*Proc. SPIE* **3034**, pp. 346-357 (1997) LA-UR-97-0880
*Medical Imaging: Image Processing*, K. M. Hanson, ed.

# 3D Tomographic Reconstruction Using Geometrical Models

X. L. Battle, G. S. Cunningham, and K.M. Hanson

Los Alamos National Laboratory, MS D454
Los Alamos, New Mexico 87545 USA

## ABSTRACT

We address the issue of reconstructing an object of constant interior density in the context of 3D tomography where there is prior knowledge about the unknown shape. We explore the direct estimation of the parameters of a chosen geometrical model from a set of radiographic measurements, rather than performing operations (segmentation for example) on a reconstructed volume. The inverse problem is posed in the Bayesian framework. A triangulated surface describes the unknown shape and the reconstruction is computed with a maximum *a posteriori* (MAP) estimate. The adjoint differentiation technique computes the derivatives needed for the optimization of the model parameters. We demonstrate the usefulness of the approach and emphasize the techniques of designing forward and adjoint codes. We use the system response of the University of Arizona Fast SPECT imager to illustrate this method by reconstructing the shape of a heart phantom.

**Keywords:** Bayesian analysis, tomographic reconstruction, geometrical models, adjoint differentiation

## 1. INTRODUCTION

Geometrical models have received increasing attention in medical imaging for tasks such as segmentation, restoration and reconstruction.[1,2] These deformable geometric models can be used in the context of Bayesian analysis to solve ill-posed tomographic reconstruction problems.[3] The geometrical structure of an object is often known before the data are taken. Bayesian methods of reconstruction take into account characteristics of the object being imaged that are known *a priori*. These methods can substantially improve the accuracy of reconstructions obtained from limited data when good geometrical information is employed in the model.[4,5]

The Bayes Inference Engine (BIE) is a flexible software tool that allows one to interactively define models of radiographic measurement systems and geometric models of experimental objects.[6,7] The BIE provides a very convenient framework to easily perform Bayesian inference. Based on an object-oriented approach, the BIE allows one to design a measurement model by connecting icons on a canvas (Fig. 1), define a geometric parameterization for the models and optimize a likelihood or posterior with respect to the user-selected parameterization.[8] As shown in Fig. 1, a set of parameters is the input of a data-flow diagram that represents a sequence of transformations referred to as the forward transformation. The output of the diagram is a scalar $\varphi = -\log(\textit{posterior})$ that is to be minimized. This optimization problem would be intractable without knowing the gradient of $\varphi$. Considering the number of parameters of a geometrical model ($10^3$ to $10^6$ or more), perturbation methods are not acceptable. Adjoint differentiation[9,10] allows one to compute these crucial derivatives in a time that is comparable to the forward calculation through the data-flow diagram.

Given a sequence of transformations,

$$\mathbf{X} \stackrel{A}{\longmapsto} \mathbf{Y} \stackrel{B}{\longmapsto} \mathbf{Z} \stackrel{C}{\longmapsto} \varphi \ ,$$

where $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$ are vectors and $\varphi$ a scalar, one wants to compute the derivatives of $\varphi$ with respect to the $x_i$ in a computationally efficient way. The chain rule for differentiation gives the derivatives of $\varphi$ with respect to $x_i$,

$$\frac{\partial \varphi}{\partial x_i} = \sum_{jk} \frac{\partial \varphi}{\partial z_k} \frac{\partial z_k}{\partial y_j} \frac{\partial y_j}{\partial x_i} \ .$$

The essence of adjoint differentiation is to perform the sum on the indices in the reverse order from that used in the forward calculation. If the sum on $k$ is done before that on $j$, the sequence of calculations is

$$\mathbf{I} \stackrel{C'}{\longmapsto} \frac{\partial \varphi}{\partial \mathbf{z}} \stackrel{B'}{\longmapsto} \frac{\partial \varphi}{\partial \mathbf{y}} \stackrel{A'}{\longmapsto} \frac{\partial \varphi}{\partial \mathbf{x}} \ ,$$

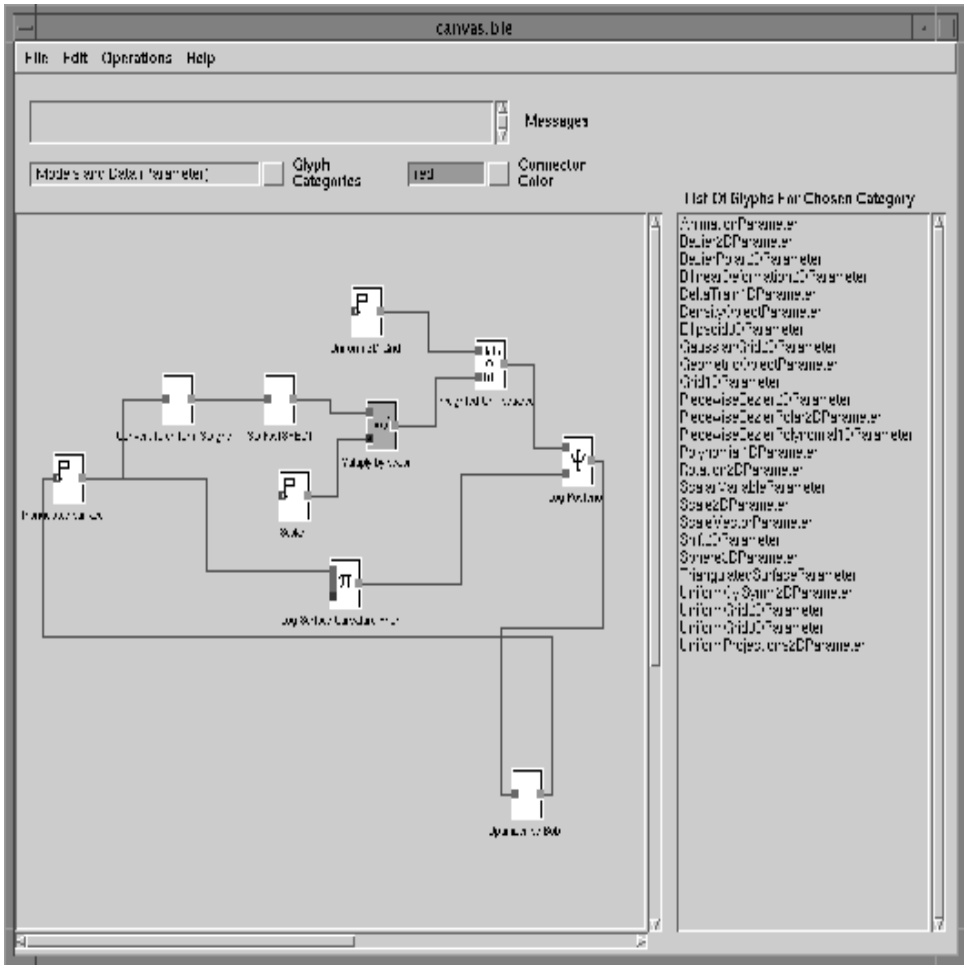


**Figure 1.** The canvas of the BIE.

where $\mathbf{I}$ is the identity vector, and $\frac{\partial \varphi}{\partial \mathbf{y}}$ denotes the vector of term $\frac{\partial \varphi}{\partial y_i}$. We see that adjoint differentiation computes the derivatives of $\varphi$ with respect to the model parameters in a way comparable to the forward calculation. The intermediate data-structures are similar (vectors) and for each forward transformation (icon in the canvas) an adjoint counterpart exists. In practice, this means that when a new transformation is added in the BIE, both forward and adjoint aspects must be implemented.

In this paper, we address the issue of reconstructing an object of constant density interior in the context of 3D tomography where there is prior knowledge about the unknown shape. We propose this problem as an illustration of the first implementation of 3D algorithms in the BIE. We chose to represent a constant density interior volume by a triangulated surface. For sake of generality and reusability, we convert this geometrical representation of a volume to an array of voxels before simulating the measurement process. This conversion step ensures the modularity of the code so that other geometrical models will be easily integrated. We will present in detail the forward algorithm, as well as its adjoint counterpart. We will emphasize the precautions that must be taken when writing forward and adjoint code, as well as some helpful techniques.

We will demonstrate the usefulness of this approach using simulated data. We have reconstructed a heart phantom using the system response of the University of Arizona Fast SPECT machine.

## 2. PRINCIPLE OF THE FORWARD TRANSFORMATION

In this study, we consider triangulated surfaces, as they are the most common representations of surfaces in space. Using other tessellations is still possible since any tessellated surface can be converted to a triangulated surface.

The goal of this transformation is to determine the voxel distribution that would describe the region contained in the interior of a closed surface in space. If a voxel is completely contained in the region, its value is set to one; if a voxel lies on the boundary of the region (i.e the surface passes through the voxel) its value is set to the value of the overlap of the surface on the considered voxel; of course, if a voxel is outside the region, its value is zero.

### 2.1. Analogy with the 2D Case

In two dimensions, the problem of converting a region $\mathcal{R}$ described by its contour to a pixelated image has the usual solution. Given a set of N vertices $P_i$ with $0 \leq i \leq N-1$, the contour of the region is defined by the N segments $[P_i, P_{i+1}]$ with $0 \leq i \leq N-2$ and $[P_{N-1}, P_0]$. The normal of a segment $[P_i, P_j]$ noted $\overrightarrow{n}(P_i, P_j)$ points towards the exterior of the region.

The pixelated image is generated by accumulating the contributions of all the segments. The accumulation is done segment by segment and line by line. The lines of pixels having an intersection with a given segment $[P_i, P_j]$ are updated. Let the $x$ axis be the direction of the lines of pixels in the image. There are two cases:

1. If $\overrightarrow{n}(P_i, P_j) \cdot \overrightarrow{x} < 0$ the pixels on the left of the segment are outside $\mathcal{R}$ whereas the ones on the right are inside. A positive contribution, equal to the surface overlap of $\mathcal{R}$ with each of the right pixels is added to them.
2. If $\overrightarrow{n}(P_i, P_j) \cdot \overrightarrow{x} > 0$ the pixels on the left of the segment are inside $\mathcal{R}$ whereas the ones on the right are outside. A negative contribution, equal to minus the surface overlap of $\mathcal{R}$ with each of the right pixels is added to them.

The degenerate case where, $\overrightarrow{n}(P_i, P_j) \cdot \overrightarrow{x} = 0$, does not require any computation. When the segment and the line of pixels are aligned, the surface overlap is null.

Figure 2 shows how the two-dimensional algorithm works. (a) shows a simple contour having three vertices. (b) shows the positive contribution of the first segment. (c) shows the accumulation of the positive contribution of the second segment. (d) shows the accumulation of the negative contribution of the last segment.

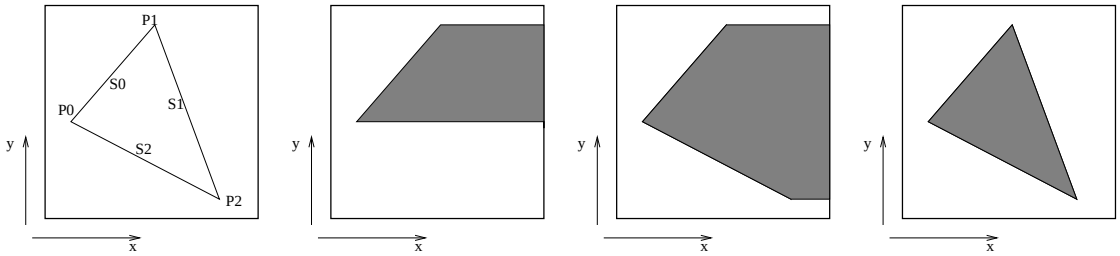


**Figure 2.** Two-dimensional conversion. A simple contour (a) having three vertices is converted to a pixelated image. The first two segments have positive contributions (b) and (c). The last segment has a negative contribution (d).

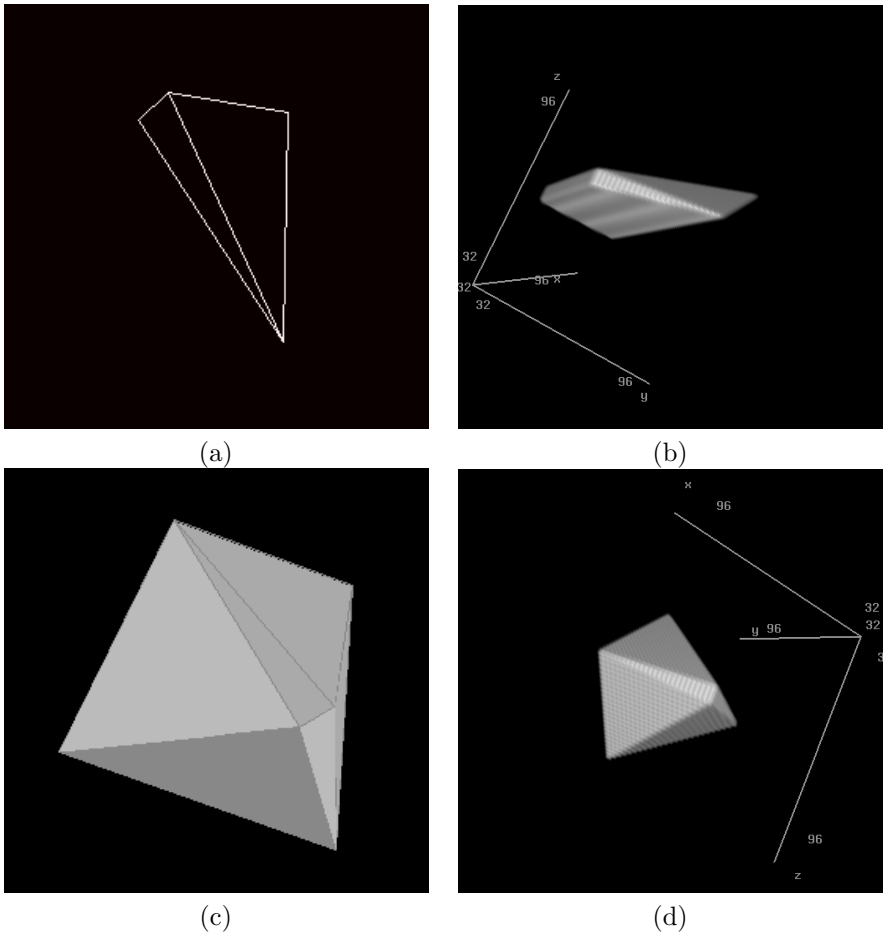


**Figure 3.** Three-dimensional conversion. Two triangular facets (a) are converted to the voxel grid (b). Their contribution is positive. When all the facets defining a diamond-like object (c) are processed, positive and negative contributions are added and define the final voxel distribution (d).

## 2.2. Proposed Method

The algorithm adopted here is a generalization of the two-dimensional algorithm. It is based on the superposition principle: each facet of the external surface is processed independently, and contributes to the voxel array. The overlap of the whole surface on the voxel grid is the sum of the overlaps of all the surface elements. Such an approach is interesting for many reasons. The complexity is *O(N)* with $N$ the number of triangles. Big datasets can easily be treated as a set of smaller ones. Moreover, this approach is valid for any region described by one or several closed surfaces. In particular, concave surfaces and cavities are handled correctly.

In the remainder of this section, we will focus on the contribution of a single triangular facet to a voxel grid. A set of three vertices $P_0, P_1, P_2$ describes the facet. The right-hand rule determines the normal of the facet, noted $\overrightarrow{n}$.

$$\overrightarrow{n} = \overrightarrow{P_0P_1} \wedge \overrightarrow{P_1P_2} \ .$$

The facets are assumed to be oriented so that the normal $\overrightarrow{n}$ points towards the exterior of the region $\mathcal{R}$ to be converted.

As in the two-dimensional case, the values of the voxels are computed by accumulating the contribution of the facets. The accumulation is done facet by facet, and line by line. In our implementation, we have chosen to work with the lines of direction $z$. There are two cases.

1. If $\overrightarrow{n} \cdot \overrightarrow{z} < 0$ the voxels in front of the facet in the $z$ direction are outside $\mathcal{R}$ whereas the ones behind are inside. A negative contribution, equal to minus the volume overlap of $\mathcal{R}$ with each of the front voxels is added to them.
2. If $\overrightarrow{n} \cdot \overrightarrow{z} > 0$ the voxels in front of the facet in the $z$ direction are inside $\mathcal{R}$ whereas the ones behind are outside. A positive contribution, equal to the volume overlap of $\mathcal{R}$ with each of the front voxels is added to them.

The degenerate case where $\overrightarrow{n} \cdot \overrightarrow{z} = 0$ does not require any computation. When the facet contains the $z$ axis, the volume overlap is null.

Figure 3 shows how the three-dimensional algorithm works. (a) shows two facets of a diamond-like shape and (b) shows their positive contribution. (c) shows the complete diamond, and (d) the final result of the conversion.

## 2.3. Contribution of a Single Facet

The computation of the contribution of a single facet is achieved in two consecutive steps. First, the facet is split into a set of smaller facets, referred to as sub-facets. The splitting is performed to make sure that each sub-facet is contained in a single voxel. The second step is the computation of the contribution of the sub-facets. Because they contribute to a single row of voxels along the $z$ axis, the contribution of the sub-facets is easy and can be done in closed form.

### 2.3.1. Splitting

In order to create a set of sub-facets that are contained in a single voxel, the facet is sliced along the three axis. The planes that cut the facet into a set of sub-facets along a given axis are the planes normal to the axis that contain the vertices of the facet and the planes between the voxels. The slicing along each axis takes place in two steps as shown in Fig. 4. The facet (a) lies across a set of cutting planes that are normal to the axis of slicing. Slicing the facet with the planes containing its vertices (solid lines) generates a set of sub-facets (b). Each of these sub-facets is then sliced by the voxel planes (dotted lines) to generate the final set of sub-facets (c).

The global splitting operation can be implemented in a recursive way. Starting on the $z$ axis, the slicing in two steps is performed. The resulting set of sub-facets is the the input of a splitting starting on the $y$ axis, and again for the $x$ axis.

Figure 5 shows the result of the recursive slicing on a triangular facet. Seen in the $z$ direction, the facet is split along the $z$ axis first. The planes normal to $z$ are not displayed and only their intersection with the plane of the facet is shown (oblique lines). The set of sub-facets generated are then split along the $y$ axis (b) and finally along the $x$ axis.

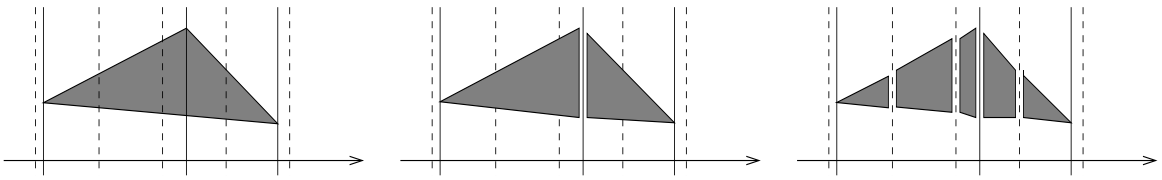


**Figure 4.** Slicing of a facet along a given axis. Represented in two dimensions for understanding, the facet (a) lies across a set of cutting planes that are normal to the axis of slicing. Slicing the facet with the planes containing its vertices (solid lines) generates a set of sub-facets (b). Each of these sub-facets is then sliced by the voxel planes (dotted lines) to generate the final set of sub-facets (c).

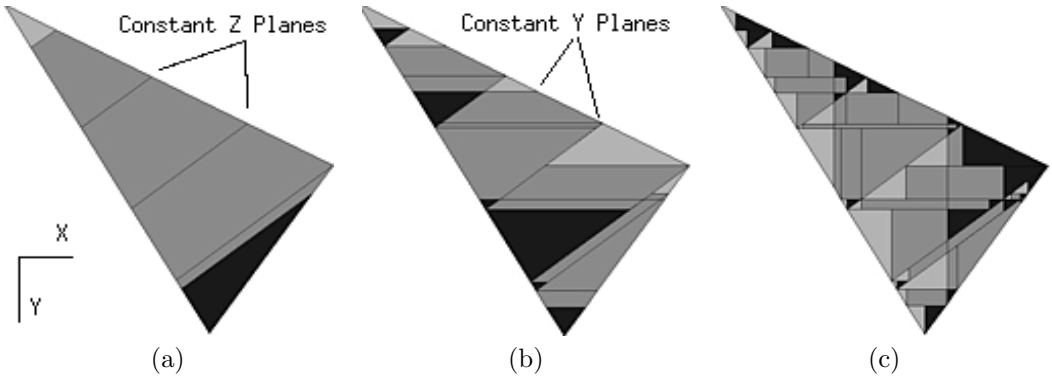


**Figure 5.** Recursive splitting of a facet. Seen in the $z$ direction, the facet is split along the $z$ axis first. The planes normal to $z$ are not displayed and only their intersection with the plane of the facet is shown (oblique lines). The set of sub-facets generated is then split along the $y$ axis (b) and finally along the $x$ axis.

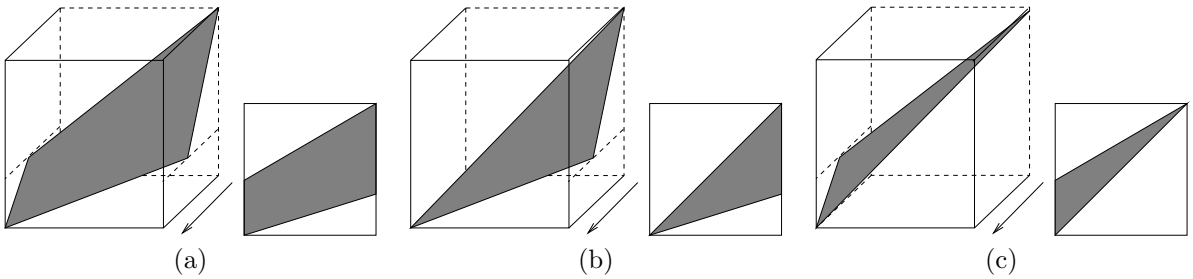


**Figure 6.** Sub-facets resulting from the splitting. Shown in perspective as well as in projection on the $(x, y)$ plane, the possibilities are well determined. The general situation is a trapezoid (a). Two edges are contained in the cutting planes, the two others are crossing the voxel. Two degenerate situations can occur when the edges contained in the cutting planes are reduced to a single vertex (b) and (c).

### 2.3.2. Overlap

Thanks to the splitting described above, the problem of computing the overlap of a facet on the voxel grid is reduced to a limited set of well determined situations. The sub-facets are either triangles or trapezoids as shown in Fig. 6. Shown in perspective as well as in projection on the $(x, y)$ plane, the possibilities are well determined. The general situation is a trapezoid (a). Two edges are contained in the cutting planes, the two others are crossing the voxel. Two degenerate situations can occur when the edges contained in the cutting planes are reduced to a single vertex (b) and (c).

The volume overlap of a sub-facet $\mathcal{F}$ on the voxel grid consists in two terms. The voxel of coordinates $(i, j, k)$ that contains the sub-facet is updated by:

$$V(i,j,k) \leftarrow V(i,j,k) \pm Overlap(\mathcal{F}, V(i,j,k)) \ ,$$

where $V(i, j, k)$ denotes the value of the voxel at the location $(i, j, k)$, $Overlap(\mathcal{F}, V(i, j, k))$ is the volume of the portion of the voxel that is in front of the facet seen in the $z$ direction and the $\pm$ sign recalls that the contribution is either positive or negative depending on the orientation of the sub-facet normal with regard to axis $z$.

All the voxels in front of the sub-facet seen in the $z$ direction are updated in the same way by:

$$(\forall p, 0 \leq p < k) V(i,j,p) \leftarrow V(i,j,p) \pm AreaOverlap(\mathcal{P}(\mathcal{F}, \overrightarrow{x}, \overrightarrow{y})) \cdot VoxelHeight(\overrightarrow{z}) \ ,$$

where $\mathcal{P}(\mathcal{F}, \overrightarrow{x}, \overrightarrow{y}))$ denotes the projection of the sub-facet $\mathcal{P}$ on the $(x, y)$ plane, $AreaOverlap(\mathcal{P}(\mathcal{F}, \overrightarrow{x}, \overrightarrow{y}))$ is the area overlap of the projected facet on the voxel face and $VoxelHeight(\overrightarrow{z})$ denotes the height of the voxel in the $z$ direction.

Figure 7 shows the two different terms. The overlap of the sub-facet on the voxel that contains it is the volume of the region shown in (a). The volume overlap of the sub-facet on all the voxels in front of the facet is the same and is shown in (b). It is a prismatic volume that is computed by multiplying the area of the base by the height.

## 3. ADJOINT DIFFERENTIATION

The adjoint counterpart of the forward conversion described in Sect. 2is the transformation that computes the derivatives of $\varphi$ with respect to the coordinates of the vertices given the derivatives of $\varphi$ with respect to the voxel values,

$$\frac{\partial \varphi}{\partial V(i,j,k)} \rightarrow \frac{\partial \varphi}{\partial \mathbf{P}_l} \ ,$$

where $V(i, j, k)$ is the value of the voxel at $(i, j, k)$ and $\mathbf{P}_l$ is the vector containing the $(x, y, z)$ coordinates of vertex $P_l$.

### 3.1. Principle

The forward conversion is based on the superposition principle, and so is the adjoint.

As described in Sects. 2.2, 2.3.1. and 2.3.2, the forward conversion is a sequence of transformations. Because of the recursive structure of the splitting algorithm (Sect. 2.3.1.) the sequence of transformation can be seen as a tree (Fig. 8). The root is the facet to be converted, and the leaves are the voxels that depend on the position of the facet. The branches represent the recursive splitting and the computation of the volume overlap. The intermediate results are the sub-facets.

The forward conversion is done going down the tree, meaning that when all the branches have been seen, the leaves contain the volume overlap of the voxels in front of the facet in the $z$ direction.

The adjoint of the conversion is done going up the tree. The leaves contain the derivatives of $\varphi$ with respect to the voxels $\frac{\partial \varphi}{\partial V(i,j,k)}$, and the root will contain the derivative of $\varphi$ with respect to the vertices of the facet $\frac{\partial \varphi}{\partial \mathbf{P}_l}$. The computation of the derivatives follows the adjoint differentiation technique described in the introduction.

### 3.2. Precautions and Other Considerations

Designing forward-adjoint couples is not an easy task, and several precautions have to be taken.

### 3.2.1. Parameterization

The choice of a good parameterization is crucial. Over-determined and non-independent parameterizations are usually helpful in the forward part, but lead to inconsistencies in the adjoint part. For example, in the algorithm described in Sect. 2, one could be tempted to parameterize a sub-facet by its four vertices. The resulting forward code is very easy to write and makes a lot of sense. The adjoint code does not.

Let $\mathbf{P}^0 \cdots \mathbf{P}^3$ be the four vertices describing a planar sub-facet. Writing $\frac{\partial \varphi}{\partial P_z^0}$ assumes that you can "move" $P_z^0$ independently of the other parameters. This is obviously wrong, since the four vertices are assumed to be on a plane.

### 3.2.2. Labeling

The degenerate cases such as the ones mentioned in Sect. 2.3.2.when two points become one (i.e. six variables become three) need to be carefully handled. This can be achieved by labeling the different instances of a general object. For example, in the case raised in Sect. 2.3.2, a label indicates the situation (trapezoid (a), first triangle (b) or second triangle (c)), so that four vertices are considered in the case of the trapezoid, and only three in the case of the triangles.

### 3.2.3. Dual design

Deriving the adjoint algorithm from the forward one is not very efficient in practice. In general, it is highly recommended to design the two algorithms at the same time, the choice in the forward influencing the adjoint and vice-versa. We found useful to write the two codes in the same file, using `#ifdef` statements to separate the forward and adjoint specific parts

## 4. PRACTICAL IMPLEMENTATION

### 4.1. Parameterization

In light of Sect. 3.2.1, we have adopted the following parameterizations for the different objects. A facet is defined by its projected contour on the $(\overrightarrow{x}, \overrightarrow{y})$ plane, and the equation of its plane:

$$\begin{cases} (\mathbf{P}^0, \mathbf{P}^1, \mathbf{P}^2) \\ z = a \cdot x + b \cdot y + c \, . \end{cases}$$

A sub-facet is defined by an interval that gives the "width" of the sub-facet along the axis of slicing, the $x$ or $y$ coordinates of the vertices defining the $(\overrightarrow{x}, \overrightarrow{y})$ projected contour, and the equation of the plane.

$$\begin{cases} (P_x^0, P_x^1, P_x^2) \\ (y_{min}, y_{max}) \\ z = a \cdot x + b \cdot y + c \, , \end{cases}$$

or

$$\begin{cases} (P_y^0, P_y^1, P_y^2) \\ (x_{min}, x_{max}) \\ z = a \cdot x + b \cdot y + c \, . \end{cases}$$

Figure 9 shows the parameterization of a facet (a) and one of the two possible parameterizations of a sub-facet (b).

### 4.2. Volume Overlap

The computation of the volume overlap can be done in a closed form, thanks to the parameterization described above. Let $z_0$ be the height of the voxel plane just below the sub-facet. The integral of the volume below the sub-facet is:

$$V = \int_{x_{min}}^{x_{max}} \mathcal{L}(x) \cdot \frac{(h_1(x) + h_2(x))}{2} dx \, ,$$

where $h_1(x)$, $h_2(x)$ and $\mathcal{L}(x)$ are represented on Fig. 9b.

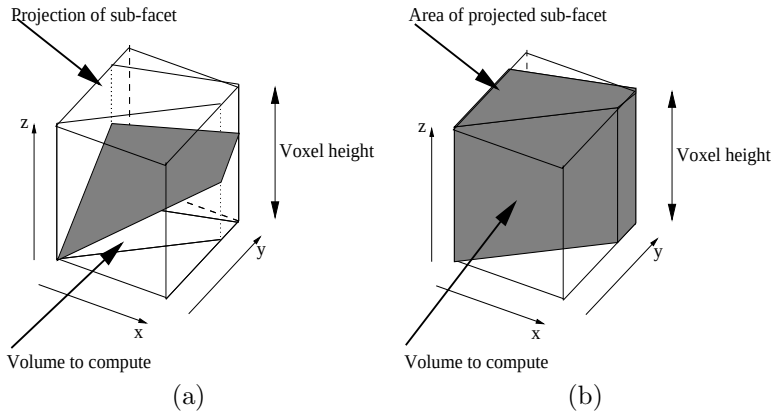


**Figure 7.** Overlap calculation. The overlap of the sub-facet on the voxel that contains it is the volume of the region shown in (a). The volume overlap of the sub-facet on all the voxels in front of the facet is the same and is shown in (b). It is a prismatic volume that is computed by multipliying the area of the base by the height.

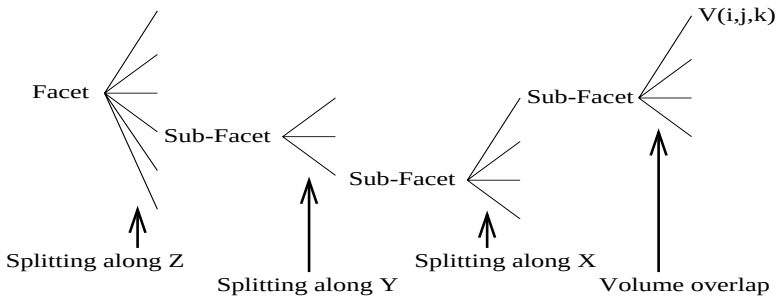


**Figure 8.** Tree structure of the forward conversion. The root is the facet to be converted, and the leaves are the voxels that depend on the position of the facet. The branches represent the recursive splitting and the computation of the volume overlap. The intermediate results are the sub-facets.

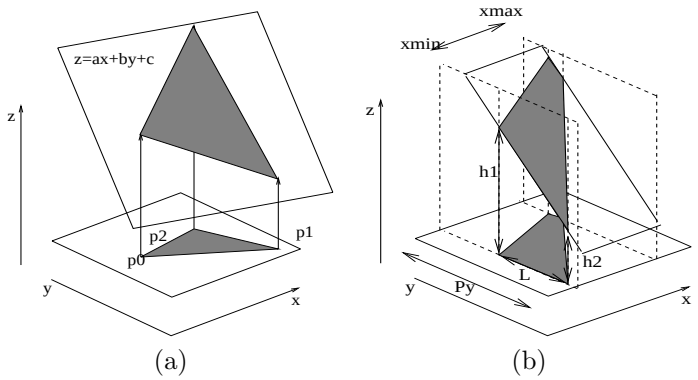


**Figure 9.** Parameterization of a facet (a) and a sub-facet (b).

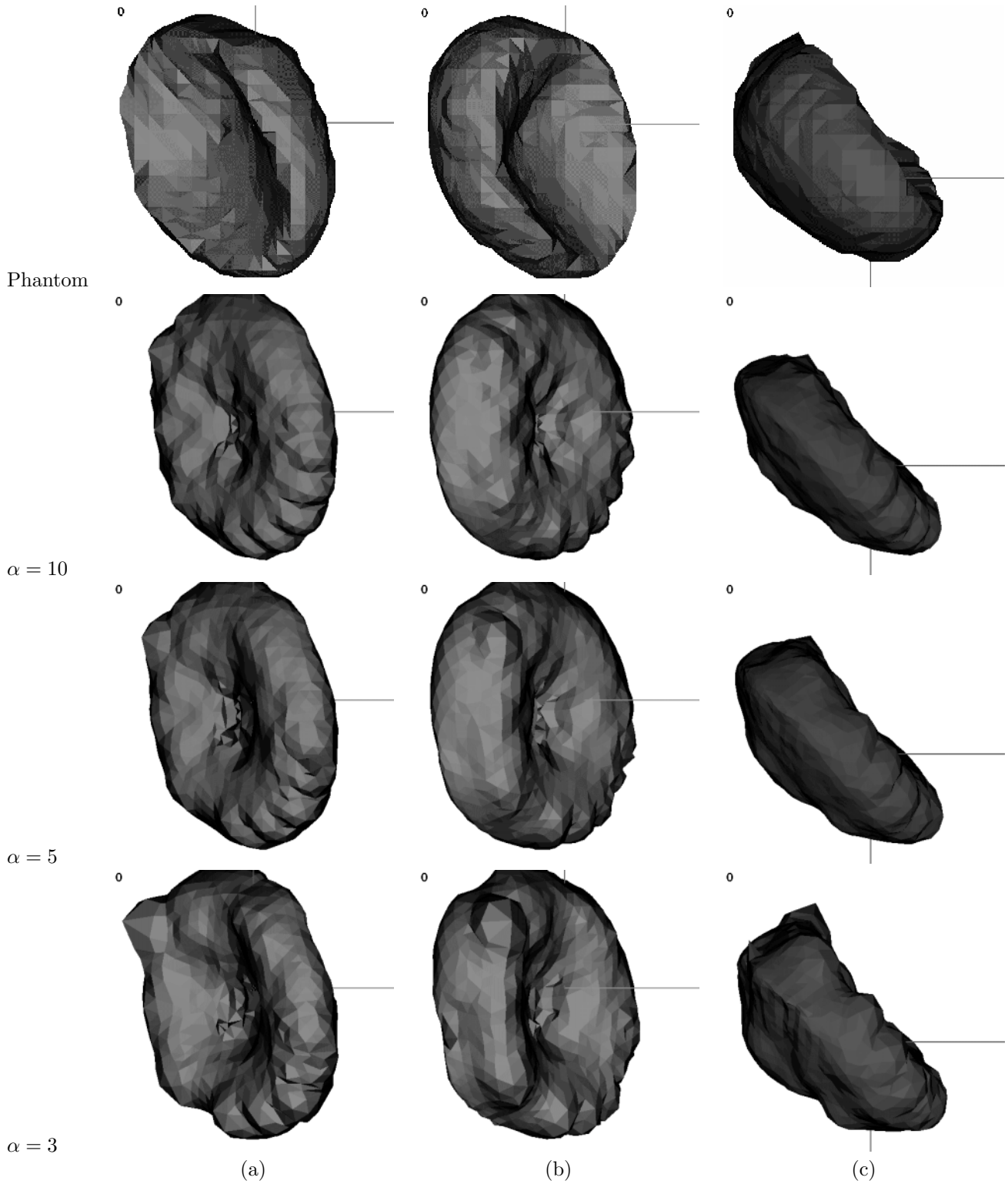


**Figure 10.** Results of reconstruction for different strengths of the prior compared with the original segmented phantom. Three different views of the objects are presented to ease the understanding of their shapes. The column (a) contains views where the $x$ axis is normal to the figure plane. The columns (b) and (c) contain views where respectively the $y$ and $z$ axis are normal to the figure plane.

## 5. RESULTS

We show the efficiency of our approach by performing three-dimensional reconstructions from simulated data. We first try to perform a tomographic reconstruction in a noiseless situation from a set of ten parallel and equally spaced projections. We generate a phantom by warping a sphere into a bean-like shape. We are able to reconstruct the external surface of the phantom from its ten parallel projections. We can achieve an arbitrary precision by increasing the resolution of the surface description. In the absence of noise and with a tesselation containing 960 triangles, the difference between the volume of the reconstruction and the volume of the original phantom was less than 0.5%. The difference between the original phantom and the voxel-converted reconstruction is less than 5% for each individual voxel.

The second set of results are obtained in a noisy situation. For this example we use the measurement matrix for the University of Arizona FASTSPECT[11] imager. This nuclear medicine device takes twenty four views of a radioactive distribution within the body. Each view is acquired by a 64-by-64-pixel detector. The measurement matrix has been measured by moving a 2mm×2mm×2mm point source through the volume of interest. The potential volume viewed by this system contains $40 \times 39 \times 47$ voxels. A phantom representing the heart as a constant interior density volume is used. It is described as an array of $40 \times 39 \times 47$ voxels and the volume of the heart occupies approximately 1800 voxels in this array of 73320. The phantom appears jagged, and does not seem to represent a "smooth" surface. Indeed, when trying to compute the external surface of the voxel-described phantom using IDL, we cannot exactly match the volume of the original phantom, we believe because of this jaggedness. The volume of the region contained in the interior of the surface always differs from the volume of the voxel-described phantom by more than 1.5%. In our simulation, the 24 images of the phantom are acquired using 10000 counts, or 420 counts per detector. The probability of a photon to hit a pixel on a detector is less than one half.

We start the global optimization with a mesh initialized to an ellipsoid fitted to the data. The mesh contains 960 triangles, is refined and smoothed after every other iteration, ending with a much finer mesh containing 2880 triangles. The smoothing operation was performed using X3D, which is general purpose mesh refinement package, and to display the meshes, we use GMV (General Mesh Viewer)*, both developed at Los Alamos.

In order to guarantee a smooth surface, we implemented a prior that penalizes large curvature:

$$\Pi = \frac{\sum_i \sum_j \tan^2(\frac{\theta_{i,j}}{2}) \times \mathcal{A}_i}{\sum_i \mathcal{A}_i} ,$$

where $\theta_{i,j}$ is the angle between the normals of two adjacent triangles, $\mathcal{A}_i$ is the area of the triangle of index $i$ and $\sum_j$ denotes a sum on the triangles that share an edge with triangle $i$.

After optimizing with a strong prior ($\alpha = 10.0$), we proceed to optimize at $\alpha = 5.0$ and $\alpha = 3.0$. The strength of the prior has a considerable influence on the reconstruction. Figure 10 shows the original segmented phantom and the different reconstructions for the various strengths $\alpha$ of the prior. Three different views of the objects are presented to ease the understanding of their shapes. The column (a) contains views where the $x$ axis is normal to the figure plane. The columns (b) and (c) contain views where respectively the $y$ and $z$ axis are normal to the figure plane.

The influence of the strength of the prior $\alpha$ on the reconstructions is obvious. The strongest prior ($\alpha = 10.0$) leads to the smoothest reconstruction, but some features are missed. The weakest prior ($\alpha = 3.0$) leads to a more jagged reconstruction, possibly caused by the high noise level, but seems to better follow the concavity in the original object. We find that the weakest of the three strengths tested leads to the best volume estimate:

| $\alpha$ | $\epsilon$ |
|---|---|
| 10.0 | 12% |
| 5.0 | 8% |
| 3.0 | 4% |

Where $\alpha$ is the strength of the prior, and $\epsilon$ the error on the volume estimate.

*http://ees-www.lanl.gov/EES5/x3d/ and http://laws.lanl.gov/XCM/gmv/GMVHome.html

## 6. CONCLUSION

This work illustrates the first implementation of 3D algorithms and techniques in the BIE. The conversion algorithm described above is the first essential brick in building fully 3D object and measurement models. The experience acquired in writing this complex forward and adjoint code is described in this paper.

The results presented are very promising. Nevertheless, many issues have been raised and need further work. The selection of the strength of the prior is a crucial issue, and an automatic selection of it from the data will be investigated soon. The curvature prior used here penalizes regions of high curvature. It seems that a prior that penalizes the derivative of the curvature might be more appropriate. The mesh-refinement and surface-smoothing codes that we have used are seperate from the BIE, and do not offer a lot of control. A better integration of these capabilities and more flexibility in the way the surface is refined would enhance the ease of use and probably the quality of the reconstructions.

Our goal is to work with real data, such as those that can be acquired with the FASTSPECT system at the University of Arizona. The problem of variable density distributions will also be addressed in the future.

## ACKNOWLEDGEMENTS

This work was supported by the United States Department of Energy under contract W-7405-ENG-36. We are indebted to Harry Barrett and Don Wilson of the University of Arizona for supplying us with their FASTSPECT measurement matrix and a suitable simulated phantom. We wish to acknowledge Harold Trease for helping us with X3D and GMV.

## REFERENCES

1. L. H. Staib and J. S. Duncan, "Model-based deformable surface finding for medical images," *IEEE Trans. Med. Imag.* **15**, pp. 720–731, 1996.
2. A. Guéziec and R. Hummel, "Exploiting triangulated surface extraction using tetrahedral decomposition," *IEEE Trans. Vis. Comp. Graph.* **1**, pp. 328–342, 1995.
3. K. M. Hanson, "Introduction to Bayesian image analysis," in *Image Processing*, M. H. Loew, ed., *Proc. SPIE* **1898**, pp. 716–731, 1993.
4. K. M. Hanson, "Bayesian reconstruction based on flexible prior models," *J. Opt. Soc. Amer. A* **10**, pp. 997–1004, 1993.
5. K. M. Hanson, G. S. Cunningham, G. R. Jennings, Jr., and D. R. Wolf, "Tomographic reconstruction based on flexible geometric models," in *Proc. IEEE Int. Conf. Image Processing, vol. II*, pp. 145–147, IEEE, 1994.
6. G. S. Cunningham, K. M. Hanson, G. R. Jennings, Jr., and D. R. Wolf, "An object-oriented optimization system," in *Proc. IEEE Int. Conf. Image Processing, vol. III*, pp. 826–830, IEEE, 1994.
7. G. S. Cunningham, K. M. Hanson, G. R. Jennings, Jr., and D. R. Wolf, "An interactive tool for Bayesian inference," in *Review of Progress in Quantitative Nondestructive Evaluation*, D. O. Thompson and D. E. Chimenti, eds., vol. 14A, pp. 747–754, Plenum, New York, 1995.
8. G. S. Cunningham, K. M. Hanson, G. R. Jennings, Jr., and D. R. Wolf, "An object-oriented implementation of a graphical-programming system," in *Image Processing*, M. H. Loew, ed., *Proc. SPIE* **2167**, pp. 914–923, 1994.
9. K. M. Hanson and G. S. Cunningham, "The Bayes inference engine," in *Maximum Entropy and Bayesian Methods*, K. M. Hanson and R. N. Silver, eds., pp. 125–134, Kluwer Academic, Dordrecht, 1996.
10. K. M. Hanson and G. S. Cunningham, "A computational approach to Bayesian inference," in *Computing Science and Statistics,* **27**, M. M. Meyer and J. L. Rosenberger, eds., pp. 202–211, Interface Foundation, Fairfax Station, VA 22039-7460, 1996.
11. W. P. Klein, H. H. Barrett, I. W. Pang, D. D. Patton, M. M. Rogulski, J. J. Sain, and W. Smith, "Fastspect: Electrical and mechanical design of a high resolution dynamic spect imager," *Conference Record of the 1995 IEEE Nucl. Sci. Symp. Med. Imag. Conf.* **2**, pp. 931–933, 1996.