An AI Approach to Computer Assisted Tomography

John F. Kolen[a,b], David A. Shamma[a,b], Thomas Reichherzer[a], and Timothy Flueharty[a,b]

Institute for Human and Machine Cognition[a] &
Department of Computer Science[b]
University of West Florida
11000 University Parkway
Pensacola, FL 32514

Abstract

Computer assisted tomography (CAT) systems demand large amounts of time and space. In this paper, we describe an approach to solving the CAT problem using several AI techniques including representation selection and pruning. Rather than backproject intensity data onto a pixel grid or voxel lattice, we solve the problem geometrically — a backprojection region is stored as a convex polygon in a planar decomposition. An algorithm for intersecting two planar decompositions is described below. This algorithm allows us to merge intensity information from multiple sources into a single coherent structure. Knowledge of the task can provide pruning information in the form of stabilized regions that need not to be decomposed.

Background

Standard medical and industrial computer assisted tomography (CAT) systems collect a series of 2-D x-ray images of an object and produce a 3-D reconstruction of the interior structure of the object (Figure 1). The 2-D images are back-projected (Herman, 1980) from the image through the reconstruction arena. The back-projection process is the inverse of the standard computer graphics methods used to project views of 3-D objects onto a 2-D view port (e.g. (Hill, 1990)). The 3-D reconstruction consists of a spacial matrix of density estimates within volume elements (voxels) of the surveyed object. CAT relies upon the assumption that the radiation traveling through the object will be linearly modulated according to the density of the path traced through the object. This assumption allows one to reduce the reconstruction problem, i.e. estimating the density of a given voxel, to solving a large number of linear equations. Other methods, which incorporate various assumptions of the nature of the objects under study, may resort to nonlinear optimization (Verhoeven, 1993). An example of this approach may be found in Figure 2.

We became interested in CAT while working on data reduction for ballistic tests (Gordon et al., 1995). The purpose of these experiments was to characterize the debris fields generated by a projectile penetrating armor. Holograms were taken of the ballistic event and provided a 180-degree view. A 3-D computer model of the debris field was to be constructed from the images. The resource requirements for such a system is very high as the number of voxels scales cubically with respect to the resolution. Assuming a resolution target of 1mm implies approximately 60 million voxels in a 500mm diameter and 300mm high cylinder. If each voxel requires an integer (2 bytes) and a floating point number (8 bytes), the voxel space would require approximately 600Mb of storage. A data structure such as this would overwhelm most workstation CPU and memory systems. Hence, we began to look for alternative methods of performing the CAT process.

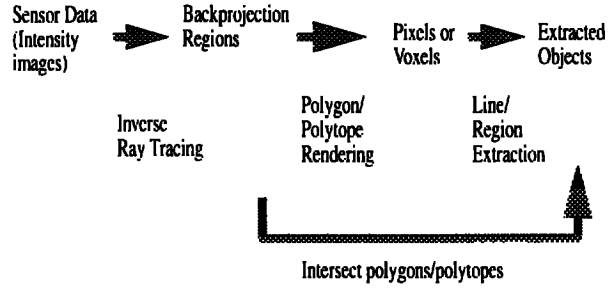

FIGURE 1. Standard CAT architecture. The last step is performed if additional processing/reasoning is required. The gray arrow indicates the approach taken in this paper.

The Planar Algorithm

Our first insight occurred when we realized that polygons were used in two places: to update pixels in the slice (rendering) and in the final product (extraction). A planar decomposition (PD) is a partition of a plane such that every point in the plane either belongs to the interior of some polygon, lies on the edge of a polygon shared by one or two polygons, or is a vertex shared by one or more polygons. A PD is labeled when there is non-geometrical information associated with the polygons composing the PD. The CAT process can be described as the overlaying of multiple labeled PDs to form a new labeled PD (Figure 6). The original labels for the regions are inferred from the image (e.g. the probability that an object lies somewhere in the polygon). The derived labels in the resulting PD can then be

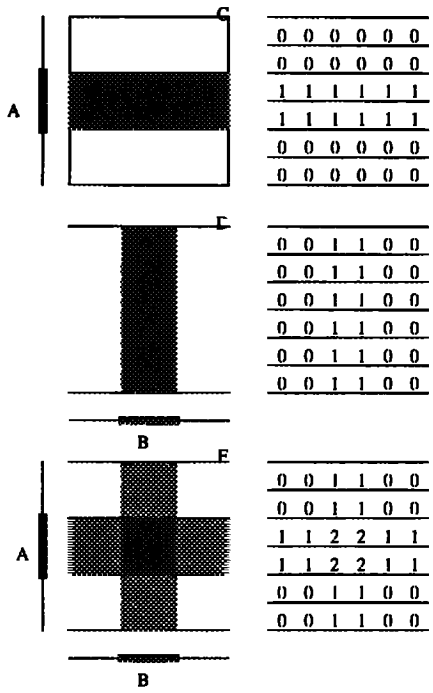

FIGURE 2. An example of CAT. Two images (A,B) are back projected from the images to form planar region C & D. These regions are then averaged to produce region E. The right column illustrates how CAT renders the regions onto a pixel/voxel discretization of the region. Note that the number of pixels/voxels depends upon the desired reconstruction scale.

inferred from original labels of all regions overlapping the given region. For instance, we might label each polygon with a sum and count, where sum is the total confidence for that region and the count is the number of regions contributing to that sum. A new label would be constructed by summing these two quantities. An average confidence can be readily calculated from these values.

The algorithm for merging two PD is fairly straight forward (Figure 3). Consider one PD the destination PD, the other the source PD. Find a common vertex between the two PDs. (If one does not exist, find two edges (one from each PD) that intersect. Given the source of the PD, an intersection should always exist.) From this registration vertex, we recursively visit the spanning tree of the vertex-edge graph defining the PD. As each source edge is visited, this edge is added to the destination PD. Since we keep track of the current vertex in the destination PD and maintain sorted edges for each vertex, it is trivial to discover destination edges that intersect the source edge. A detailed version of the algorithm appears in the Appendix.

In the worst case, a n region PD merged with an m region PD can produce an mn region PD. We use two methods for minimizing this exponential growth effect. First, when an edge crosses a polygon we may or may not divide the existing polygon. One reason for not adding the new edge is that the polygon is indivisible, i.e. we have enough data to conclude that the region is of a certain density. Another reason for not dividing the polygon is that the two polygons separated by the edge have no impact on the existing polygon. This can occur when the involved polygons convey the similar labeling information.

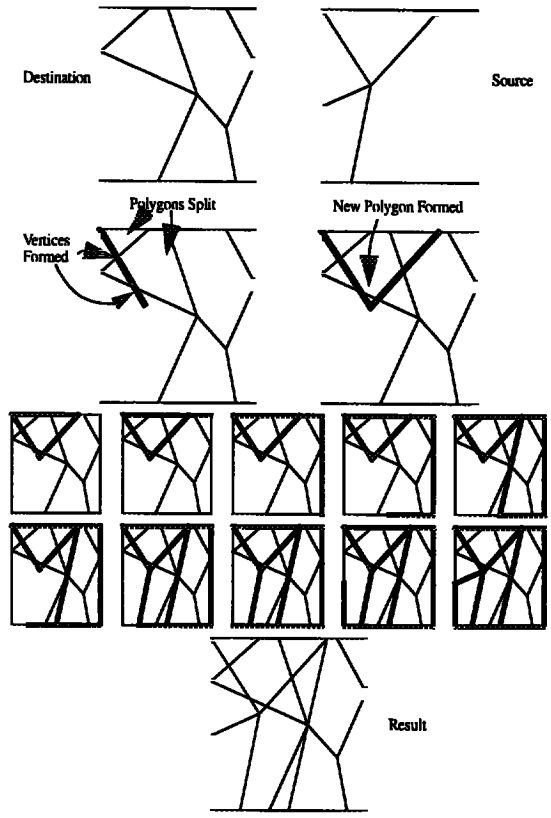

FIGURE 3. The PD merge process. The destination PD is shown after adding each edge from the spanning tree generated by the source PD.

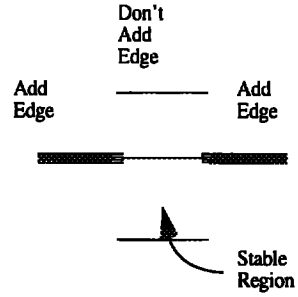

FIGURE 4. Preventive edge pruning. When travelling across a stable region (i.e. a region where new information will not change the final classification) do not add an edge to create two new polygons.

A second method for reducing the number of regions produced by the merge is to perform a post hoc cleanup of the PD. Two adjacent polygons can be combined if their density information is similar to one another. For instance, P_1 may have an average intensity of 100 over 10 images and P_2 may have an average intensity of 99 over 11 images. If this

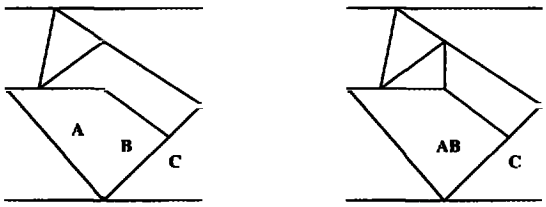

FIGURE 5. Merging Adjacent Polygons. When two polygons are adjacent and have the similar intensity values merge them together if the resulting polygon is convex. Assuming that A, B, and C have similar intensity values, we can merge A and B, but not B and C.

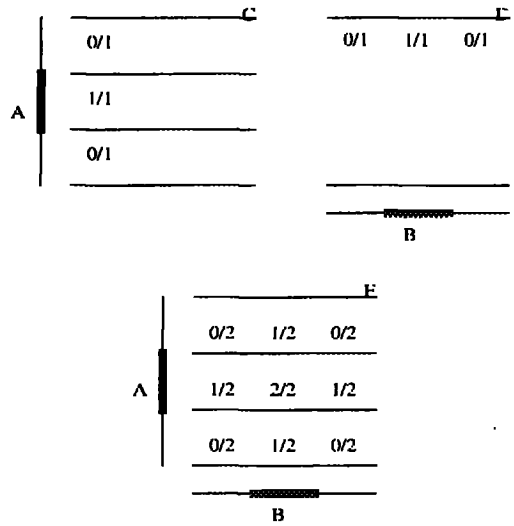

FIGURE 6. CAT using the labeled planar decomposition method described in the text. (X/Y signifies sum over count.) Note that the data needed to specify a given region is independent of the desired reconstruction scale.

difference is insignificant (a problem specific relation), then it is rational to merge P_1 and P_2 into a single polygon. However, we must maintain the convexity of the polygons and only allow the merge if the resulting polygon is convex. If this constraint is not maintained, it is possible to create islands within polygons.

The final 3-D model is constructed by pasting several 2-D planar decompositions together.

Evaluation

A preliminary implementation of the algorithm was evaluated using a synthetic data set described in Figure 7. Images from two latices of different sizes were overlapped and averaged. This implementation did not include merge pruning. We performed the merge process with both the conventional pixel representation and the geometrical approach described above. The implementation of the conventional approach, however, did not include the final step of extracting poly-

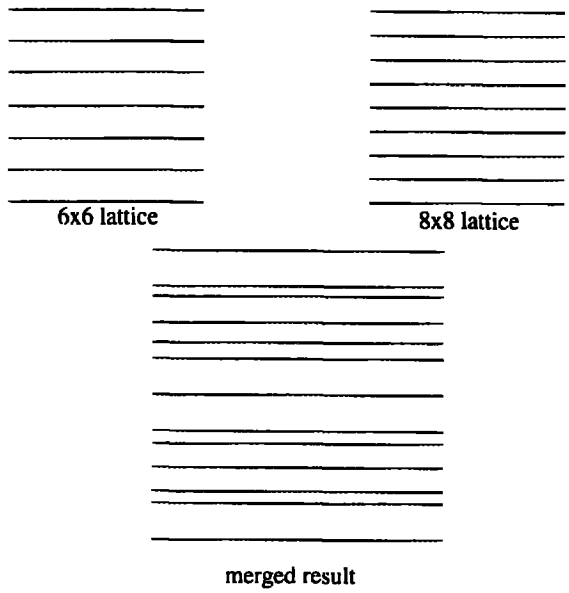

FIGURE 7. The sample data sets are listed as $n \times m$ pairings. The first decomposition is an $n \times n$ lattice, while the second is a $m \times m$ lattice. This figure illustrates the 6-8 pairing. Polygon intensities have been removed for sake of clarity.

gons from the image. The independent variables consisted of image resolution (measured in pixels), lattice granularity of the two original images (the number of regions), and method. The dependent variable was the number of seconds the implementation used to perform the planar decomposition merge. Table 1, Table 2, and Figure 8 summarize the results of the experiment. Run-times reported in the table are averages over five runs on an SGI Indigo^2 (100MHz R4000, 64MB RAM).

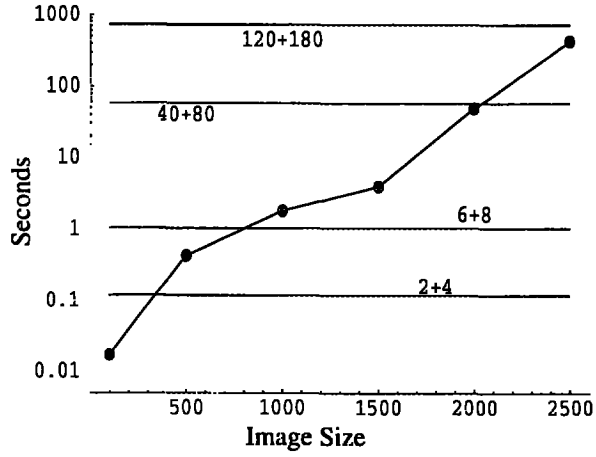

FIGURE 8. Comparison of the two approaches. Since the conventional approach is insensitive to the complexity of the images and the geometric approach is insensitive to the resolution of the images, the run-times for the geometric approach appear as straight lines when plotted against resolution.

The conventional pixel approach is a $\Theta(r^2)$ algorithm, where r is the resolution of the image. The geometric

Pixels	Time
100x100	0.0171
500x500	0.438
1000x1000	1.81
1500x1500	3.97
2000x2000	49.1
2500x2500	432.

TABLE 1. Conventional Method

n-m	Time
2+4	0.120
6+8	1.04
40+80	58.1
120+180	728

TABLE 2. Geometrical Method

approach is a $\Theta(s+d)$ algorithm, where s and d are the number of edges in the source and destination decompositions. The empirical results demonstrate this relationship as well. The run-times for the geometric method are inflated by recursive call overhead. This recursion can be eliminated by providing a stack to manage the spanning tree traversal of the edge graph.

Summary

Appropriate choice of representation and domain knowledge can simplify the CAT process. The major advantage of this method is that it dispenses with the traditional intermediate spatial representation via pixels and voxels. Originally, the final product of a CAT operation was an intensity image of some slicing plane. However, in many applications the tomographic reconstruction is only a small part of a much larger data analysis problem. For instance, we are interested in extracting computer-aided-design-like 3-D models from our reconstructions (e.g. wire-frame models). The planar decomposition representation is much closer to this goal than pixels or voxels as it avoids any additional image processing necessary for the extraction of lines and regions. We also see the planar decompositions as an intermediate representation in that they must go through another processing step (layer gluing). As such, we are exploring a 3-D version of the algorithm that manipulates spatial decompositions (SDs) of convex polytopes.

The method described above is currently being used to develop a data collection system for analyzing behind the armor debris of munitions (Anderson, Gordon, Watts, & Marsh, 1995; Gordon, Watts, & Talbot, 1995). Intensity data is collected from holograms of munition tests. Holograms do not suffer from point of view problems associated with camera-based data collection--viewing angle selection can be performed after the event. Data extracted from the static holograms of test via CCD camera is further reduced to a 3-D structural model of debris. The CAT technique was chosen due to its ability to produce high resolution reconstructions (Cheeseman, Kanefsky, Hanson, & Stutz, 1994) in the presence of noise. These 3-D models will provide quantitative debris data for use in hydrocode modeling of munition/target interactions and lethality/vulnerability modeling of fragment interaction with threat target components and structures.

Acknowledgments

This work was funded by the U.S. Air Force under contract FO8630-96-K-0013.

References

Anderson, C., J. Gordon, D. Watts, J. Marsh. (1995) "Measurement of Behind Armor Debris Using Cylindrical Holograms", In Benton, S. (Ed.), *Practical Holography IX*. SPIE Proceedings Series, Volume 2406. 132-146.

Cheeseman, P., B. Kanefsky, R. Hanson, & J. Stutz, (1994) "Super-Resolved Surface Reconstruction from Multiple Images", Technical Report FIA-94-12, NASA Ames Research Center, Artificial Intelligence Branch, October 1994.

Gordon, J., D. Watts, & I. Talbot, (1995) "Behind-the-armor-debris Data Collection Using Holography". Presented at Test & Evaluation Symposium and Exhibition XI, Austin TX, January 31 to February 3, 1995.

Herman, G. T. (1980) *Image Reconstruction from Projections*. Academic, New York.

Hill, F. S., (1990) *Computer Graphics*. Macmillan Publishing Company, New York.

Verhoeven, D. (1993) "Limited-data computed tomography algorithms for the physical sciences". *Applied Optics*. 32:20. 3736-3754.

Appendix

The planar decomposition merge algorithm is described below.

Data structures

- Points: A point in a two-dimensional Cartesian coordinate system. Points can be marked when visiting them.
- Vertex: A vertex in a graph G(V, E). A vertex maintains angle reference information about each edge with which the vertex is connected.
- Edge: An edge in a graph G(V, E) implemented as a pair of vertices. An edge maintains information about the two possible polygons of which it is a member for both the source planar decomposition (source poly info) and the destination planar decomposition (destination poly info). An edge maintains also the angles between the endpoints of an edge and the horizontal line. Edges can be marked when visiting them.
- Polygon: A list of edges of which the polygon consists and information about the intensity and the cardinality of a polygon.

- Vertex pool: A set of vertices. We impose a less-order on the set. A vertex v is less than a vertex w if $v.x < w.x$ or $v.y < w.y$ when $v.x = w.x$.
- Edge pool: A set of edges. We impose a less-order on the set. An edge a is less than an edge b if the lesser vertex in a is less than the lesser vertex in b. If the lesser vertex in a is equal to the lesser vertex in b then the greater vertex in a must be less then the greater vertex in b.
- Polygon pool: A list of polygons.
- Planar Decomposition (PD): A set of vertices (Vertex pool), a set of edges (Edge pool), and a set of polygons (Polygon pool).

Algorithm

Given is a source PD *srcPD*, and a destination PD *destPD*.

Merge(PlanarDecomp *destPD*, PlanarDecomp *srcPD*)

Merges the source PD with the destination PD. The result will be stored in destPD.

Set *vd* to a vertex in *destPD* whose location is common to both PDs.
Set *vs* to a vertex in *srcPD* whose location is common to both PDs.
Unmark all vertices in the vertex pool of *destPD*.
Mark all edges in the edge pool of *destPD* with OLD.
Unmark all edges in the edge pool of *srcPD*.
MergeCommonVertex(*vd*, *vs*)

MergeCommonVertex(Vertex *v*, Vertex *vNext*)

Merges all outgoing edges from v with outgoing edges from vd

For all unmarked outgoing edges *e* of *vNext* (start with smallest angle of the edge with the horizontal line).
  Mark edge *e*.
  Compute a polygon pair *pPair* consisting of two polygons which *e* may reference.
  Set *vOther* to the vertex in *e* that is different from *vNext*.
  MergeFromCommonVertex(*v*, *vOther*, *pPair*).

MergeFromCommonVertex(Vertex *v*, Vertex *vNext*, PolyPair *pPair*)

Merges an edge defined by v and vNext with the destination planar decomposition.

Set the line segment *ls* to the line between *v* and *vNext*.
Traverse the outgoing edges of *v* and identify:
  The edge *e* that overlaps the line segment *ls* or
  The two edges *e1* and *e2* of which the line segment *ls* is in-between if no outgoing edge overlaps the line segment.
If edge *e* overlaps the line segment *ls* then:
  MergeEdge(*e*, *v*, *vNext*, *pPair*).
else:
  Set *pCommon* to the common polygon of the two edges *e1* and *e2*.
  MergePolygon(*pCommon*, *v*, *vNext*, *pPair*).

MergeEdge(Edge *er*, Vertex *v*, Vertex *vNext*, PolyPair *pInfo*)

Merges two overlapping edges.

Set *vOther* to the vertex in *e* that is different from *v*.
If the location of *vOther* and *vNext* is the same then:
  Set the source polygon info in edge *er* to *pInfo*.
  MergeCommonVertex(*vOther*, *vNext*).
If the location of *vNext* is between the location of *v* and the location of *vOther* then:
  Create a new vertex *vNew* with the location of vertex *vNext*.
  Add *vNew* to the vertex pool of the destination planar decomposition.
  Replace edge *er* with two new edges *e1* = *v vNext*, *e2* = *vNext vOther*.
  Set the source polygon info in the two edge *e1* and *e2* to *pInfo*.
  MergeCommonVertex(*vNew*, *vNext*).
Otherwise the location of vOther must be between the location of *v* and the location of vNext. Then:
  Set the source polygon info in edge *er* to *pInfo*.
  MergeFromCommonVertex(*vOther*, *vNext*, *pInfo*).

MergePolygon(Polygon *pDest*, Vertex *vPoly*, Vertex *vNext*, PolyPair *pInfo*)

Splits a polygon. A single edge e may intersect a polygon completely or partially. In the latter case the outgoing edges from the endpoint of e including their outgoing edges eventually split the polygon into several pieces.

Set the line segment *ls* to the line between *vPoly* (a vertex of the polygon) and *vNext*.
Determine the edge *e* in Polygon *pDest* that intersects *ls*.
If no such edge *e* exists then the line segment must terminate inside the polygon. In this case:
  Create a new vertex *vNew* using the location of vertex *vNext*.
  Add *vNew* to the vertex pool of the destination PD.
  Create a new edge *eNew* = *vPoly vNext*.
  Set the source polygon info in the edge *eNew* to *pInfo*.
  Set the destination polygon info in the edge *eNew* to the pair (*pDest*, *pDest*).
  Mark *eNew* with NEW.
  For all unmarked outgoing edges *e* of vertex vNext in the source planar decomposition do:
  Mark *e* with VISITED.
  Set *pairSrc* to the poly pair info in *e*.
  Set *vOther* to the vertex in *e* that is different from *vNext*.
  MergePolygon(*pDest*, *vNext*, *vOther*, *pairSrc*).
If line segment *ls* intersects *e* in a vertex *vIntersect* then:
  Create a new edge *eNew* = *vPoly vIntersect* and add it to the edge pool of the destination PD.
  Adjust source polygon info of the two edges in polygon *pDest* that are adjacent to *eNew* with *pInfo*.
  Set source and destination polygon info of *eNew* using *pInfo* and pair (*pDest*, *pDest*).
  If the location of *vIntersect* and *vNext* is identical then MergeCommonVertex(*vIntersect*, *vNext*).
  Otherwise MergeFromCommonVertex(*vIntersect*, *vNext*, *pInfo*).
Otherwise do the following:
  Add a new vertex *vIntersect* to the vertex pool of the destination PD. The location of *vIntersect* is the intersection of *ls* with *e*.
  Add a new edge *eNew* = *vPoly vIntersect* to the edge pool of the destination PD.
  Replace edge *e* by the two edges *e1* and *e2*. Both edges result from the splitting of *e* with *vIntersect*.
  Update source and destination polygon info for *eNew*, *e1*, and *e2* using pair (*pDest*, *pDest*) and *pInfo*.
  If line segment *ls* intersects *e* and the endpoint of *ls* is located on *e* (T-intersection) then:
    MergeCommonVertex(*vIntersect*, *vNext*).
  Otherwise line segment *ls* intersects *e* and the endpoint of *ls* is located outside of the polygon (X-intersection). Then:
  If the endpoint of *ls* is located in another polygon then:
    Set *nextPoly* to be the polygon of destination PD in which the endpoint of *ls* is located.
    MergePolygon(*nextPoly*, *vIntersect*, *vNext*, *pInfo*).
  otherwise:
    MergeFromCommonVertex(*vIntersect*,*vNext*, *pInfo*).