aprendtech.com >> blog >> this post

If you have trouble viewing this, try the pdf of this post. You can download the code used to produce the figures in this post.

A projection simulator–Matlab implementation

edit: Sep 9, 2011 I extended the code to include convex polygons and renamed the function as $CTProjSim$.

I previously discussed the rationale, the C++ implmentation, and the the Matlab interface for a computed tomography projection simulator. In this post, I discuss a Matlab-only implementation of a simulator. The simulator is includes ellipses and convex polygons.The projection lines are assumed parallel but it is simple and can be (fairly) easily extended to other object types and geometries.

Ellipse

The geometry for the ellipses is shown in Fig. 1. With this geometry, the length of the intersection of the projection line with an ellipse is

$$T = \begin{cases} 2ab\sqrt{s_m^2 - s^2}/s_m^2 & |s| \leq s_m \\ 0 & |s| > s_m \end{cases}$$

where s is the distance of the line L from the center of the ellipse and θ is the angle from the perpendicular of the line to the principal axis of the ellipse. Also, the maximum offset of the ellipse perpendicular to the projection line is $s_m^2 = a^2\cos^2(\theta) + b^2\sin^2(\theta)$.The code of the function $CTProjSim$ is a straightforward implementation of these formulas.

I used the function to compute the projections of the Shepp-Logan phantom and reconstructed with my CTrecon function. The results are shown in Fig. 2.

Convex polygon

We can use concepts from my previous posts on intersection of line segments and polylines to compute the intersection of a line with a polygon. I assume the polygon is convex so the line can intersect with at most two segments of the polygon. The geometry is shown in Fig. 3. In two dimensions, we can specify a line by a vector $\mathbf{n}$ through the origin and perpendicular to the line. We can write $\mathbf{n}$ as $n\hat{\mathbf{n}}$, where $\hat{\mathbf{n}}$ is a unit length vector. If the line passes through the origin, $n = 0$, but we still know its direction from $\hat{\mathbf{n}}$.

A point on the line is

$$\mathbf{r_L} = n\hat{\mathbf{n}} + t\hat{\mathbf{s}}$$

while a point on the line segment is

$$\mathbf{r_S} = \mathbf{P_1} + u\mathbf{d}.$$

At the intersection,

$$\mathbf{r_L} = n\hat{\mathbf{n}} + t\hat{\mathbf{s}} = \mathbf{r_S} = \mathbf{P_1} + u\mathbf{d}.$$

Taking the dot product of this equation with $\hat{\mathbf{n}}$

$$n = \hat{\mathbf{n}} \cdot \mathbf{P_1} + u\hat{\mathbf{n}} \cdot \mathbf{d}$$

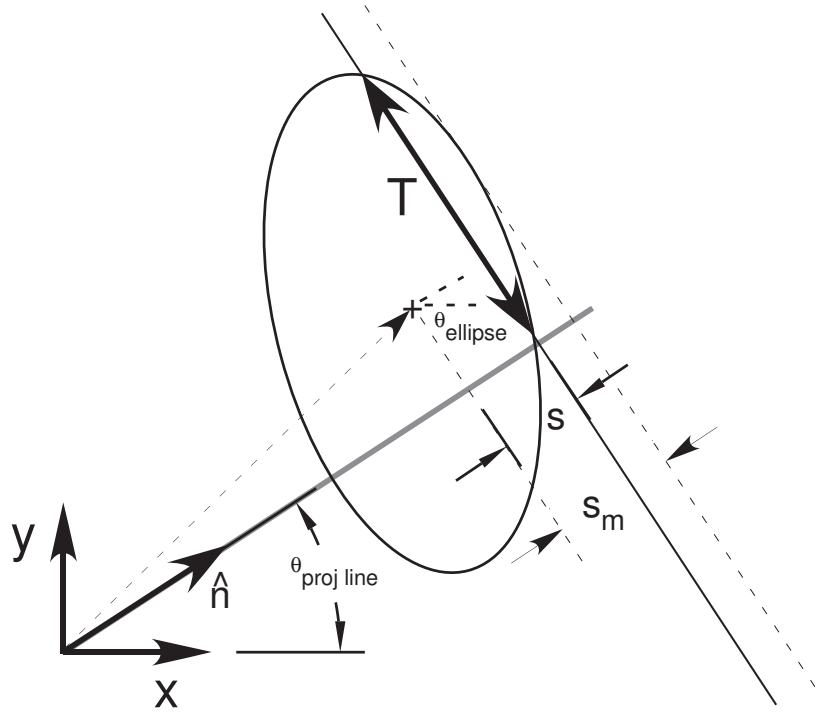

Figure 1: Geometry of simulator. The ellipse has major and minor semi-axes a and b and the major axis is at angle $\theta_{ellipse}$ with the x-axis. The ellipse center is $\mathbf{z_{ell-center}}$and the perpendicular to the projection line is $\hat{\mathbf{n}}$. Therefore the offset of the center along the projection line perpendicular is $ellipse\_center\_offset = \hat{\mathbf{n}} \cdot \mathbf{z_{ell-center}}$.

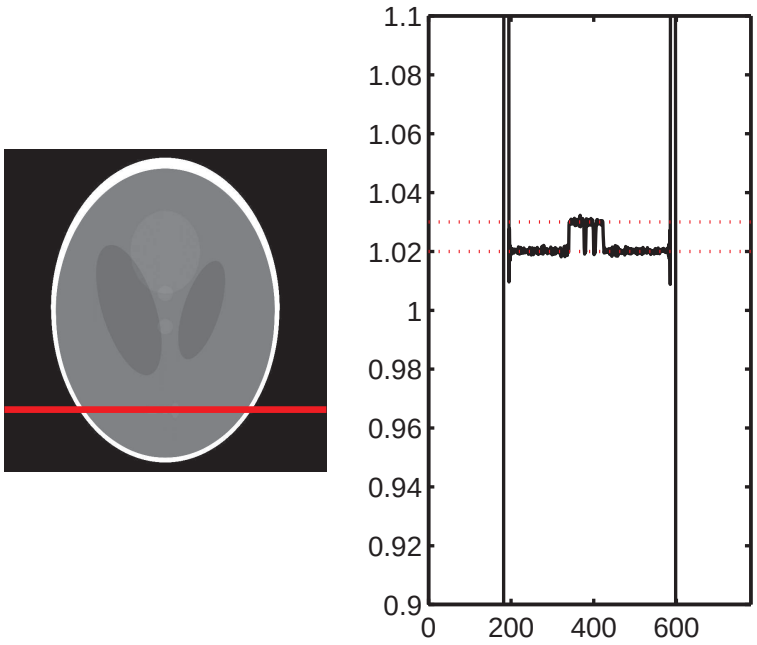

Figure 2: Reconstruction of projections of Shepp-Logan phantom produced with $CTProjSim$. My CTrecon function was used to reconstruct so the results do not have the offsets introduced by the Matlab $iradon$ function. The red lines are the accurate values of the phantom density.

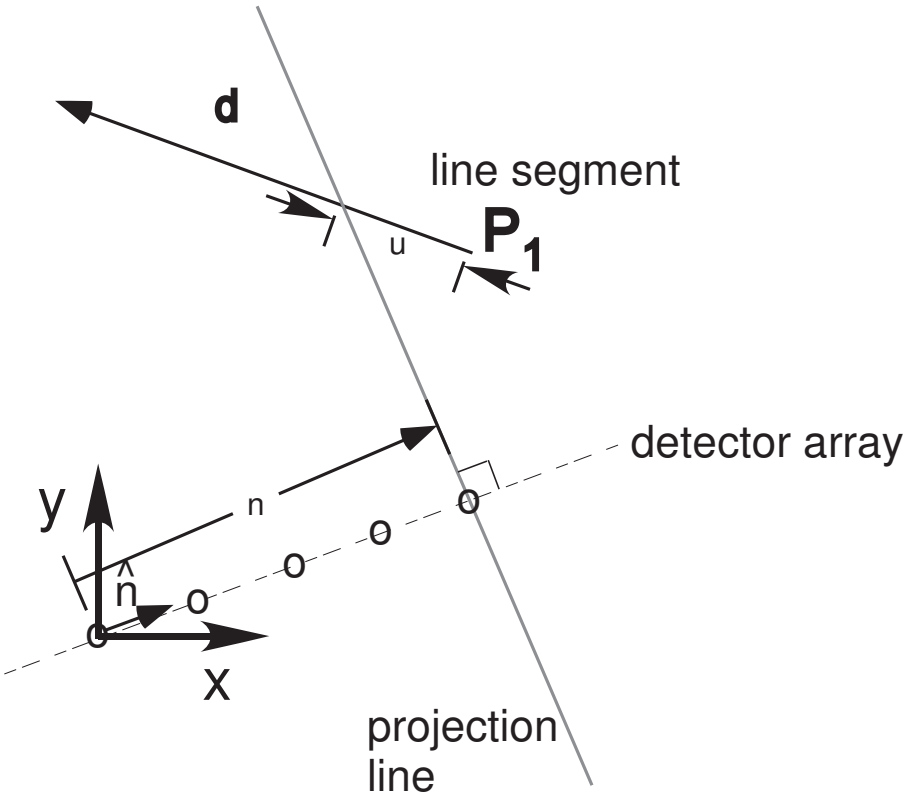

Figure 3: Intersection of a line and a line segment. The line is specified by a vector $\mathbf{n}$ through the origin and perpendicular to it. We can express $\mathbf{n}$ is $n\hat{\mathbf{n}}$, where $\hat{\mathbf{n}}$ is a unit length vector. The line segment is specified by an endpoint $\mathbf{P_1}$ and a vector $\mathbf{d}$ to the other endpoint. The intersection of the line with the extended line segment is at $\mathbf{P_1} + u\mathbf{d}$. The intersection is within the line segment if $0 \leq u < 1$.

Solving for u

$$u = \frac{n - \hat{\mathbf{n}} \cdot \mathbf{P_1}}{\hat{\mathbf{n}} \cdot \mathbf{d}}$$

To find the intersection of a line with a polygon, we can test $0 \leq u < 1$ for all the segments of the polygon sides. If there are two intersections, then the line overlaps the polygon. We can compute the intersect length T as

$$T = |\mathbf{r_{intersect,1}} - \mathbf{r_{intersect,2}}|$$

Multiplying T by the (possibly vector) density gives the line integral.

These formulas are implemented in the $CTProjSim$ along with the code for the intersection with ellipses. Line integrals are linear so we can add the line integrals for different shapes for each projection line.

I created the projections of a hexagon embedded in an ellipse and reconstructed them to produce the image in Fig. 4. The sharp vertexes of the polygon cause large aliasing artifacts in the reconstruction. I reduced them somewhat by filtering them with the $'filter\_type','hamming','freq\_cutoff',0.9$ parameters in $CTrecon$. You can download the code to reproduce the figures in this post.

Last edited Sep. 9, 2011

Figure 4: Reconstructed projections of a hexagon embedded in an ellipse.