculate the final performance score, this ratio (or overlap threshold) is set successively to `0.51`, `0.51`, `0.60`, `0.65`, `0.70`, `0.75`, `0.8`, `0.85`, `0.90` and `0.95`. Resulting comparison values are plotted to draw a performance curve, which is then normalized. The area under the curve (AUC) is then calculated, which is the final performance score, an area between 0 and 100.

2.2 Parameter tuning

One of the main enhancements of the evaluation tool since its first use [1] is its ability to tune segmenter parameters (typically minimum fitting residual, etc...) to favor optimal performance in evaluation. Performance using this method is has been shown to be generally more effective than manual parameter setting in [2].

To train a set of parameters, an extra set of images is used; it is separate from the evaluation set and is used solely to train the algorithm's parameters. It is essential that the data used to evaluate the segmenter is of the same nature as the data used to train it or results may suffer; the range data must not only be of the same type (noise levels, quantization or other, can vary considerably between different range scanner acquisitions) but must also have similar objects in the scenes that they represent. The parameter training acts as a form of adaptive search: in a first step, a range is given for each parameter (e.g. between `0.04` and `1.6`). This range is then uniformly sampled at 5 points for each parameter. For n parameters to be trained, the tuning algorithm starts with 5^n initial parameter settings to be considered. Each setting is used in the segmenter, resulting in 5^n MS images per scene. After a run though the compare tool, the highest-scoring settings are kept. A 3^n sampling around the best settings is performed, the new settings are used with the segmenter, the highest scoring settings are kept, and so on and so forth until the difference in performance is no higher than 5% between

iterations. At the end of the tuning, parameters should be optimal.

Although this method provides good results, it is very time consuming, and it is not practical to tune more than 4 parameters with it, hence the importance of choosing the most significant parameters. For instance, tuning 4 parameters on 14 train images represents $5^4x14 = 8750$program executions for the first iteration only. If the execution of the segmenter happens to take 15 minutes, total execution time for the first step alone would span over 3 months! This is not good.

2.3 Validation

Once the parameters have been tuned, the actual evaluation process takes place. Parameters are tuned for each set train set (individual sets of train images). In the end, we have m parameter settings (for each train set) and n validation sets. The final validation result is mxn AUC performance results. Parameters are tuned with an increasing number of dimensions. First only one parameter is trained, then 2 together, then 3, etc. At the end of the parameter tuning for each dimension, a test is made to see if the performance between the last 2 tunings is significantly better. A statistical sign test is carried out, checking if performance has a significance $\alpha = 0.05$ in normal approximation of binomial distribution. If the performance is not significantly better, the next dimension is not tuned, otherwise, training is resumed adding a new dimension.

2.4 Test data

In this particular package, there are 2 separate data sets; A set of planar scenes in ABW format and a set of curved surface scenes (containing planes, cylinders, cones, spheres and torii) in Cyberware (CW) format. Both data sets comprise 40 range images, along with their ground truth specification. Within these 40 images, 14 are used for training, 13 for validation and another 13 for testing. For each group, 10 sets are defined, each containing 6 of the images (a same image can be in several groups).

2.4.1 ABW

All the planar data is in ABW format. These are 512 x 512 PGM (Portable GrayMap) 8-bit (256 levels of gray) images. They are extracted using the ABW structured light scanner (a depth acquisition technique using striped light patterns). An example is shown in fig.1.1 Data is represented in a perspective view, and Cartesian coordinates for each pixel of the range file can be extracted using the calibration (`filename.cal`) file, which specifies the camera offset, scale, focus and correction. The data is characterized by the presence of shadows which

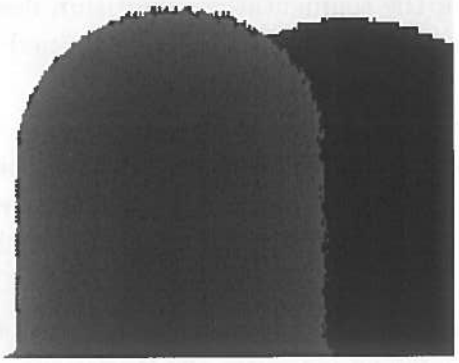

Figure 2.2: An example Cyberware image

are a side-effect of the scanning technique. Furthermore, the low color resolution results in quantization noise.

2.4.2 CW

All the curved-surface data is in Cyberware format.These are 256 x 256 PGMs. An example is shown in fig. 2.2. Data is represented in an orthographic view. Cartesian coordinates are extracted from the echo (`filename.eko`) file, a compressed binary version of the range image file; the `filename.range` file is not actually used. As with ABW images, the low color resolution causes a noticeable level of quantization noise.

2.5 Package organisation

The evaluation package (downloadable from http://marathon.csee.usf.edu/range/segcomp/SegComp.html) is a composed of several individual binaries. Each is briefly described in this section.

Compare `usage -g ground truth -m machine segmentation -t tolerance -r result file name`
This is the tool which performs the comparison operation described in 3.1 and

writes the five overlap measurements based on the defined `tolerance` to a file `result file name`

Segtrain `usage: segtrain -c configfile -d dimension`
This is the tool which trains the parameters as described in 3.2. the configfile is a text file containg information about the location of various binaries and data, including the segmentation algorithm itself. the dimension option sets the maximum number of parameters to be trained

Segtest `usage: segtest -c configfile -d dimension`
This is the tool which evaluates the segmentation algorithm once the parameters have been changed, as described in 3.3. Dimension defines for which dimension of trained parameters the segmentation algorithm should be evaluated.

Signtest `usage: signtest -c configfile -d1 dimension -d2 dimension`
This is the tool which measures the statistical sign test between training of 2 successive parameter dimensions, as described in 3.3 `-d1` is the first dimension and `-d2` the second.

Scrpt `usage: scrpt -t traindirectory -c configfile -d dimension`
This is the tool which successively runs training, sign testing and validation. traindirectory is the directory where the aforementionned binaries binaries are located.

Segmentation algorithm requirements Finally, to be properly evaluated, the segmentation algorithm must take the following options:
`segmenter -r rangeimage -p paramfile -m msimage`

3. RangeSeg

This chapter introduces the RangeSeg application, Edinburgh Universitys range data segmentation algorithm, developed as a part of the Imagine II project. In order to understand the steps taken in evaluating RangeSeg, it is necessary to explain the applications method of execution, parameters and main outputs.
RangeSeg is a state of the art range data segmenting algorithm which runs on Sun stations as well as X. It is characterised by its interactive interface, which graphically presents the segmentation at run time, as well as its precision, thanks to a least squares curve-fitting algorithm using true Euclidean distance (as opposed to an approximation).

In a first section, RangeSegs 3-step segmentation method is explained. A second section describes RangeSegs modifiable parameters (which will be tuned in the evaluaiton process). A third section describes the array graphical and textual output/feedback provided by RangeSeg. Finally a fourth section covers RangeSeg's file input and output.

3.1 Segmentation method

In order to achieve accurate segmentation of range data, RangeSeg uses a 3-step method, using both pre-processing and initial shape classifications. These 3 steps are described in this section. The properties that result from this method, as outlined in[rbf] are also described.

The first step involves noise reduction in the range data using smart pre-processing. Firstly, surface boundaries are determined; these are can either be blade edges, defined as significant depth changes in the data (what is perceived as the extremity of an object), or fold edges, defined as large variations between the normals of two adjacent pixels (what is perceived as a fold or corner in the object). Once surface edges are detected, diffusion smoothing is performed within these set boundaries, hence stopping surfaces from bleeding into each other and preserving the general shape of the object. In the second step of segmentation, the principal curvatures k_1 and k_2 are extracted from the data, and from these the relative H (mean) and K (Gaussian) curvatures are calculated for each pixel; based on its H and K property, each pixel is then classified as belonging to a specific shape class, as classified in fig. 3.1.

Inner and outer H thresholds are defined , effectively classifying each H as planar (under the inner threshold), curved (above the outer threshold) or unlabeled (any value between the inner and outer threshold). Using proximity, patches are defined by grouping labeled pixels of the same shape class. A morphology schedule, a string composed of the characters + and −, is then applied, as a sequence of dilations, + (labeled regions are expanded to unlabeled regions) and erosions ,

Curvature Signs $K = k_1 \times k_2$	$H = (k_1 + k_2)/2$	Shape Class
0	0	plane
0	−	negative cylindrical
0	+	positive cylindrical
+	−	negative elliptic
+	+	positive elliptic
−	any	hyperbolic

Figure 3.1: Class shapes by H and K curvatures

− (small labeled regions are reverted to unlabeled regions). This has the effect of filling small unlabelled gaps and removing small isolated labeled regions, which are usually there due to noise. A final operation removes all regions which are smaller than a certain size (again, usually spots of curvy noise).

The third step takes each of the previously defined pixel regions and expands them as follows: if a pixel is 8-connected to the region (i.e. a region pixel is one of the 8 pixels surrounding the original pixel, diagonals included), its Euclidean distance from the surface being fitted is within an allowed deviation, and its normal is in good enough agreement with the current fitted surface, it is added to the region or slurped. Previously slurped pixels can be stolen from a region if the new surface provides a better fit (in terms of normals and least squares fitting).

For each region, a selection of primitive shapes (plane, sphere, cone, cylinder, elliptic cylinder, or general quadric) is considered to fit the data. Since the equations of most of these shapes (spheres, cylinders, etc.) are just a special form of quadric, a penalty, the Aikaike metric, is put in place to avoid a systematic fitting of general quadrics and favor primitive shapes. This number is subtracted from the estimated fitting error of all shapes but the general quadric. Once optimal parameters for each implicit surface equation are found through a least-squares method, the region pixels are compared to each decision surface. The surface equation yielding the lowest fitting residual (post Aikaike penalty) in the least-squares fit is chosen. Finally, the region is contracted to best fit the shape. Using the new surface equation as a basis, the region is grown again, and so on and so forth...These iterations are called slurp refinements. The region growng step is acted out for each region defined in the second segmetnation step. Once all regions have been grown, the segmentation is completed and a final labeled image is written.

RangeSegs particular segmentation method makes it particularly resistant to noise, and by using Euclidean distance for least-squares fitting, it is very precise, at the cost of greater calculation time. It was also shown[buyers guide]that the use of Euclidean distance initiated greater robustness (can deal with a reduced

point density) and as a consequence also pose-invariant (segmentation does not change in accuracy when data is moved in space), properties which did not apply when distance approximations (such as the Taubin method) were used.

3.2 Parameters

In the previous section, RangeSegs operation was described. In this section, we list what metrics used in this segmentation method can be tuned for best results; these are of particular importance for evaluation . Each modifiable parameter is listed, along with its default value and its general effect on the segmentation process:

Sculptured: Determines whether the input scene has spline surfaces (default: no)

Background threshold: Sets what depth values should be considered as infinite background or not segmented (e.g. shadows). Typically these are the lowest 2 or 3 grayscale values of a scene (default 2)

Depth disc. threshold: The maximum amount (mm) which can vary between 2 pixels before a depth discontinuity edge is declared. If this value is set too small, steep slopes will be considered as blade (or jump) edges, if it is set too large, real depth discontinuity between separate surfaces might not be detected. (Default 5; should be adjusted according to image dimension)

Fold edge threshold: The maximum amount (degrees) adjacent surface normals can vary before a fold edge is declared. This value must not be too big (e.g. a value of 90 means fold edges will only be detected when 2 adjacent surfaces from a sharper angle than 90 degrees), but not too small, or noise will create spurious fold edges everywhere. (Default 25; should be as low as can be afforded without the interference of noise)

Number of smoothings: The number of smoothing iterations applied using boundary-preserving diffusion smoothing. Too few smoothings leaves noisy, difficult to segment data, too many smoothings distorts the data shape, especially if fold edge detection is disactivated (typical with noisy data). (Default 2)

Morphology schedule: The series of dilations and erosions made the initial shape class regions of step 2, represented by a string of $+$ and $-$ (dilation and erosion, respectively). This is used to remove falsely labeled noise pixels, and expanding unlabelled pixels to labeled pixels. (Default $+ - ++$)

H curvature Threshold Hi/Lo: Any pixel with an absolute H value higher than the Hi threshold is labeled as positive or negative, depending on the sign of the curvature, any pixel below the Lo threshold is labeled as planar. Pixels between Hi and Lo are declared to be unlabeled pixels. Typically, pixels will be mislabeled due to image noise. This indirectly affects the minimum curvature a surface can have before it is considered a plane (e.g. the maximum radius of a sphere). Both values are set to the same value, and should be a factor of the noise level in the scene (default `0.02`)

Fitting residual tolerance: The maximum deviation (mm) allowed between a pixel in the data and the surface which is trying to fit it (if the deviation is higher than this value, the pixel will not be added to the fitted surface). Too small a tolerance will only accept very close pixels, missing out much data if the data is noisy. Too large a tolerance will allow too many pixels to be added to the fitted surface. Again, this value should be a factor of noise level in the scene (default `1.0`)

Max norm. deviation: The maximum, in degrees, the normals from 2 adjacent pixels can deviate from each other and still be fitted to the same surface. (Default: 90)

Eigenratio for plane: The ratio defined as distance of a pixel from plane/length of plane. the smaller the value, the further away a pixel can be from the plane and be fitted. (default: 250)

Max slurp refinements: Maximum number of times the fitted surface shape parameters should be re-estimated when surface-fitting. As pixels are added, the least squares fit will change. This means that the pixels satisfying the residual test will change. Typically, after 2 refinements too few pixels are added to really make a significant difference in the parameters resulting from a least-squares fit. Since refinements are still as calculation-intensive regardless, it is more efficient time-wise to stick to low values (default: 2)

Minimum region size: The minimum size, in pixels, a region must before it can be considered for growing in the third step. If this is set too small, small patches of noise will be considered as independent surfaces. If set too large, regions identifying genuinely small surfaces will not be considered for growing. This value should be set taking in account the morphology schedule, which will previously grow and shrink the regions prior to the decison. (Default: 30)

Number of slurps: The number of times the whole surface fitting process is used on the data. Typically, this is once. (Default: 1)

Aikaike: The level of penalty put in place to avoid the systematic fitting of general quadrics to the data. Too low a value will not stop general quadrics from being the best fit; too high a value will prevent the fitting of any general quadric. (Default: `0.00025`)

3.3 Graphical and textual output

As an interactive graphical application , one of the characteristics of RangeSeg is the rich level of feedback which is sent the users way. This provides a significant advantage when manually adjusting parameters, as the effect of most changes can be seen in real time. Beyond the graphical interface, RangeSeg also provides textual output, which is useful in estimating how well current data is being fitted. This section covers what graphical and textual output is provided and how it can be used to optimize parameters.

3.3.1 Graphical output

In order to assist the user in the tweaking of parameters, RangeSeg provides a series of displays which show intermediate steps in segmentation graphically. These are listed here, categorized by the 3 main steps they belong too.

Step 1 (boundary-preserving surface smoothing)

Non-background: shows which pixels in the scene are being considered in the segmentation process (green). If this does not seem right, the background threshold parameter must be adjusted.

Blade edges: shows the blade edges of the scene (green). If these do not seem to correspond to the actual extremities of objects in the scene, the depth discontinuity threshold must be adjusted.

Normals: shows a cosine-shaded version of the scenes surface normals. Brighter means the surface points more towards user. This image can help give a good idea of where fold edges in the scene should be.

Fold edges: shows the fold edges of the scene (blue). If these appear in more places than they should, the fold edge threshold should be adjusted to a higher level

Closed edges: shows the gaps in both blade and fold edges filled using a morphology technique

Smoothed depth: shows the resulting smoothed range data

curvature-based segmentation)

H + K values: images shaded by the values of the H and K curvatures, respectively. In both cases, brightness represents a larger magnitude

H + K histogram: shows the histogram of all H and K values, respectively. Here the H Hi and H Lo threshold bars are displayed.

H thresholds: shows the pre-morphed H value in terms of Hi and Lo thresholds. If not all supposed planar data is displayed as yellow and not all supposed curved data is displayed as red, H Hi and Lo thresholds should be adjusted.

K thresholds: shows the pre-morphed K value in terms of Hi and Lo thresholds. If not all supposed planar or cylindrical data is displayed as yellow and not all supposed u and v (e.g. sphere) curved data is displayed as red, H Hi and Lo thresholds should be adjusted.

H + K morph: shows the result of applying the morphology schedule on both H and K thresholds images, respectively. If the image is too eroded or too much noise subsists, the morphology schedule should be modified. Shape classes: shows the 6 different types of regions that were identified in the image based on the respective H and K values of each pixel

Unique labels: divides the previously defined patches into the individual regions, which will be grown in the region-growing step. Ideally, these should provide an approximation to the final image segmentation; parameters should be set so that at least one region should be present per surface. Too many regions may mean the data is still too noisy (either more smoothing erosion or a higher minimum region size should be applied).

Step 3. (Region-growing/ surface fitting)

Slurping: shows the region growing process; each pixel is assigned the color of its final surface fit.

Slurped pixels: shows what pixels have been already successfully fitted to a surface. For each pixel, red indicates a positive fitting residual, blue a negative fitting residual. This can help determine if a shape fitted to the pixels is in fact correct (e.g. a plane fitted to a curve and vice versa). The H Hi/Lo threshold can be accordingly adjusted.

3.3.2 textual output

As well as providing a graphical display, RangeSeg also outputs useful information to the terminal. This is the feedback provided with verbosity set to the minimum. For each shape being fitted (ordered by size), each surface type is considered in the least squares fitting, yielding different sets of parameters. Other information such as radii are shown for cyliders, elliptical cylinders and spheres. The fitting residual for each likely surface is shown, before and after the Aikaike penalty is removed. The chosen surface type with the smallest error will is eventually shown. the codes for each surface type are :
4-plane
5-cylinder
6-elliptical cylinder
7-sphere
11-general quadric

3.4 File input and output

This section briefly describes the type of range data RangeSeg accepts as input, and what segmentation output files it produces.

3.4.1 Image input

RangeSeg takes HIPS images as range data input. A typical Hips header contains image height and width, as well as color depth. Hips range data should additionally include the projection type (orthographic, perspective), the range (max and min) of x, y and z values, respectively. Camera location w.r.t. the origin, and camera orientation (x- and z-axes). Here data is represented as floating point values.

In its current implementation, RangeSeg only accepts range data using an orthographic projection.

3.4.2 File output

Once the range data has been successfully segmented, RangeSeg generates the following files:

filestem.labels: Hips file showing the original range data pixels labeled according to what individual surface/segment they have been fit to. This is typical labeled data output equivalent to most segmentation algorithms `file.ms-seg` (machine segmentation)

filestem.ms-nor: ASCII text file detailing the normal directions for each surface/segment Each segment is described in the form: Surface/segment number, normal x direction, normal y direction, normal z direction

filestem.ms-seg: Rasterfile containing pixel labels plus additional surface/segment information.

filestem.rawsurfs: ASCII text file listing the parameters for the implicit equations of each individual surface fitted to the data.

filestem.seg: Hips file containing information about discontinuity in the data.

4. The adaptation process

Although the primary objective of this project is to extract meaningful performance results from RangeSeg, most of the effort was in fact put into adapting RangeSeg to the evaluation framework. In the two last chapters, both RangeSeg and the University of South Florida evaluation framework were described. By comparing various inputs and outputs of one and the other, it is soon obvious that a minimum of adaptation is necessary. For a start, the evaluation package requires that the segmentation algorithm be executed as `segmenter -r rangeimage -p paramfile -m msimage`; these are not the options used by RangeSeg. Less trivially, we know that the image format accepted by RangeSeg is HIPS; both sets of test data are in PGM format. As we shall see the next section, the list does not stop there.

The first section of this chapter lists the various incompatibilities that arose in applying the evaluation package on RangeSeg, as well as other constraints and limitations that applied due to the nature of RangeSeg and of this project. The section after that covers the problem of adapting the data; a task, which we shall see, is non-trivial. A third section describes the software that was written in order to seamlessly interface RangeSeg to the evaluation package. Finally, a fourth section is reports the process of choosing parameters for evaluation. This covers which most significant parameters were chosen for training, as well as the values chosen for the remaining parameters.

4.1 incompatibilities, constraints and limitations

As is common with standard evaluation tools, the University of South Florida evaluation framework requires a set input and output format from the algorithm to be evaluated. As a consequence, some adaptation was required in order to successfully use the evaluation package. In this section we list what initial incompatibilities needed to be bridged, as well as what limitations prevented a perfect evaluation of RangeSeg.

As mentioned in the introduction to this report, the scope of this project has been considering RangeSeg as a black box, this implies that incompatibilities are not dealt with by altering the program, but by creating an interface between it and the evaluation package. This in itself can be considered as constraint in evaluating Rangeseg. The main incompatibility between RangeSeg and the evaluation package lies in the difference in data formats. Whilst the evaluation package provides data of type PGM, RangeSeg accepts images solely of type HIPS. Along the same line, the evaluation package relies on the segmentation algorithm to produce a PGM segmentation; RangeSeg outputs HIPS. Nevertheless, the data incompatibility problem is deeper than a mere file-conversion issue; in the case

of ABW scanned images, the topology of the data is different from RangeSegs. ABW range data is stored in a perspective view, whereas the current implementation of RangeSeg only reads range data presented in an orthographic view (i.e. x is the pixels x position, y the pixels y position and z the pixels intensity level). The data therefore needs to be converted from a perspective to an orthographic view, as well as being converted from one image type to another. Of course, once processed, the orthographic ABW data must be converted back to perspective, as well as being converted from HIPS to PGM. These data conversions and transformations and the side-effects they incur on the data are covered in the following chapter.

A second class of constraints arises when we consider training parameters for RangeSeg. The first issue is that RangeSegs use of Euclidean geometry for better precision (as mentioned in the previous chapter) makes it a particularly slow algorithm. Segmentation of a scene on a SunBlade 100 Sparc machine can take any time between 15 minutes and 1h30 hours depending on its complexity. This execution time is approximately divided by 5 on a DICE Pentium 4 machine (using Rangesegs X-windows implementation). This still means that an average run of the algorithm takes in the neighborhood of 15 minutes to execute; as a measure of comparison, a single execution of the University of Bern planar segmentation algorithm takes a little less than a minute. Using the calculation from chapter 2, section 2, it becomes obvious that we must limit the amount of trained parameters in consequence. Since parameters had to be trained for both planar and curved-surface scenes, we decided to limit the number of trained parameters to 2 (an approximate total of 500 runs of RangeSeg per scene type). Despite this choice, there is still scope for training more parameters for future evaluation, as is covered in section 4. The second parameter issue is that not all of RangeSegs parameters are numeric, or for that matter, linear. This is specifically the case for the morphology schedule parameter, which is a string of $+$ and $-$. A range for the various combinations cannot be reliably specified: uniformly sampling 5 values between $-----$ (5 erosions) and $+++++$ (5 dilations) is not really possible. The only possible recourse could be to manually sort all the different permutations of $+$ and $-$ by their qualitative effect on the data, and then to create a table mapping each morphology schedule to number between 1 and 5^2, which could be used as input for the training algorithm. Since this method is still quite approximate, and the number of parameters that could be trained was limited, we have decided not to cover this problem. Fortunately, the training tool supports both integer and floating point types of parameter values, so all the other parameters can potentially be trained in a more traditional manner. The remaining incompatibilities, such as the required execution format of the segmenter, are quite trivial. This particular issue is covered in section 3, RangeSeg interfacing programs

4.2 Adapting the data

In this section we present the general method to for converting HIPS to PGM and back, then we look at the specific topological transformations that need to be made for each kind of data conversion.

4.2.1 General PGM to HIPS cross-conversion

All the image transformations are from PGM to HIPS format, or from HIPS top PGM. Here the format of both image types is described, as well as the simple conversion operation.

PGM is very basic image type: the header has a simple format:

`char * magic number`	(here: `P5`)
`int rows int cols`	(number of rows, number of columns)
`int depth`	(intensity depth)

The pixel values are then displayed as a series of unsigned bytes (as specified by the P5 header - P2 denotes pixel values displayed as ASCII numbers) HIPS is a more complex format, although it is not compressed. The header is organized as follows (courtesy of Craigs Hips page [reference]):

`char * orig_name`	(originator of this sequence. here defaults to Array2D::init_header())
`char * seq_name`	(name of this sequence. here defaults to `No Name`)
`int num_frame`	(number of frames in this sequence. here `1`)
`char * orig_date`	(date the sequence was originated)
`int rows`	(number of rows in each image)
`int cols`	(number of columns in each image)
`int bits_per_pixel`	(number of significant bits per pixel. here: `32`)
`int bit_packing`	(Nonzero if bits were packed contiguously. here: `0`)
`int pixel_format`	(format of each pixel, here defaults to empty string)
`char * seq_history`	(sequence's history of transformations here `ABW2Hips`)
`char * seq_desc`	(Descriptive information, ending with a single '.')

The descriptive information section is used for RangeSeg range data as:

```
+COLORMAP range
CAMERA-INFO
char * projection type              (here : orthographic)
x-range float min float max
y-range float min float max
z-range float min float max
location float x float y float z    (here 0 0 0)
z-axis float x float y float z      (here 0 0 -1)
x-axis float x float y float z      (here 1 0 0)
.
```

The pixel values are then displayed as a series of floating point numbers.
To convert PGM to HIPS, we extract rows and cols from the PGM file. We then write these to the HIPS header, as well as the date of image creation, and x, y and z ranges. If the original PGM view/topology is orthographic, x and y ranges are 0-512 and the Z range is the minimum and maximum pixel intensities. If it perspective, x, y and z ranges are the maximum and minimum values of the extracted Cartesain coordinates. The data is then cast from byte to float. To convert HIPS to PGM, we extract rows and cols from the HIPS file, write these to the PGM header, then cast the data from float (or short) to byte.

4.2.2 ABW conversion

ABW data is characterised by the fact that the data is organised in a perspective view. To convert it to an orthographic representation, 2 steps must be taken:
Firstly, extracting the cartesian coordinates from the data
This is performed using the following equations[ABW-desc]:

$$x[i,j] = (y_j - rows) * (range[x_i, y_j]/scal + offset)/|fk|$$
$$y[i,j] = (cols - x_i)/c * (range[x_i, y_j]/scal + offset)/|fk|$$
$$z[i,j] = (cols - range[x_i, y_j])/scal$$

where range[xi,yj] are the pixels in the range (`file.range`) image and *scal*, *fk*, *c* and *offset* are the calibration metrics obtained from the calibration (`file.cal`) file.

Secondly, mapping the data to an orthographic view.
Once we have a table containing the cartesian coordinates for each point, the orthographic representation for each point is defined as :

$$hips[x_{ij}, y_{ij}] = z_{ij}$$

,where hips$[x, y]$ are the pixels in the resulting hips image.
Unfortunately, as the x and y values represent pixel postions here, they are re-

quired to be integer values, which they are not necessarily. An approximation of an orthographic representation can be defined as:

$$hips[round(x_{ij}), round(y_{ij})] = z_{ij};$$

As the points are now represented with an error margin of 0.5, this adds a proportional amount of noise in the scene; methods for dealing with this are discussed in chapter 5.

Because of the change of projection, the sampled points are displayed in a different manner, which creates 3 problems. The first is that holes appear in the data, namely in areas of the scene which are further away (this is explained by the fact that further objects are smaller in perspective and are described by less pixels). Filling in the holes with surrounding pixels using 8-connectivity solves this problem. A hole is filled with the mean value of pixels on either side of it vertically, horizontally or diagonally. The process is repeated until all holes are filled. It must be noted that this only works because the surfaces represented are planar; if the surfaces were curved the interpolation would be wrong.
The second problem is the logical, opposite side effect: areas of the scene which are nearer produce a condensation of pixels; pixels are written on top of each other (this is explained by the fact that closer objects are larger in perspective and hence more pixels are used to describe them). This is less of a serious problem as most of these pixels are part of the same surface, so proper segmentation is not affected, at the exception that a higher level of noise is created. This is solved by creating a mean of the values of all the points that are written to a single pixel. The third problem is more serious: with the change of view, some objects are partially occluded by others. This happens when several pixels have similar x and y coordinates, but are at a different depths in the scene. A case of this is highlighted in fig. . Here the occluded pixels have no relation to the pixels overwriting them (unlike the previous case), so information is definitely lost. This particular case of occlusion can be separated from the previous case by using a depth threshold; if the difference between both pixels is higher than a certain value, it is definitely a case of an object hiding another, otherwise we consider that the pixels are part of the same surface. The problem could be partially fixed by rotating the scene slightly, but this would incur the occlusion of other pixels. There would be ways of fixing this problem entirely, such as creating multiple images taken from different angles and segmenting them separately, but since the amount of data lost is not particularly significant, we have chosen to ignore this problem.
These last three issues could have been avoided if we had modified the RangeSeg program to extract Cartesian coordinates from the ABW directly. In this sense this was the major drawback of using the black box paradigm.

On an aside note, the problems experienced here underline the general properties of perspective range data: the distribution of the sampled points is far

from being uniform in Cartesian space. Surfaces further away will generally suffer from being sampled less than close ones, and as a consequence will risk being less accurately segmented.

4.2.3 Planar machine segmentation data conversion

Once the ABW-originating Hips file has been processed by RangeSeg, a resulting machine-segmented file (`filename.labels`) will be created. We need to convert it from HIPS back to PGM, but also from orthographic to its original perspective view, so that the image can be compared to the ground truth image, which is based on the original perspective ABW image. To do this, we use a similar method to the one used in converting perspective to orthographic. The first step is the same; using the original ABW image and caliber file, we extract Cartesian coordinates for each pixel/point of the original range file as before. We then perform the previously described rounding operation to figure out what pixel that point is mapped to in an orthographic image. We can now map a pixel of the perspective image to a pixel in an orthographic image, and vice versa. Using this, we can write each label pixel of the orthographic segmented image to its equivalent perspective pixel; we now have a perspective labeled image. Again, the problem of multiple perspective pixels mapped to a single orthographic pixel appears. The artifact circled in figure a appears. This can be partially solved by mapping an orthographic pixel to a single perspective pixel only, yielding the result shown in figure b. Once we have the labeled perspective image, a further modification must be made; the evaluation packages compare tool requires that the first 10 intensity values be reserved for nonregion pixels. Therefore we leave the 0 value (used for non-scene pixels by RangeSeg) as it is, and add +9 to each label from 1 upward.

4.2.4 Cyberware conversion

Unlike ABW images, Cyberware (CW) images contain orthographic range data. Despite this fact, converting them to HIPS was not a trivial deed. Whereas much documentation was provided on extracting cartesian coordinates from ABW, none was available for doing this with CW images. Finally, a c module borrowed from the baseline segmenter, cyfile.o,was used to obtain the coordinates; these are extracted from the compressed echo (`filename.eko`) file. The depth values returned were nevertheless a bit bizarre, and included negative values. These were removed by shifting all the values past the z origin. By using a graphical debugging system, it became apparent that the z values also needed to be multiplied by 22, despite being quite arbitrary, seemed like the best value (fig).

4.2.5 Curved surface machine segmentation data conversion

Converting CW-originated machine-segmentation images is the simplest conversion: the only requirement beyond the HIPS to PGM conversion was to add 9 to all the label values, except for 0, as in the ABW case.

4.3 RangeSeg interfacing programmes

4.3.1 ABW planar data evaluation

ABW2Hips This is the program that converts ABW images to Hips, as described in 4.2.1. As well as producing a HIPS file which can be used by RangeSeg, it also produces a text file, filename.pixloss which describes the ratio of lost pixels, for hidden and "recoverable" pixels respectively.
Usage : `ABW2Hips [-hpfco] <filename.range>`
The `-p` option forces ABW2Hips to convert the ABW to HIPS conversion without converting to data to orthographic (i.e. keeping it in perspective).
The `-f` option defines how many hole-filling iterations are performed to fill the gaps left by the perspective to orthographic conversion. -f 0 disactivates the hole-filling. default is 2.
The `-c` option is used to define the path to the calibration file, if it is not in the same location as the range image.
The `-o` option defines the path where the HIPS image is written to.

pseg This is the program which converts an ABW-originated machine-segmentation `filename.labels` image to a PGM format `filename.ms-seg` image, as described in 4.2.2 Since the segmentation results can be quite noisy, a noise reducing/ edge smoothing algorithm can be used. This is fully described in Chapter 5.
Usage : `pseg [-hfo] <filename.labels>`
The `-f` option activates the edge-smoothing algorithm. Default is off.
The `-o` option defines the path where the ms-seg image is written.

nr Stands for not really RangeSeg.This program is the interface between RangeSeg and the evaluation package for planar data. It acts as a segmentation algorithm, accepting `segmenter -r rangeimage -p paramfile -m msimage` as its input. Internally, it parses the parameter file, then successively runs ABW2Hips on rangeimage, then runs RangeSeg (in text mode) with the specified parameters and the Hips image generated by ABW2Hips, then runs pseg on the resulting .labels file with its output sent to msimage. This pipelining is illustrated in fig. 4.4.

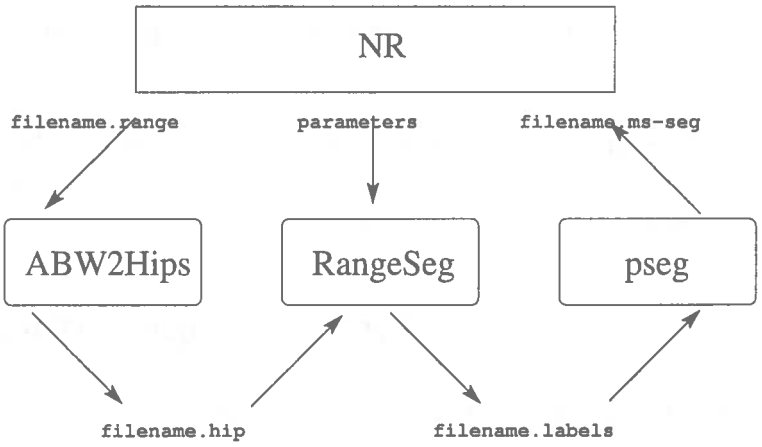

Figure 4.1: The nr interface (planar data)

4.3.2 Cyberware planar data evaluation

CW2Hips This is the program that converts ABW images to Hips, as described in 4.2.3.
Usage : `CW2Hips [-heo] <filename.range>`
The `-e` option is used to define the path to the echo file, if it is not in the same location as the range image.
The `-o` option defines the path where the HIPS image is written to.

CWpseg This is the program that converts an CW-originated machine-segmentation `filename.labels` image to a PGM format `filename.ms-seg` image, as described in 4.2.4. Unlike pseg, CWpseg does not incorporate the edge-smoothing algorithm, which is more adapted to planar data.
Usage: `CWpseg [-ho] <filename.labels>`
The `-o` option defines the path where the ms-seg image is written to.

CWnr This program is the interface between RangeSeg and the evaluation package for curved-surface data. It is analogous to the nr program.

4.4 Parameter selection

Once RangeSeg is set up to run seamlessly with the evaluation framework, the next step is to select the right parameters for evaluation. A similar process is described in [1], where each parameter was selected manually and the most significant parameters were given 2 to 7 different values, so as to find the best combination. Here our work is similar, except that the framework now provides the automated training function. As mentioned already, for the purpose of time

efficiency, we will only train the 2 most significant parameters and manually select values for the others. Here is a list of parameters listed by estimated greatest to least significance (refer to chapter 3 for a detailed description of each parameter):

1. H-K Hi-Lo threshold
2. Minimum fitting residual
3. Morphology schedule
4. Number of smoothings
5. Minimum region size
6. Fold edge threshold
7. Depth discontinuity threshold
8. Number of slurp refinements
9. Aikaike

In the next 2 subsections, we describe how the value for each parameter was chosen, using the visual displays provided by the RangeSeg GUI.

4.4.1 Planar data segmentation

In [1], the following parameters were chosen for RangeSeg using the ABW data: Most significant parameter ranges:

H-K inner threshold: `[.011 .012 .013 .014 .015 .016 .017]`
Minimum fitting residual: `[1.5 2.0 2.5 3.0]`
Morphology schedule: `[+-+ +-++ +-++-]`
Number of smoothings: `[2 3]`
Minimum region size: `[20 25 30]`

Least significant parameters:

Depth discontinuity threshold: `15`
Fold edge threshold: `180`
H-K inner threshold: `infinity`
Eigenratio for plane: `0`
Slurp refinements: `3`

In this evaluation, we decided to use a slightly different set of parameters. Firstly here, it must be noted that RangeSeg was fixed in order to only fit planar surfaces.

This was done by setting the HK inner threshold to infinity, classifying all H values as planar or undefined. Furthermore, setting the Eigenratio for plane fitting to 0 effectively disactivated this parameter. Here, we decided to set both Hi and Lo values to the same value, which is ultimately more organic. Furthermore, the Eigenratio was left to its default value of 250. The morphology schedule was also set to its default value $+-++$ (region dilate, erode, dilate, dilate), which seemed to be the most effective. Again this is an influential parameter, as it defines the size of initial regions to be grown, and other schedules might have been better, but due to its non-numerical, non-linear nature, this parameter was not trained. Whilst in [1] the fold edge threshold is disactivated with a setting of 180 degrees, by inspection of the displays it seemed that a value of 90 detected real fold edges without picking up noise.
Despite the particularly noisy nature of the data (due to the original quantization noise and the subsequent view transformation noise), a value of 2 was chosen for the number of smoothings. The rationale behind this is the lack of fold edge detection; since not all of these are identified with a value of 90, the edge boundaries are not properly detected, and a large amount of diffusion smoothing would erode the data. This underlines a paradox in the method used by RangeSeg: if the data is too noisy, fold edges cannot be reliably identified, hence using a large amount of smoothing (initially used to remove this noise) ultimately blurs the surface boundaries and erodes the data, so a low value is preferable. On the other hand, a low value of smoothing does not effectively remove noise!
As in [1], the depth distance threshold was set to 15. The minimum region size parameter was set to 50, as many small regions are created due to the noisy data. Finally, the number of slurp refinements was set to 2 to speed up execution time.
The values trained were the H Hi-Lo threshold and fitting residual.
The range specified for H Hi-Lo was 0.001 0.04
The range specified for the fitting residual was: 0.6 2.5

4.4.2 Curved-surface data segmentation

The parameter values chosen for the curved-surface segmentation are similar to the ones used in planar, with the exception that the data is considerably less noisy, but also 4 times smaller. Also the curvature H Hi-Lo must be more precise, as we are dealing with both curved and non-curved surfaces. While running tests on data to find the best parameters, a cylinder-fitting bug was detected, wherein a cylinder is only fitted if its length is bigger than its width. In the opposite case, a cylinder-like general quadric is fitted instead, causing a slightly irregular shape being fitted. These are the values that were used for evaluation:

Depth discontinuity threshold: 15

Number of smoothings: 4
Morphology schedule: "+-++"
Eigenratio for the plane: 250
Minimum region size: 50
Number of slurp refinements: 2
Again the parameters trained were the H Hi-Lo threshold and the fitting residual
The range specified for H Hi-Lo was: 0.0001 0.01
The range specified for the fitting residual was: 0.6 2.5

5. Pre and post-processing

In this chapter we consider the effects of pre-processing and post-processing to obtain a better evaluation result. 2 different goals are sought: The first, with pre-processing, is to make the input data as noise-free as possible, maximizing the chances of an accurate segmentation. The second, with post-processing, is to remove obvious noise in the labeling of the segmented data, maximizing the score yielded by the compare tool.

The first section describes a strategy to reduce the noise created by the perspective to orthographic transformation in the ABW data, before RangeSeg processes it. The second section describes a noise removing/edge-smoothing algorithm, which can be used to perfect the labeled data before comparison with ground truth. The positive and negative effects of each technique are covered.

5.1 ABW preprocessing

Inherently, the ABW data, due to its limited color depth of 256 level of gray, presents a certain level of quantization noise $(\max z - \min z)/256$. When the data is converted to an orthographic representation as described in chapter 4, section 2 the rounding of the x and y values creates additional noise of variation 0.5. Although it was not used in this project, this section explains a method to reduce this noise to a minimum.

Rather than round each extracted Cartesian point to an orthographic pixel of coordinates x,y, a much more accurate way of converting the data would be to start from the integer x, y pixel coordinates of the orthographic image, and sample the perspective data at that point. An issue with this is that the perspective data is not continuous, but itself a collection of discrete points. This is not so much a problem as the fact that these points are not uniformly distributed in Cartesian space. Applying the rules of sampling theory, we know that to sample the perspective data accurately for each orthographic pixel (x, y), we need to convolute the perspective points around the orthographic point (x, y) using a pulse function. In practice, this would be used as illustrated in fig 5.1 The orthographic point would be an average of the values of the perspective points, weighted by their level of intersection with the cone function (much easier to wield than a pulse function). This works well in areas where there are many points in the perspective data (the closer points), but badly in areas where the points are sparse (the points further away). To solve this, a threshold should be set where if only a minimal value of weighted point value is sampled, the orthographic pixel should be written as a hole in the data. The holes in the data (typically in the further away regions) could then be filled in using the 8-connectivity hole-filling technique described in chapter 4, section 2.

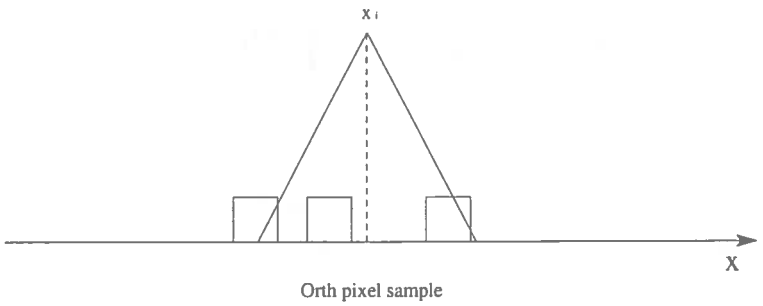

Figure 5.1: Sampling an orthographic pixel from non-uniform perspective data

Although this technique would reduce noise considerably, we chose not to use it, as the ABW image segmentation is already biased by the fact that data must be converted from perspective to orthographic; since the data is transformed, the segmentation performance can only be taken as a approximate result. Furthermore, the added noise provides a good test for measuring the robustness of RangeSeg.

5.2 ABW post-processing

A segmentation image, be it machine segmentation or ground truth, has a characteristic property: it is organized as patches of similarly colored regions; each pixel fitted to a surface will be colored with the surface label. In practice, segmentation of noisy data produces noisy segmentation: boundaries between 2 regions will be rough and occasionally small (less than 30 pixels big) regions spurious may be created. Ultimately, this noise can go against a good performance when the machine segmentation is compared to the ideal ground truth specification. This section presents a post-processing algorithm, which eliminates such noise without eroding the data. If such an algorithm invariably creates better results in evaluation, it can be considered as a useful addition to any segmentation algorithm. A simple noise-reducing algorithm would be to evaluate, for each pixel if it is sufficiently surrounded by pixels of an identical color. If it is, then this pixels color is changed to this surrounding color. This works well to a certain extent, but can result in a deadlock in certain situations see. This problem is fixed by using the concept of dominant label. A first pass of the algorithm counts the amount of instances where a pixel of color A is surrounded by a pixel of color B. this is repeated for each pixel in the scene. At the end of this, we have a N x N table where N is the total amount of labels. For each label, the number of instance where it is surrounded by another label is counted. For 2 adjacent labels, the dominant label is the one that surrounds the other more than it is surrounded. The aforementioned noise-reducing algorithm is then used so: evaluate, for each

pixel, if it is sufficiently surrounded by pixels of an identical color. If it is, and the surrounding color dominates the pixel color, the pixels color is changed to this surrounding color, otherwise, it is unchanged. This strategy avoids deadlocks, and stops spurious regions from growing (these regions are always dominated by the surrounding regions) A problem occurs with corners: these are surrounded but should not be re-colored. A fix to this is based on the observation that the region around the corner is often larger than the corner region. Based on this observation, we can give an advantage to smaller regions, which save corners, but not noise patches. This algorithm is quite effective, but we have chosen not to use it for the training process, Indeed, a noisy segmentation (using a low fitting residual) looks like a non-noisy (using a higher fitting residual) one once the algorithm is used. The training algorithm will not see the difference and as a consequence will keep the low fitting residual.

6. Evaluation results and discussion

Although most of the effort dedicated to this project was put into interfacing Rangeseg and the evaluation framework, the most important aspect is the outcome of the evaluation. This chapter presents these results in a first section, then places them in context by comparing them to previous RangeSeg results as well as results from other segmentation algorithms.

6.1 Results

Two separate evaluations were performed, one on the planar data and the other on the curved surface data.

6.1.1 ABW planar scenes

The ABW data, despite its various degrees of noisiness and data loss, did quite well. The final trained parameters were 0.014 for the H Hi-lo threshold and 1.25 for the fitting residual. the final results for each validation set was (out of 100):

validation set 0 : 74.68
validation set 1 : 66.52
validation set 2 : 75.74
validation set 3 : 76.08
validation set 4 : 75.7147
validation set 5 : 74.4322
validation set 6 : 76.1195
validation set 7 : 80.8344
validation set 8 : 69.0096
validation set 9 : 66.9448

yielding an overall score of 73.602
This relatively high score despite the large quantity of error in the data confirms that RangeSeg is very robust. Further levels of performance was achieved when applying the noise-reducing algorithm on the segmented data:

validation set 0 : 75.2563
validation set 1 : 67.3428
validation set 2 : 75.7393
validation set 3 : 76.2401

validation set 4 : 76.9828
validation set 5 : 75.6755
validation set 6 : 77.6755
validation set 7 : 81.5668
validation set 8 : 69.8280
validation set 9 : 67.5527

yiedling an overall score of 74.381
As we can see, performance goes up by approximately 0.8 points when the post-processing algorithm is used. The individualmachine segmentation images can be found at `amd/partition/u41/s9904184-temp-expansion/RangesegProject/results`

6.1.2 CW curved-surface scenes

Performance of curved-surface scene segmentation was surprisingly quite low. Several factors are present:
a) he data itself was approximate (an arbitrary number, 17, was used to multiply the z values)
b) Rangeseg does not fit torii, yielding unusually high over-segmentation metrics
c)Rangeseg produces a bug in the segmented output, where the rim of the data is not labelled. This adds to degrading comparison performance.
Surprisingly, whilst spheres and cylinders were fitted with low levels of error, no cones were succesfully mapped to the range data, even when the cone-shaped nature of the data was quite explicit.
A further, and in fact simple, explanation to the poor results is that the data itself

a) is too complex (many real-life objects are used)
b) has unrealistically specified ground truth; the walls in the background do not always consist of a single plane, as we are laed to believe. Even very high fitting residual settings do not manage to segment some walls as a single plane.
The final trained parameters were 0.004 for the H Hi-lo threshold and 2 for the fitting residual. These are the scores for the various validation sets:

validation set 0: 42.9955
validation set 1: 49.1331
validation set 2: 42.2055
validation set 3: 43.9903
validation set 4: 39.7955
validation set 5: 42.3850
validation set 6: 39.7547
validation set 7: 47.8277

validation set 8: 44.1752
validation set 9: 37.6568

The average score was 42, 91

6.2 Comparison

In this section, we compare our current performance results to those of the baseline University of Bern (UB) segmenter, and also to previous results obtained by rangseg on the ABW data

6.2.1 Baseline algorithms

The UB "baseline" segmenters (they are, along with RangeSeg, the most effective segmenters around) yield the following results on Planar and curved-surface data: UB planar data algorithm

validation set 0: 79.9708
validation set 1: 76.5129
validation set 2: 81.7753
validation set 3: 80.5969
validation set 4: 79.2726
validation set 5: 82.7415
validation set 6: 81.5183
validation set 7: 82.4716
validation set 8: 78.0811
validation set 9: 80.8150

overall score 80, 3

UB planar curved data algorithm

validation set 0: 50.2382
validation set 1: 52.6223
validation set 2: 67.6987

validation set 3: 46.6517
validation set 4: 40.9566
validation set 5: 52.0409
validation set 6: 55.5572
validation set 7: 58.7663
validation set 8: 44.9967
validation set 9: 52.5342

overall score 52,2

As we can see, both of these yield better performances. Whereas the Curved-surface algorithm beats RangeSeg by a solid 10 points, it still is far from performing at the same level as its Planar counterpart. This further confirms that the Cyberware data is simply too complex and at times unrealistically specified. Interestingly, both Rangeseg and UBC received similar scores (around 44) for validation set 8.

RangeSegs Planar data segmentation is beaten by only 6 points, which further underlines RangeSegs robust performance on the "deteriorated" ABW images. Whether this is due to Rangesegs new Euclidean fitting can only be found out by running the ABW data intact and comparing results to RangeSegs last run on that data.

6.2.2 Rangeseg PAMI 96 results

In [1] a different system for measurement, taking in account all 5 fitting criteria was used, neverthelss, we know that RangeSeg performed slightly better than the UB Planar algorithm, which has not been changed, so one can imagine that this score was anywhere between 80 and 85. As described in chapter 4, the parameters used, had "fixed" Rangeseg to only accept planar data. Perhaps using such a strategy this time round would have collected a couple more points...

7. Conclusion

Several conclusions can be made for this project: firstly on RangeSegs performance: During the adaptation process, both planar and curved-surface data lost a certain level of information (be it because of erosion, in the case of the ABW data, or distortion, in the case of the CW data). Therefore, the results cannot be truly compared with other performances on the intact data. Nevertheless, we can still observe some trends in RangeSegs surface-fitting. We already know that RangeSeg performed well on the "intact" ABW data in [1], it is interesting to see that Rangeseg also performs well on a seriously noisier version of the same data. For many reasons, namely the combined complexity and small resolution of the data, RangeSeg performed quite poorly on the curved data. Still, some trends in RangeSegs behaviour were identified, such as a cylinder-fitting bug (cylinders are only fit if their length is greater than their width), a general reticence in fitting cones, and the lack of torii in the surface -fitting. These are all problems that can be worked on in future implementations of Rangeseg.
We can also discuss the usefulness of post-processing the labelled segmentation data; As we saw, using such an algorithm improves the measured performance; should such techniques generally be added to range data segmentation algorithms? If this kind of hole-filling and smoothing of edges can make a labelling of data more accurate, can it useful for specific applications? Although it has no impact on potential applications such as extracting models from the range data (the only important output here is the surface equations), its main selling point is that is makes the labelled data look more coherent to the user. Finally, although it improves performance in evaluation, this sort of algorithm can only serve to hide the true effectiveness of a segmentation algorithm, so ultimately, its usefulness is limited.

Future work on this project might involve performing actual modifications on RangeSeg so that it could read point coordinates from various types of data directly without the damaging side-effects of image conversion. A suitable system would be to implement a plugin system, where a module can be written for each type of range data. Each module would deal with the specifics extracting the correct Cartesian coordinates, as well as provide some other information such as the intial projection type, data maximum and minimum, etc. Plugins specifiying different types of range data could be written then seamlessly added.This System would counter the currently quite rigid HIPS-only image input in Rangeseg.
Further work would also involve responding to the problems which were identified with Rangeseg segmentation; cone and cylinder fitting would have to be tweaked, and torii fitting using least-squares implemented. Could maybe also be worked on is the inital boundary-preserving smoothing technique, whose effectiveness on removing noise from the data without eroding it is paradoxically a factor of the

existing noise level.

As far as further evaluation is concerned, more parameters could be trained simultaneously if time was not an issue. With the advent of RangeSegs new X-windows implementation, much faster machines can be used, so there is scope for simultaneously training 4 or even 5 parameters at once. A suitable solution for training the morphology schedule, an important parameter could also be figured out. Training a large number parameters at once can be interesting as it may yield some unintuitive yet powerful combinations.

Another area that can be explored is evaluating RangeSeg with other types of test data, such as K2T and Perceptron, which have been much more widely tested on other segmentation algorithms. (It seems the Cyberware data set is less popular due to its complexity and unrealistic ground truth specifications)

Bibliography

[1] An experimental comparison of range image segmentation algorithms, 1996. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol 18(7).

[2] K. Bowyer J. Min, M. Powell. Automated performance evaluation of range image segmentation, 2000.

[3] R. Fisher P. Faber. A buyer's guide to euclidean elliptical cylindrical and conical surface fitting, 2001. Proc. British Machine Vision Conference BMVC01 ,pp 521-530.

[4] M. Powell. Comparing curved-surface range image segmenters, 1997.