A fast but ill-conditioned formal inverse to Radon transforms in 2D and 3D

Ricardo Oliva-García[a, b], José G. Marichal-Hernández[a,*], Óscar Gómez-Cárdenes[a], Nelson Suárez-Martín[a], and José M. Rodríguez-Ramos[a,b]

[a]Universidad de La Laguna, Industrial Engineering Department, ETSI, La Laguna, Spain, 38200
[b]Wooptix S.L., Av. Trinidad, 61, La Laguna, Spain, 38204

ABSTRACT

We present a formal inversion of the multiscale discrete Radon trasform, valid both for 2D and 3D. With the transformed data from just one of the four quadrants of the direct 2D Radon transform, or one of the twelve dodecants, in case of 3D Radon transform, we can invert exactly and directly, with no iterations, the whole domain. The computational complexity of the proposed algorithms will be O(N log N). With N the total size of the problem, either square or cubic. But this inverse transforms are extremely ill conditioned, so the presence of noise in the transformed domain turns them useless. Still we present both algorithms, and characterize its weakness against noise.

Keywords: Radon transform, DRT, Numerical transforms, Invertibility

https://doi.org/10.1117/12.2617409

Ricardo Oliva-García, José G. Marichal-Hernandez, Óscar Gómez-Cárdenes, Nelson Suarez-Martin, and José M. Rodríguez-Ramos "A fast but ill-conditioned formal inverse to Radon transforms in 2D and 3D", Proc. SPIE 12102, Real-Time Image Processing and Deep Learning 2022, 1210209 (27 May 2022) *Contact author, jmariher@ull.edu.es

1. INTRODUCTION

Radon transform[1] is an integral transform useful to deal with problems where instead of sensing directly a magnitude, it is possible to sense its interaction with a physical medium in a straight line within a two-dimensional domain, a plane. That is the reason why this mathematical tool is the foundation for medical imaging: it can undo the integrals of the rays attenuation as they travel through body tissues. When used in this way, it is called inverse transform. The direct transform, which is not of interest in medical imaging, since it is taken care of by the scanner in that context, calculates the integrals of any line that traverses a plane. It is assumed that the direct problem is much simpler than the inverse problem, which, depending on sampling and

measurement noise, may not have a solution; and even when it is invertible, the inverse solution is achieved by slower methods than the direct one.

In 3D, Radon transform extends to plane integrals within a cube.[2] Radon transform also has been extended to reduce the 4D plenoptic space of rays by mimicking lens focusing –an integration– to generate a 3D focal stack.[3]

In the field of medical imaging the two-dimensional inverse transform is achieved thanks to the slice-projection theorem, which allows to express Radon in terms of the Fourier transform.[4] Thus, Radon's computation benefits from the existence of an optimal algorithm for computing the Fourier transform: the FFT algorithm.[5]

To accelerate the computation of the two-dimensional direct Radon transform beyond what Fourier-based methods achieve, a divide-and-conquer algorithm was proposed in the 1990s,[6–8] but on integer mathematics, which is called the discrete Radon transform, DRT. Fourier-based methods sometimes also receive that name, but by DRT we are referring specifically to the multi-scale method in integer mathematics, which has advantages over any other method implemented on computers.[9]

In 2006, it was proven that DRT could be computed for the reverse path, but with a higher computational cost. The solution, despite being iterative, was accurate and fast.[10] More recently, in 2021, a hybrid formulation was found to solve the inverse transform with only two iterations.[11] Although Fourier deconvolution was used to filter the result, the method was not related to the slice-projection theorem.

By the same time, a method was found to exactly invert the DRT with the same number of operations as that of the forward DRT.[12,13] Moreover, it did not need projections covering the whole semicircumference, 180 angles, but only one quadrant: 45 degrees of projection.

In this contribution we will deepen in the expression as an algorithm of that method.[12] Then we will extend it to three dimensions. We will make some considerations about its application with noise.

1.1 Discrete Radon transform

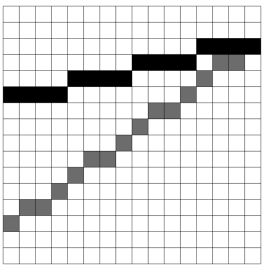

Figure 1: Two discrete lines of parameters (s, d) in a 16×16 grid. In black, the line (3, 10); in gray the line (11, 2).

The code that actually computes the forward DRT of a quadrant is shown in the listing 1. The algorithm computes the line integrals of the quadrant formed by the angles between 0 and 45 degrees, which are those that can be expressed in the form $y = s \cdot x + d$, where d is the

*d*isplacement on the vertical axis and s represents the *s*lope. Actually, the s variable is expressed in ascents between 0 and N-1, while x runs along the side of an image of size N. So to interpret it as slope, instead of ascent, $s/(N-1)$ should be used, which is in the range 0 to 1.

Listing 1: Matlab code to compute the forward DRT of a quadrant

```
%% Forward Radon transform of a quadrant
function Rf = qDRTforward(f)
[N, N] = size(f);
% we will go from coordinates (y, x)
% to coordinates (d, s): d displacement, s slope
% output size: double of displacements than slopes
fm = zeros(2*N, N);
fmp1 = fm;

n = log2(N);

% We start with f on the upper half of the data
fm(N +1:2*N,:) = f;

for m = 0:n-1 % % stages, log2(N)
    for d = 0:2*N-1 % for every displacement
         % slope variable, comprise 3 elements: {v: n-m-1 bits; sigma: m-1 bits; s0: 1 bit}
        for v = 0:(pow2(n - m - 1) -1)   % for every band not transformed yet
            for sigma = 0:(pow2(m) -1) % within a band, already transformed
                for s0 = 0:1  % last bit of slope, the one being transformed
                   if ((d+s0+sigma) <= (2*N -1))
                       aux = fm(d+s0+sigma +1, sigma+(1*pow2(m))+(v*pow2(m+1)) +1);
                   else
                       aux = 0;
                   end
                   fmp1(d +1, s0+(sigma*2)+(v*pow2(m+1)) +1) = ...
                       fm(d +1, sigma+0+(v*pow2(m+1)) +1) + aux;
                end
            end
        end
    end
    fm = fmp1;
end

Rf = fm;
end
```

The projections sweeping the whole semicircumference are obtained by repeating this computation for another 3 quadrants, corresponding to the lines formulated as: $y = -s \cdot x + d$, $x = s \cdot y + d$ y $x = -s \cdot y + d$; again with the slope, $s/(N-1) \in [0..1]$. This is achieved, without modifications to the algorithm, but by applying it to suitably rotated versions of the image.

The algorithm has complexity $O(N^2 logN)$ because the lines are approximated by stripes of width 1 pixel, which traverse the image in integer positions ascending without interpolation from the cartesian coordinate (0, d) to (N-1, d+s). And the line integrals are approximated by the summations of the intensity of the values over those stripes. These stripes can be formulated recursively so that the computation of the summation over any line of arbitrary length is achieved by adding the summation on two line segments of half that length. No intermediate adds are calculated more than once. Two of those loose discrete lines are shown in figure 1.

The core of the DRT formulation is constituted by the following 3 equations.

The recursive definition of a discrete line, which ascends $s \in [0..2^n - 1]$ over a domain u ranging between 0 and $2^n - 1$, expressed in binary, $\mathbf{u} = \{u_0, u_1, \ldots, u_{n-1}\}$:

$$l_s^n(u_0, \ldots, u_{n-1}) \quad = \quad l_{\lfloor s/2 \rfloor}^{n-1}(u_0, \ldots, u_{n-2}) \; + \; u_{n-1} \left\lfloor \frac{s+1}{2} \right\rfloor \quad = \quad \sum_{i=0}^{n-1} u_{n-1-i} \; \cdot \; \left\lfloor \frac{\frac{s}{2^i}+1}{2} \right\rfloor \qquad (1)$$

The definition of partial transform, $\widetilde{f}^{\,m}$, of an input f up to stage m, that contains the summations on all the lines that ascend $s \in [0..2^{m-1}]$, in all horizontal bands $v \in [0..2^{n-m}-1]$, each of width 2^m, starting from any displacement d. Note that $\lambda(\cdot)$ function returns the decimal value from a set of binary indices.

$$\widetilde{f}^{\,m}(\, s_{n-m}, \overbrace{s_{n-m+1},\ldots,s_{n-1}}^{\boldsymbol{\sigma}} \,|\, \overbrace{v_m,\ldots,v_{n-1}}^{\mathbf{v}} \,|\, d) \;=\; \sum_{\mathbf{u}\in\mathbb{Z}_2^m} f(\lambda(\mathbf{u},\mathbf{v})|l^m_{\lambda(\boldsymbol{s})}(\mathbf{u}) + d) \quad (2)$$

And the mapping equation between partial transforms in two consecutive stages. Thanks to the recursive definition of lines, eq. (1), and due to the definition of partial transforms in eq. (2), the computation of the solution up to any stage, $m+1$, for any horizontal band v, and any ascent s, can be expressed as the sum of the summations of two segments of half the length that ascend $\sigma = \lfloor \frac{s}{2} \rfloor$; both of which must be already computed in the partial transform up to stage m.

$$\widetilde{f}^{\,m+1}(\, s_{n-m-1}, \overbrace{s_{n-m},\ldots,s_{n-1}}^{\boldsymbol{\sigma}} \,|\, \overbrace{v_{m+1},\ldots,v_{n-1}}^{\mathbf{v}} \,|\, d) =$$
$$\widetilde{f}^{\,m}(\boldsymbol{\sigma}|\, 0,\mathbf{v}|\, d) + \widetilde{f}^{\,m}(\boldsymbol{\sigma}|\, 1,\mathbf{v}|\, d + s_{n-m-1} + \lambda(\boldsymbol{\sigma})) \quad (3)$$

So in order to arrive to the final solution:

$$\widetilde{f}\,(s|d) = \widetilde{f}^{\,n}(\, s_0,\ldots,s_{n-1} \,|\, d) = \sum_{u=0}^{N-1} f(\, u \,|\, l^n_{\lambda(\boldsymbol{s})}(u) + d)$$

from an input $\widetilde{f}^{\,0}(s|-|d) = f(s,d)$, the aforementioned mapping equation must be enclosed in a series of for loops on the variables m, s, v and d. Note that the parameters in the functions are separated by vertical bars (|), while commas (,) are reserved for separating binary indices within a parameter.

Algorithm 1 is the translation of equation (3) into code, –adding the necessary loops and zero padding the input to take into account those lines starting below displacement 0 that reach the input due to their ascent.

The mapping equation is depicted in figure 2 without the mathematical apparatus. It relates –for a fixed m, d, σ and v but unrolled for $s_{n-m-1} \in [0,1]$, the last bit of ascent up to that stage–, two elements of the partial transform at stage $m+1$ with three elements from the previous stage of solution, m. In the figure we have designated as α a segment participating in stage $m+1$, with ascent s that starts at d in the vertical band v. It is calculated by joining the segments designated as A and C on the previous stage.

Segment A, of half the length than α, ascends $\sigma = \lfloor \frac{s}{2} \rfloor$ from d in vertical band $2v$; and segment C, ascends $\sigma = \lfloor \frac{s}{2} \rfloor$ in vertical band $2v+1$ but starting from $d+\sigma+1$. Meanwhile the

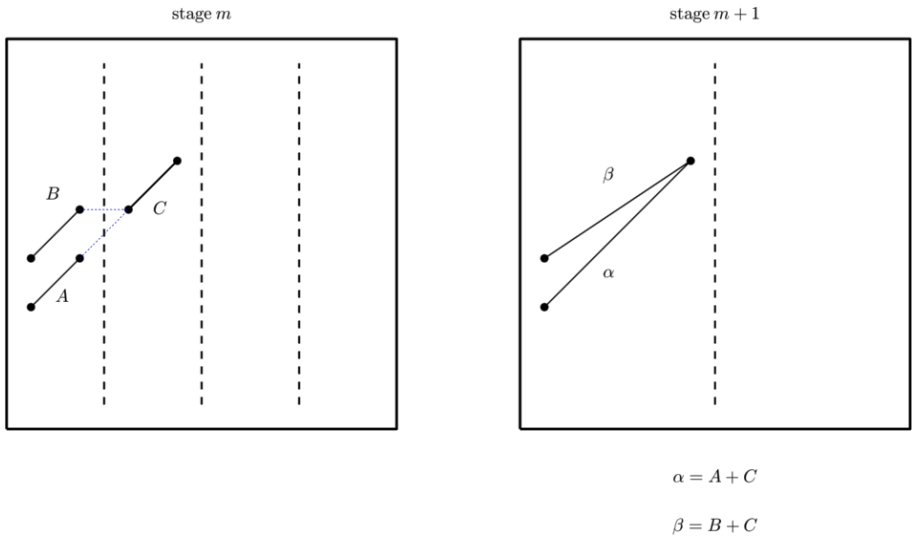

Figure 2: Scheme of computation of forward Radon transform.

computation of β, that ascends $s-1$ from $d+1$, shares C with α, but additionally needs B, which is the segment that shares band, and slope with A but starts on $d+1$.

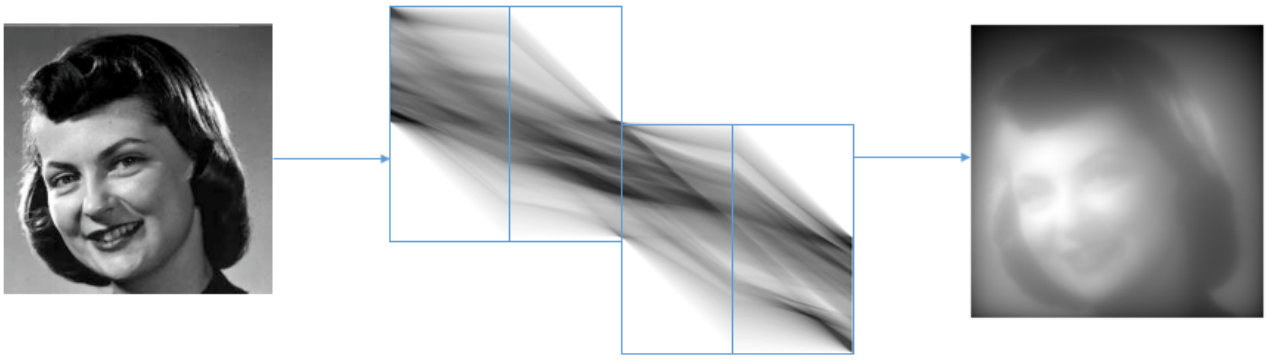

Figure 3: Left to right: an image, its forward DRT constituted by 4 quadrants, its backprojection.

The effect of applying forward discrete Radon transform on an image can be seen in figure 3. The explained algorithm accounts for the third quadrant from left to right in the complete DRT.

1.2 Inversion methods

There are two published methods that deal with the inverse or backward discrete Radon transform.[10,11] Both use as intermediate step the adjoint transform, also called backprojection. While in other transforms it is possible to return from the transformed domain to the initial domain with an algorithm of the same complexity as the one used for the direct path; in the discrete Radon transform, where summations of the initial data have to be undone, this is not the case. If we try to revert the steps of the forward transform algorithm, we do not arrive to the backward

algorithm, but to the adjoint operator, or backprojection. A backprojection is illustrated on the right of the figure 3, and coincides with the 'unfiltered' backprojection in Fourier methods, where there is an imbalance between low and high frequencies unless a ramp filter is applied.

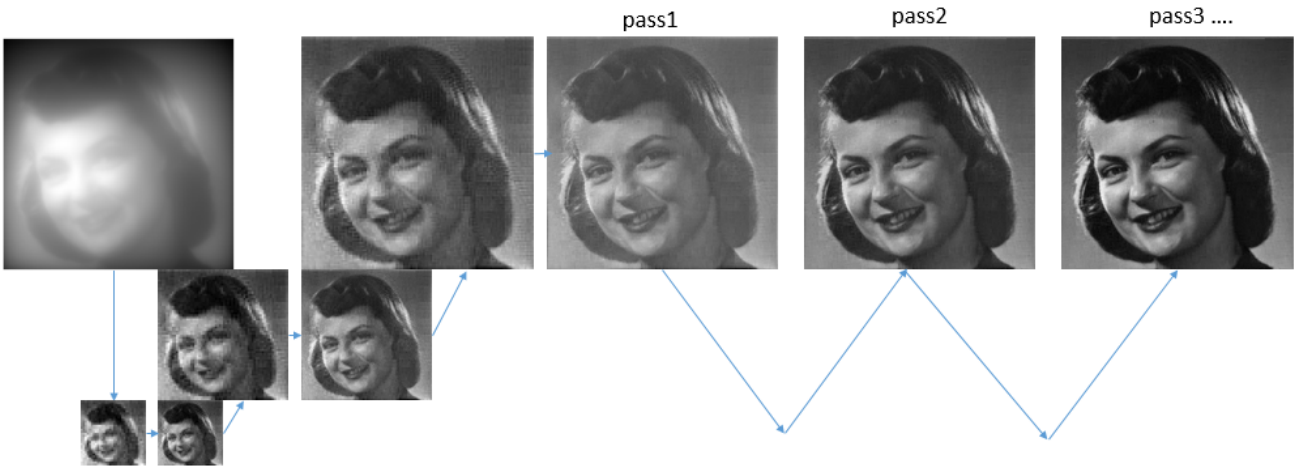

Figure 4: Scheme of the inversion method based on multigrid reduction of error.

The two existing inverse DRT algorithms start by applying backprojection, which will be covered in subsection 1.3, and then use different strategies to clean that blurred intermediate result, to arrive to the solution.

Press[10] proposes a multigrid method to accelerate an iterative error cleanup in a loop of successive applications of the direct+adjoint operator. It is illlustrated on figure 4. Starting from the backprojected image, an iterative error substraction is applied, taking advantage that the adjoint operator approximates to the inverse operator. The method is further accelerated solving, for each iteration, at all possible subscales of the problem (illustrated in the figure only for the first pass).

On the other hand, the method proposed by Marichal et al.[11] expands the sinogram to backproject to an image size that is 3 times bigger than the original. This expanded backprojection can then be cleaned by deconvolving with the point spread functions, PSFs, of the direct+adjoint combined operator. It is not exactly a time invariant operator, but the diversity and distribution of the different PSFs is known before-hand so that it can be cleaned with just two deconvolution stages, instead of the several ones required by Press' method, as shown in figure 5.

1.3 Adjoint operator

The adjoint operator aim is to revert what the forward DRT accomplishes, but as there are data that has been added together, there is no way to tell them apart. What it does is illustrated in figure 6, and the corresponding code is supplied in listing 2. The algorithm advances from stage m to stage $m-1$. Three segments of the m-th stage and how they backproject their energy to segments of half their length in the $m-1$ stage, are depicted.

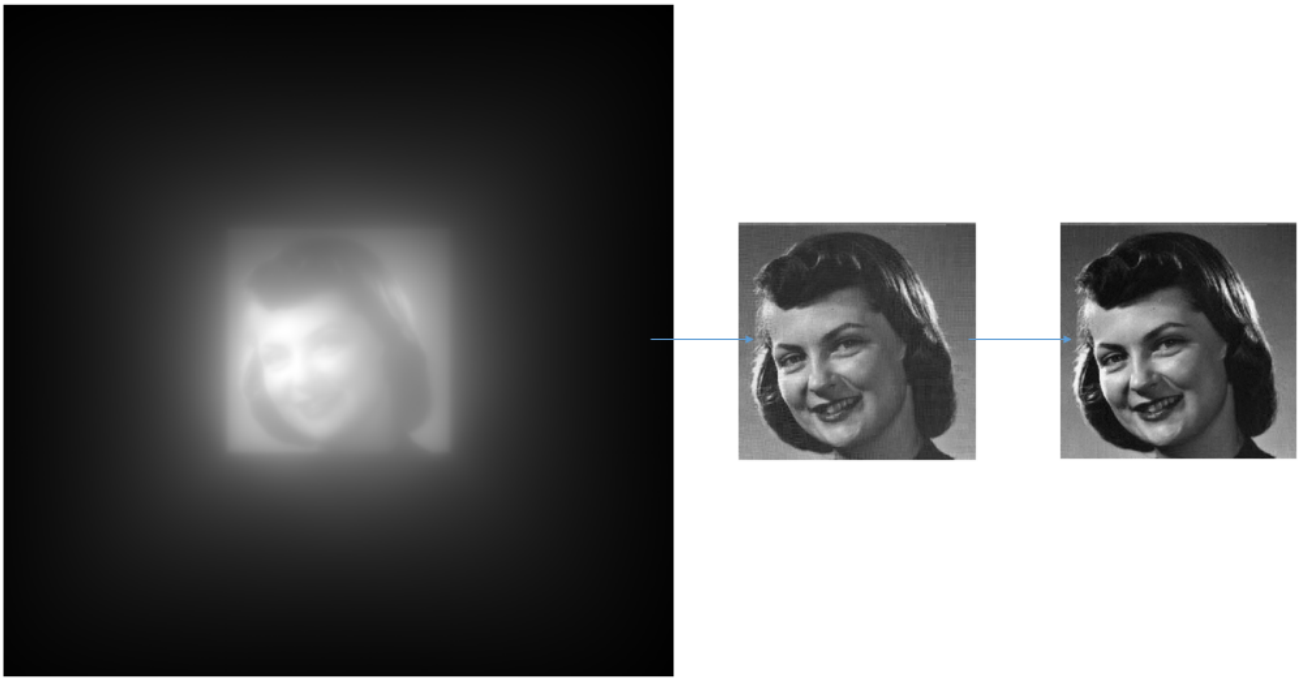

Figure 5: Scheme of the inversion method based on expanded backprojection and PSFs deconvolution.

In the forward stage, as shown in figure 2, C was half the contribution used to compute α and β. Now C will be computed as the average of both, α and β. Similarly B can now be estimated as half β and half γ. Since it is not possible to know what B contributed and what C contributed in the forward algorithm, we can now only backproject them equally. This data averaging is responsible for the blurring that needs to be cleaned up for the input image to be fully recovered.

In the algorithm there is not an explicit division by 2 at each stage, because it is performed at the end as a unique division by N.

Listing 2: Matlab code to compute the backprojection DRT of a quadrant

```
%% Radon backprojection or adjoint transform of a quadrant
function f = qDRTbackproject_FOR(Rf)
[N, N] = size(Rf);
Rfm = Rf; clear Rf;
Rfmm1 = zeros(2*N, N, 'like', Rfm);

n = log2(N);

for m = n-1:-1:0 % stages, log2(N), now in reverse order
    for d = 0:2*N-1 % for every displacement
        % slope variable, comprise 3 elements: {v: n-m-1 bits; sigma: m-1 bits; s0: 1 bit}
        for v = 0:(pow2(n - m - 1) -1);  % for every band not transformed yet
            for sigma = 0:(pow2(m) -1) % within a band, already transformed
                for s0 = 0:1  % last bit of slope, the one being transformed
                    Rfmm1(d +1, sigma+0+(v.*pow2(m+1)) +1) = ...
                        Rfmm1(d +1, sigma+0+(v.*pow2(m+1)) +1) + ...
                        Rfm(d +1, s0+2*sigma+(v.*pow2(m+1)) +1);

                    if ((d+s0+sigma) <= (2*N -1))
                        Rfmm1(d+s0+sigma +1, sigma+(1*pow2(m))+(v.*pow2(m+1)) +1) = ...
                            Rfmm1(d+s0+sigma +1, sigma+(1*pow2(m))+(v.*pow2(m+1)) +1) + ...
                            Rfm(d +1, s0+2*sigma+(v.*pow2(m+1)) +1);
                    end
                end
```

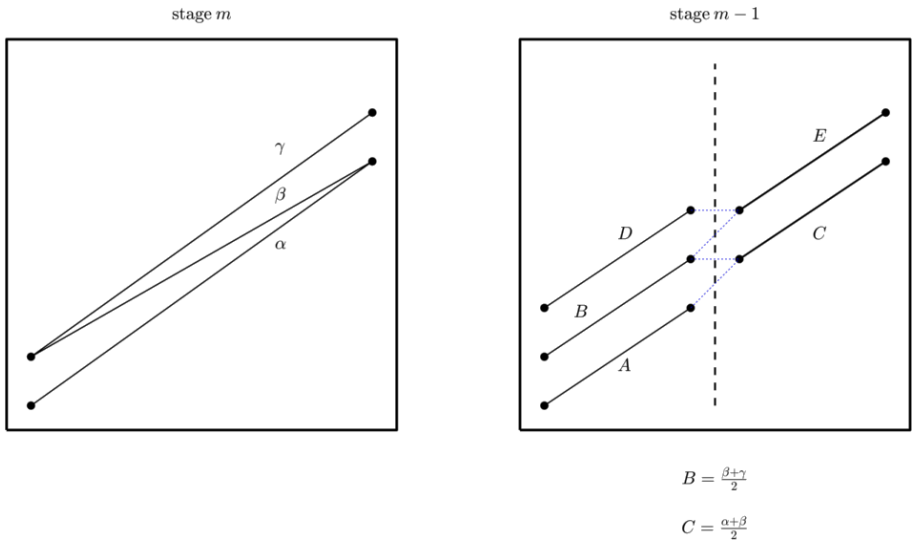

Figure 6: Scheme of computation of adjoint Radon transform.

```
            end
        end
    end
    Rfm = Rfmm1; % last stage output is next stage input
    Rfmm1 = zeros(2*N, N);
end

% The output f ends in the upper half of the data
f = Rfm(N +1:2*N,:);
end
```

2. FORMAL INVERSE IN TWO-DIMENSIONS

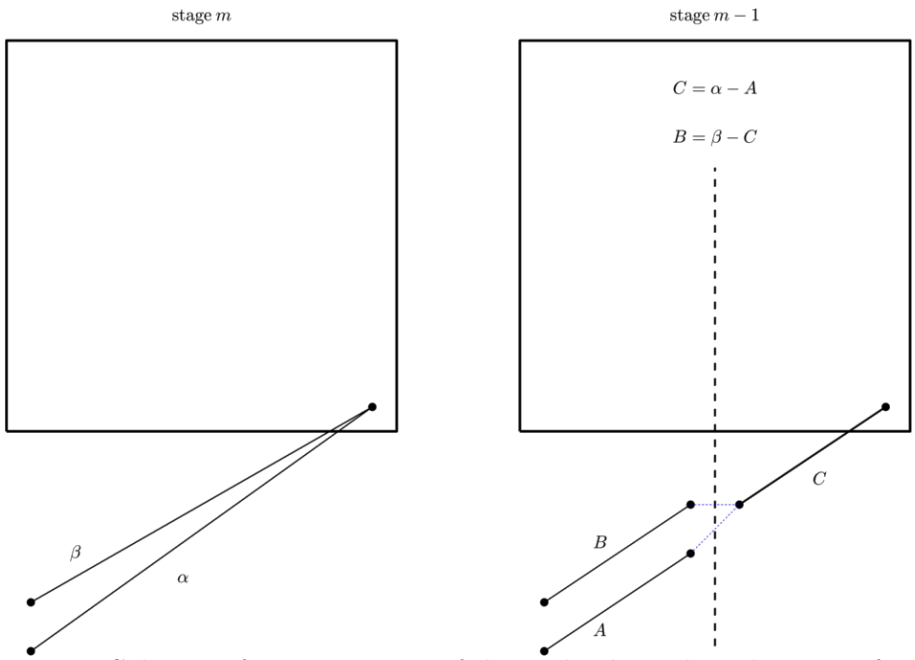

Figure 7: Scheme of computation of direct backward Radon transform.

The first novel contribution of this work is to express as an algorithm the direct reversal of forward DRT. The derivation of the same has been already published.[12] What we present here is an explicit implementation as code of the ideas set forth therein, as listing 3.

The key idea to arrive to that code is to reconsider the statement given to support the backprojection internal mechanism: *"Since it is not possible to know what B contributed and what C contributed in the forward algorithm, we can now only backproject them equally."* The figure 7 suggests us a way to remove that impediment.

Note that we have represented α and β segments in the m-th stage, displaced to $d = -s$, this is, they are the first segments on their stage, band and slope that touch the image from below. In that case, we can be sure that A, half the contribution to α in the forward path, comes from the zero-padded zone, and therefore $C = \alpha$. This creates a starting point –a thread to pull on to unravel the tangle– for calculating the values in stage $m - 1$ from not only the segments of stage m, but also from the newly calculated values of this stage: once C has been calculated we can take out B by subtracting C from β, etc

The concreteness of the algorithm, and its inner workings, which may be too obscured in Rim's elucidation of the method are clearly exposed with this figure and the accompanying code.

Listing 3: Matlab code to compute the direct reversal of DRT from a single quadrant

```
%% Radon direct inversion from a quadrant
function f = qDRTdirectReverse2D(Rf)
[Nd, N] = size(Rf);

fmp1 = Rf;
fm = zeros(2*N, N);

n = log2(N);

for m = n-1:-1:0 % stages, log2(N), now in reverse order
    for v = 0:(pow2(n - m - 1) -1) % for every vertical band
        for sigma = 0:(pow2(m) -1) % slopes within a band
            for d =  N-2*sigma-1:2*N-2   % for every displacement
                aux = fmp1(d +1, 1+(sigma*2)+(v*pow2(m+1)) +1)- fm(d +1, sigma+0+(v*pow2(m+1)) +1);
                if d+1+sigma <= 2*N-1
                    fm(d+1+sigma +1, sigma+(1*pow2(m))+(v*pow2(m+1)) +1) = aux;
                end
                fm(d+1 +1, sigma+(0*pow2(m))+(v*pow2(m+1)) +1) = ...
                    fmp1(d+1 +1, 0+(sigma*2)+(v*pow2(m+1)) +1) - aux;
            end
        end
    end
    fmp1 = fm; % last stage output is next stage input
    fm = zeros(2*N, N);
end
% The output f ends in the upper half of the data
f = fmp1(N+1:2*N,:);
end
```

3. FORMAL INVERSE IN THREE-DIMENSIONS

In a similar way than for two dimensions, the 3D Radon transform can be reversed with an algorithm of the same complexity than the forward path and attending to just a single portion of projection angles.

In the case of 3D Radon transform,[2] which computes the complete set of plane integrals of a volume, the solution is composed of 12 dodecants, instead of 4 quadrants. The 3D Radon transform have been found to be invertible with an adaptation of the multigrid method proposed by Press. This means that a backprojection algorithm had to be defined, and the necessary functions to feedback from the DRT domain to the initial domain, in a multigrid form.[2]

The second novel contribution on this work is that we now establish that this inverse 3D DRT can be computed much more efficiently with the same intuition that guided the 2D direct inversion.

In the unrotated dodecant the formulation of a plane has an expression of the form: $z - x \cdot slope_x - y \cdot slope_y - d = 0$, or in discrete, using the same definition of discrete loose line, that considers ascents instead of slopes: $z = l^n_{s_x}(\mathbf{x}) + l^n_{s_y}(\mathbf{y}) + d$. There are 2 ascents, but only one displacement, through z. Let's take $\sigma_x = \lfloor \frac{s_x}{2} \rfloor = 0$, and similarly $\sigma_y = 0$, that is, we nullify sigmas for a simpler explanation and depiction. Even then, as there are ascents in both directions, the total ascent can reach height 2, due just to the last bit of ascent per dimension.

Figure 8: Planes of two consecutive stages touching the volume from below.

The equivalent to line α –the first one to touch the initial data, at height $z = 0$, from below–, depicted in figure 7 is the plane that ascends from $d = -2$ in figure 8: $z = l^m_1(\mathbf{y}) + l^m_1(\mathbf{x}) - 2$. There are another two planes, –for the same σ_x, σ_y, band v_x, and v_y, and m–, that reach height 0 from $d = -1$. One plane being $z = l^m_0(\mathbf{y}) + l^m_1(\mathbf{x}) - 1$ and the other being $z = l^m_1(\mathbf{y}) + l^m_0(\mathbf{x}) - 1$. Finally there is a plane, of null ascents, that starts and ends at height 0: $z = l^m_0(\mathbf{y}) + l^m_0(\mathbf{y}) + 0$. Let's designate those planes as α, β, γ and δ.

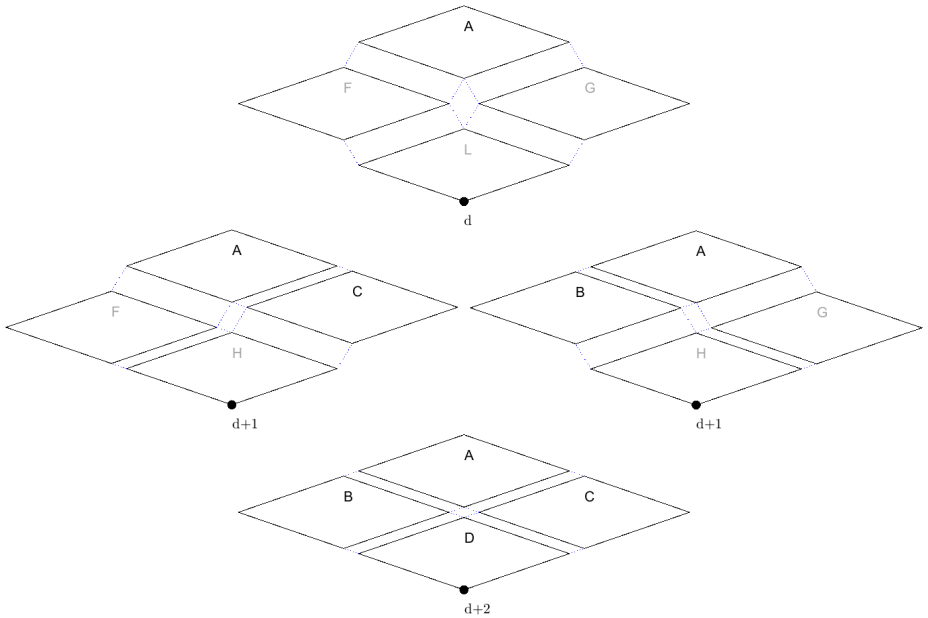

Figure 9: Planes from which the patches computation can be started at the m stage.

The right of that picture illustrates that those planes are constituted each internally by 4 plane patches of half the length per dimension. Many of those plane patches are shared among

α, β, γ and δ. If we had not cancelled the sigmas, then all the subplanes on the right would themselves be tilted. The next figure, 9, illustrates that again we have "a thread to pull on to unravel the tangle". α is constituted by the four subplanes: A, F, G and L. But all those contributions, except for A, when we reach the volume from below for the first time are zero. With $A = \alpha$, we can extract C and B, from planes β and γ, and so on. The actual code to perform direct inversion of 3D Radon transform, starting from a transformed dodecant is given in listing 4.

Listing 4: Matlab code to compute the direct reversal of 3D DRT from a single dodecant

```
%% Radon direct inversion from a dodecant
function f = qDRTdirectReverse3D(Rf)
[Nx, N, Nd] = size(Rf);

fmp1 = Rf;              % f at stage m plus 1
fm = zeros(N, N, 3*N); % f at stage m

n = log2(N);

for m = n-1:-1:0 % stages, log(N), in reverse order
    for v0 = 0:(pow2(n - m - 1) -1)  % not transformed yet
    for v1 = 0:(pow2(n - m - 1) -1)  % not transformed yet
        for sigma0 = 0:(pow2(m) -1) % already transformed
        for sigma1 = 0:(pow2(m) -1) % already transformed
            for d =  2*N-2*sigma0-1-2*sigma1-1:3*N-2 %displacements
                alpha = fmp1(1+(sigma0*2)+(v0*pow2(m+1)) +1, ...
                         1+(sigma1*2)+(v1*pow2(m+1)) +1, ...
                         d +1);
                if d+1<3*N
                  beta = fmp1(1+(sigma0*2)+(v0*pow2(m+1)) +1, ...
                           0+(sigma1*2)+(v1*pow2(m+1)) +1, ...
                           d+1 +1);
                  gamma = fmp1(0+(sigma0*2)+(v0*pow2(m+1)) +1, ...
                           1+(sigma1*2)+(v1*pow2(m+1)) +1, ...
                           d+1 +1);
                else beta=0; gamma=0; end;
                if d+2 < 3*N
                  delta = fmp1(0+(sigma0*2)+(v0*pow2(m+1)) +1, ...
                           0+(sigma1*2)+(v1*pow2(m+1)) +1, ...
                           d+2 +1);
                else delta = 0; end;


                if (d+1+sigma0 < 3*N)
                  G = fm(sigma0+(1*pow2(m))+(v0*pow2(m+1)) +1, ...
                         sigma1+(0*pow2(m))+(v1*pow2(m+1)) +1, ...
                         d+1+sigma0 +1);
                else G = 0; end;
                if (d+1+sigma1 < 3*N)
                  F = fm(sigma0+(0*pow2(m))+(v0*pow2(m+1)) +1, ...
                         sigma1+(1*pow2(m))+(v1*pow2(m+1)) +1, ...
                         d+1+sigma1 +1);
                else F = 0; end;
                L = fm(sigma0+(0*pow2(m))+(v0*pow2(m+1)) +1, ...
                       sigma1+(0*pow2(m))+(v1*pow2(m+1)) +1, ...
                       d +1);
                A = alpha-F-G-L;
                if (d+2+sigma0+sigma1 < 3*N)
                    fm(sigma0+(1*pow2(m))+(v0*pow2(m+1)) +1, ...
                       sigma1+(1*pow2(m))+(v1*pow2(m+1)) +1, ...
                       d+2+sigma0+sigma1 +1) = A;
                end;

                H = fm(sigma0+(0*pow2(m))+(v0*pow2(m+1)) +1, ...
                       sigma1+(0*pow2(m))+(v1*pow2(m+1)) +1, ...
                       d+1 +1);
                C = beta-A-F-H;
                if (d+2+sigma0 < 3*N)
                    fm(sigma0+(1*pow2(m))+(v0*pow2(m+1)) +1, ...
                       sigma1+(0*pow2(m))+(v1*pow2(m+1)) +1, ...
                       d+2+sigma0 +1) = C;
                end;

                B = gamma-A-G-H;
                if (d+2+sigma1 < 3*N)
                    fm(sigma0+(0*pow2(m))+(v0*pow2(m+1)) +1, ...
                       sigma1+(1*pow2(m))+(v1*pow2(m+1)) +1, ...
                       d+2+sigma1 +1) = B;
                end;

                D = delta-A-C-B;
                fm(sigma0+(0*pow2(m))+(v0*pow2(m+1)) +1, ...
```

```
                    sigma1+(0*pow2(m))+(v1*pow2(m+1)) +1, ...
                    d+2 +1) = D;
            end % for d
        end % for sigma1
        end % for sigma0
    end % for v1
    end % for v0
    fmp1 = fm;
    fm = zeros(N, N, 3*N);
end  % for m

% The output is in the upper side of the data
f = fmp1(:, :, 2*N+1:3*N);
end
```

4. CHARACTERIZATION OF DIRECT REVERSAL AGAINST NOISE

The algorithms proposed for direct 2D and 3D inversion of the DRT do recover perfectly those images, or 3D volumes, that gave rise to the transforms presented to them as input. But only if the inputs to revert came exactly from an unmodified output of the direct DRT algorithm.

In 2D, any image, of size $N \times N$, has a discrete Radon transform where each quadrant has size $N \times (N(N-1)/2) = 3N^2/2 - N/2$ and considering the 4 quadrants: $6N^2 - 2N$. The size and shape of a 2D DRT was illustrated by the intermediate image in figure 3. Since the domain and codomain of the direct DRT operator are not the same size, the mapping between elements among them cannot be bijective: the DRT is an injective non-surjective operator. There will be codomain elements to which do not map any element of the domain.

The elements of the codomain $\mathbb{R}^{4\times N\times N(N-1)/2}$ have to fulfill a number of constraints if they came from the application of direct DRT to an image. In particular, note that the minimum alteration of an image: modifying the value of a single pixel by a minimum amount, generates changes in its DRT in 4N positions: the coordinates of all those lines to which the point belongs. See figure 10 (a).

Also note that $\sum_{\forall d} \widetilde{f}\,(s,d) = \sum_{\forall x,y} f(x,y)$, i.e. the sum of columns of a given slope, s, of a quadrant, must be equal to each other and equal to the total cumulative value of the input image. In that sense, modifying an isolated value of $\widetilde{f}\,(s,d)$ will cause this rule not to be fulfilled.

(a) (b)

Figure 10: (a) DRT of a delta. (b) Inverse DRT of a delta.

Figure 10 illustrates the direct and inverse transform of a delta, for $N = 16$. The addition of an unity in a point in an, otherwise, null image causes an increment by an unity of N elements, per quadrant, in the DRT codomain: marking the displacements where this activated point is seen from each slope. In the reversal DRT path, the addition of an unity in a transformed quadrant, causes an absolute increment of over 300 unities, when instructed to determine which image generated that 'imposible' DRT. The already known inversion methods, which act by refinement of the backprojection, would have returned an almost null image, with a very faint line at the slope and offset coordinates of the 'incorrectly' activated delta.

One of the future lines of this contribution is to establish whether the inverse computation algorithm itself can be modified so that while it computes, it verifies the fulfillment of the constraints that the DRTs must meet, and if divergences are found, whether they can be corrected on the fly.

5. CONCLUSIONS

We have introduced, in some detail, the two-dimensional direct and adjoint Radon transforms by multiscale method: the 2D DRT. We have also analyzed the two already known methods of inversion in the 2D case. We also discussed the existing methods in 3D, for the direct and inverse path.

We have unified the notations with simple ideas that explain how an inversion method is possible with exactly the same computational burden as in the direct path. Which was not the case with the already known methods. We have given such inversion algorithms for 2D and 3D DRT.

They are faster than previously known backward algorithms, and can invert from only a quadrant or dodecant. But they incur in major error when applied to elements of the forward DRT codomain to which no element maps. It remains to be seen whether it is possible that such non-bijective codomain elements can be mapped backward to a nearest neighbor image, without a computational cost that impacts on the acceleration they exhibit compared to conventional methods.

Disclosures

The authors declare that there is no conflict of interest.

Acknowledgements

This work has been partially supported by the project ProID2020010066 of 'Estrategia de Especialización inteligente de Canarias RIS-3', cofunded by Government of the Canary Islands and European Regional Development Fund (ERDF). J.G.M.-H. has been partially supported by 'Convenio de investigación Wooptix-ULL: Asesoramiento técnico sobre *consumer electronic* con tecnología *lightfield*, 2022'. R.O.-G. attendance has been supported by University of La Laguna's research personnel training program, funded by 'Universidad de La Laguna' and 'Banco de Santander'.

REFERENCES

[1] J. Radon, "Über die Bestimmung von Funktionen durch ihre Integralwerte längs gewisser Mannigfaltigkeiten," *Akad. Wiss.* **69**, pp. 262–277, 1917.

[2] J. G. Marichal-Hernandez, Óscar Gómez-Cárdenes, F. L. R. González, D. H. Kim, and J. M. Rodríguez-Ramos, "Three-dimensional multiscale discrete Radon and John transforms," *Optical Engineering* **59**(9), pp. 1 – 23, 2020.

[3] J. G. Marichal-Hernández, J. P. Lüke, F. L. R. González, and J. M. Rodríguez-Ramos, "Fast approximate 4-d/3-d discrete radon transform for lightfield refocusing," *Journal of Electronic Imaging* **21**(2), p. 023026, 2012.

[4] A. Averbuch, R. R. Coifman, D. L. Donoho, M. Israeli, and Y. Shkolnisky, "A Framework for Discrete Integral Transformations I-The Pseudopolar Fourier Transform," *SIAM J. Scientific Computing* **30**(2), pp. 764–784, 2008.

[5] C. F. Van Loan, *Computational Frameworks for the Fast Fourier Transform*, SIAM, Society for Industrial and Applied Mathematics, 1992.

[6] W. Götz and H. Druckmüller, "A fast digital Radon transform—An efficient means for evaluating the Hough transform," *Pattern Recognition* **29**(4), pp. 711–718, 1996.

[7] M. L. Brady, "A fast discrete approximation algorithm for the Radon transform," *SIAM Journal on Computing* **27**(1), pp. 107–119, 1998.

[8] A. Brandt and J. Dym, "Fast calculation of multiple line integrals," *SIAM Journal on Scientific Computing* **20**(4), pp. 1417–1429, 1999.

[9] A. Kingston, I. Svalbe, and J.-P. Guédon, "The discrete Radon transform: a more efficient approach to image reconstruction?," in *Proc.SPIE*, **7078**, pp. 7078 – 7078 – 10, 2008.

[10] W. H. Press, "Discrete Radon transform has an exact, fast inverse and generalizes to operations other than sums along lines," *Proceedings of the National Academy of Sciences* **103**(51), pp. 19249–19254, 2006.

[11] J. G. Marichal-Hernández, R. Oliva-García, Ó. Gómez-Cárdenes, I. Rodríguez-Méndez, and J. M. Rodríguez-Ramos, "Inverse multiscale discrete radon transform by filtered backprojection," *Applied Sciences* **11**(1), 2021.

[12] D. Rim, "Exact and fast inversion of the approximate discrete radon transform," *Appl. Math. Lett.* **102**, p. 106159, 2020.

[13] W. Li, K. Ren, and D. Rim, "A range characterization of the single-quadrant ADRT," *ArXiv* **abs/2010.05360**, 2020.