Telerobotic Scene Description using Generalized Cylinders

Denis Dion Jr., Denis Laurendeau and Robert Bergevin
Computer Vision and Systems Lab, Laval University, Quebec, G1K 7P4
{ddion,laurend,bergevin}@gel.ulaval.ca

Abstract

This paper describes an approach for the extraction of generalized cylinders from a single range image. Generalized cylinders are an excellent tool for modeling parts of complex objects. They are defined with 1) an axis, 2) a cross-section (a 'slice') at constant angle with respect to the axis and 3) a 'sweeping rule' which describes how the cross-section evolves along the axis. The algorithms were tested on a wide variety of range images in a telerobotic context.

1. Introduction.

In '*model-based*' systems, object recognition in range images depends on the construction of a model of the objects in the scene. When these objects are complex, or when there are many different objects in the scene, this modeling process can be quite difficult. One important step prior object modeling is to search for the different parts that may belong to objects and then to group them in higher-level models.

Generalized cylinders can be used to model parts of complex objects. A generalized cylinder is composed of 1) an axis, 2) a base cross-section (a '*slice*' at constant angle with respect to the axis), and 3) a sweeping rule which describes in closed-form how the cross-section evolves along the axis of the cylinder. Straight Homogenous Generalized Cylinders (SHGC) are a particular case of generalized cylinders having a straight axis and a constant cross-section which can only be scaled along the axis according to the sweeping rule. When the cross-section is perpendicular to the axis, these generalized cylinders are referred to as Right Generalized Cylinders (RGC). This is the type of generalized cylinder that is studied this work.

Our objective is to design an approach for the extraction of generalized cylinders in a single range image in order to identify parts that could be modeled properly by this type of descriptor. In a telerobotic context, we focused on a class of scenes that contain a wooden beam, an insulator and an electrical conductor. The image in Figure 1 is computer-generated with a software called *Smog* that was developped at Laval University (see [3]). Using *Smog* and a software called *Ametist*, it is possible to synthesize a range image of the scene (shown in figure 1) from any point of view. The range image generated with 'Smog' shown in Figure 1 is shown as an intensity (2D) image for better visualisation. See [2].

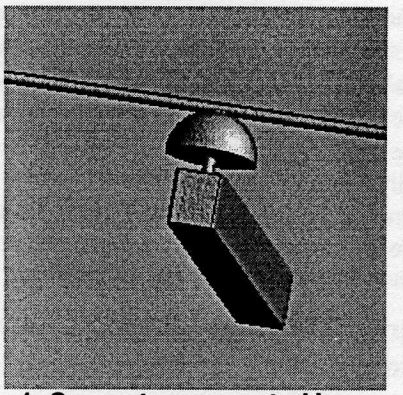

Figure 1: Computer-generated image of a telerobotic scene (using *Smog* and *Ametist*).

The algorithm was also tested on a wide variety of range images included in the National Research Council of Canada (NRC) image database. (see [10])

This paper is divided in 7 sections. Section 2 briefly defines the range image format and describes preprocessing steps. Section 3 describes how to extract 'step-edges' in the range images. Sections 4 and 5 are concerned with the most crucial part of the algorithm: the extraction of axis points and the description of the axis with a parametric 3D-space equation. Sections 6 and 7 deal with the construction of the generalized cylinders: cross-section modeling and sweeping rule estimation. Finally, section 8 shows how extracted generalized cylinders models can be used to reconstruct the scene at different resolution levels.

2. Range Images and background thresholding

Range images used for this work are 256 pixels in width by 256 pixels in height. Each pixel contains 16-bits integer range information, that is, the distance between the object's surface and the range-finder (where 'higher' range values corresponds to 'closer' surface points). The range images provided by the NRC use this file format. For images saved

by 'Ametist', some file format conversion must be done, but details will not be presented in this paper (see [4], section 1.2.2.1).

One important preprocessing step is to set the image background to a *unique* value. For every pixel z(l,c) in the image,

$$z(l,c)=max(z(l,c),t_b) \quad (1)$$

which means that every pixel in the image cannot be lower than a threshold value 't_b'. In an image, '*l*' stands for the '*row*' component (y-axis) and '*c*' is '*column*' component (x-axis).

A useful result of this background thresholding is the elimination of shadow effects which occur when the laser cannot reach the surface of the object or when its reflection on the surface cannot reach back the range-finder. But, most of all, when the threshold 't_b' is properly selected, a unique range value is assigned to every pixel which does not belong to any object. Background pixels thus share the same unique range value. This threshold value can be selected easily using z-value histograms.

For images provided by '*Smog*', background pixels already share the same unique value. After adding a proper offset, the background value can be set to *zero*.

3. Contour detection through step-edge extraction.

The first step for extracting generalized cylinders in the thresholded range image is to detect step-edges. Even for range images, contours convey significant information on object shape. Step-edges are a good way to extract these contours. Let equation (2) define an arbitrary horizontal line *n* in the range image (z is the range value).

$$z(i) = z(n,i),\ i=0,1,...,255 \quad (2)$$

First differences are defined by (3):

$$z(i)' = z(i) - z(i-1),\ i=1,2,...,255 \quad (3)$$

Second differences are similarly defined by (4):

$$z(i)'' = z(i)' - z(i-1)',\ i=2,3,...,255 \quad (4)$$

Pixel $z(i)$ is labeled as a step discontinuity pixel if:

$$z(i)' >= t_s \quad (5)$$

and

$$z(i)'' > 0 \text{ and } z(i+1)'' < 0 \quad (6)$$

Similarly, pixel $z(i-1)$ is labeled as a step discontinuity pixel if:

$$z(i)' <= (-t_s) \quad (7)$$

and

$$z(i)'' < 0 \text{ and } z(i+1)'' > 0 \quad (8)$$

Figure 2 shows an example of first and second differences of a 1-D signal.

Step-edge extraction is a two-step process: horizontal scanning (as illustrated above) and vertical scanning. The two resulting images are then combined with a logical OR.

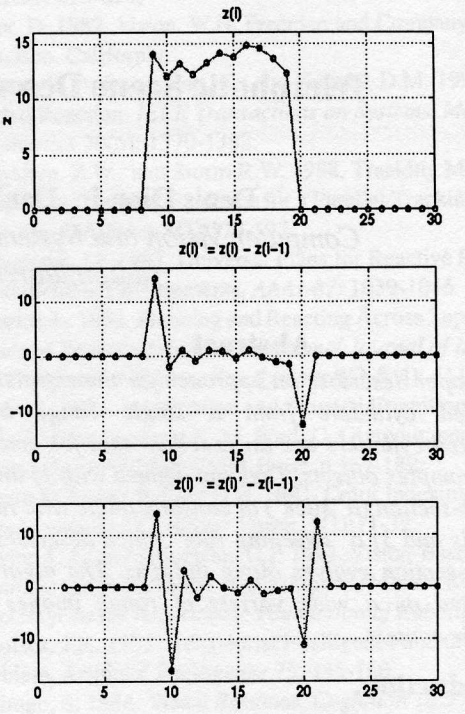

Figure 2: First and second differences.

The threshold value 't_s' governs the resolution (or precision) of the step-edges. A value of t_s between 200 and 250 usually yields good results (see figure 3).

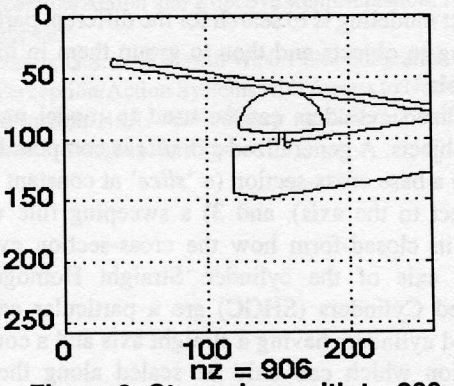

Figure 3: Step-edges with t_s=200

A major drawback of using only step-edges is that it is sometimes difficult to select a unique threshold that can guarantee one-pixel-wide step-edges. Based on the approach reported in [5], using hysteresis thresholding, it is possible to extract reliable step-edges using a low-threshold step-edge image to enhance a high-threshold step-edge image. Basically, high-threshold step-edges are allowed to 'grow' in specific directions, using step-edges from the low-threshold image. The algorithm also garantees one-pixel-wide contours. The reader is invited to refer to [4] and [5] for a complete description of the algorithm. In our case only the occluding exterior contours

(transitions background/object) of objects are kept. (see figure 4).

The iterative segmentation algorithm of Lejeune and Ferrie (see [6] and [7]) also provides good results for our application. However, it was abandoned in favor of the hysteresis thresholding approach because it is more computationally intensive for the following processing step.

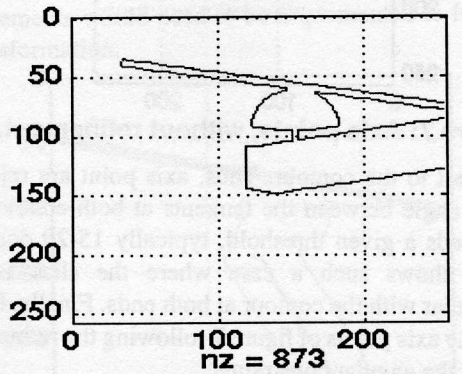

Figure 4: Occluding exterior contour of the object (see fig. 3)

4. Axis points extraction.

Even though there exists a broad variety of approaches to extract the 'skeleton' of objects for 2D (intensity) images based on their contour, ('Smooth Local Symmetries' proposed by Brady [1], 'Symmetric Axis Transform', 'Medial Axis Transform', 'Process Inferring Symmetry Analysis' by Leyon [8]), they cannot adequately be extended in a straightforward manner to 2.5D (range) images since they do not fully take advantage of the third dimension ('z').

In [9], Nevatia and Binford developped an algorithm to extract axis points on range images. This has been the starting point for the design of our algorithm.

Nevatia and Binford's approach also use the contour of the objects. Axis points are approximated by the mid-point between the projection (in the x-y plane) of two contour points. These mid-points are computed in n directions (scanning directions, typically $n=8$, $360/n$ degrees between each direction). Then, iterative processing is performed to reject invalid axis points and to link good ones. This approach also depends on the initial orientation of the object(s) in the image, since only 'n' scanning directions are considered.

It is clear that, for some directions, mid-points between contour points are valid, and for some directions, they are incorrect. In fact, it is important to note that the mid-points are good approximations of the axis points when the 'scanning direction' is *taken perpendicularily with respect to the length* of the object (or part of it). Our algorithm takes advantage of this fact. It also gives each axis point estimate a proper 'z'-component, which mid-points cannot achieve.

4.1. Extraction of tangent segments on the contour.

The basic idea of our algorithm is to take a 3D 'slice' perpendicularily with respect to the contour of the object for each contour point and to compute the center of gravity of this 'slice'. This center of gravity is considered as a reliable estimate of the axis points. It is worth noting that actual axis points cannot be computed using a single range image since the object is only partially visible from the range-finder.

For each contour point, an edge following is performed, for $\pm N$ points on each side of the contour point (see figure 5). The segment composed of these two endpoints provides an estimate of the tangent at this location. Generally, a value of 'N' between 2 and 4 yields good results.

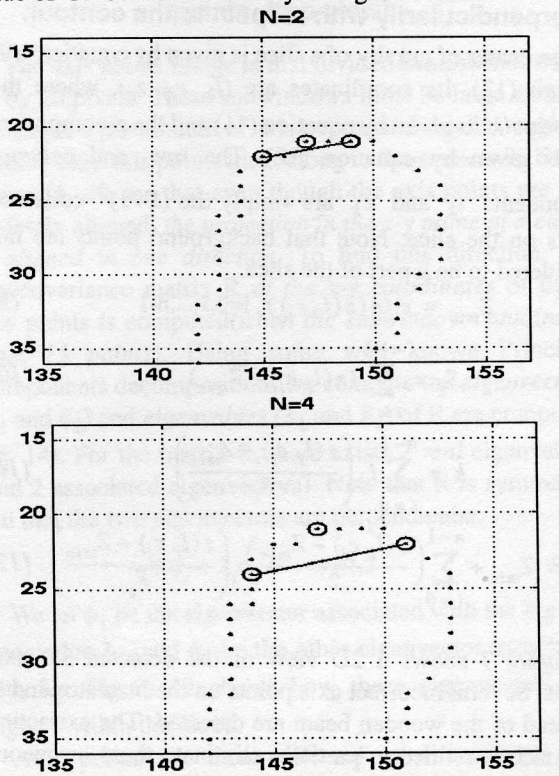

Figure 5: Edge following for a)N=2 and b)N=4 points. Tangent approximation.

4.2. Computation of the center of gravity of a slice.

The next step is to decide on which side of the tangent is the object (and on which side is the background) and then collect each (l_i, c_i, z_i) coordinates of the 'slice' taken perpendicularily with respect to the tangent until another contour point is reached (see figure 6). The subscript 'i'

goes from *0* through *n-1*, assuming that '*n*' points were collected for the slice.

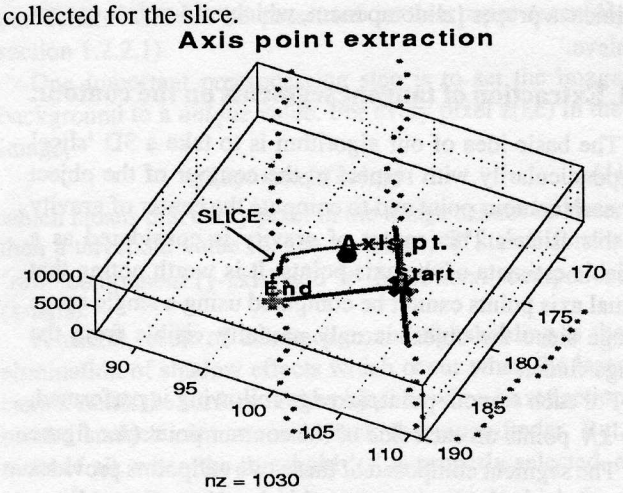

Figure 6: 3D view of a 'slice' taken perpendicularily with respect to the contour.

The center of gravity of a slice is given by equations (9) through (12). Its coordinates are (l_k, c_k, z_c), where the subscript 'k' is given by equation (11) and the z-component 'z_c' is given by equation (12). The *row-* and *column-*components 'l_k' and 'c_k' are simply the $(k+1)^{th}$ collected points on the slice. Note that background points are not considered to be a part of the slice.

$$Z_{min} = min\,(z\,(l_i, c_i))\,,\ i=0,1,...,n-1 \qquad (9)$$

$$S_z = \sum_{i=0}^{n-1} (z\,(l_i, c_i) - Z_{min}) \qquad (10)$$

$$k = \sum_{i=0}^{n-1} i \cdot \left(\frac{z\,(l_i, c_i) - Z_{min}}{S_z} \right) \qquad (11)$$

$$z_c = Z_{min} + \sum_{i=0}^{n-1} \left(\frac{z\,(l_i, c_i) - Z_{min}}{2} \right) \cdot \left(\frac{z\,(l_i, c_i) - Z_{min}}{S_z} \right) \qquad (12)$$

Figure 7 shows a 2D view of the detected 3D axis points. Several incorrect axis points on the insulator and at one end of the wooden beam are detected. The extraction approach is refined to partially eliminate these erroneous axis points.

4.3. Axis points refinement through angular constraints.

As described above, a slice is taken perpendicularily to one side of the contour without taking into account the angle between the slice and the contour, on the other side. We should (must) ensure that the angle at the other end of the slice is almost 90 degrees, to be coherent with the basic idea that lies beneath our algorithm, that is, the best axis points are found when the slice is taken perpendicularily

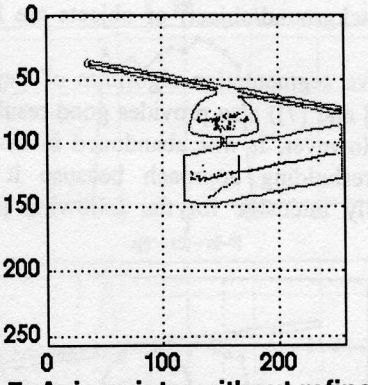

Figure 7: Axis points, without refinement.

with respect to the contour. Thus, axis point are rejected when the angle between the tangents at both ends of the slice exceeds a given threshold, typically 15-20 degrees. Figure 8 shows such a case where the slice is not perpendicular with the contour at both ends. Finally, figure 9 shows the axis points of figure 7 following the refinement step using the angular constraints.

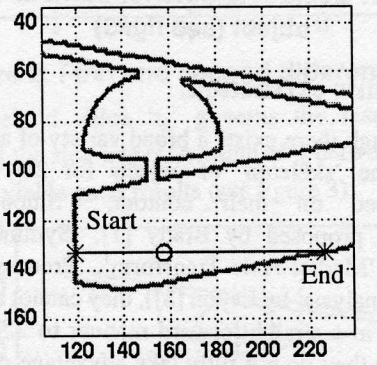

Figure 8: Invalid axis point.

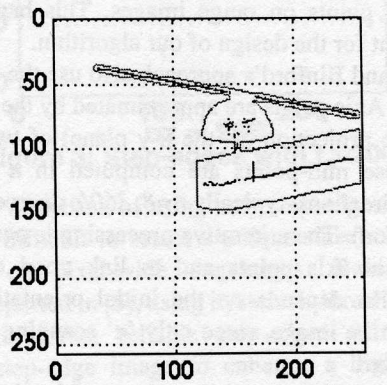

Figure 9: Axis points, after refinement. (parallelism within 20 degrees)

4.4. Results.

The angular refinement constraint helps to eliminate invalid axis points. The drawback is that the algorithm

yields good results for elongated parts having almost parallel contours but fails for contours with high curvature. For a wide variety of objects, this is often true, but unfortunately, the insulator is an exception. Figure 10 shows results for a second view of the beam scene. In order to extract axis points correctly for any object, the algorithm should be far more complex. It could use Smooth Local Symmetries [1] as a starting point but some major improvements would need to be implemented to fully use range information.

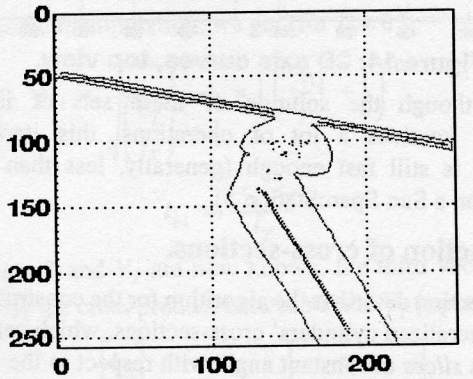

Figure 10: Axis points after refinement (other results).

5. Estimation of the parametric equation of the axis.

Axis points found at the previous step are not very useful by themselves and we must find a way to compute a 3D curve that can describe them in a compact manner. A wide variety of equations can describe a 3D curve. Third-degree polynomials for each three components of the curve (see equation (13)) were adopted in this current application. There is a third-degree polynomial for the y-component ($l(t)$), one for the x-component ($c(t)$) and one for the z-component ($z(t)$). There are thus 3 equations with 12 unknowns (l_i, c_i, z_i, for i=0,1,2,3). This set of linear equations can be solved/estimated with at least 4 axis points. The beginning of the curve is a $t=0$ and the end is at $t=1$. For each axis point, we must assign a proper 't' value in order to fix the position of each axis point on the 3D curve. This is the main difficulty of this step.

$$(l(t), c(t), z(t)) = \begin{pmatrix} l_0 t^3 + l_1 t^2 + l_2 t + l_3, \\ c_0 t^3 + c_1 t^2 + c_2 t + c_3, \\ z_0 t^3 + z_1 t^2 + z_2 t + z_3 \end{pmatrix} \qquad (13)$$

The extracted axis points are not perfectly aligned but are rather scattered in elongated cloud-like shapes (see figure 11). Each axis point must be assigned a proper 't_i' value. The third dimension (z-component) makes pixel following difficult. Only axis points which belong to the same axis must be considered. For instance, axis points that belong the *wooden beam* must not be confused with axis points that belong to the *wire* or the *insulator.*

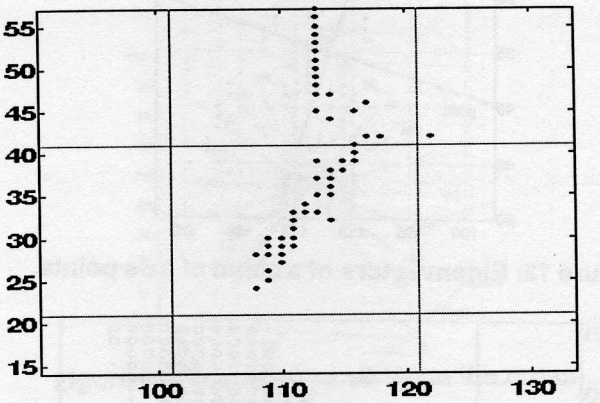

Figure 11: Cloud-like distribution of the axis points. (top view)

The axis points image is first divided in subwindows of 20 by 20 pixels. These subwindows must be large enough to include a fair amount of axis points, and small enough to include only axis points that belong to the same axis. From figure 11, we see that even though the axis points are not perfectly aligned, *the projection in the x-y plane of a cloud is aligned in one direction.* To find this direction, the autocovariance matrix R *of the x-y coordinates* of these axis points is computed(*when the subwindow contains at least 15 points*). Using some well known Principal Components decomposition, we compute the *eigenvectors* (ϕ_1 and ϕ_2) and *eigenvalues* (λ_1 and λ_2) of R are computed (eq. 14). For the matrix R, there exists 2 *real* eigenvalues (and 2 associated eigenvectors). Note that R is symmetric and that the two eigenvectors are perpendicular.

$$R\phi_i = \lambda_i \phi_i \qquad i = 1,2 \qquad (14)$$

We let ϕ_1 be the eigenvector associated with the *highest* eigenvalue λ_1, and ϕ_2 be the other eigenvector associated with λ_2. Figure 12 shows how these eigenvectors are aligned within the cloud.

Axis points are then collected in direction ϕ_1, by scanning lines in direction ϕ_2, for fixed length and width (see figure 13). For each collected axis point, all three component are saved and the axis point is removed from the image.

Suppose that these 'N' axis points are collected almost in perfect order, they're assigned an approximative 't_i' value, with equation (15) (where $t_0=0$ and $t_{N-1}=1$, for the *first* and *last* axis points respectively).

$$t_i = \frac{i}{N-1}, \text{ for } i = 0, 1, \ldots, (N-1) \qquad (15)$$

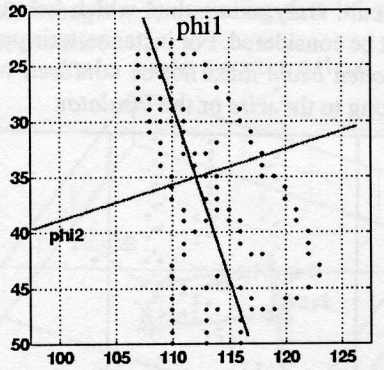

Figure 12: Eigenvectors of a cloud of axis points.

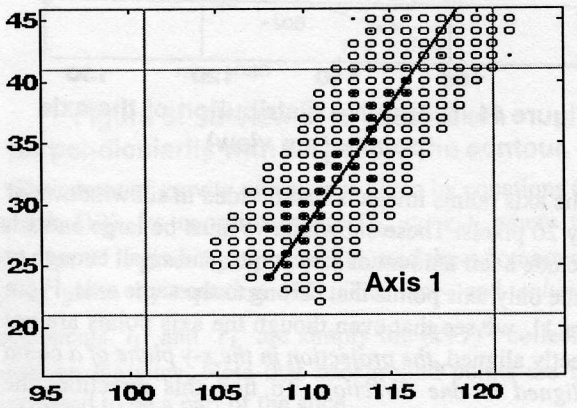

Figure 13: Axis points gathering.

Three sets of overdetermined (if N>4) linear equations are built using equation (13), where the unknowns are l_i, c_i and z_i (for $i=0,1,2,3$). An approximative 't_i' value is good enough, since these sets of linear equations are always overdetermined and N is generally high.

When these sets of linear equation are solved, the axis equations are known and is it then easy to evaluate the tangent at any axis point. The tangent is evaluated *at one end of the axis* and axis points are collected *in this direction* (again, for a fixed length and width). These new axis points are properly added to the previous ones. The sets of linear equations are built again, solved and the process repeats *until no axis points are found in a fixed number of consecutive scan lines*. Then, the tangent *at the other end* of the axis is evaluated and axis points are collected in this direction. Again, these new axis points are properly added to the ones at previous step(s). Every axis point has its own influence on the estimation of the 12 unknowns. Thus, both ends of the axis are allowed to expand. When there is no more axis points to be found at both ends of the axis, we move on with the next 20x20 subwindow. As a final result, figure 14 show the 3D curves obtained from axis points of figure 9.

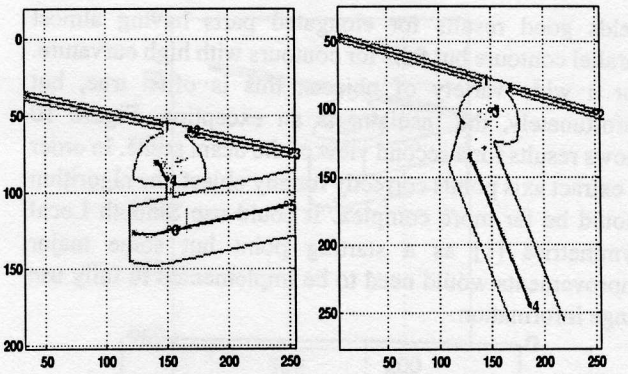

Figure 14: 3D axis curves, top view.

Even though the solution of these sets of linear equations involves a lot of operations, this iterative algorithm is still fast enough (generally, less than two seconds, on a Sun SparcStation 5).

6. Extraction of cross-sections.

This section describes the algorithm for the construction of the generalized cylinders' cross-sections, which can we viewed as *slices* at constant angle with respect to the axis. In this work, this angle is 90 degrees (Right Generalized Cylinders) for convenience. For different angles about the axis, the objective is to find the points on the object that are at the intersection with cross-section's plane. Figure 15 shows such cross-sections, at two different angular resolution levels.

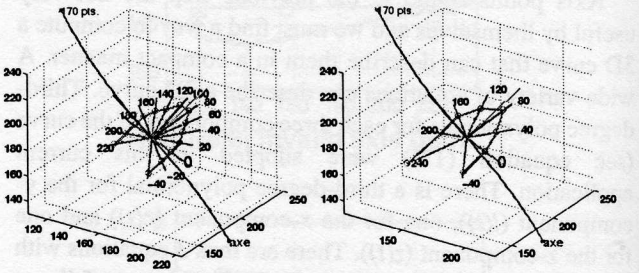

Figure 15: Cross-sections at two different angular resolution levels.

Using equation (13), it is easy to obtain the tangent at each point on the axis by computing the derivative with respect to 't'. The cross'section's plane is then found. Let us define a Serret-Frenet (along the axis) of three orthogonal *unit* vectors T, V_1 and V_2, where T is the axis' tangent vector and is fixed by the axis curve's equations. Vectors V_1 and V_2 will the cross-section's plane. A fixed number of cross-sections are built at equally spaced 't' values along the axis. Let's denote by equations (16) and (17) the components of T, V_1 and V_2 (for the 't' value being considered).

$$T = \left[l_T \; c_T \; z_T\right]^T \tag{16}$$

$$V_1 = \left[l_{v1} \; c_{v1} \; z_{v1}\right]^T \qquad V_2 = \left[l_{v2} \; c_{v2} \; z_{v2}\right]^T \tag{17}$$

There is an infinity of vectors V_1 and V_2 that can be perpendicual with each other and with T also. Let us fix the z-component of V_1 to be zero.

$$z_{v1} \equiv 0 \tag{18}$$

Since the dot product between V_1 and T must be zero, after some manipulation, we get (for $l_T \neq 0$):

$$c_{v1} = \sqrt{\frac{1}{\left(\frac{c_T}{l_T}\right)^2 + 1}} = \left(\left(\frac{c_T}{l_T}\right)^2 + 1\right)^{-0.5} \tag{19}$$

$$l_{v1} = \frac{-c_{v1} c_T}{l_T} \tag{20}$$

Since T and V_1 are now known, the third vector V_2 is given by the cross product between T and V_1 (equation 21).

$$\hat{V}_2 = \begin{bmatrix} l_{v2} \\ c_{v2} \\ z_{v2} \end{bmatrix} = \hat{T} \times \hat{V}_1 = \begin{bmatrix} -z_T c_{v1} \\ z_T l_{v1} \\ l_T c_{v1} - c_T l_{v1} \end{bmatrix} \tag{21}$$

We impose the z-component of V_2 to be positive. If it is not, we negate each component of V_2.

If $l_T=0$, V_1 and V_2 are given simply by equation (22):

$$V_1 = \left[1 \; 0 \; 0\right]^T \qquad V_2 = \left[0 \; z_T \; -c_T\right]^T \tag{22}$$

Still, there is the possibility to have two vectors V_1 that respect all the above conditions. We must finally verify that T, V_1 and V_2 satisfy equation)23).

$$\hat{T} \cdot (\hat{V}_1 \times \hat{V}_2) < 0 \tag{23}$$

If equation (23) is not satisfied, we simply negate each component of V_1. Now, the ortogonal vector set is unique for a given tangent vector T. Figure 16 show such vectors.

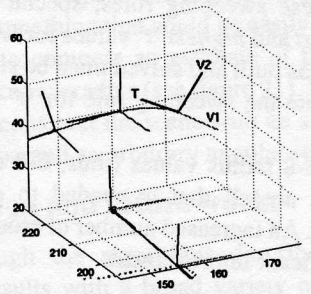

Figure 16: Orthogonal vector set.

Vector V_1 represents the 0-degree angle ($V(0)$) on the cross-section and V_2 represents the 90-degree angle ($V(90)$), as shown on figure 17. Each other direction $V(\theta)$ will be a linear combination of V_1 and V_2, given by equation (24). Vector $V(\theta)$ is (of course) coplanar with V_1 and V_2.

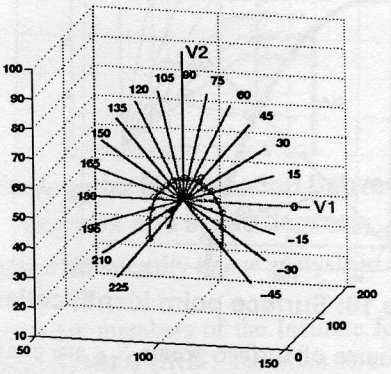

Figure 17: Vectors V_1 and V_2 on the cross-section.

$$\hat{V}(\theta) = \begin{bmatrix} l_\theta \\ c_\theta \\ z_\theta \end{bmatrix} = \begin{bmatrix} l_{v1} \\ c_{v1} \\ z_{v1} \end{bmatrix} \cos(\theta) + \begin{bmatrix} l_{v2} \\ c_{v2} \\ z_{v2} \end{bmatrix} \sin(\theta) \tag{24}$$

6.1. Algorithm description.

Let A be, the coordinate of the axis points for which the cross-section is being constructed (equation 25).

$$A = \left[l_A \; c_A \; z_A\right]^T \tag{25}$$

Starting from this axis point, we move in each direction $V(\theta)$ until the surface of the object is traversed. First, the vector V(θ) (equation (26)) is normalized so that, a move in this direction would be in at most one unit.

$$V(\theta) = \left[l_\theta \; c_\theta \; z_\theta\right]^T = \frac{V(\theta)}{max(|l_\theta|, |c_\theta|, |z_\theta|)} \tag{26}$$

Let $P_\theta(i)$ be the coordinate of the i^{th} *point* in direction $V(\theta)$. It is given by equation 27, where for $i=0$, $P_\theta(i) \equiv A$, the axis point.

$$P_\theta(i) = \left[l_i \; c_i \; z_i\right]^T = A + i \cdot V(\theta) = \begin{bmatrix} l_A + l_\theta i \\ c_A + c_\theta i \\ z_A + z_\theta i \end{bmatrix} \tag{27}$$

Since $z(l_i,c_i)$ is the z-value at the coordinate (l_i,c_i) in the range image, the gap between the z-value on the object's surface and the z-value of $P_\theta(i)$ is given by equation 28 and is noted $\Delta z(i)$.

$$\Delta z(i) = z(l_i, c_i) - z_i \tag{28}$$

For an increasing value of 'i', the surface of the object will be traversed when $\Delta z(i)$ changes sign (from positive to negative). Then, the surface point $S_{t,\theta}$ is simply identified as $P_\theta(i)$ (see figure 18). This process is repeated for each

angle (θ) on the cross-section, and then, for each equally distant cross-sections ('*t*' values) along the axis.

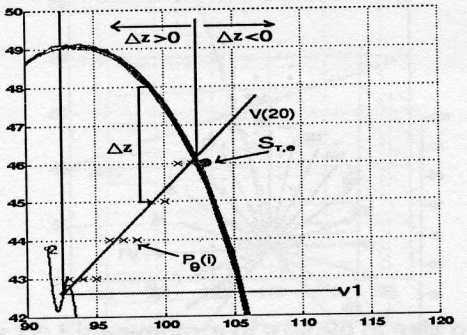

Figure 18: Surface point identification.

The coordinate of surface points $S_{t,\theta}$ are converted as radii values $r_{t,\theta}$, that is, distance between the axis point and the surface point.

6.2. Results

This step provides interesting results without requiring intensive computation. The number of cross-sections, the starting/ending angles and the angle steps allow to choose the precision level at which the generalized cylinders are to be built. Figure 19 shows generalized cylinders of the same scene, but at two precision levels (top: 6 cross-sections, 40 degrees of angular resolution, bottom: 8 cross-sections, 20 degrees of angular resolution).

7. Sweeping rules.

This step is concerned with the modeling of how the equally distant cross-sections evolve along the axis, in order to gain the possibility to reconstruct the scene with an arbitrary number of *cross-sections*(i.e. with arbitrary precision level).

For a particular angle θ on all cross-sections (that belong to the same axis), we need to relate the radius values to some parametric equation $r_\theta(t)$ called the "sweeping rule" (see equation 29). Generally, the radius variation along the axis is continuous and slow. We used third-degree polynomials to describe the sweeping rules.

$$r_\theta(t) = R_0 t^3 + R_1 t^2 + R_2 t^1 + R_3 \qquad (29)$$

For every cross-section, the '*t*' value for which it was built is known, thus, it is easy to construct and solve the set of linear equations that yields all four coefficients of the third degree polynomial (see equation 30, where N_c is the number of cross-sections that were built).

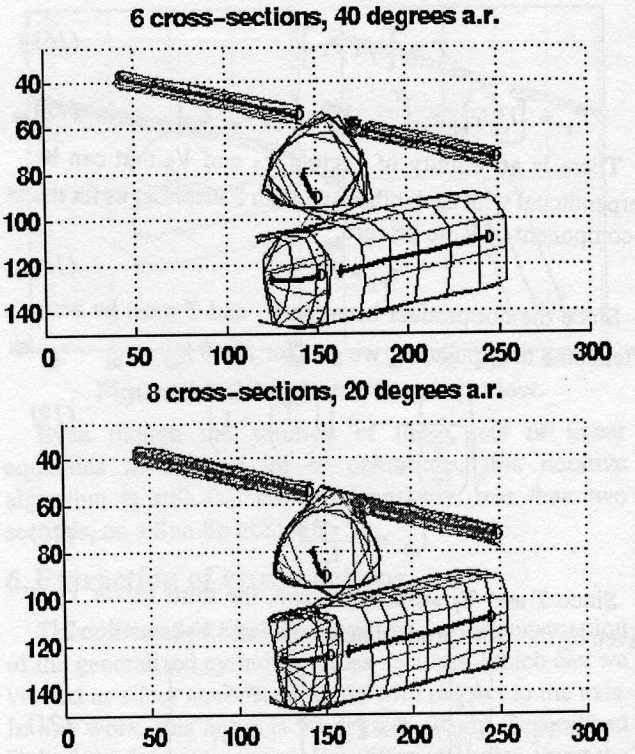

Figure 19: Generalized cylinders at two different precision levels

$$\begin{bmatrix} t_0^3 & t_0^2 & t_0 & 1 \\ t_1^3 & t_1^2 & t_1 & 1 \\ \dots & \dots & \dots & \dots \\ t_{(N_c-1)}^3 & t_{(N_c-1)}^2 & t_{(N_c-1)} & 1 \end{bmatrix} \begin{bmatrix} R_0 \\ R_1 \\ R_2 \\ R_3 \end{bmatrix} = \begin{bmatrix} r_{t_0,\theta_i} \\ r_{t_1,\theta_i} \\ \dots \\ r_{t_{(Nc-1)},\theta_i} \end{bmatrix} \qquad (30)$$

At least, 4 radius values are required to solve the equations set given in (30). A sweeping rule must be estimated for every angle on the cross-section.

The cross-sections construction algorithm may not work properly in some situations and, some radius values may be missing. In order to avoid numerical instability when using these sweeping rules, special care must be taken: the *lowest* and *highest* '*t*' values for which the linear equations set was built and solved must be stored. We must avoid to evaluate the sweeping rule for '*t*' values outside this range.

Generally, 6-8 radius values (thus, 6-8 cross-sections) along the axis are all that is needed to compute valid sweeping rules. An increased number of cross-sections will not yield significant improvement.

8. Scene reconstruction using generalized cylinders models.

The scene reconstruction is pretty straightforward. For every generalized cylinder, the axis equation is known,

thus, we can compute back (at any point 't' along the axis) the orthogonal vector set T, V_1 and V_2 (V_1 and V_2 being the new cross-section's plane). For every angle on the cross-section, the sweeping rule is known, thus providing estimated radius values for this cross-section. Since the coordinates of the axis point and vectors V_1 and V_2 are available, the conversion from radius values to absolute surface points coordinates (l,c,z) is easily obtained.

Figure 20 shows two reconstructed scenes. It must be remembered that the scene can be reconstructed with an arbitrary number of cross-sections. The spacing between each of them can be small, yielding a more precise description. A more distant spacing between cross-sections yields a coarser description.

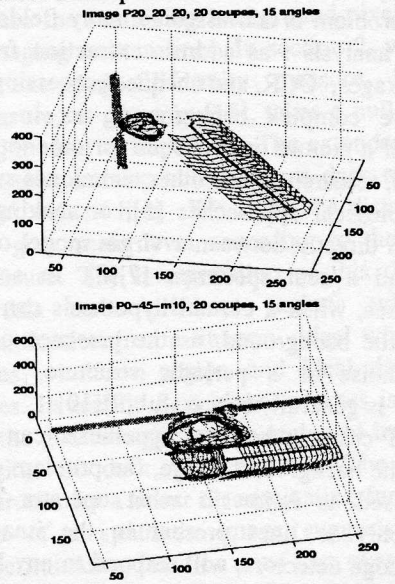

Figure 20: Reconstructed scenes. (20 cross-sections, 20 degrees of angular resolution)

Since we only have sweeping rules for particular angles on the cross-sections, it is possible to reconstruct the scene only for these specific angles. For now, nothing has been done to model the shape of the cross-section with respect to the angle θ. Something that could be easily done is a linear interpolation to compute an estimate of the radius value between two known radius (see figure 21). It would then be possible to reconstruct the scene with an arbitrary number of cross-section and an arbitrary angular resolution level.

9. Conclusion and future work.

Even though the algorithms described above yield interesting results with a broad variety of range images, some improvement must be made to the axis points extraction step. The modified algorithm should give good results, even when the contours are not parallel but are rather symmetric. Some additional information could also

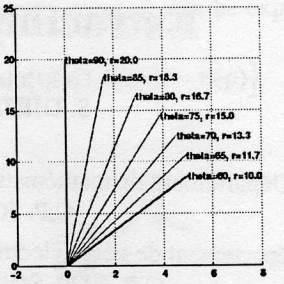

Figure 21: Angular interpolation between θ_1=60 degrees and θ_2=90 degrees.

be supplied by a 2D intensity image analysis in order to get better results.

The authors are members of the Institute for Robotics and Intelligent Systems (IRIS) and wish to acknowledge the support of the Networks of Centres of Excellence Program of the Government of Canada, the National Science and Engineering Research Council and the participation of PRECARN Associates Inc.

10.References

[1] Asada, H., Brady, M., "The Curvature Primal Sketch.", IEEE Transactions on Pattern Analysis and Machine Intelligence, 1986.

[2] Côté, J. et al., "Ametist: Users guide.", Laval University, 1992.

[3] Croteau, A., "Estimation par vision 3-D de l'occupation de l'espace de travail d'un robot.", M. Sc. thesis in electrical engineering, Laval University, 1991.

[4] Dion, Denis Jr., "Extraction de cylindres généralisés à partir d'images télémétriques.", M.Sc thesis in electrical engineering, Laval University, 1996.

[5] Laurendeau, Denis, "Construction du modèle d'objets tridimensionnels à partir d'images 3D.", Ph.D. thesis in electrical engineering, Laval University, 1985.

[6] Lejeune, A., Ferrie, F.P., "Finding the Parts of Objects in Range Images.", Centre for Intelligent Machines, McGill University, McConnell Engineering Building, CIM-93-8, August 1993.

[7] Lejeune, A., Ferrie, F.P., "Partitioning Range Images Using Curvature and Scale.", IEEE Transactions on IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1993.

[8] Leyton, M., "A Process-Grammar for Shape.", Artificial Intelligence, 1988.

[9] Nevatia, R., Binford, T. O., "Description and Recognition of Curved Objects", Artificial Intelligence, 8, 77-98 (1977).

[10] Rioux, M., Cournoyer, L., "Les fichiers d'images tridimensionnelles du CNRC.", CNRC no. 29077, June 1988.