CS448f: Image Processing For Photography and Vision

Wavelets and Compression

ImageStack Gotchas

- Image and Windows are pointer classes
- What’s wrong with this code?

```
Image sharp = Load::apply(“foo.jpg”);
Image blurry = foo;
FastBlur::apply(blurry, 0, 5, 5);
Subtract::apply(sharp, blurry);
```

ImageStack Gotchas

- Image and Windows are pointer classes
- What’s wrong with this code?

```
Image sharp = Load::apply(“foo.jpg”);
Image blurry = foo.copy();
FastBlur::apply(blurry, 0, 5, 5);
Subtract::apply(sharp, blurry);
```

ImageStack Gotchas

- Images own memory (via reference counting), Windows do not.
- What’s wrong with this code?

```
class Foo {
  public:
    Foo(Window im) {
      Image temp(im);
      ... do some processing on temp ...
      patch = temp;
    };
    Window patch;
};
```

ImageStack Gotchas

- Images own memory (via reference counting), Windows do not.
- What's wrong with this code?

```
class Foo {
  public:
    Foo(Window im) {
      Image temp(im);
      ... do some processing on temp ...
      patch = temp;
    };
    Image patch;
};
```

Using Windows Wisely

```
float sig = 2;
Image pyramid = Upsample::apply(gray, 10, 1, 1);

// pyramid now contains 10 copies of the input
for(int i = 1; i < 10; i++) {
  Window level(pyramid, i, 0, 0, 1, pyramid.width, pyramid.height);
  FastBlur::apply(level, 0, sig, sig);
  sig *= 1.6;
}
// 'pyramid' now contains a Gaussian pyramid

for(int i = 0; i < 9; i++) {
 Window thisLevel(pyramid, i, 0, 0, 1, pyramid.width, pyramid.height);
 Window nextLevel(pyramid, i+1, 0, 0, 1, pyramid.width, pyramid.height);
 Subtract::apply(thisLevel, nextLevel);
}
// ‘pyramid’ now contains a Laplacian pyramid
// (except for the downsampling)
```

The only time memory gets allocated

```
float sig = 2;
Image pyramid = Upsample::apply(gray, 10, 1, 1);

// pyramid now contains 10 copies of the input
for(int i = 1; i < 10; i++) {
  Window level(pyramid, i, 0, 0, 1, pyramid.width, pyramid.height);
  FastBlur::apply(level, 0, sig, sig);
  sig *= 1.6;
}
// 'pyramid' now contains a Gaussian pyramid

for(int i = 0; i < 9; i++) {
 Window thisLevel(pyramid, i, 0, 0, 1, pyramid.width, pyramid.height);
 Window nextLevel(pyramid, i+1, 0, 0, 1, pyramid.width, pyramid.height);
 Subtract::apply(thisLevel, nextLevel);
}
// ‘pyramid’ now contains a Laplacian pyramid
// (except for the downsampling)
```

Select each layer and blur it

```
float sig = 2;
Image pyramid = Upsample::apply(gray, 10, 1, 1);

// pyramid now contains 10 copies of the input
for(int i = 1; i < 10; i++) {
  Window level(pyramid, i, 0, 0, 1, pyramid.width, pyramid.height);
  FastBlur::apply(level, 0, sig, sig);
  sig *= 1.6;
}
// 'pyramid' now contains a Gaussian pyramid

for(int i = 0; i < 9; i++) {
 Window thisLevel(pyramid, i, 0, 0, 1, pyramid.width, pyramid.height);
 Window nextLevel(pyramid, i+1, 0, 0, 1, pyramid.width, pyramid.height);
 Subtract::apply(thisLevel, nextLevel);
}
// ‘pyramid’ now contains a Laplacian pyramid
// (except for the downsampling)
```

Take the difference between each layer and the next

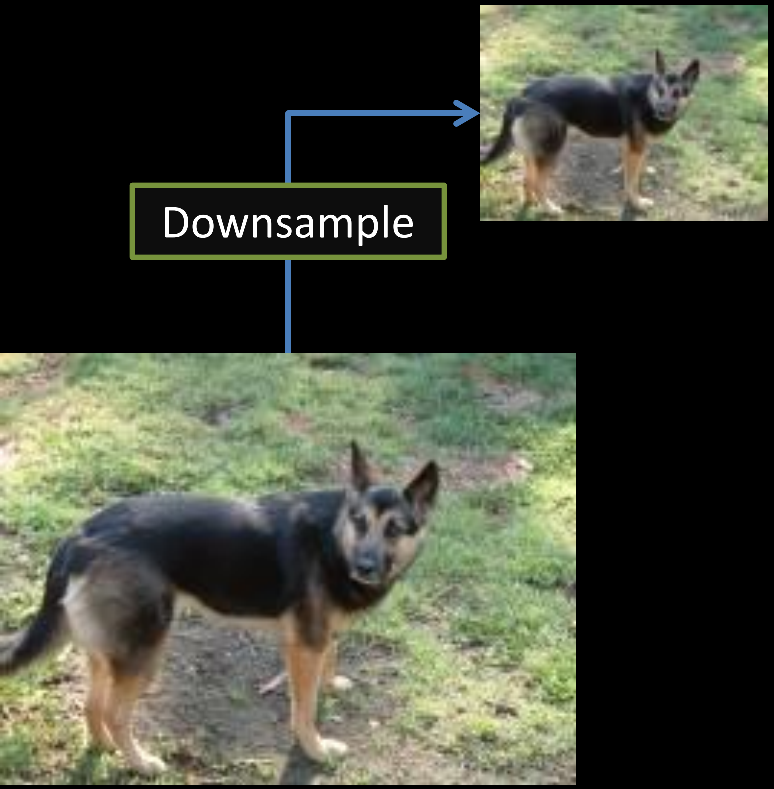
```
float sig = 2;
Image pyramid = Upsample::apply(gray, 10, 1, 1);

// pyramid now contains 10 copies of the input
for(int i = 1; i < 10; i++) {
  Window level(pyramid, i, 0, 0, 1, pyramid.width, pyramid.height);
  FastBlur::apply(level, 0, sig, sig);
  sig *= 1.6;
}
// 'pyramid' now contains a Gaussian pyramid

for(int i = 0; i < 9; i++) {
 Window thisLevel(pyramid, i, 0, 0, 1, pyramid.width, pyramid.height);
 Window nextLevel(pyramid, i+1, 0, 0, 1, pyramid.width, pyramid.height);
 Subtract::apply(thisLevel, nextLevel);
}
// ‘pyramid’ now contains a Laplacian pyramid
// (except for the downsampling)
```

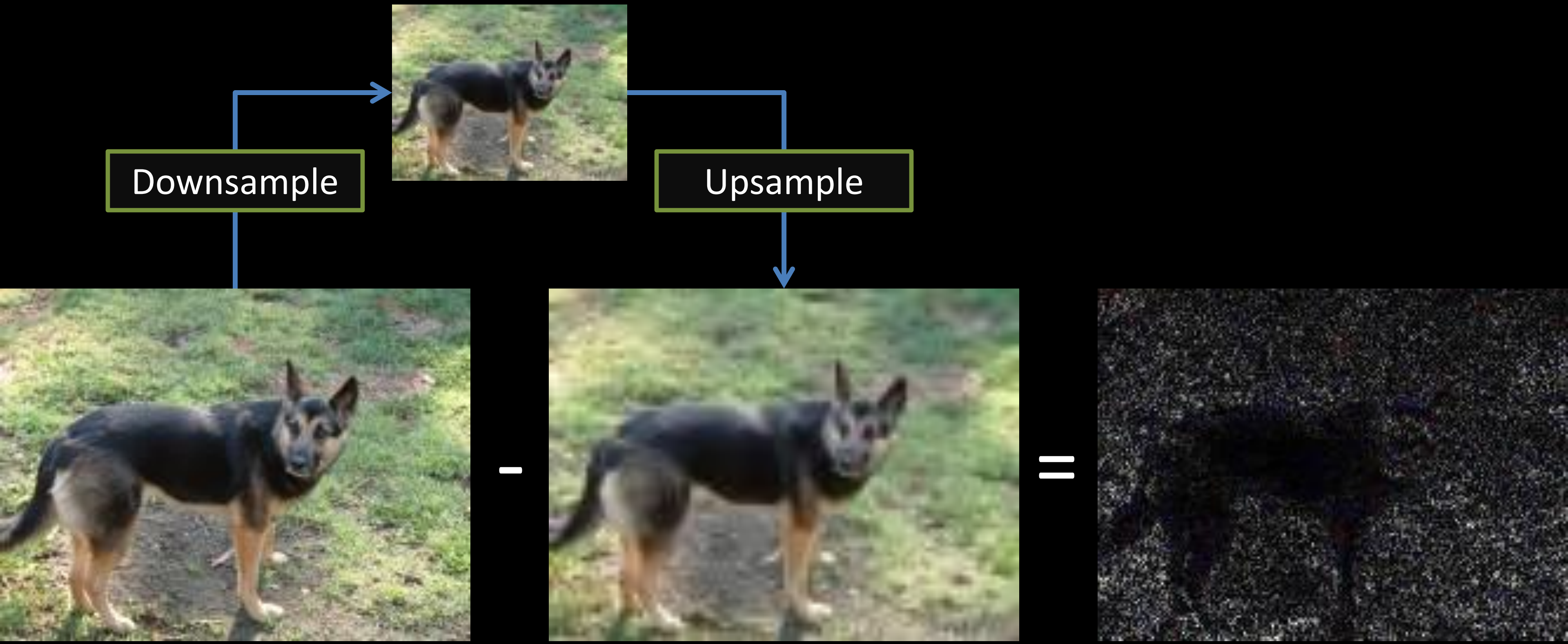
Review: Laplacian Pyramids

- Make the coarse layer by downsampling



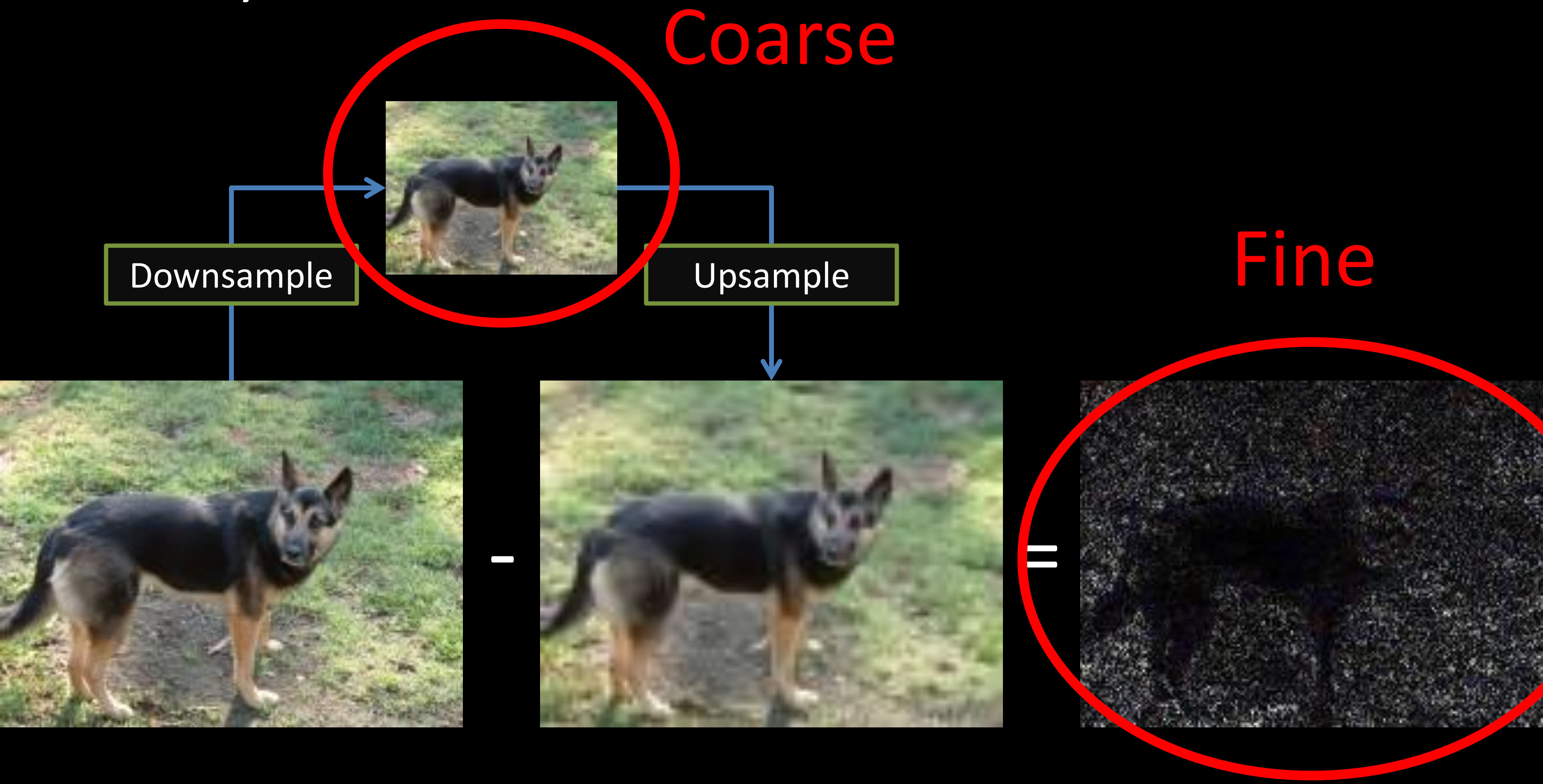
Review: Laplacian Pyramids

- Make the fine layer by upsampling the coarse layer, and taking the difference with the original



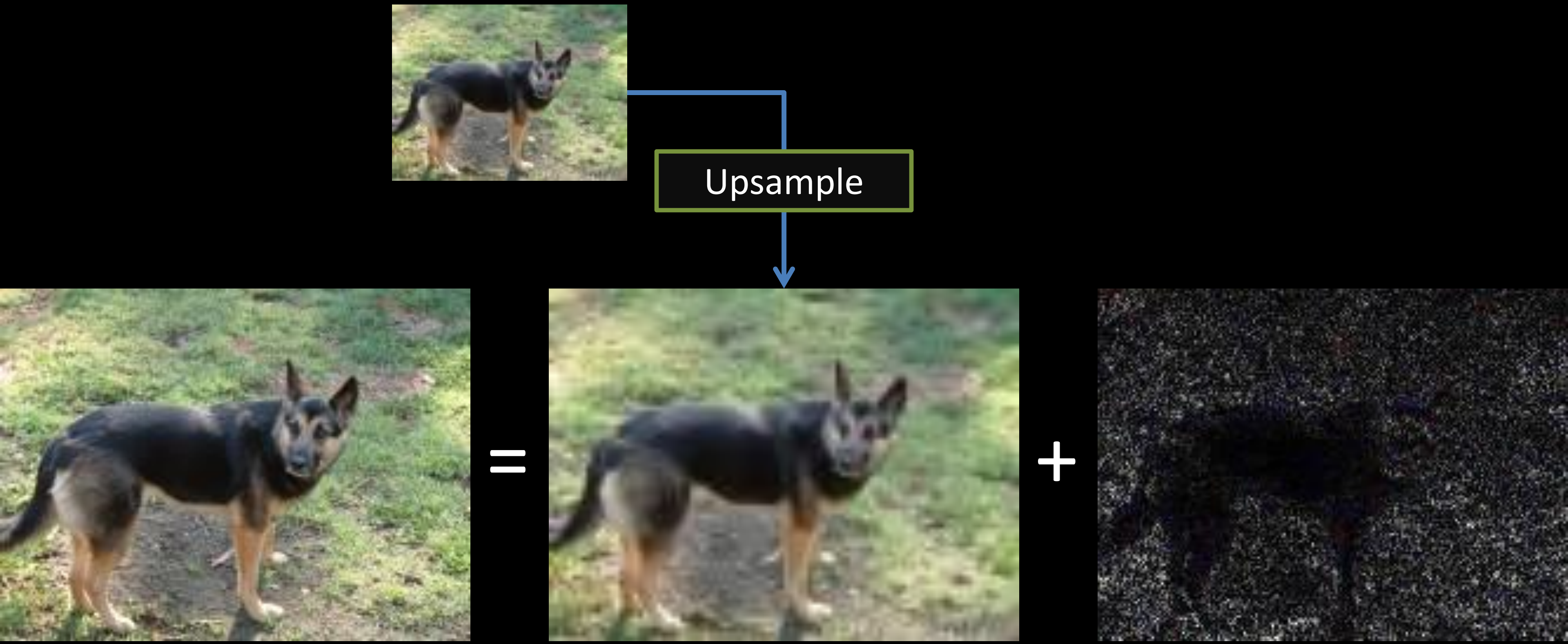
Review: Laplacian Pyramids

- Only store these



Review: Laplacian Pyramids

- Reconstruct like so:



Laplacian Pyramids and Redundancy

- The coarse layer has redundancy - it's blurry. We can store it at low resolution
- In linear algebra terms:
  - coarse = Upsample(small)
  - c = Us
  - c is a linear combination of the columns of U
  - How many linearly independent dimensions does c have?

Laplacian Pyramids and Redundancy

- The fine layer should be redundant too
- What constraint does the fine layer obey?
- How much of the fine layer should we actually need to store?

Laplacian Pyramids and Redundancy

- The fine layer should be redundant too
- What constraint does the fine layer obey?
- How much of the fine layer should we actually need to store?
  - Intuitively, should be ¾n for n pixels

<MATH>

Laplacian Pyramids and Redundancy

- What constraint does the fine layer obey?

  f = m - c

  f = m - UDm

  Kf = Km - KUDm

  if KUD = K

  then Kf = 0

  K(UD-I) = 0

  K is the null-space (on the left) of UD-I

  May be empty (no constraints)

  May have lots of constraints. Hard to tell.

m = input image
c = coarse
f = fine
U = upsampling
D = downsampling
K = some matrix

Laplacian Pyramids and Redundancy

- What if we say DU = I
  - i.e. upsampling then downsampling does nothing
- Then $(UD)^2 = (UD)(UD) = U(DU)D$
- f = m - UDm
- UDf = UDm - UDUDm = UDm - UDm = 0
- f is in the null-space of UD
- Downsampling then upsampling the fine layer gives you a black image.

DU = I

- How about nearest neighbor upsampling followed by rect downsampling?

- How about lanczos3 upsampling followed by lanczos3 downsampling?

DU = I

- How about nearest neighbor upsampling followed by nearest neighbor downsampling?
  - Yes, but this is a crappy downsampling filter ☹
- How about lanczos3 upsampling followed by lanczos3 downsampling?
  - No ☹
- This is hard, if we continue down this rabbit hole we arrive at...

Wavelets

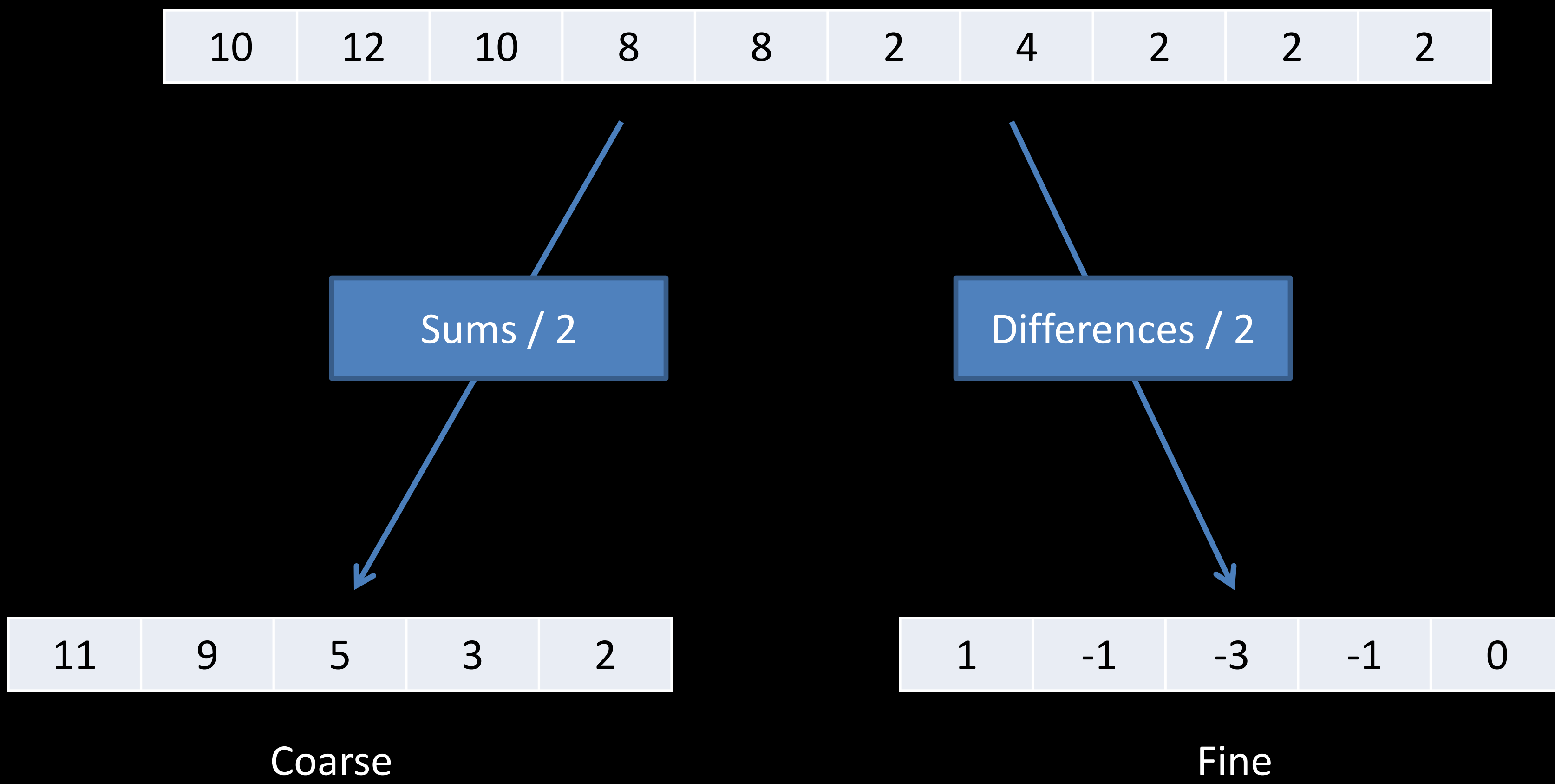
- Yet another tool for:
  - Image = coarse + fine
- So why should we care?
  - They don't increase the amount of data like pyramids (memory efficient)
  - They're simple to compute (time efficient)
  - Like the Fourier transform, they're orthogonal
  - They have no redundancy

The Haar Wavelet

- Equivalent to nearest neighbor downsampling / upsampling.
- Take each pair of values and replace it with:
  - The sum / 2
  - The difference / 2

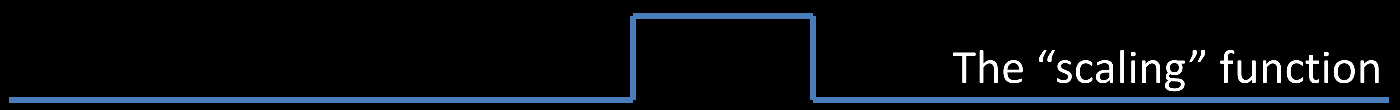
- The sums form the coarse layer
- The differences form the fine layer

The 1D Haar Transform

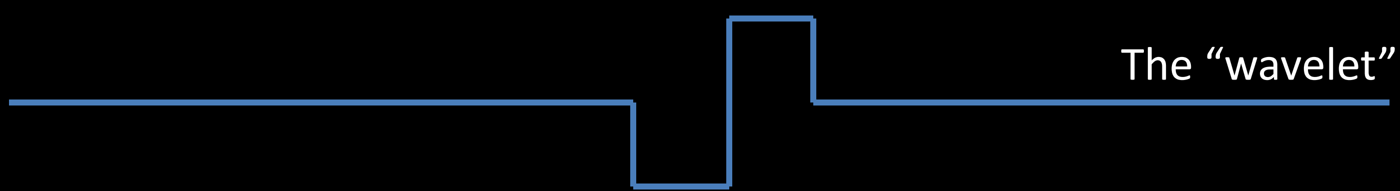


Equivalently...

- The coarse layer is produced by convolving with [½ ½] (then subsampling)

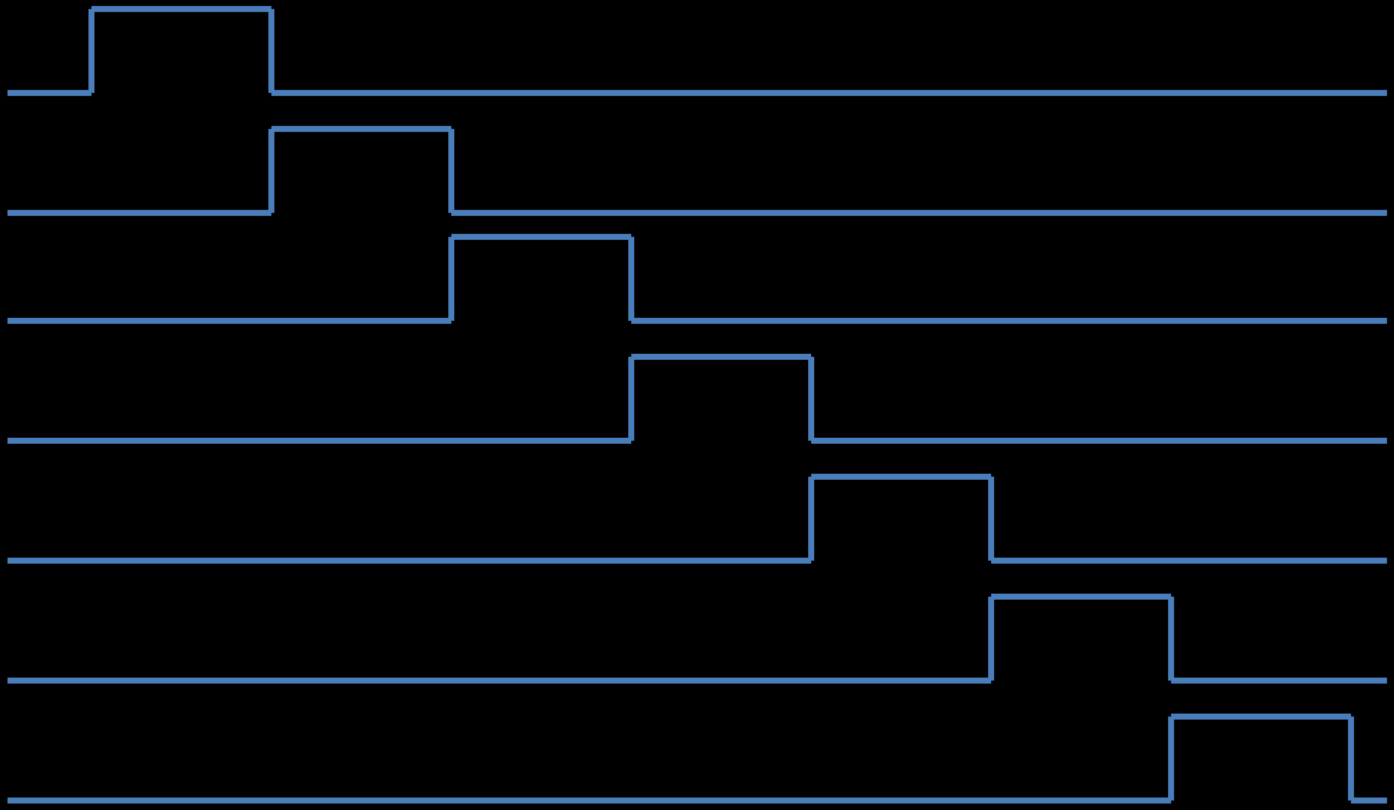


- The fine layer is produced by convolving with [-½ ½] (then subsampling)



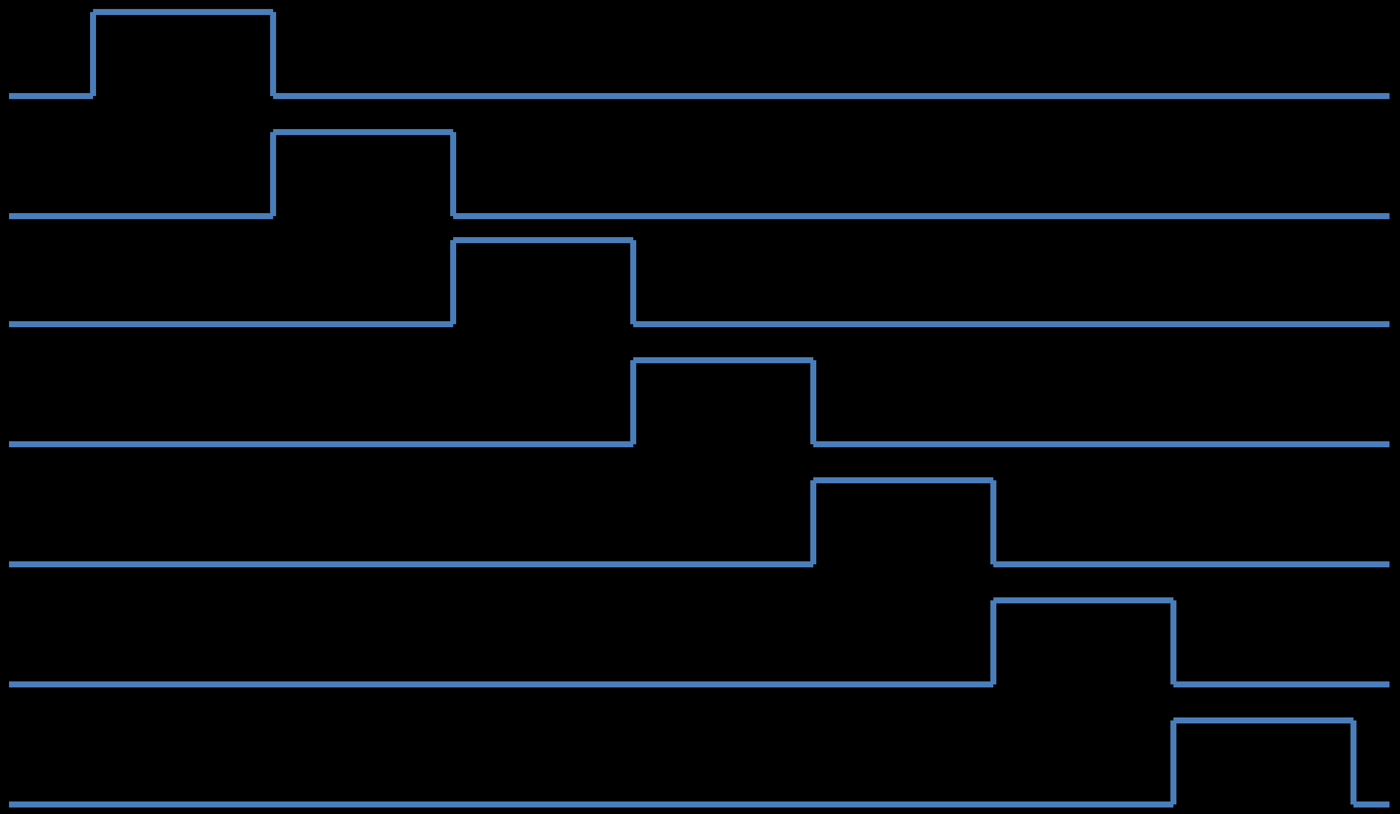
DU = I

- In this case, D =



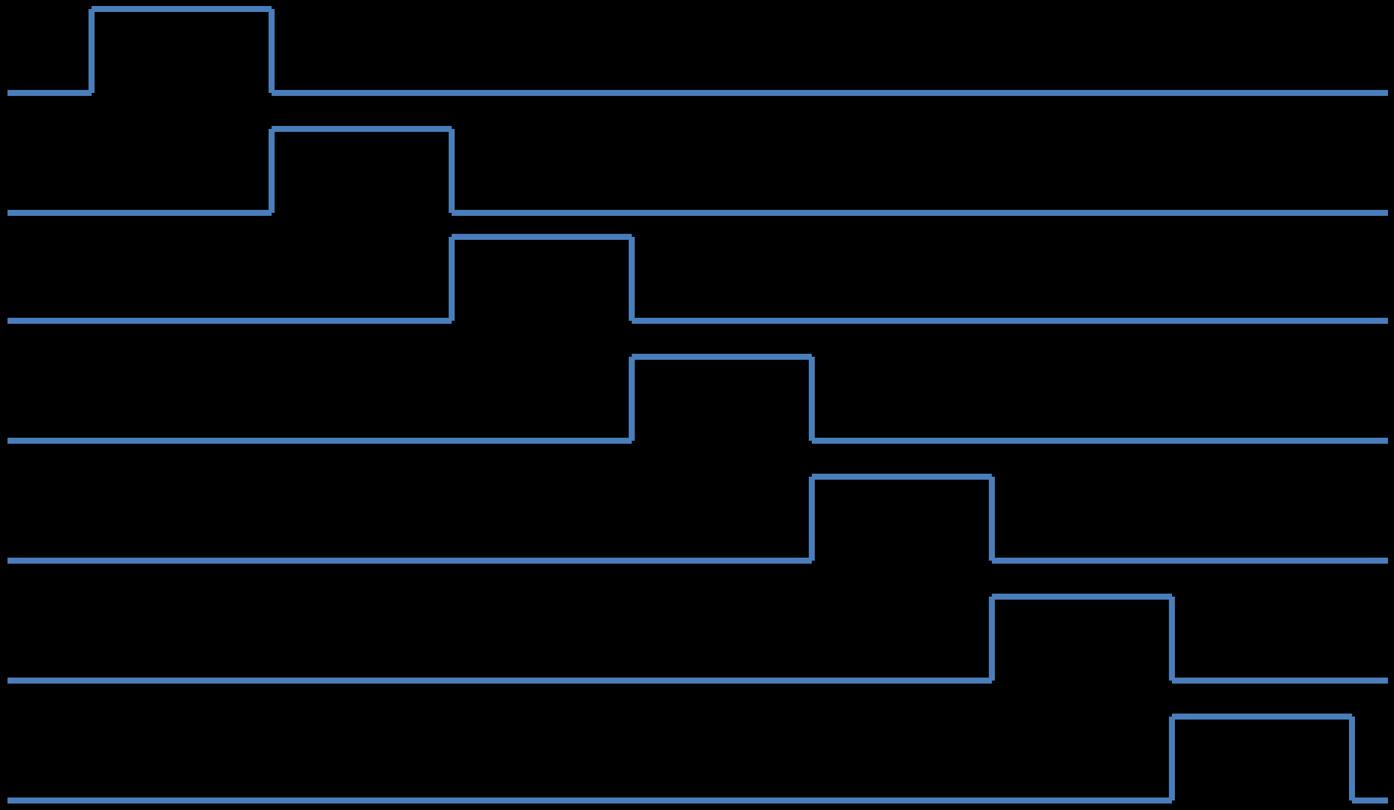
DU = I

- Note each row is orthogonal



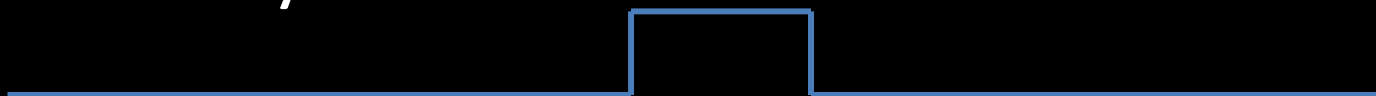
DU = I

- So Let $U = D^T$. Now $DU = DD^T = I$
- What kind of upsampling is U?

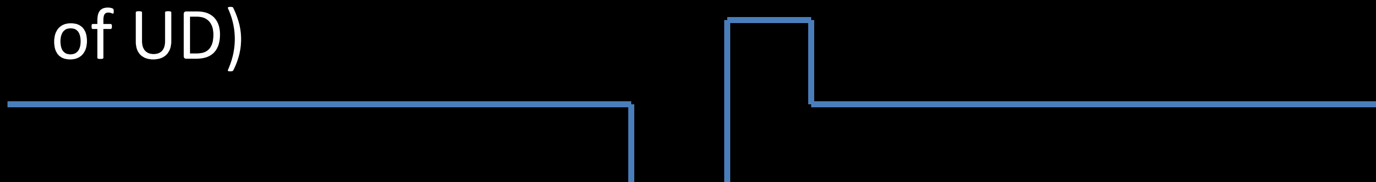


Equivalently…

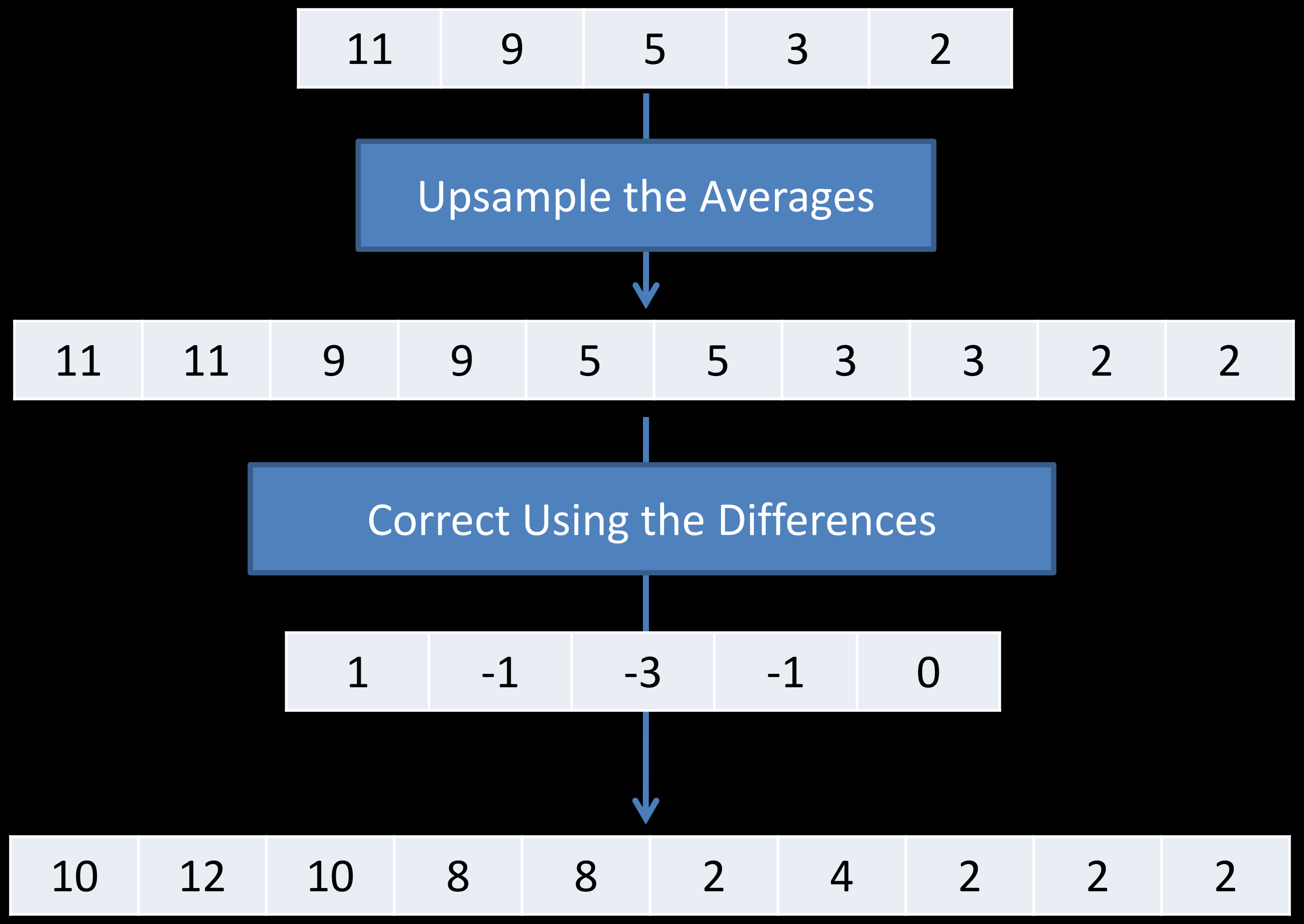
- The scaling function is the downsampling filter. It must be orthogonal to itself when shifted by 2n.



- The wavelet function parameterizes what the downsampling throws away
  - i.e. the null-space of UD (orthogonal to every row of UD)



The 1D Inverse Haar Transform



Recursive Haar Wavelet

- If you want a pyramid instead of a 2-level decomposition, just recurse and decompose the coarse layer again
  - $O(n \log(n))$

2D Haar Wavelet Transform

- 1D Haar transform each row
- 1D Haar transform each column
- If we’re doing a full recursive transform, we can:
  - Do the full recursive transform in X, then the full recursive transform in Y (standard order)
  - Do a single 2D Haar transform, then recurse on the coarse layer (non-standard order)

2D Haar Wavelet Transform

- (demo)

Problem with the Haar Wavelet

- Edges at a certain scale may exist in one of several levels, depending on their position.

10	10	10	10	10	0	0	0	0	0

10	10	5	0	0

Averages (Coarse)

0	0	**-5**	0	0

Differences (Fine)

10	10	10	10	0	0	0	0	0	0

10	10	0	0	0

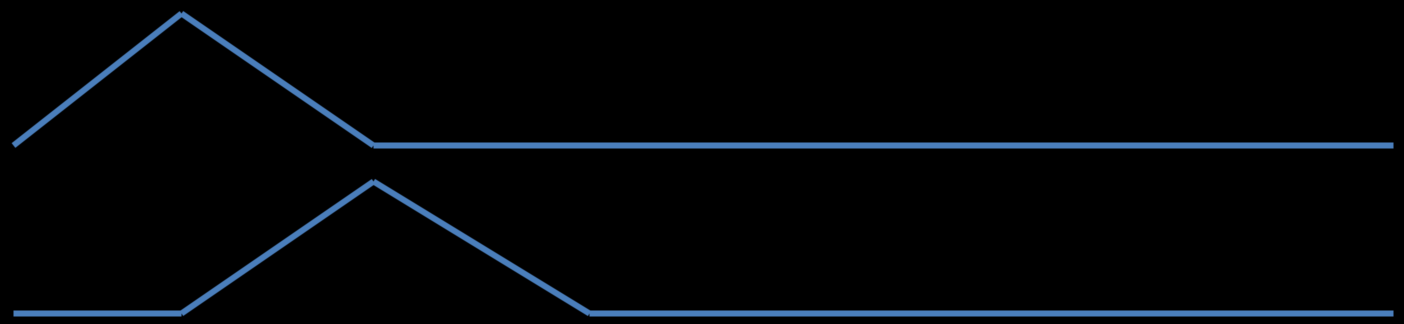
Averages (Coarse)

0	0	**0**	0	0

Differences (Fine)

Better Wavelets

- Let’s try to pick a better downsampling filter (scaling function) so that we don’t miss edges like this
  - Needs a wider support
  - Still has to be orthogonal
- Tent: [¼ ½ ¼]?



Better Wavelets

- Lanczos3 downsampling filter:

```
[0.02 0.00 -0.14 0.00 0.61 1.00 0.61 0.00 -0.14 0.00 0.02]
```

- Dot product = 0.1987
  - not orthogonal to itself shifted

Let’s design one that works

- Scaling function = [a b c d]
- Orthogonal to shifted copy of itself
  - [0 0 a b c d].[a b c d 0 0] = ac + bd = 0
- If we want $DD^T = I$, then should be unit length...
  - [a b c d].[a b c d] = $a^2 + b^2 + c^2 + d^2 = 1$
- That’s two constraints...

more constraints

- Let’s make the wavelet function use the same constants but wiggle: [a -b c -d]
  - Just like the Haar, but 4 wide
- Wavelet function should parameterize what the scaling function loses, so should be orthogonal (even when shifted)
- [a b c d].[a -b c -d] = $a^2 - b^2 + c^2 - d^2 = 0$
- [0 0 a b c d].[a -b c -d 0 0] = $ac - bd = 0$

Wavelet function should also be orthogonal...

- [0 0 a -b c -d].[a -b c -d 0 0] = ac + bd = 0
- Good, we already had this constraint, so we're not overconstrained

The constraints

- $ac + bd = 0$
- $a^2 + b^2 + c^2 + d^2 = 1$
- $a^2 - b^2 + c^2 - d^2 = 0$
- $ac - bd = 0$
- Adding eqs 1 and 4 gives us $a = 0$ or $c = 0$, which we don’t want...
- In fact, this ends up with Haar as the only solution

Try again...

- Let’s reverse the wavelet function
  - Wavelet function = [d -c b -a]
- ac + bd = 0
- $a^2 + b^2 + c^2 + d^2 = 1$
- [a b c d].[d -c b -a] = ad - bc + bd - ad = 0
  - trivially true
- [0 0 a b c d].[d -c b -a 0 0] = ab - ba = 0
  - also trivially true
- [a b c d 0 0].[0 0 d -c b -a] = cd - cd = 0
  - Also trivially true

Now we can add 2 more constraints

- Considerably more freedom to design
- Let's say the coarse image has to be the same brightness as the big image:

  $a + b + c + d = 1$

- And the fine layer has to not be effected by local brightness (details only):

  $d - c + b - a = 0$

Solve:

- $ac + bd = 0$
- $a^2 + b^2 + c^2 + d^2 = 1$
- $a + b + c + d = 1$
- $d - c + b - a = 0$
- Let’s ask the oracle...

No Solutions

- Ok, let's relax $U = D^T$
- It's ok for the coarse layer to get brighter or darker, as long as **DU = I** still holds
- $a^2 + b^2 + c^2 + d^2 = 1$
- ~~$a + b + c + d = 1$~~
- $a + b + c + d > 0$

Solve:

- $ac + bd = 0$
- $a^2 + b^2 + c^2 + d^2 = 1$
- $a + b + c + d > 0$
- $d - c + b - a = 0$
- We’re one constraint short...

Solve:

- $ac + bd = 0$
- $a^2 + b^2 + c^2 + d^2 = 1$
- $a + b + c + d > 0$
- $d - c + b - a = 0$
- We're one constraint short...
- Let's make the scaling function really smooth
  - minimize: $a^2 + (b-a)^2 + (c-b)^2 + (d-c)^2 + d^2$
  - or maximize: $ab + bc + cd$

Solution!

- a = 0.482963
- b = 0.836516
- c = 0.224144
- d = -0.12941

Ingrid Daubechies Solved this Exactly

- a = (1 + sqrt(3)) / (4 sqrt(2))
- b = (3 + sqrt(3)) / (4 sqrt(2))
- c = (3 - sqrt(3)) / (4 sqrt(2))
- d = (1 - sqrt(3)) / (4 sqrt(2))
- Scaling function = [a b c d]
- Wavelet function = [d -c b -a]
- The resulting wavelet is better than Haar, because the downsampling filter is smoother.

</MATH>

Applications

- Compression
- Denoising

Compression

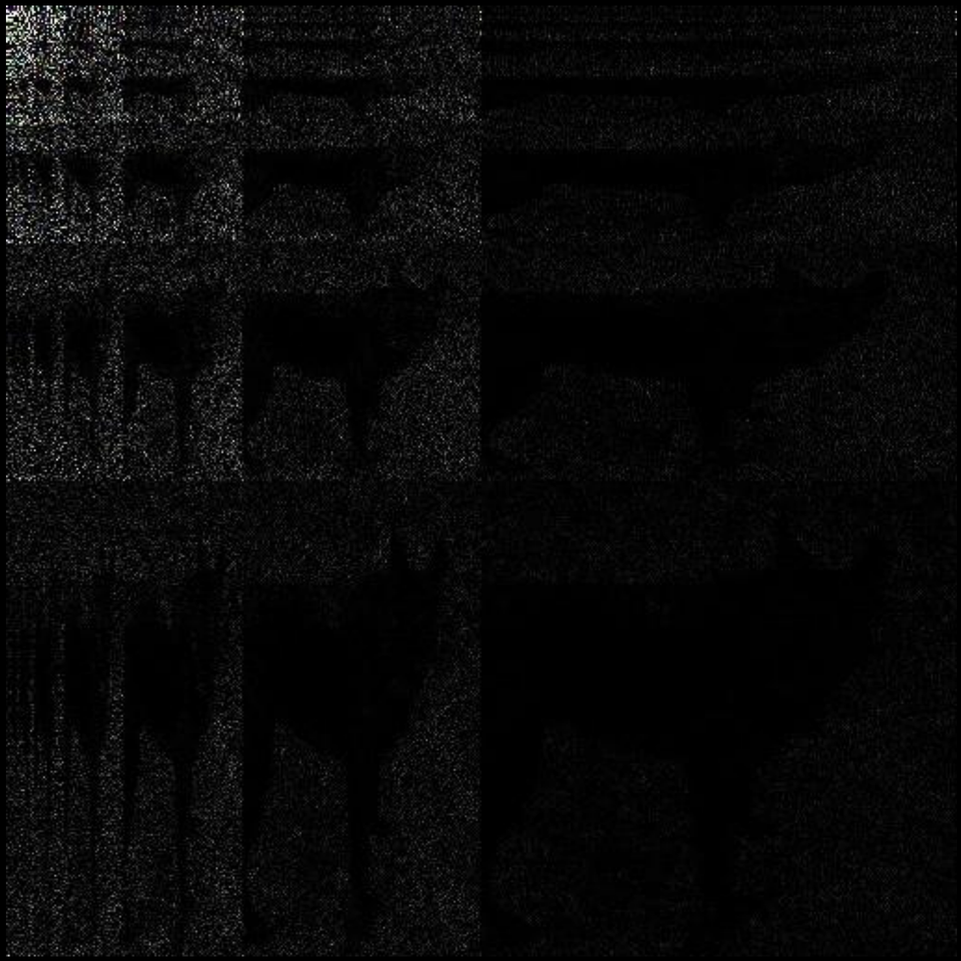
- Idea: throw away small wavelet terms
- Algorithm:
  - Take the wavelet transform
  - Store only values with absolute value greater than some threshold
  - To recontruct image, do inverse wavelet transform assuming the missing values are zero

Compression

- ImageStack -load pic.jpg -daubechies
-               -eval “abs(val) < 0.1 ? 0 : val”
-               -inversedaubechies -display

Input:

Daubechies Transform:



Dropping coefficients below 0.01

30% less data

Dropping coefficients below 0.05

65% less data

Dropping coefficients below 0.1

82% less data

Dropping coefficients below 0.2

94% less data

Daubechies vs Haar at 65% less data

Daubechies vs Reducing Resolution

Denoising

- Similar Idea: Wavelet Shrinkage
  – Take wavelet coefficients and move them towards zero
- E.g.
  – 0.3 -> 0.25
  – -0.2 -> -0.15
  – 0.05 -> 0
  – 0.02 -> 0

Input vs Output

Wavelet Shrinkage vs Bilateral

- Wavelet shrinkage much faster
- Denoised at multiple scales at once

Lifting Schemes

- Turns out there's a better way to derive orthogonal wavelet bases
- We've done enough math for today
- Next Time

Edge-Avoiding Wavelets

- Laplacian Pyramid : Wavelets
- as Bilateral Pyramid : Edge-Avoiding Wavelets

Projects

- Rest of Quarter:
  - Project proposal, due 1 week after due date of assn3
  - 1 Paper presentation on your chosen paper (20 minutes of slides, 15 minutes of class discussion)
  - Final project demo (after thanksgiving break)
  - Final project code due at end of quarter.
  - Intent: rest of quarter is 50-75% of the workload of start of quarter.

Project Ideas:

- http://cs448f.stanford.edu/