Range synthesis for 3D Environment Modeling

Luz Abril Torres Méndez

August 13, 2002

Problem statement

In this technical report, we present ongoing work to the problem of range synthesis for 3D environment modeling. We propose an efficient algorithm for extrapolating sparse range data to provide a dense 3D environment model. The extraction of range data from intensity images has been one of the key problems for computer vision, as addressed by numerous “shape-from” methods. Many of these methods provide incomplete data. More prosaically, while laser range sensors (for example based on LIDAR) have become well-established technology they are costly and often return data that is sparse relative to video images (most typically providing only dimensional-stripes of range data). In robotics, for example, the use of range data for navigation and mapping has become a key methodology, but it is often hampered by the fact that range sensors that provide complete (2-1/2D) depth maps with a resolution akin to that of a camera, are prohibitively costly.

The method

Our method for range synthesis consists in using a statistical learning method for interpolating/extrapolating range data from only one intensity image and a relatively small amont of range data and without strong *a priori* assumptions on either surface smoothness or surface reflectance.

We base our range estimation process on inferring a statistical relationship between intensity variations and existing range data. This is elaborated using Markov Random Field (MRF) methods akin to those used in texture synthesis.

A single range value at a time is estimated where only intensity is known. This is done by using neighboring locations where both range and intensity data is available.

Assumptions:

- The intensity and range data is already registered
- The sampling of both images is the same.

The Algorithm

We use Markov Random Fields (MRF) as a model that captures characteristics of the relationship between intensity and range data in a neighborhood of a given voxel, i.e. the data in a voxel are determined by its immediate neighbors (and prior knowledge) and not on more distant voxels (the locality property). The other property that we exploit is limited stationarity, i.e. different regions of an image are always perceived to be similar. This property is true for textures but not for more general classes of images representing scenes containing one or more objects. In our algorithm, we synthesize a depth value so that it is locally similar to some region not very far from its location. The process is completely deterministic, meaning that no explicit probability distribution needs to be constructed.

Synthesizing range

We focus on our development of a set of **augmented voxels V** that contain intensity and range information (where the range is initially unknown for some of them). Thus, $\mathbf{V} = (\mathrm{I}, \mathrm{R})$, where I is the matrix of known pixel intensities and R denotes the matrix of incomplete pixel depths. We are interested only in a set of such augmented voxels such that one voxel lies on each ray that intersects each pixel of the input image I, thus giving us a registered range image R and intensity image I.

Let $Z_m = (x, y) : 1 \leq x, y \leq m$ denote the $m \times m$ integer lattice (over which the images are described); then $\mathrm{I} = \{I_{x,y}\}$, $(x, y) \in Z_m$, denotes the gray levels of the input image, and $\mathrm{R} = \{R_{x,y}\}$, $(x, y) \in Z_m$ denotes the depth values. We model $\mathbf{V}$ as an MRF. Thus, we regard I and R as a random variables. For example, $\{\mathrm{R} = r\}$ stands for $\{R_{x,y} = r_{x,y}, (x, y) \in Z_m\}$. Given a *neighborhood system* $\mathcal{N} = \{\mathcal{N}_{x,y} \in Z_m\}$, where $\mathcal{N}_{x,y} \subset Z_m$ denotes the neighbors of (x, y), such that, (1) $(x, y) \not\in \mathcal{N}_{x,y}$, and (2) $(x, y) \in \mathcal{N}_{k,l} \Longleftrightarrow (k, l) \in \mathcal{N}_{x,y}$. An MRF over $(Z_m, \mathcal{N})$ is a stochastic process indexed by Z_m for which, for every (x, y) and every $v = (i, r)$ (i.e. each augmented voxel depends only on its immediate neighbors),

$$\begin{aligned} &P(V_{x,y} = v_{x,y} \,|\, V_{k,l} = v_{k,l}, (k, l) \neq (x, y)) \\ &\quad = P(V_{x,y} = v_{x,y} \,|\, V_{k,l} = v_{k,l}, (k, l) \in \mathcal{N}_{x,y}), \end{aligned} \tag{1}$$

The choice of $\mathcal{N}$ together with the conditional probability distribution of $P(\mathrm{I} = i)$ and $P(\mathrm{R} = r)$, provides a powerful mechanism for modeling spatial continuity and other scene features. On one hand, we choose to model a neighborhood $\mathcal{N}_{x,y}$ as a square mask of size $n \times n$ centered at the voxel location (x, y). This neighborhood is causal, meaning that only those voxels already containing both, intensity and range information are considered for the synthesis process. On the other hand, calculating the conditional probabilities in an explicit form is an infeasible task since we cannot efficiently represent or determine all the possible combinations between augmented voxels with its associated neighborhoods. Therefore, we synthesize a depth value $R_{x,y}$ deterministically by selecting the range value $R_{k,l}$ from the augmented voxel whose data most resemble the (partial) data from location (x, y), i.e.,

$$\underset{(k,l) \in \mathcal{A}}{\textbf{argmin}} \ ||V_{x,y} - V_{k,l}||, \tag{2}$$

where $\mathcal{A}$ is the set of those augmented voxels located at distance d to the augmented voxel to be synthesized. The similarity measure $||.||$ is described over the partial data about locations (x, y) and (k, l) and is calculated as follows,

$$\sum_{\vec{v} \in \mathcal{N}} G(\sigma, \vec{v} - \vec{v}_0)[(I_{\vec{v}} - I'_{\vec{v}})^2 + (R_{\vec{v}} - R'_{\vec{v}})^2], \tag{3}$$

where $\vec{v}_0$ is the voxel located at the center of the neighborhood $\mathcal{N}$, $\vec{v}$ is a neighboring voxel of $\vec{v}_0$ that contains both intensity and range information. I and R are the intensity and range values of the neighboring voxels of the depth value $R_{x,y} \in \vec{v}_0$ to synthesize, and I' and R' are the intensity and range values to be compared with and in which, the center voxel $\vec{v}_0$ has already assigned a depth value. G is a 2-D Gaussian kernel that gives more weight to those pixels near the center than those at the edge of the window.

In our algorithm we synthesize one depth value at a time. The order in which we choose the next depth value to synthesize will reflect the final result. We have implemented different scan orderings for assigning depth values. These are:

- Spiral-scan ordering (growing inwards or outwards)
- Based on the maximum number of already assigned neighboring voxels. The selection among those voxels having the same number of neighbors is done:
  - Sequentially (left-to-right, top-to-bottom)
  - Randomly

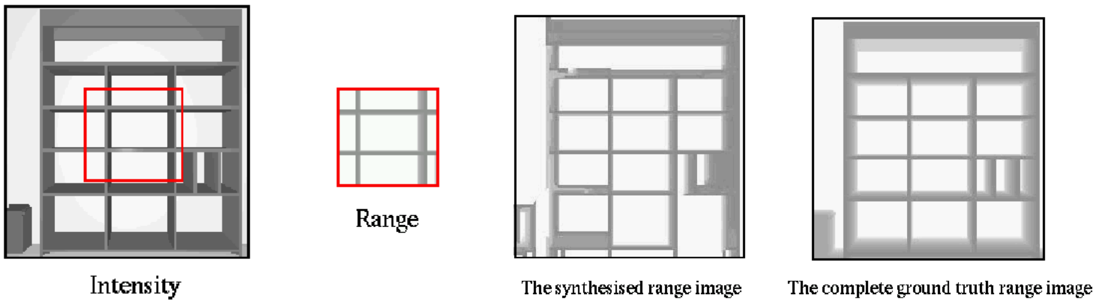
Part I

Range synthesis using only already existing range values

In this initial work we estimate the depth values where missing by using the already existing range values. Although this does not always represent the true range, we have found that our experimental results are quite satisfactory.

Experimental Results

We have tested our algorithm using synthetic and real data. The synthetic data were generated with the 3D rendering package PovRay using natural lighting conditions. In the left side of Figure 1a, a synthetic intensity image of an empty bookshelf is shown; the subwindow in the right side is the associated range image taken from the position indicated by the red rectangle in the intensity image. These images are given as an input to our algorithm. For this case, the size of the neighborhood is set to be 9×9 pixels. The left side of Figure 1b shows the range synthesis results, and to the right, as a way of visual comparison, the complete synthetic range data is shown. It can be seen that the algorithm captures most of the changes involved in the intensity information and are reflected in the range synthesis process.



(a) Input. (b)

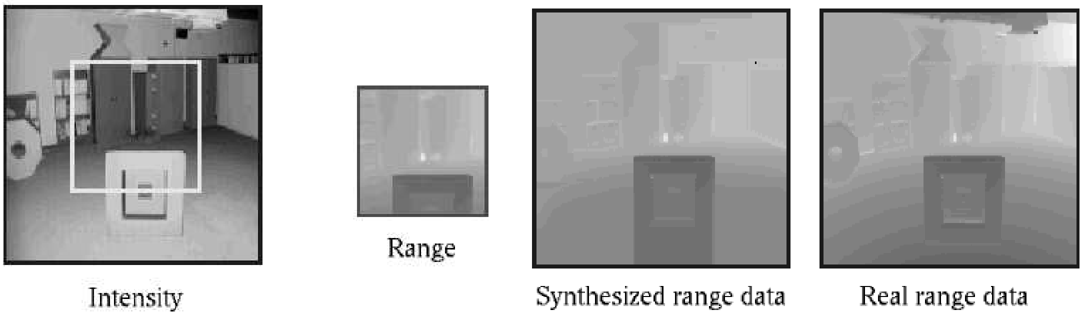
Figure 1: Results of range synthesis on synthetic data. Panel (b) shows the comparison of synthesized range with ground truth for this artificial scene.

Representative results based on data acquired in a real-world environment are shown in Figures 2 through 9. The real intensity (reflectance) and range images of indoor scenes were acquired by an Odetics laser range finder mounted on a mobile platform. Images are 128×128 pixels and encompass a $60^o \times 60^o$ field of view. As with the synthetic data, we start with the complete range data set as ground truth and then hold back most of the data to simulate the sparse sample of a real scanner and to provide input to our algorithm. This allow us to compare the quality of our reconstruction with what is actually in the scene. In the following we will consider two strategies for subsampling the range data.

Limited dense range

The first type of experiment involves the range synthesis when the initial range is a window of size $p \times q$ and at position (r_x, r_y) on the intensity image. Figure 2a shows the intensity image (left) of size 128×128 and the initial range (right), a window of size 64×64, i.e. only the 25% of the total range is known. The size of the neighborhood is 5×5 pixels. The synthesized range data obtained after running our algorithm is shown in the left side of Figure 2b; for purposes of comparison, we show the complete real range data (right side).

It can be seen that the synthesized range is very similar to the real range. The Odetics LRF uses perspective projection, so the image coordinate system is spherical. To calculate the residual errors, we first convert the range images to the Cartesian coordinate system (range units) by using the equations in [**?**]. For this example, the average residual error is 7.98.

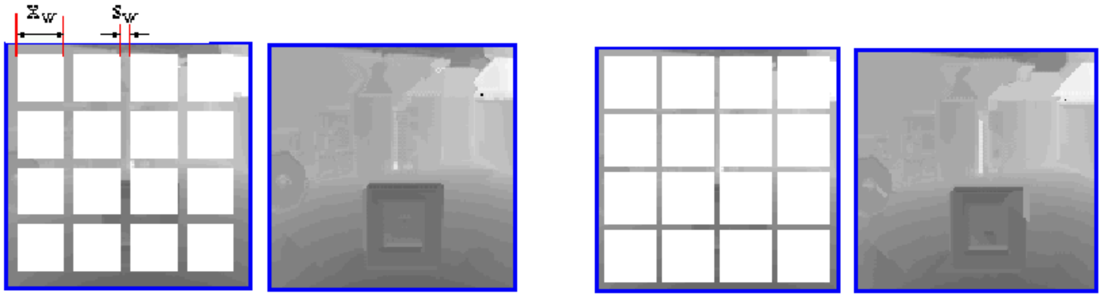


(a) (b)

Figure 2: Results on real data. (a) Input. (b) Results comparing synthesized range data to ground truth.

Sparse range measurements

In the second type of experiment, the initial range data is a set of stripes with variable width along the $x-$ and $y-$axis of the intensity image. We tested with the same intensity image used in the previous section in order to compare both results. Two experiments are shown in Figure 3. The initial range images are shown in the left column, and to their right are synthesized results. In Figure 3a, the width of the stripes s_w, is 5 pixels, and the area with missing range data ($x_w \times x_w$) is 25×25, i.e., 39% of the range image is known. For Figure 3b, the values are $s_w = 3$, $x_w = 28$, in this case, only 23% of the total range is known. The average residual error (in range units) for the reconstruction are 2.37 and 3.07, respectively. In Figure 4 a graph of the density of pixels at different depth values (scale from 0 to 255) of the original and synthesized range of Figure 3a. Figure 5 displays two different views using the real range and the synthesized range results of Figure 3.



(a) $s_w = 5$, $x_w = 25$. (b) $s_w = 3$, $x_w = 28$.

Figure 3: Results on real data. The left column shows the initial range data and to their right is the synthesized result (the white squares represent unknown data to be estimated). Since the unknowns are withheld from genuine ground truth data, we can estimate our performance.

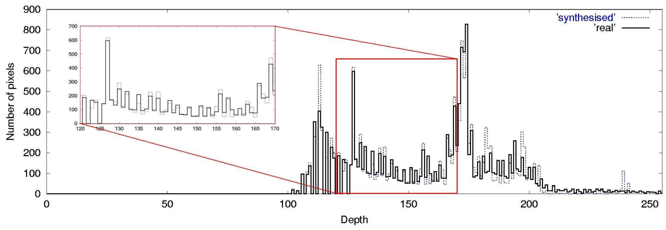


Figure 4: Histogram of pixels at different depth values (scale from 0 to 255) of the original and synthesized range of Figure 3a.

The results are surprisingly good in both cases. Our algorithm was capable of recovering the whole range of the image. We note, however, that results of experiments using stripes are much better than those using a window as the initial range data. Intuitively, this is because the sample spans a broader distribution of range-intensity combinations than in the local window case.

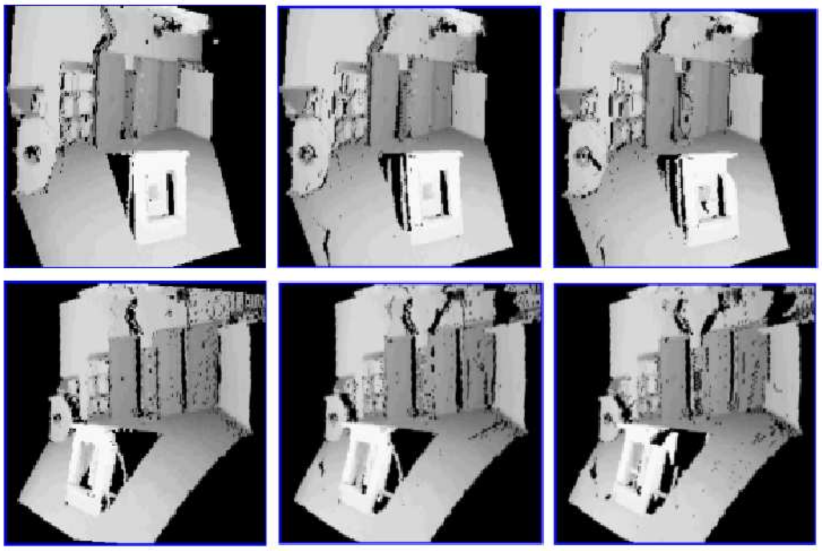
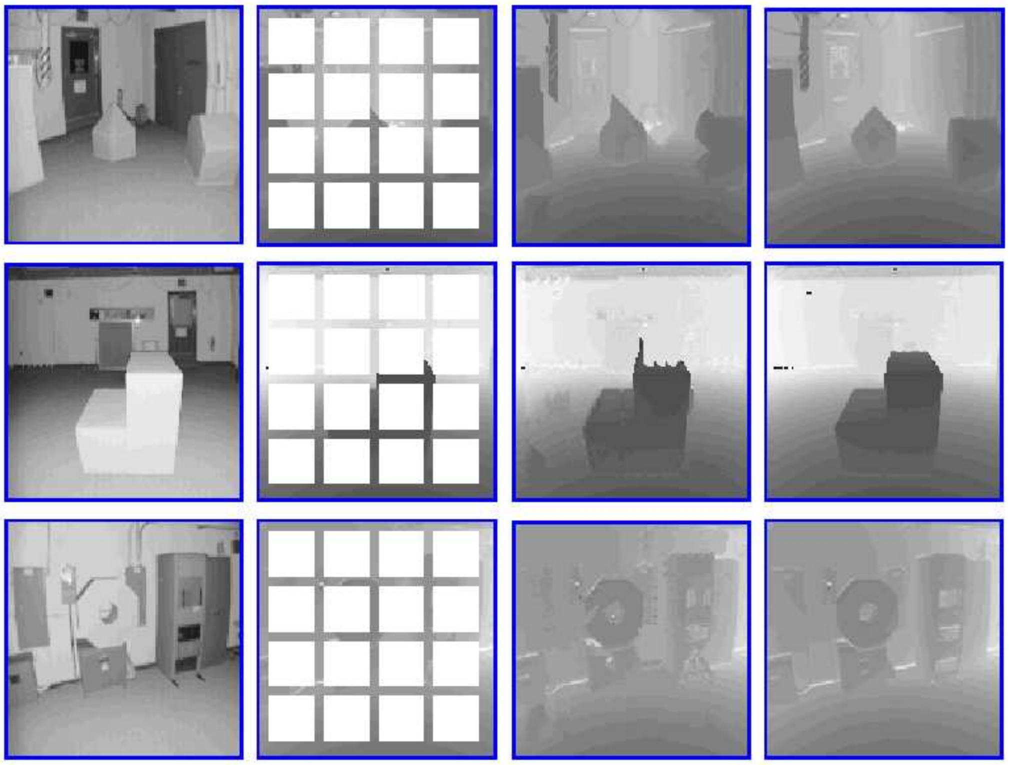


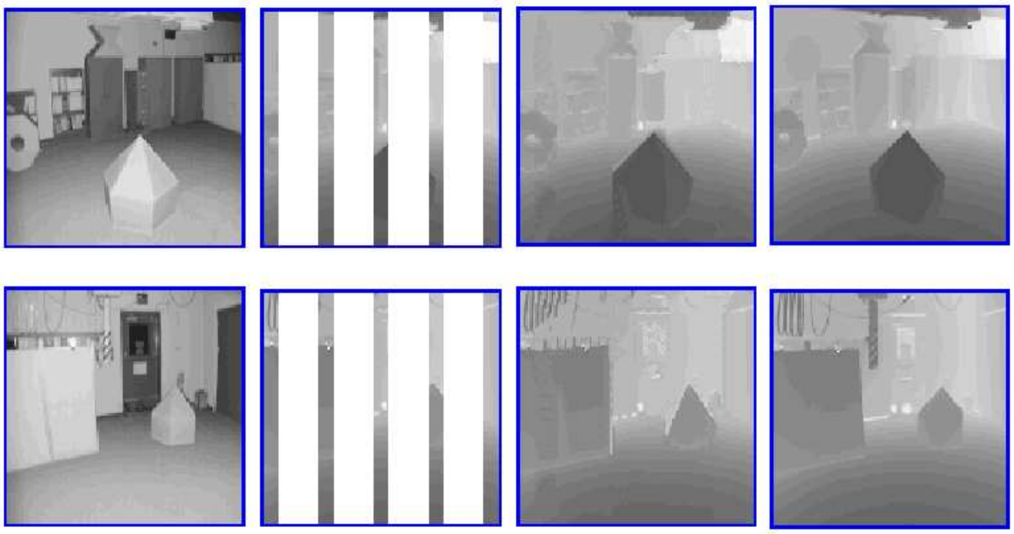
Figure 5: Results in 3D. Two views of the real range (left column) and the synthesized results (middle and right columns) of Figure 3.

Our algorithm was tested on 30 images of common scenes found in a general indoor man-made environment. Two cases of subsampling were used in our experiments. Case 1 is as one of the subsampling previously described, with $r_w = 5$ and $x_w = 25$, applied along $x-$ and $y-$axis. Due to space limitations, we are only showing 3 more examples of this case in Figure 6a, the average residual errors are, from top to bottom, 2.84, 4.53 and 3.32. For Case 2, $r_w = 8$ and $x_w = 22$, but applied only along the $x-$axis. Figure 6b shows 2 examples of this case. Here the average residual errors are 4.17 and 5.25, respectively. Once again, it can be seen that the results are good in both cases. The maximum average residual errors obtained from all 30 test images were for Case I, 6.52 and for Case II, 11.85.

It is important to note, that the initial range data given as an input is crucial to the quality of the synthesis, that is, if no interesting changes exist in the range and intensity, then the task becomes difficult. However, the results presented here demonstrate that this is a viable option to facilitate environment modeling.



(a) Case 1: $r_w = 5$, $x_w = 25$.



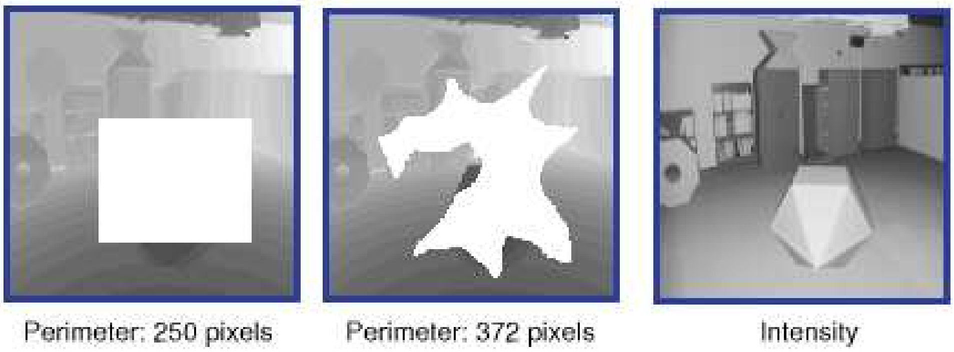
(b) Case 2: $r_w = 8$, $x_w = 22$ along the x-axis.

Figure 6: Examples on real data. The first and second columns are the input intensity and range data, respectively. White regions in the input data are unknown data to be inferred by the algorithm. The synthesized results are shown in the third column and, the real range images are displayed in the last column for visual comparison.

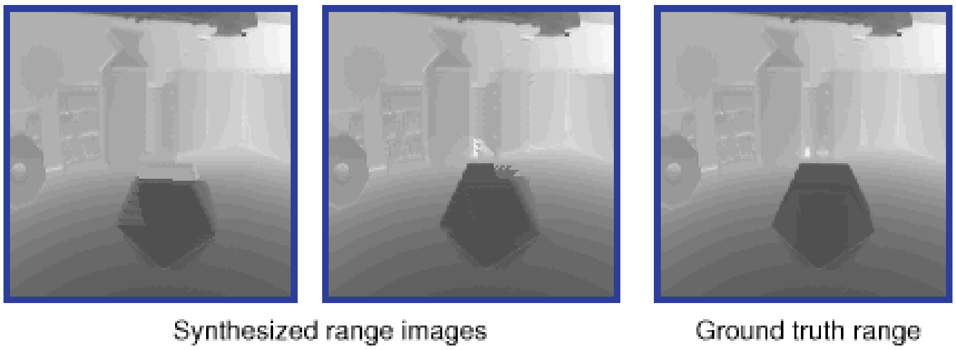
Arbitrary shape of missing range area

We now show some experiments where the missing range is of arbitrary shape. The input parameters are the left superior coordinates and the size in $x-$ and $y - axis$ of the maximum rectangle that surrounds the arbitrary shape.

Our hypothesis is that the less compact the shape that contains the missing range the better the range estimation. We show this in the following figures. In Figure 7a, two input range images (the left and middle images) and input intensity are given. Both range images have the same area, which is 3864 pixels^2, but different shapes and perimeters, 250 and 372 pixels, respectively. Figure 7b shows the synthesized range images (left and middle) and the ground truth range image for comparison purposes. The residual average errors are 7.17 and 3.99, respectively.



(a) Input (the white areas represent unknown data to be estimated).

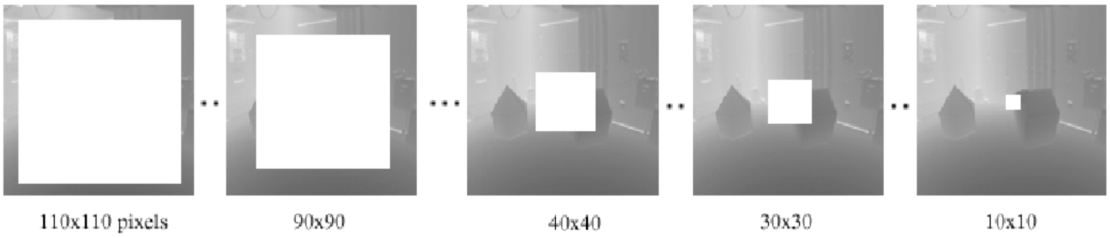


(b) Results.

Figure 7: Results on two different shapes of missing range with same area 3864 pixels^2.

How the amount of the initial range affects the range synthesis process

We now show how the amount of input range affects the estimation of range where missing. In Figure 8a we show some of the input range images, the white squares represent the area of missing range, which varies from 12100 (110×110) to 100 pixels[2]. The synthesized results are shown in Figure 8b.



(a) Input.

(b) Synthesized range data.

Figure 8: Results showing how the amount of initial range affects the range synthesis process.

We calculate the residual average error of each synthesized image but only in a fixed area in order to be able to evaluate the range synthesis process. This fixed area is the 10×10 square show in the rightmost image of Figure 8a. In Figure 9, we plot the residual average error of each synthesized image versus the amount of missing range.

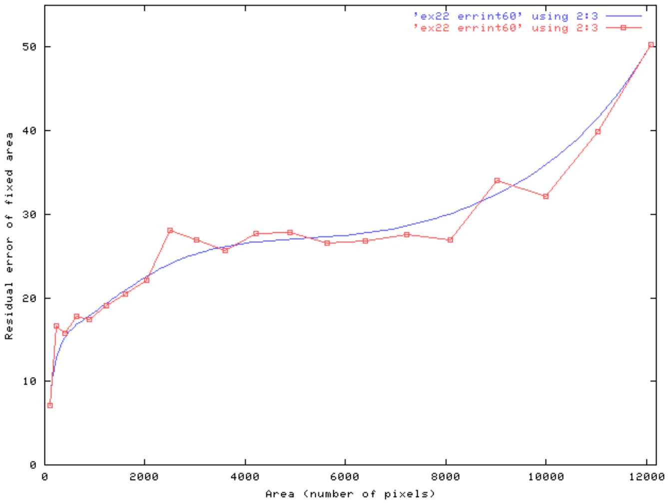


Figure 9: A graph showing how the amount of the missing range affects the range synthesis process.

Observations and aspects to consider

Size of the mask (neighborhood) The size of the neighborhood plays an important role in the range synthesis process. If it is too small then we may find many similar neighborhoods since it does not capture much about the relationship between the intensity and range. If it is too large then it may be difficult to find a similar neighborhood at all, since we are dealing with multiple objects of different sizes in the scene. It is also important to consider the amount of range that is available initially. In cases where we have a very sparse range available, the size of the mask will be set by the width of the stripes of available range, as were the case in the experiments shown.

Size of the area where looking for the best match This is another aspect that needs to be considered. Neigborhoods that are located at a close distance from the voxel to be synthesized are more probable to contain similarity information in both intensity and range than neighborhoods located far away. This parameter will help to reduce the running time of our algorithm.

Order of the estimation (spiral, maximum number of neighbors) As it was already mentioned, the order in which the depth values are assigned influences in the quality of the estimation process.

Confidence of the estimation

The estimation of range values Currently, the estimation range values is done by selecting the range value of that voxel with the most similar neighborhood to the voxel to be synthesized. However this range value may not suitable for cases in which the normal direction in the $z-axis$ of the surface containing the voxel to be synthesized is one. In the next Part, we are dealing with this problem.

List of the main programs, classes and subroutines

Name: PartI.cpp

Description: This is the main program for Part I: Range synthesis from already existing range values.

Sintaxis:

%p1 -m[#] [i/r] [fpn] [rw_x] [sw_x] [rw_y] [sw_y] [dmin] st

%p1 -m[#] [i/r] [fpn] [R_x] [R_y] [w_s] [h_s] [dmin] ms

%p1 -m[#] [i/r] [fpn] [min_x] [$width_x$] [min_y] [$height_y$] [dmin] as

%p1 -m[#] [i/r] [fpn] -n[#] [rx_1] [ry_1] [wx_1] [hy_1] ... [rx_n] [ry_n] [wx_n] [hy_n]

where,

-m[#] mask size (spatial neighborhood).

[i/r] intensity or range

[fpn] file prefix name, for example: ex10, ex22, etc.

[rw_x]/[rw_y] the width of stripe with known intensity/range along the $x-$ and $y-$axis, respectively.

[sw_x]/[sw_y] the width of stripe with unknown intensity/range along the $x-$ and $y-$axis, respectively.

[min_x]/[min_y] The minimum coordinates in (x, y) of the arbitrary shape to synthesize.

[$width_x$]/[$height_y$] The maximum width and height of the arbitrary shape.

$[R_x]/[R_y]$ The initial position in (x, y) of the moving subwindow to synthesize.

$[w_s]/[h_s]$ The width and height of moving subwindow.

$[dmin]$ the minimum distance to consider a voxel's neighborhood (explain more).

st/as/ms the type of unknown range: st-stripes, as-arbitrary shape, ms-moving subwindow.

-n[#] number of subwindows with different size to be synthesize in one image.

$[rx_1]/[ry_1]...[rx_n]/[ry_n]$ The initial position in (x, y) of each subwindow.

$[sx_1]/[sy_1]...[sx_n]/[sy_n]$ The width and height of each subwindow.

Name: StageM1.h

Description: This class contains all the main procedures and subroutines for formating the input intensity and missing range according to its shape (subwindow, stripes or arbitrary). Depending on the shape of missing range is the procedure selected.

Class procedures and their descriptions:

MovingSubwindow(.) The missing range is a subwindow of size $w_s \times h_s$ which is moving inside the main image. The start position is at (Rx, Ry). The increment along the x-axis is w_s, and along the y-axis is h_s.

StripesShape(.) The area of missing range is a set of stripes. It can be either along both the x- and y- axis or along only one of them. The width of the stripes is defined by rw and the width of the area to estimate is s_w.

ArbitraryShape(.) The area of missing range is of arbitrary shape. Its limit coordinates in the x- and y-axis are given as an input.

Name: MaskM1.h

Description: This class contains all the main procedures and subroutines for the estimation of missing data. Depending the scan ordering type (spiral, max. number of neighbors in sequential or random order) is the procedure selected:

Class procedures and their descriptions:

SpiralOutwards(.) The estimation is done in spiral-scan order going outwards, meaning that the missing range is all around a given subwindow. It starts at the left superior corner and going to the right.

SpiralInwards(.) The estimation is done in spiral-scan order going inwards, meaning that the missing range is a closed shape. It starts at the left superior corner and going to the right.

MaxNumNeighb(.) The estimation is done by assigning first those voxels which have the maximum number of neighbors already assigned. The selection among those voxels with the same number of neighbors can be done sequentially (left to right and top to bottom) or randomly. This is specified in *type-sqrnd* parameter. The procedure works only for closed shapes with missing range, where the estimation must be done inwards.

Subroutines for the range estimation:

BestMatch() This subroutine is called for the above procedures and finds the best match by comparing the current mask (neighbourhood) values with all the existing neighborhoods.

GetNm(.) It assigns the voxel values to N_m of the current neighborhood with center voxel at (v_x, v_y).

GetNsM1(.) It gets all the neighborhoods to be compared to N_m.

Similarity(.) It calculates the similarity between two neigborhoods. Similarity is calculated by using the least squares measure between the voxels of each neighborhood.

MatchValue(.) It finds the best match value.

SetGaussian(.) It assigns a 2D Gaussian kernel to each mask. It starts at 1.0 and decreases by $1/maskhalf$.

Part II

Range synthesis using surface normal information

As it was described in Part I, we were not assigning new depth values to the voxels with missing range, i.e. only the already existing values were considered in the algorithm. However, if we want to have a realistic range synthesis, we need to consider additional information to be able to compute new depth values based on the available information. In Part II, we now consider the surface normals as additional information. Surface normals give the orientation of a given surface and if there exists a voxel which belongs also to this surface, but its depth value is missing, then we can use its surface normal to estimate its new depth value.

Computing the surface normals

We use the eigenvalue method to compute surface normals. In this method, we find the eigenvectors and their correspondent eigenvalues of a surface patch. The two eigenvectors corresponding to the two largest eigenvalues are the two tangent vectors which determine the tangent plane to the surface patch, while the eigenvector corresponding to the smallest eigenvalue is pointing in the normal direction.

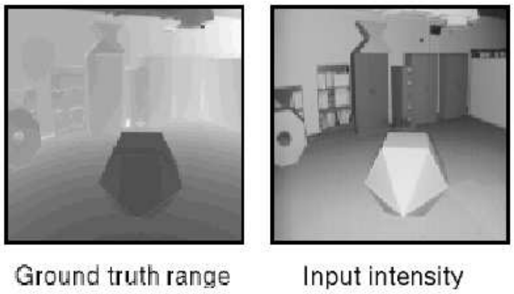
Experimental results

For comparison reasons, we are using the same set of range and intensity data and carrying on the same type of experiments as in Part I, with the only difference that now we are estimating first the surface normals and then the new depth values.

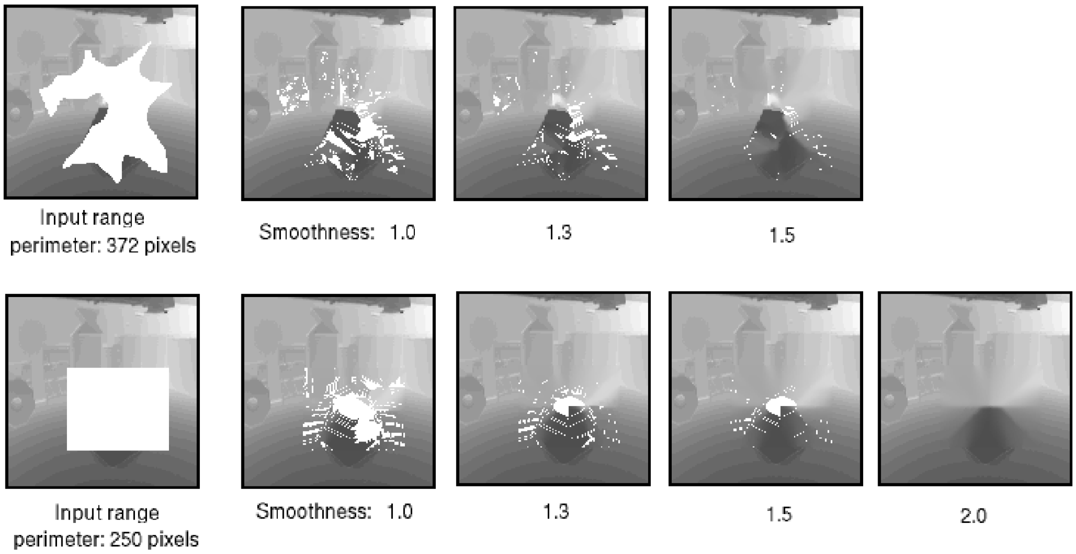
Arbitrary shape of missing range area

We now show some experiments where the missing range is of arbitrary shape. The input parameters are the left superior coordinates and the size in $x-$ and $y-axis$ of the maximum rectangle that surrounds the arbitrary shape.

Figure 10a shows the ground truth range and input intensity images. In Figure 10b, two input range images are shown in the left column. Both missing (white) areas have the same area, which is 3864 pixels^2, but different shapes and perimeters, 250 and 372 pixels, respectively. The synthesized images are shown in the rest of the columns, each of them with its respective smoothness value used in the calculation of the new depth values. (explain later) Another example is shown in Figure 11. In this case the missing (white) areas have an area of 1824 pixels^2, and perimeters of 222 and 122 pixels, respectively.

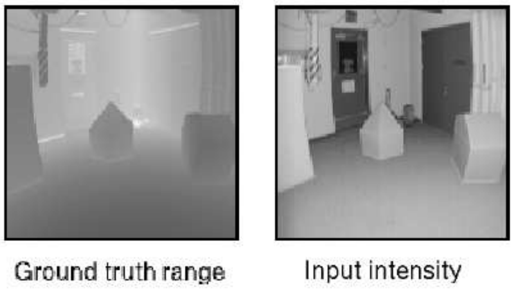


(a)

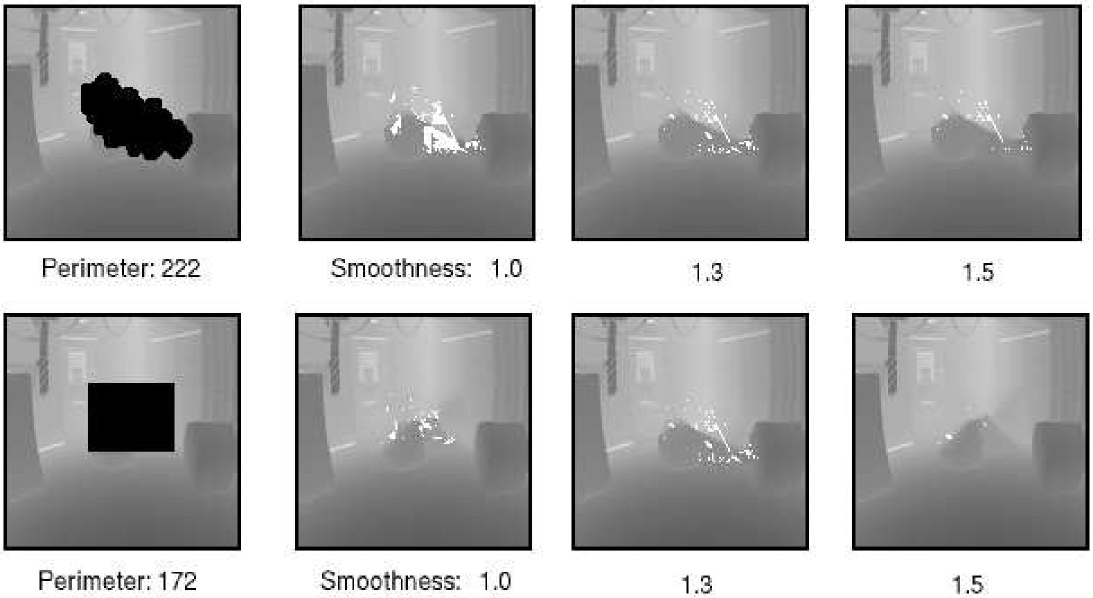


(b)

Figure 10: Results on two different shapes of missing range with same area 3864 pixels^2.



(a)



(b)

Figure 11: Results on two different shapes of missing range with same area 1824 pixels^2.

Observations and Important aspects to consider

Basically, all aspects mentioned in **??** of Part I are also important in this Part II, plus the following:

Size of the cluster (mask) used for calculating the normals

The co-normality threshold ($alpha$) When calculating the surface normal and when doing the normal estimation.

Calculation of the new depth values

Programs, classes and their descriptions

Name: PartII.cpp

Description: This is the main program for Part II: Range synthesis using surface normal information.

Sintaxis: Basically the same as in PartI.cpp.

Name: StageM2.h

Description: This class contains all the main procedures and subroutines for formating the input intensity and missing range according to its shape (subwindow, stripes or arbitrary). Depending on the shape of missing range is the procedure selected. It also has the procedures to calculate the surface normals and the new depth values.

Class procedures and their descriptions:

MovingSubwindow(.) The missing range is a subwindow of size $w_s \times h_s$ which is moving inside the main image. The start position is at (Rx, Ry). The increment along the x-axis is w_s, and along the y-axis is h_s.

StripesShape(.) The area of missing range is a set of stripes. It can be either along both the x- and y- axis or along only one of them. The width of the stripes is defined by rw and the width of the area to estimate is s_w.

ArbitraryShape(.) The area of missing range is of arbitrary shape. Its limit coordinates in the x- and y-axis are given as an input.

CalcNormalsCaseI(.) It computes the surface normals of the resulting image from Part I.

CalcNormalsCaseII(.) It computes the surface normals of the initial range image, i.e. the range image with little associated range.

CalcNewDepthValues(.) It computes the new depth values in the missing areas by using the available surface normals previously estimated.

Name: NormalsClass.h

Description: This class contains all the main procedures to compute the surface normals of a given image according to its available information. The input is an array containing the 3D coordinates of a range image. The output is a vector $N[n_x, n_y, n_z]$ associated to each point containing the surface normal along each axis. In case the surface normal was not possible to compute, this vector N contains $[255, 255, 255]$.

Class procedures and their descriptions:

Clusters(.) It arrranges the 3D points in clusters of size $MaskN \times MaskN$. The clusters are assigned by getting patches of size $MaskN \times width$.

Obtlpp(.) It test if the center pixel of a given cluster is a local planar point (lpp). It calls the subroutine *Eigen()* which returns the smallest eigenvalue according the plane fitness measure $Pthr$.

Normalizing(.) It normalizes the normal directions n_x, n_y, n_z. We consider n_z always positive (outside the plane).

Name: MaskM2.h

Description: This class contains all the main procedures and subroutines for the estimation of surface normals where missing. Depending the scan ordering type (spiral, max. number of neighbors in sequential or random order) is the procedure selected:

Class procedures and their descriptions:

SpiralOutwards(.) The estimation is done in spiral-scan order going outwards, meaning that the missing range is all around a given subwindow. It starts at the left superior corner and going to the right.

SpiralInwards(.) The estimation is done in spiral-scan order going inwards, meaning that the missing range is a closed shape. It starts at the left superior corner and going to the right.

MaxNumNeighb(.) The estimation is done by assigning first those voxels which have the maximum number of neighbors already assigned. The selection among those voxels with the same number of neighbors can be done sequentially (left to right and top to bottom) or randomly. This is specified in *type-sqrnd* parameter. The procedure works only for closed shapes with missing range, where the estimation must be done inwards.

Subroutines for the surface normal estimation:

BestMatch() This subroutine is called for the above procedures and finds the best match by comparing the current mask (neighbourhood) values with all the existing neighborhoods.

GetNm(.) It assigns the voxel values to N_m of the current neighborhood with center voxel at (v_x, v_y).

GetNsM1(.) It gets all the neighborhoods to be compared to N_m.

Similarity(.) It calculates the similarity between two neigborhoods. Here, the similarity measure is based on the co-normality measure *alpha*, which gives information about how much co-normal the two neighborhoods being compared are from each other. The smaller *alpha* is, the more it indicates the neighborhoods belong to the same surface.

MatchValue(.) It finds the best match value.

SetGaussian(.) It assigns a 2D Gaussian kernel to each mask. It starts at 1.0 and decreases by $1/maskhalf$.

Name: IntDepth.h

Description: This class contains the main procedures to compute the new depth values by integration using the surface normals previously estimated.

Class procedures and their descriptions:

SpiralOutwards(.) The estimation is done in spiral-scan order going outwards, meaning that the missing range is all around a given subwindow. It starts at the left superior corner and going to the right.

SpiralInwards(.) The estimation is done in spiral-scan order going inwards, meaning that the missing range is a closed shape. It starts at the left superior corner and going to the right.

MaxNumNeighb(.) The estimation is done by assigning first those voxels which have the maximum number of neighbors already assigned. The selection among those voxels with the same number of neighbors can be done sequentially (left to right and top to bottom) or randomly. This is specified in *type-sqrnd* parameter. The procedure works only for closed shapes with missing range, where the estimation must be done inwards.

Obtznew(.) It computes the new depth value by integrating all the 8-connected and *info available* neighbors.

Part III

To-do list

Here we list some future work:

- Include intensity edges as additional information to help in the estimation of surface normals and new depth values [?].
- We would like to analyze the following case: Having as an input an intensity image with associated range data (complete and/or incomplete), range synthesize a *new* intensity image with no associated range data. For this case, of course, the two intensity images must show some similarity (the type of scene they represent), in order to be able to recover the depth values.
- albedo instead of intensity?
- Implementation of a *space carving* algorithm. The input are two different views of the same scene. The algorithm finds those augmented voxels from each view that intersect each other, i.e. resembles the same space information, and then carve away those voxels that are not consistent.

  The aspects to analyze would be:

  - Sampling of the augmented voxels =¿ calibration.
  - The distance between the viewpoints.
  - Registration and Integration.