Some applications of fractal methods in imaging

Franklin Mendivil

Department of Mathematics and Statistics, Acadia University

Multifractal Analysis: From Theory to Applications and Back, BIRS, Winter 2014
Outline

1. Basics of fractal block-coding
2. A Fractal-wavelet hybrid algorithm
3. Fractal denoising and zooming

We will only discuss grayscale images. Colour images are done in a similar way.
IFS on functions

The first ingredient for fractal image coding is a fractal operator on images.

The idea is we represent the given image by the fixed point $\bar{f}_\lambda$ of $T_\lambda$, where $\lambda \in \Lambda$ (a parameter space).

The hope is that it is simpler/cheaper to store $\lambda$ than it is to store the original image.

We represent images as functions, so need an operator on functions.
IFS on functions

Abstractly, our functions lie in $\mathcal{F} := \{ f : \square \rightarrow \mathbb{R} \}$. 
IFS on functions

Abstractly, our functions lie in $\mathcal{F} := \{ f : \Box \to \mathbb{R} \}$.

For $w_i : \Box \to \Box$ and $\phi_i : \mathbb{R} \to \mathbb{R}$ we define

$$T(f)(x) = \sum_{x \in w_i(\Box)} \phi_i(f(w_i^{-1}(x))).$$
IFS on functions

Abstractly, our functions lie in $\mathcal{F} := \{ f : \square \rightarrow \mathbb{R} \}$.

For $w_i : \square \rightarrow \square$ and $\phi_i : \mathbb{R} \rightarrow \mathbb{R}$ we define $T(f)(x) = \sum_{x \in w_i(\square)} \phi_i(f(w_i^{-1}(x)))$.

For image compression the common choice is $\phi_i(t) = \alpha_i t + \beta_i$.

With this choice, $\alpha$ is a “contrast adjustment” and $\beta$ is a “brightness” adjustment.
IFS on functions

Abstractly, our functions lie in $\mathcal{F} := \{ f : \square \to \mathbb{R} \}$.

For $w_i : \square \to \square$ and $\phi_i : \mathbb{R} \to \mathbb{R}$ we define

$$T(f)(x) = \sum_{x \in w_i(\square)} \phi_i(f(w_i^{-1}(x))).$$

For image compression the common choice is $\phi_i(t) = \alpha_i t + \beta_i$.

With this choice, $\alpha$ is a “contrast adjustment” and $\beta$ is a “brightness” adjustment.

This type of operator is not flexible enough, as small parts of a given image are rarely transformed copies of the entire image.
Local IFS on functions

The key idea is to change the “geometric” maps to “local” maps, $w_i : D_i \rightarrow \mathbb{R}^d$ where $D_i \subseteq \mathbb{R}^d$. Contractivity conditions can be quite complicated in this abstract formulation because of the possible overlaps in the domains and ranges of the $w_i$. 
Local IFS on functions

The key idea is to change the “geometric” maps to “local” maps, $w_i : D_i \rightarrow \Box$ where $D_i \subseteq \Box$.

For $T(f) : \Box \rightarrow \Box$ we need $\Box = \bigcup_i w_i(D_i)$. 
Local IFS on functions

The key idea is to change the “geometric” maps to “local” maps, \( w_i : D_i \to \square \) where \( D_i \subseteq \square \).

For \( T(f) : \square \to \square \) we need \( \square = \bigcup_i w_i(D_i) \).

Contractivity conditions can be quite complicated in this abstract formulation because of the possible overlaps in the domains and ranges of the \( w_i \).
Example

The $D_i$ are the larger squares (such as the shaded one).

Each $w_i$ reduces by a factor of 2, so maps a “large” block to a “small” block. We use 16 maps $w_i$, one for each “small” square.
“Collage” theorem

Given a “target” image \( g \) and a family of contractive fractal operators \( T_\lambda \), how do we find which \( \lambda \) minimizes \( \| \bar{f}_\lambda - g \| \)?

The problem is that we need to compute the fixed point \( \bar{f}_\lambda \) before we can measure this approximation error.
“Collage” theorem

Given a “target” image $g$ and a family of contractive fractal operators $T_\lambda$, how do we find which $\lambda$ minimizes $\|\bar{f}_\lambda - g\|$?

The problem is that we need to compute the fixed point $\bar{f}_\lambda$ before we can measure this approximation error.

The usual solution is to use a simple consequence of the triangle inequality, the Collage Theorem.

$$d(g, \bar{f}_\lambda) \leq \frac{d(g, T_\lambda(g))}{1 - c_\lambda},$$

where $c_\lambda$ is the contractivity of $T_\lambda$. 
“Collage” theorem

Given a “target” image \( g \) and a family of contractive fractal operators \( T_\lambda \), how do we find which \( \lambda \) minimizes \( \| \bar{f}_\lambda - g \| \)?

The problem is that we need to compute the fixed point \( \bar{f}_\lambda \) before we can measure this approximation error.

The usual solution is to use a simple consequence of the triangle inequality, the Collage Theorem.

\[
d(g, \bar{f}_\lambda) \leq \frac{d(g, T_\lambda(g))}{1 - c_\lambda}, \text{ where } c_\lambda \text{ is the contractivity of } T_\lambda.
\]

The result is very simple, but powerful, since it lets us trade the hard \( d(g, \bar{f}_\lambda) \) for the easier \( d(g, T_\lambda(g)) \).

“If \( g \) is close to \( T_\lambda(g) \), then it is also close to \( \bar{f}_\lambda \).”
Collage theorem illustrated
Collage theorem illustrated
The basic idea is to partition the image into “big” blocks and also into “small” blocks.

We match each “small” block with a “big” block which is similar (when suitably transformed by an $\alpha$ and $\beta$).
More formally:

for Each “small” block $R_i$ do
  for Each “large” block $D_j$ do
    Compute least-squares optimal $\alpha, \beta$ to minimize $\|R_i - \alpha \hat{D}_j - \beta\|$. ($\hat{D}_j$ is a downsampled version of $D_j$)
  end for
  Record the index of which $D_j$ gave the smallest error along with the corresponding $\alpha$ and $\beta$.
end for

The collection of triples ($index, \alpha, \beta$) defines a local IFS operator. Every pixel is in the range of some $w_i$. 
Reconstruction, iteration 1.
Reconstruction, iteration 2.
Reconstruction, iteration 3.
Reconstruction, iteration 4.
Reconstruction, iteration 5.
Reconstruction, iteration 6.
Reconstruction, iteration 7.
Fractal compression example

Reconstruction, iteration 8.
Reconstruction, iteration 9.
Reconstruction, iteration 10. We have reached the fixed point.

There are $32 \times 32$ “small” blocks in this decomposition.
Fractal compression example

Reconstruction, iteration 1.

You can see the $32 \times 32$ “small” blocks here.
Practical considerations

It is necessary to have a fine partition to get reasonable results. In the example, there are $32 \times 32$ small blocks, and so we do $32^2 \times 16^2 = 262,144$ block comparisons.

This searching makes the compression process relatively slow. There has been a large amount of work done to speed this up (using block classifiers, adaptive partitions, etc).
Practical considerations

It is necessary to have a fine partition to get reasonable results. In the example, there are $32 \times 32$ small blocks, and so we do $32^2 \times 16^2 = 262,144$ block comparisons.

This searching makes the compression process relatively slow. There has been a large amount of work done to speed this up (using block classifiers, adaptive partitions, etc).

Regenerating (uncompressiong) the image is very fast.
We are using the Collage theorem when we only compute $\|R_i - \alpha \hat{D}_i - \beta\|$. The solution is usually suboptimal.

Research has shown that it is pretty good – doing gradient descent from the “collage” solution does not result in substantial improvement.
We are using the Collage theorem when we only compute $\| R_i - \alpha \hat{D}_i - \beta \|$. The solution is usually suboptimal.

Research has shown that it is pretty good – doing gradient descent from the “collage” solution does not result in substantial improvement.

The non-overlapping nature of the “small” block partition also makes the error estimate simple.
The operator $T$ is affine.

We can combine the $\beta_i$'s into one piecewise constant function $\beta(x)$.

$$T(f) = A(f)(x) + \beta(x),$$

where $A$ combines the downsampling and multiplication by $\alpha_i$. 
Theoretical considerations

The operator $T$ is affine.

We can combine the $\beta_i$’s into one piecewise constant function $\beta(x)$.

$$T(f) = A(f)(x) + \beta(x),$$

where $A$ combines the downsampling and multiplication by $\alpha_i$.

$$\bar{f} = (I - A)^{-1}\beta = \beta + A\beta + A^2\beta + \cdots.$$  

We could see this in our example, where for each “small” block we have the constant, then the first image of the larger block, then the second image, etc.

This progressively extrapolates the finer-scale details.
In the discrete (image) setting, $A$ is represented by an $N \times N$ matrix, where $N$ is the number of pixels in the image ($512^2$ in our example).
In the discrete (image) setting, $A$ is represented by an $N \times N$ matrix, where $N$ is the number of pixels in the image ($512^2$ in our example).

This matrix is very sparse as each row has only 4 non-zero entries (corresponding to the 4 pixels in the “large” block which are downsampled).
Convergence

In the discrete (image) setting, $A$ is represented by an $N \times N$ matrix, where $N$ is the number of pixels in the image ($512^2$ in our example).

This matrix is very sparse as each row has only 4 non-zero entries (corresponding to the 4 pixels in the “large” block which are downsampled).

The iterations converge iff the spectral radius of $A$ is less than 1. Not a practical criteria!

In practice, we don’t usually check a convergence criteria. Sometimes we force $|\alpha| < 1$. 
The method suffers from blocking artifacts. This can be alleviated somewhat by using postprocessing or by using overlapping “small” blocks.

Another way of dealing with blocking is to use a fractal-wavelet hybrid method.
Fractal-wavelet compression

Because a wavelet basis is inherently multi-scale it is natural to combine fractal compression with wavelets.

The important property is that $\psi(2x + i)$ is an element of the basis if $\psi(x)$ is a basis element.
Fractal-wavelet compression

Because a wavelet basis is inherently multi-scale it is natural to combine fractal compression with wavelets.

The important property is that $\psi(2x + i)$ is an element of the basis if $\psi(x)$ is a basis element.

The fractal-wavelet algorithm works by defining an IFS operator on the collection of wavelet coefficients organized as a tree.
Wavelet coefficient tree

We illustrate with the 1D version for simplicity.

In addition, we have one scaling function coefficient (the overall mean).
Wavelet coefficient tree

It is also convenient to illustrate this in block form.

Let \( B_{i,j} \) represent the part of the tree below \( \psi(2^i x - j) \), so \( B_{0,0} \) is the entire tree.
Wavelet coefficient tree

It is also convenient to illustrate this in block form.

Let $B_{i,j}$ represent the part of the tree below $\psi(2^i x - j)$, so $B_{0,0}$ is the entire tree.

If $w(t) = (t - k)/2^n$ then $B_{i,j} \circ w^{-1} = 2^{n/2}B_{i+n, 2^i k + j}$ since $2^i(2^n x - k) - j = 2^{i+n}x - 2^i k - j$.

That is, $w^{-1}$ maps blocks of the tree to other blocks which are lower down on the tree.

This is the basis for the “geometric contractive” part of the IFS on wavelets and represents a spatial contraction.
Mapping on wavelet tree

Here $B_{1,0}$ gets mapped to $B_{2,3}$.
In addition to mapping \( B_{i,j} \) to \( B_{i+n, 2^i k + j} \), we allow a multiplication of all the coefficients in \( B_{i,j} \) by a single number \( \alpha \).

We do not add a constant \( \beta \) as we previously did.
Fractal wavelet transform

In addition to mapping $B_{i,j}$ to $B_{i+n,2^i k + j}$, we allow a multiplication of all the coefficients in $B_{i,j}$ by a single number $\alpha$.

We do **not** add a constant $\beta$ as we previously did.

We choose two “levels”, the “large” level and the “small” level (corresponding to the two different block partitions).

For each subtree on the small level, we try to find a matching subtree on the large level and the optimal $\alpha$ for this match.
Fractal wavelet transform

We store the indices, the $\alpha_i$s, and all the coefficients in the tree up to and including the “large” level.

The values in the lower levels (starting at the “small” level) are cascaded from the upper levels.

This is the extrapolation (or the filling in of fine details) by the algorithm.
Mapping on wavelet tree

Just for illustration we choose a mapping with

\[ \begin{align*}
B_{1,0} \times a_0 & \rightarrow B_{2,1} \\
B_{1,0} \times a_1 & \rightarrow B_{2,3} \\
B_{1,1} \times a_2 & \rightarrow B_{2,0} \\
B_{1,1} \times a_3 & \rightarrow B_{2,2}
\end{align*} \]
Mapping on wavelet tree

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_{0,0}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$c_{0,0}$</td>
</tr>
<tr>
<td>$c_{1,0}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$c_{1,1}$</td>
</tr>
</tbody>
</table>
### Mapping on wavelet tree

<table>
<thead>
<tr>
<th>b_{0,0}</th>
<th>c_{0,0}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>c_{1,0}</th>
<th>c_{1,1}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>\alpha_2 c_{1,1}</th>
<th>\alpha_0 c_{1,0}</th>
<th>\alpha_3 c_{1,1}</th>
<th>\alpha_1 c_{1,0}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

THANKS
Mapping on wavelet tree

<table>
<thead>
<tr>
<th></th>
<th>$b_{0,0}$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$c_{0,0}$</td>
</tr>
<tr>
<td>$c_{1,0}$</td>
<td></td>
<td>$c_{1,1}$</td>
</tr>
<tr>
<td>$\alpha_2 c_{1,1}$</td>
<td>$\alpha_0 c_{1,0}$</td>
<td>$\alpha_3 c_{1,1}$</td>
</tr>
<tr>
<td>$\alpha_2 \alpha_3 c_{1,1}$</td>
<td>$\alpha_2 \alpha_1 c_{1,0}$</td>
<td>$\alpha_0 \alpha_2 c_{1,1}$</td>
</tr>
</tbody>
</table>
Example of fractal-wavelet compression
Example of fractal compression
Comparison: fractal vs. fractal wavelet
Relation between fractal coding and fractal wavelet coding

For a separable Haar basis, fractal-wavelet compression is exactly the same as fractal compression.

For other wavelets the relationship is more complicated.
Relation between fractal coding and fractal wavelet coding

For a separable Haar basis, fractal-wavelet compression is exactly the same as fractal compression.

For other wavelets the relationship is more complicated.

The coefficients which are kept at the top of the tree correspond to the piecewise function $\beta(x)$; instead of piecewise constant it is a finite linear combination of wavelets.
Relation between fractal coding and fractal wavelet coding

For a separable Haar basis, fractal-wavelet compression is exactly the same as fractal compression.

For other wavelets the relationship is more complicated.

The coefficients which are kept at the top of the tree correspond to the piecewise function $\beta(x)$; instead of piecewise constant it is a finite linear combination of wavelets.

The disjoint “small” blocks $R_i$ correspond to orthogonal subspaces spanned by disjoint subtrees.
For both the fractal and fractal-wavelet methods, we are fitting a type of multiplicative cascade to our data.
For both the fractal and fractal-wavelet methods, we are fitting a type of multiplicative cascade to our data.

Interestingly, it is “local” in that parts influence other parts. The fixed point equation can be organized into a simultaneous linear system of fixed point equations.
For both the fractal and fractal-wavelet methods, we are fitting a type of multiplicative cascade to our data.

Interestingly, it is “local” in that parts influence other parts. The fixed point equation can be organized into a simultaneous linear system of fixed point equations.

Of course, the model for the fractal (spatial) representation also includes a translation (the $\beta_i$s).
Non-separable wavelet basis: twin-dragon Haar
Non-separable wavelet basis: smooth wavelet
What else can we do with fractal methods?

Fractal methods can be used to represent/compress an image. Great. What’s next? Can anything else be done with these methods?

There are many image processing tasks other than compression. Can we use the fractal code/process for anything else?

We discuss only two possible applications, image denoising and zooming, but we discuss them together.
Fractal denoising

The downsampling which occurs as part of the fractal compression is a smoothing operation. Thus fractal compression naturally smooths additive noise.

In addition, usually $|\alpha| < 1$ and so this also reduces the magnitude of the noise.

Finally, in fractal compression $\beta$ is the mean of the “small” block, so this quantity is somewhat insensitive to additive noise.
Fractal imaging

Fractal zooming

In principle, a fractal representation of an image is size invariant.

Once you have the “fractal code”, you just render the image in whatever size you wish.

Of course reality is more complicated than this. Any version of the image larger than the original involves data extrapolation.

So the question is, is fractal extrapolation “more natural” than other methods?
Our discussion is taken from

*Solving the inverse problem of image zooming using ‘self-examples’*

and

*Multi-frame super-resolution with no explicit motion estimation*

by M. Ebrahimi and E.R. Vrscay.

and *Joint Fractal-Wavelet Image Denoising and Interpolation*

Denoising and zooming using self-examples

We start with a noisy image $\hat{u}$ of size $m \times n$ and wish to estimate a denoised $n^z \times m^z$ (where $z > 1$) version $U$. 
Denoising and zooming using self-examples

We start with a noisy image \( \hat{u} \) of size \( m \times n \) and wish to estimate a denoised \( n_z \times m_z \) (where \( z > 1 \)) version \( U \).

The idea is to take all possible “large” blocks in \( \hat{u} \) and compute a weighted average where the weight is a function of similarity.

This is similar to fractal compression where we take multiple “large” blocks and use a weighted average.

We don’t care about efficiency of representation, only accuracy.
Some details

We will up-sample each pixel in $\hat{u}$ to a $z \times z$ block in $U$.

Let $U(x)$ be a $z \times z$ block in $U$ and $y$ a $d z \times d z$ block in $\hat{u}$. 

The weight is given by $w(x, y) = \exp(-\|u(N(D(x))) - u(\hat{y})\|_2^2 h^2)$, where $\hat{y}$ is a downsampled $(d z \times d z)$ $y$ and $N(D(x))$ is the $d z \times d z$ centered neighborhood of the pixel $D(x)$ in $\hat{u}$. The form of the weight is taken from the non-local means denoising literature.
Some details

We will up-sample each pixel in $\hat{u}$ to a $z \times z$ block in $U$.

Let $U(x)$ be a $z \times z$ block in $U$ and $y$ a $d \times d$ block in $\hat{u}$.

$$U(x) \approx \frac{1}{\sum_y w(x, y)} \sum_y w(x, y)y \text{ (sum over all } y).$$
Some details

We will up-sample each pixel in $\hat{u}$ to a $z \times z$ block in $U$.

Let $U(x)$ be a $z \times z$ block in $U$ and $y$ a $d \times d$ block in $\hat{u}$.

$$U(x) \approx \frac{1}{\sum_y w(x,y)} \sum_y w(x,y)y \text{ (sum over all } y).$$

The weight is given by

$$w(x, y) = \exp \left( - \frac{\|u(N(D(x)))-u(\hat{y})\|^2}{h^2} \right),$$

where $\hat{y}$ is a downsampled (to $d \times d$) $y$ and $N(D(x))$ is the $d \times d$ centered neighborhood of the pixel $D(x)$ in $\hat{u}$.

The form of the weight is taken from the non-local means denoising literature.
Example results

**Fig. 2.** Original, Pixel replication, Bilinear, Self-examples. In the first row $h = 0.05$ is applied while in the second row the standard deviation of noise $\sigma = 0.1$ and $h = 0.15$.

Here $z = 2$ and $d = 3$. 
We start with several noisy images \( \hat{u}_i \) (frames from a video) of size \( m \times n \) and wish to estimate denoised \( U_i \) of size \( n_z \times m_z \).
Multi-frame super-resolution with no explicit motion estimation

We start with several noisy images $\hat{u}_i$ (frames from a video) of size $m \times n$ and wish to estimate denoised $U_i$ of size $n_z \times m_z$.

We assume that the noise is independent and that the frames are close to each other.

We use several frames to estimate the noise and also to estimate an interpolation.
Multi-frame super-resolution with no explicit motion estimation

We start with several noisy images $\hat{u}_i$ (frames from a video) of size $m \times n$ and wish to estimate denoised $U_i$ of size $nz \times mz$.

We assume that the noise is independent and that the frames are close to each other.

We use several frames to estimate the noise and also to estimate an interpolation.

The method is very similar to the previous one and is inspired by the same work.
Some details

To estimate $U_i$, we use $\frac{1}{G_i} \sum_j g(|i - j|)E[U_i|\hat{u}_j]$.

$E(U_i|\hat{u}_j)$ is the expectation of $U_i$ given $\hat{u}_j$ and is modeled in the same way as a single image was in the previous method.

$g$ is a function which decays so that the influence of “later” frames decays with time.

In the numerical example they used a characteristic function for $g$. 
Example

Pixel duplication  Bilinear interpolation  New method

The original frames were $32 \times 32$ pixels, expanded to $96 \times 96$. 
Joint fractal-wavelet image denoising and interpolation

We start with a noisy image $\hat{u}$ of size $m \times n$ and wish to estimate a denoised $n_z \times m_z$ (where $z = 2^\ell$) version $U$.

We assume that the noise is additive and iid normal at each pixel, so $\hat{u} = u + n$.

We want to estimate the “fractal wavelet” parameters of the denoised version from the noisy observations.
Some details

For given “parent”, $\vec{x}$, and “child”, $\vec{y}$, wavelet subtrees, the optimal scaling parameter $\alpha$ is

$$\alpha = \frac{\sum_i x_i y_i}{\sum_i x_i^2} = \frac{E(XY)}{E(X^2)}.$$ 

We only have direct access to the noisy versions of $x$ and $y$. 

From our independence assumption it can be shown that $\alpha = \hat{\alpha}(1 + 1/\gamma)$ where $\gamma = E(X^2)/\sigma_n^2 \approx E(\hat{X}^2)/\sigma_n^2 - 1$. 

To estimate $\sigma_n^2$, we assume that the image is mostly smooth (with some discontinuities near texture and edges). Take all blocks of some appropriate size, compute variance, and then find the median of this collection. This is a reasonable estimate for $\sigma_n^2$. 

Some details

For given “parent”, $\vec{x}$, and “child”, $\vec{y}$, wavelet subtrees, the optimal scaling parameter $\alpha$ is $\alpha = \frac{\sum_i x_i y_i}{\sum x_i^2} = \frac{E(XY)}{E(X^2)}$.

We only have direct access to the noisy versions of $x$ and $y$.

From our independence assumption it can be shown that $\alpha = \hat{\alpha}(1 + 1/\gamma)$ where $\gamma = \frac{E(X^2)}{\sigma_n^2} \approx \frac{E(\hat{X}^2)}{\sigma_n^2} - 1$.

To estimate $\sigma_n^2$, we assume that the image is mostly smooth (with some discontinuities near texture and edges).

Take all blocks of some appropriate size, compute variance, and then find the median of this collection. This is a reasonable estimate for $\sigma_n^2$. 
The next problem is that the parent-child matchings are probably not the same in \( \hat{u} \) and \( u \), so we need to estimate the approximation error in \( u \) via a measurement in \( \hat{u} \).
Some details

The next problem is that the parent-child matchings are probably not the same in $\hat{u}$ and $u$, so we need to estimate the approximation error in $u$ via a measurement in $\hat{u}$.

This can also be done. The analysis is elementary but involved and consists of several cases (for instance depending on if $E(\hat{X}^2) < \sigma_n^2$ or not).
Some details

The next problem is that the parent-child matchings are probably not the same in $\hat{u}$ and $u$, so we need to estimate the approximation error in $u$ via a measurement in $\hat{u}$.

This can also be done. The analysis is elementary but involved and consists of several cases (for instance depending on if $E(\hat{X}^2) < \sigma_n^2$ or not).

In this way we estimate the fractal-wavelet code for $u$ from our observation of $\hat{u}$. 
Some details

The next problem is that the parent-child matchings are probably not the same in $\hat{u}$ and $u$, so we need to estimate the approximation error in $u$ via a measurement in $\hat{u}$.

This can also be done. The analysis is elementary but involved and consists of several cases (for instance depending on if $E(\hat{X}^2) < \sigma_n^2$ or not).

In this way we estimate the fractal-wavelet code for $u$ from our observation of $\hat{u}$.

To expand $u$ to $U$, when we regenerate the image we simply add levels to the wavelet tree.
Example

(a) Original image: 512 x 512 pixels

(c) FW interpolation of the original image
1024 x 1024 pixels

(d) FW interpolation of the noisy image
1024 x 1024 pixels

(b) Noisy test image: \( \sigma_n = 25 \)

(e) Simultaneous FW denoising and interpolation without the cycle spinning algorithm
1024 x 1024 pixels

(f) Simultaneous FW denoising and interpolation with the cycle spinning algorithm: \( N=50 \) shifts,
1024 x 1024 pixels

THANKS
Thanks

I want to thank the organizers and participants for this great workshop.

I also want to thank Ed Vrscay for letting me use the images from his papers for my talk.
Fractal Imaging
Franklin Mendivil
Basics of fractal-block coding
Fractal-wavelet hybrid
Fractal denoising and zooming
THANKS
Questions?