Discrete Haar Wavelet Transforms

Catherine Bénéteau

University of South Florida
Tampa, FL USA

UNM - PNM Statewide Mathematics Contest, 2011

A MINI HISTORY OF WAVELETS

- In the late 1970s, Morlet (trained at the Ecole Polytechnique), a geophysicist, became interested in signals that carried information related to geological layers.

A MINI HISTORY OF WAVELETS

- In the late 1970s, Morlet (trained at the Ecole Polytechnique), a geophysicist, became interested in signals that carried information related to geological layers.
- **The problem:** Fourier analysis techniques lacked the ability to *zoom in* on a signal and find short varying high frequencies.

A MINI HISTORY OF WAVELETS

- In the late 1970s, Morlet (trained at the Ecole Polytechnique), a geophysicist, became interested in signals that carried information related to geological layers.
- **The problem:** Fourier analysis techniques lacked the ability to *zoom in* on a signal and find short varying high frequencies.
- Morlet represented his signals via a special kind of function that is an ancestor of what we would call a wavelet today.

A MINI HISTORY OF WAVELETS

- In the late 1970s, Morlet (trained at the Ecole Polytechnique), a geophysicist, became interested in signals that carried information related to geological layers.
- **The problem:** Fourier analysis techniques lacked the ability to *zoom in* on a signal and find short varying high frequencies.
- Morlet represented his signals via a special kind of function that is an ancestor of what we would call a wavelet today.
- But Morlet was a geophysicist, and wanted to make sure his work made sense mathematically, so he contacted Grossman (a quantum physicist), who gave an elegant proof that Morlet's representation worked.

A Mini History of Wavelets

- In the late 1970s, Morlet (trained at the Ecole Polytechnique), a geophysicist, became interested in signals that carried information related to geological layers.
- **The problem:** Fourier analysis techniques lacked the ability to *zoom in* on a signal and find short varying high frequencies.
- Morlet represented his signals via a special kind of function that is an ancestor of what we would call a wavelet today.
- But Morlet was a geophysicist, and wanted to make sure his work made sense mathematically, so he contacted Grossman (a quantum physicist), who gave an elegant proof that Morlet's representation worked.
- In 1984, Yves Meyer(a mathematician) became acquainted with Morlet's work, and noticed right away that Morlet's functions were connected to some deep mathematics that had been worked on in the 1960s, and the subject took off from there.

ENOUGH OF THIS HISTORY!

What would a math lecture be without an exam? It's time to take a test!

You are going to see three images. One is the original image consisting of 149604 bytes of information. A second image is a wavelet-compressed version of the original using 12253 bytes (about 8% of the original size), and another image is a wavelet-compressed version of the original using only 4452 bytes (about 3% of the original size)!

ENOUGH OF THIS HISTORY!

Question: Which is which?

The following images come from a really neat site developed and maintained by Osmar R. Zaïane at Simon Fraser University in Canada.

ENOUGH OF THIS HISTORY!

Image A

ENOUGH OF THIS HISTORY!

Image B

ENOUGH OF THIS HISTORY!

Image C

ENOUGH OF THIS HISTORY!

The Answer Key:

Image	Number of Bytes	Transfer Time*
Image A	4452	0.15 seconds
Image B	149604	5.00 seconds
Image C	12253	0.40 seconds

**This rate assumes a standard 56K dial up modem that typically connects at about 30K.*

SO WHY WAVELETS?

- It is definitely new math - the subject took off in the late 80's/early 90's.

SO WHY WAVELETS?

- It is definitely new math - the subject took off in the late 80's/early 90's.
- If you ask engineering or physics students to name some of the top accomplishments in their respective fields in the past 100 years, they can.

SO WHY WAVELETS?

- It is definitely new math - the subject took off in the late 80's/early 90's.
- If you ask engineering or physics students to name some of the top accomplishments in their respective fields in the past 100 years, they can.
- Mathematics students, when asked the same question, typically struggle to name one or two.

SO WHY WAVELETS?

- It is definitely new math - the subject took off in the late 80's/early 90's.
- If you ask engineering or physics students to name some of the top accomplishments in their respective fields in the past 100 years, they can.
- Mathematics students, when asked the same question, typically struggle to name one or two.
- The reason is most of the math we cover was developed hundreds of years ago and most of those famous mathematicians are ...

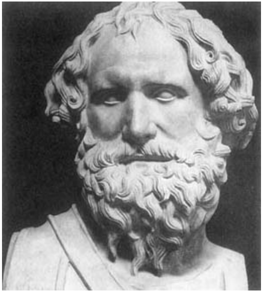
SO WHY WAVELETS?

Isaac Newton - Dead ... for a VERY long time.

SO WHY WAVELETS?

Joseph Fourier - Dead.

SO WHY WAVELETS?



Archimedes - Fossilized.

SO WHY WAVELETS?

C.F. Gauss - Dead .... But Brain is Still Around.

SO WHY WAVELETS?

Alfred Haar - Dead .... Ok, so we're still talking about him.

SO WHY WAVELETS?

But wavelet theory is *new* math and the major players are...

SO WHY WAVELETS?

Ingrid Daubechies - Still Alive!

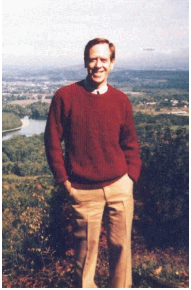
SO WHY WAVELETS?

David Donoho - Still Alive!

SO WHY WAVELETS?

Yves Meyer - Blurry, But Still Alive!

SO WHY WAVELETS?



Gil Strang, Still Alive!

SO WHY WAVELETS?

Applications involving wavelets are numerous. Here are two:

- The FBI uses wavelets to digitally compress fingerprints. The compression factor is between 40 and 100 times the original size of the fingerprint!

SO WHY WAVELETS?

Applications involving wavelets are numerous. Here are two:

- The FBI uses wavelets to digitally compress fingerprints. The compression factor is between 40 and 100 times the original size of the fingerprint!
- Starting in 2000, the Joint Photographic Experts Group, designers of the popular .jpg image format used on the web, began implementing wavelet techniques in their format scheme.

SO WHY WAVELETS?

And two more:

- Kodak Polychrome Graphics, a Twin Cities' based company, produces extremely high resolution digital images for advertisements that appear in publications like Time Magazine. They use wavelets to allow their clients to quickly analyze parts of the image.

SO WHY WAVELETS?

And two more:

- Kodak Polychrome Graphics, a Twin Cities' based company, produces extremely high resolution digital images for advertisements that appear in publications like Time Magazine. They use wavelets to allow their clients to quickly analyze parts of the image.
- Wavelets have been heavily utilized in the modeling of distant galaxies. Wavelets have helped astronomers locate a subcluster of galaxies in the Coma supercluster of 1,400 galaxies! (*Quantum Vol. 15 No. 3*)

SO WHY WAVELETS?

And one more:

- Musicologists restore an 1889 recording of Brahms playing his Hungarian Dance Number 1 from a wax cylinder recording in such bad shape that many listeners failed to recognize that a piano was even being played. (*Quantum Vol. 15 No. 3*)

DIGITAL IMAGES

GRAYSCALE IMAGE BASICS

Digital Images

Grayscale Image Basics

- A *bit* is a fundamental unit on a computer - either 0 or 1.

DIGITAL IMAGES

GRAYSCALE IMAGE BASICS

- A *bit* is a fundamental unit on a computer - either 0 or 1.
- A *byte* is 8 bits. There are $2^8 = 256$ possible bytes. There are also 256 characters on a standard keyboard!

DIGITAL IMAGES

GRAYSCALE IMAGE BASICS

- A *bit* is a fundamental unit on a computer - either 0 or 1.
- A *byte* is 8 bits. There are $2^8 = 256$ possible bytes. There are also 256 characters on a standard keyboard!
- The American Standard Code for Information Interchange (ASCII) assigns to each byte a *character*.

Digital Images

Grayscale Image Basics

- A *bit* is a fundamental unit on a computer - either 0 or 1.
- A *byte* is 8 bits. There are $2^8 = 256$ possible bytes. There are also 256 characters on a standard keyboard!
- The American Standard Code for Information Interchange (ASCII) assigns to each byte a *character*.
- Some of these characters are visible on a standard computer keyboard (eg., *y* is 121 and *0* is 48).

DIGITAL IMAGES

GRAYSCALE IMAGE BASICS

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

DIGITAL IMAGES

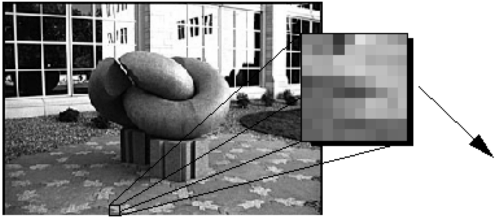
GRAYSCALE IMAGE BASICS

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	φ
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	□

DIGITAL IMAGES

GRAYSCALE IMAGE BASICS

An *8-bit* digital image can be viewed as a matrix whose entries (known as *pixels*) range from 0 (black) to 255 (white).



$$\begin{bmatrix} 129 & 128 & 121 & 51 & 127 & 224 & 201 & 179 & 159 & 140 \\ 148 & 116 & 130 & 75 & 184 & 191 & 182 & 185 & 186 & 180 \\ 175 & 169 & 166 & 195 & 195 & 192 & 168 & 173 & 166 & 158 \\ 157 & 171 & 169 & 182 & 199 & 205 & 191 & 191 & 180 & 172 \\ 73 & 89 & 96 & 100 & 122 & 143 & 166 & 190 & 188 & 180 \\ 93 & 107 & 103 & 81 & 70 & 77 & 106 & 139 & 165 & 181 \\ 106 & 105 & 112 & 132 & 144 & 147 & 189 & 183 & 158 & 184 \\ 102 & 100 & 106 & 124 & 140 & 157 & 179 & 175 & 168 & 175 \\ 91 & 105 & 112 & 93 & 86 & 85 & 100 & 104 & 110 & 106 \\ 97 & 97 & 112 & 102 & 113 & 111 & 105 & 94 & 103 & 104 \end{bmatrix}$$

DIGITAL IMAGES

GRAYSCALE IMAGE BASICS

- Thus if we store an intensity value, say 121, to disk, we don't store 121, we store its ASCII character *y*.

DIGITAL IMAGES

GRAYSCALE IMAGE BASICS

- Thus if we store an intensity value, say 121, to disk, we don't store 121, we store its ASCII character *y*.
- *y* also has a binary representation: 01111001_2.

Digital Images

Grayscale Image Basics

- Thus if we store an intensity value, say 121, to disk, we don't store 121, we store its ASCII character y.
- y also has a binary representation: 01111001_2.
- Thus if an image has dimensions $N \times M$, we need $8MN$ bits to store it on disk (modulo some header information).

DIGITAL IMAGES

GRAYSCALE IMAGE BASICS

- Thus if we store an intensity value, say 121, to disk, we don't store 121, we store its ASCII character y.
- y also has a binary representation: 01111001_2.
- Thus if an image has dimensions $N \times M$, we need $8MN$ bits to store it on disk (modulo some header information).
- We will refer to the stored image as the *bitstream* and note that the bits per pixel (*bpp*) is 8.

HUFFMAN CODING

- In 1952, David Huffman made a simple observation:

HUFFMAN CODING

- In 1952, David Huffman made a simple observation:
- *Rather than use the same number of bits to represent each character, why not use a short bit stream for characters that appear often in an image and a longer bit stream for characters that appear infrequently in the image?*

HUFFMAN CODING

- In 1952, David Huffman made a simple observation:
- *Rather than use the same number of bits to represent each character, why not use a short bit stream for characters that appear often in an image and a longer bit stream for characters that appear infrequently in the image?*
- He then developed an algorithm to do just that. We refer to his simple algorithm as Huffman coding. We will illustrate the algorithm via an example.

HUFFMAN CODING

- Suppose you want to perform Huffman coding on the word seesaws.

HUFFMAN CODING

- Suppose you want to perform Huffman coding on the word seesaws.
- First observe that s appears three times (24 bits), *e* appears twice (16 bits), and *a* and *w* each appear once (16 bits) so the total number of bits needed to represent *seesaws* is 56.

HUFFMAN CODING

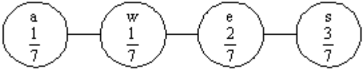
Char.	ASCII	Binary	Frequency
s	115	01110011_2	3
e	101	01100101_2	2
a	97	01100001_2	1
w	119	01110111_2	1

So in terms of bits, the word seesaws is

01110011 01100101 01100101 01110011 01100001 01110111 01110011

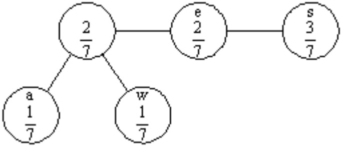
HUFFMAN CODING

The first step in Huffman coding is as follows: Assign probabilities to each character and then sort from smallest to largest. We will put the probabilities in circles called nodes and connect them with lines (branches).



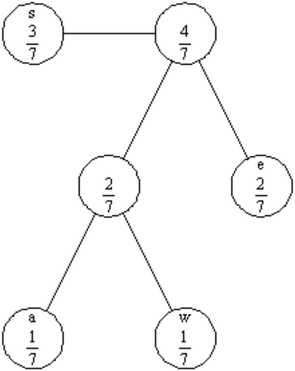
HUFFMAN CODING

Now simply add the two smallest probabilities to create a new node with probability $2/7$. Branch the two small nodes off this one and resort the three remaining nodes:



HUFFMAN CODING

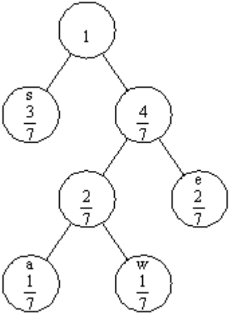
Again we add the smallest two probabilities on the top row ($2/7 + 2/7 = 4/7$), create a new node with everything below these nodes as branches and sort again:



HUFFMAN CODING

Since only two nodes remain on top, we simply add the probabilities of these nodes together to get 1 and obtain our finished tree:

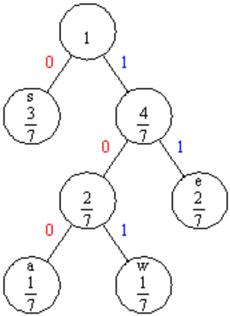
HUFFMAN CODING



HUFFMAN CODING

Now assign to each left branch the value 0 and to each right branch the value 1:

HUFFMAN CODING



HUFFMAN CODING

- We can read the new bit stream for each character right off the tree!

Huffman Coding

- We can read the new bit stream for each character right off the tree!
- Here are the new bit streams for the four characters:

HUFFMAN CODING

Char.	Binary
s	0_2
e	11_2
a	100_2
w	101_2

HUFFMAN CODING

- Since *s* appears three times in *seesaws*, we need 3 bits to represent them. The character *e* appears twice (4 bits), and *a* and *w* each appear once (3 bits each).

HUFFMAN CODING

- Since *s* appears three times in *seesaws*, we need 3 bits to represent them. The character *e* appears twice (4 bits), and *a* and *w* each appear once (3 bits each).
- The total number of bits we need to represent the word *seesaws* is 13 bits! Recall without Huffman coding, we needed 56 bits so we have reduced the number of bits needed by a factor of 4!

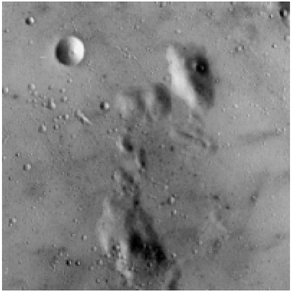
HUFFMAN CODING

- Since *s* appears three times in *seesaws*, we need 3 bits to represent them. The character *e* appears twice (4 bits), and *a* and *w* each appear once (3 bits each).
- The total number of bits we need to represent the word *seesaws* is 13 bits! Recall without Huffman coding, we needed 56 bits so we have reduced the number of bits needed by a factor of 4!
- Here is the word *seesaws* using the Huffman codes for each character:

$$0111101001010$$

ENCODING AN IMAGE

- The example is a bit of a sales job - of course we will enjoy great savings with only 4 distinct characters. What happens when we apply Huffman coding to a digital image?
- Consider the 200×200 image



Encoding an Image

- Unencoded, we need 320000 bits or 8 *bits per pixel* (bpp) to represent the image.

Encoding an Image

- Unencoded, we need 320000 bits or 8 *bits per pixel* (bpp) to represent the image.
- Using Huffman encoding, we only need 266993 bits to store the image.

Encoding an Image

- Unencoded, we need 320000 bits or 8 *bits per pixel* (bpp) to represent the image.
- Using Huffman encoding, we only need 266993 bits to store the image.
- This constitutes a 16.5% savings or an average of 6.67bpp.

ENCODING AN IMAGE

- Unencoded, we need 320000 bits or 8 *bits per pixel* (bpp) to represent the image.
- Using Huffman encoding, we only need 266993 bits to store the image.
- This constitutes a 16.5% savings or an average of 6.67bpp.
- We should be able to do better . . .

ENCODING AN IMAGE

- Unencoded, we need 320000 bits or 8 *bits per pixel* (bpp) to represent the image.
- Using Huffman encoding, we only need 266993 bits to store the image.
- This constitutes a 16.5% savings or an average of 6.67bpp.
- We should be able to do better ...
- What really helps an encoding method is a *preprocessor* that transforms the image to a setting that is a bit more amenable to the encoding scheme.

ENCODING AN IMAGE

- Unencoded, we need 320000 bits or 8 *bits per pixel* (bpp) to represent the image.
- Using Huffman encoding, we only need 266993 bits to store the image.
- This constitutes a 16.5% savings or an average of 6.67bpp.
- We should be able to do better . . .
- What really helps an encoding method is a *preprocessor* that transforms the image to a setting that is a bit more amenable to the encoding scheme.
- That's where the discrete wavelet transform comes in!

CHALLENGE PROBLEMS

- Get with a partner, and each partner should choose a word or short phrase and determine the Huffman codes for each letter.

CHALLENGE PROBLEMS

- Get with a partner, and each partner should choose a word or short phrase and determine the Huffman codes for each letter.
- Write the word as a binary string using the Huffman codes.

CHALLENGE PROBLEMS

- Get with a partner, and each partner should choose a word or short phrase and determine the Huffman codes for each letter.
- Write the word as a binary string using the Huffman codes.
- Give the string and the codes (dictionary) to your partner.

CHALLENGE PROBLEMS

- Get with a partner, and each partner should choose a word or short phrase and determine the Huffman codes for each letter.
- Write the word as a binary string using the Huffman codes.
- Give the string and the codes (dictionary) to your partner.
- The partner should determine the word or phrase.

CHALLENGE PROBLEMS

- Get with a partner, and each partner should choose a word or short phrase and determine the Huffman codes for each letter.
- Write the word as a binary string using the Huffman codes.
- Give the string and the codes (dictionary) to your partner.
- The partner should determine the word or phrase.
- Given the Huffman codes g = 10, n = 01, o = 00, space key=110, e = 1110, and i = 1111, decode the bit stream 1000111101101101000111101101101000011110.

CHALLENGE PROBLEMS

- Are Huffman codes unique? In other words, given a word or a phrase, is it possible to have more than one Huffman code for that word or phrase?

CHALLENGE PROBLEMS

- Are Huffman codes unique? In other words, given a word or a phrase, is it possible to have more than one Huffman code for that word or phrase?
- Are Huffman codes invertible? In other words, given a bit stream and a Huffman code tree or dictionary, can I *always* translate the bit stream?

CHALLENGE PROBLEMS

- Are Huffman codes unique? In other words, given a word or a phrase, is it possible to have more than one Huffman code for that word or phrase?
- Are Huffman codes invertible? In other words, given a bit stream and a Huffman code tree or dictionary, can I *always* translate the bit stream?
- Is it possible to design a word so that one letter has as its code 0 and the other letter has as its code 1?

CHALLENGE PROBLEMS

- Are Huffman codes unique? In other words, given a word or a phrase, is it possible to have more than one Huffman code for that word or phrase?
- Are Huffman codes invertible? In other words, given a bit stream and a Huffman code tree or dictionary, can I *always* translate the bit stream?
- Is it possible to design a word so that one letter has as its code 0 and the other letter has as its code 1?
- For a word of length greater than 2, is it possible for all the letters to have the same code length?

Naive Data Approximation

- Suppose you are given N values

$$\mathbf{x} = (x_1, x_2, \ldots, x_N)$$

where N is even.

Naive Data Approximation

- Suppose you are given N values

$$\mathbf{x} = (x_1, x_2, \ldots, x_N)$$

  where N is even.
- Your task: Send an approximation **s** (a list of numbers) of this data via the internet to a colleague.

NAIVE DATA APPROXIMATION

- Suppose you are given N values

$$\mathbf{x} = (x_1, x_2, \ldots, x_N)$$

  where N is even.
- Your task: Send an approximation $\mathbf{s}$ (a list of numbers) of this data via the internet to a colleague.
- In order to reduce transfer time, the length of your approximation must be $N/2$.

Naive Data Approximation

- Suppose you are given N values

$$\mathbf{x} = (x_1, x_2, \ldots, x_N)$$

  where N is even.
- Your task: Send an approximation $\mathbf{s}$ (a list of numbers) of this data via the internet to a colleague.
- In order to reduce transfer time, the length of your approximation must be $N/2$.
- How do you suggest we do it?

Naive Data Approximation

- One solution is to pair-wise average the numbers:

$$s_k = \frac{x_{2k-1} + x_{2k}}{2}, \qquad k = 1, \ldots, N/2$$

Naive Data Approximation

- One solution is to pair-wise average the numbers:

$$s_k = \frac{x_{2k-1} + x_{2k}}{2}, \qquad k = 1, \dots, N/2$$

- For example:

$$\mathbf{x} = (6, 12, 15, 15, 14, 12, 120, 116) \to \mathbf{s} = (9, 15, 13, 118)$$

- Suppose now you were allowed to send extra data in addition to the pair-wise averages list **s**.

- Suppose now you were allowed to send extra data in addition to the pair-wise averages list **s**.
- The idea is to send a second list of data **d** so that the original list **x** can be recovered from **s** and **d**.

- Suppose now you were allowed to send extra data in addition to the pair-wise averages list **s**.
- The idea is to send a second list of data **d** so that the original list **x** can be recovered from **s** and **d**.
- How would you do it?

- There are a couple of choices for d_k (called directed distances):

- There are a couple of choices for d_k (called directed distances):
- We could set

$$d_k = \frac{x_{2k-1} - x_{2k}}{2}, \qquad k = 1, \ldots, N/2$$

- There are a couple of choices for d_k (called directed distances):
- We could set

$$d_k = \frac{x_{2k-1} - x_{2k}}{2}, \qquad k = 1, \ldots, N/2$$

- or

$$d_k = \frac{x_{2k} - x_{2k-1}}{2}, \qquad k = 1, \ldots, N/2$$

- There are a couple of choices for d_k (called directed distances):
- We could set

$$d_k = \frac{x_{2k-1} - x_{2k}}{2}, \qquad k = 1, \ldots, N/2$$

- or

$$d_k = \frac{x_{2k} - x_{2k-1}}{2}, \qquad k = 1, \ldots, N/2$$

- We will use the second formula.

$$\boxed{d_k = \frac{x_{2k} - x_{2k-1}}{2}, \qquad k = 1, \ldots, N/2}$$

- The process is invertible since

$$s_k + d_k = \frac{x_{2k-1} + x_{2k}}{2} + \frac{x_{2k} - x_{2k-1}}{2} = x_{2k}$$

and

$$s_k - d_k = \frac{x_{2k-1} + x_{2k}}{2} - \frac{x_{2k} - x_{2k-1}}{2} = x_{2k-1}$$

- The process is invertible since

$$s_k + d_k = \frac{x_{2k-1} + x_{2k}}{2} + \frac{x_{2k} - x_{2k-1}}{2} = x_{2k}$$

  and

$$s_k - d_k = \frac{x_{2k-1} + x_{2k}}{2} - \frac{x_{2k} - x_{2k-1}}{2} = x_{2k-1}$$

- So we map $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ to $(\mathbf{s} \,|\, \mathbf{d}) = (s_1, \ldots, s_{N/2} \,|\, d_1, \ldots, d_{N/2})$.

- The process is invertible since

$$s_k + d_k = \frac{x_{2k-1} + x_{2k}}{2} + \frac{x_{2k} - x_{2k-1}}{2} = x_{2k}$$

  and

$$s_k - d_k = \frac{x_{2k-1} + x_{2k}}{2} - \frac{x_{2k} - x_{2k-1}}{2} = x_{2k-1}$$

- So we map $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ to $(\mathbf{s} \,|\, \mathbf{d}) = (s_1, \ldots, s_{N/2} \,|\, d_1, \ldots, d_{N/2})$.
- Using our example values we have

$$(6, 12, 15, 15, 14, 12, 120, 116) \to (9, 15, 13, 118 \,|\, 3, 0, -1, -2)$$

- The process is invertible since

$$s_k + d_k = \frac{x_{2k-1} + x_{2k}}{2} + \frac{x_{2k} - x_{2k-1}}{2} = x_{2k}$$

  and

$$s_k - d_k = \frac{x_{2k-1} + x_{2k}}{2} - \frac{x_{2k} - x_{2k-1}}{2} = x_{2k-1}$$

- So we map $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ to $(\mathbf{s} \,|\, \mathbf{d}) = (s_1, \ldots, s_{N/2} \,|\, d_1, \ldots, d_{N/2})$.
- Using our example values we have

$$(6, 12, 15, 15, 14, 12, 120, 116) \to (9, 15, 13, 118 \,|\, 3, 0, -1, -2)$$

- Why might people prefer the data in this form?

- We can identify large changes in the the differences portion **d** of the transform.

- We can identify large changes in the the differences portion **d** of the transform.
- It is easier to *quantize* the data in this form.

- We can identify large changes in the the differences portion **d** of the transform.
- It is easier to *quantize* the data in this form.
- The transform concentrates the information (*energy*) in the signal in fewer values.

- We can identify large changes in the the differences portion **d** of the transform.
- It is easier to *quantize* the data in this form.
- The transform concentrates the information (*energy*) in the signal in fewer values.
- And the obvious answer: fewer digits!!

- We can identify large changes in the the differences portion **d** of the transform.
- It is easier to *quantize* the data in this form.
- The transform concentrates the information (*energy*) in the signal in fewer values.
- And the obvious answer: fewer digits!!
- We will talk about quantizing and image compression a little later.

- The transformation

$$\mathbf{x} = (x_1, \ldots, x_N) \to (\mathbf{s} \,|\, \mathbf{d}) = (s_1, \ldots, s_{N/2} \,|\, d_1, \ldots, d_{N/2})$$

is (almost!) called the (1-dimensional) Discrete Haar Wavelet Transformation. (Actually, usually we would multiply by $\sqrt{2}$ - why do you think we might do that?)

- The transformation

$$\mathbf{x} = (x_1, \ldots, x_N) \to (\mathbf{s} \,|\, \mathbf{d}) = (s_1, \ldots, s_{N/2} \,|\, d_1, \ldots, d_{N/2})$$

  is (almost!) called the (1-dimensional) Discrete Haar Wavelet Transformation. (Actually, usually we would multiply by $\sqrt{2}$ - why do you think we might do that?)
- What does the transform look like as a matrix?

Consider applying the transform to an 8-vector. What is the matrix that works?

$$\begin{bmatrix} & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \\ & & & & & & & \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} x_1 + x_2 \\ x_3 + x_4 \\ x_5 + x_6 \\ x_7 + x_8 \\ \hline x_2 - x_1 \\ x_4 - x_3 \\ x_6 - x_5 \\ x_8 - x_7 \end{bmatrix}$$

Consider applying the transform to an 8-vector. What is the matrix that works?

$$\left[\begin{array}{cccccccc} \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \hline -\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{2} & \frac{1}{2} \end{array}\right] \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix} = \frac{1}{2}\left[\begin{array}{c} x_1 + x_2 \\ x_3 + x_4 \\ x_5 + x_6 \\ x_7 + x_8 \\ \hline x_2 - x_1 \\ x_4 - x_3 \\ x_6 - x_5 \\ x_8 - x_7 \end{array}\right]$$

We will denote the transform matrix by W_8.

What about W_8^{-1}? That is, what matrix solves

$$\begin{bmatrix} & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \end{bmatrix} \cdot \left(\frac{1}{2} \begin{bmatrix} x_1 + x_2 \\ x_3 + x_4 \\ x_5 + x_6 \\ x_7 + x_8 \\ \hline x_2 - x_1 \\ x_4 - x_3 \\ x_6 - x_5 \\ x_8 - x_7 \end{bmatrix} \right) = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix}$$

What about W_8^{-1}? That is, what matrix solves

$$\left[\begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right] \cdot \left(\frac{1}{2} \left[\begin{array}{c} x_1 + x_2 \\ x_3 + x_4 \\ x_5 + x_6 \\ x_7 + x_8 \\ \hline x_2 - x_1 \\ x_4 - x_3 \\ x_6 - x_5 \\ x_8 - x_7 \end{array}\right] \right) = \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{array}\right]$$

- The matrices W_8^{-1} and W_8^T are very closely connected!

- The matrices W_8^{-1} and W_8^T are very closely connected!
- In fact, $2W_8^T = W_8^{-1}$.

- The matrices W_8^{-1} and W_8^T are very closely connected!
- In fact, $2W_8^T = W_8^{-1}$.
- This makes W_8 very close to being what we call an orthogonal matrix. (That's why we usually throw in a $\sqrt{2}$!)

Consider the 480×640 image (call it A)

If $\mathbf{a}^1, \ldots, \mathbf{a}^{640}$ are the columns of A, then computing $W_{480}A$ is the same as applying the HWT to each column of A:

$$W_{480}A = (W_{480} \cdot \mathbf{a}^1, \ldots, W_{480} \cdot \mathbf{a}^{640})$$

Graphically, we have

- $C = W_{480}A$ processes the columns of A.

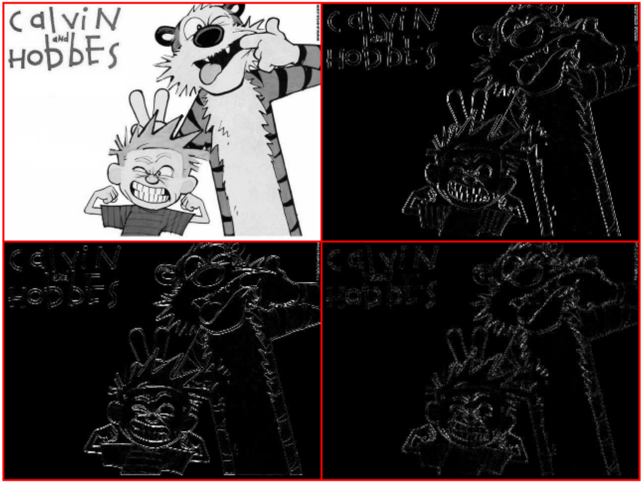
- $C = W_{480}A$ processes the columns of A.
- How would we process the rows of C?

- $C = W_{480}A$ processes the columns of A.
- How would we process the rows of C?
- We compute $CW_{640}^T = W_{480}AW_{640}^T$.

- $C = W_{480}A$ processes the columns of A.
- How would we process the rows of C?
- We compute $CW_{640}^T = W_{480}AW_{640}^T$.
- The two-dimensional Haar transform of the $M \times N$ matrix A is

$$\boxed{B = W_M A W_N^T}$$

Graphically, we have



- Can we interpret what the transformation does to the image?

- Can we interpret what the transformation does to the image?
- Suppose A is the 4×4 matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

- Can we interpret what the transformation does to the image?
- Suppose A is the 4×4 matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

- Partitioning $W_4 = \begin{bmatrix} H \\ - \\ G \end{bmatrix}$, we have

$$
\begin{aligned}
W_4 A W_4^T &= \begin{bmatrix} H \\ - \\ G \end{bmatrix} A \left[H^T \mid G^T \right] \\
&= \begin{bmatrix} HA \\ - \\ GA \end{bmatrix} \left[H^T \mid G^T \right] \\
&= \left[\begin{array}{c|c} HAH^T & HAG^T \\ \hline GAH^T & GAG^T \end{array} \right]
\end{aligned}
$$

Let's look at each 2×2 block individually:

- $HAH^T = \frac{1}{4}\left[\begin{array}{c|c} a_{11}+a_{12}+a_{21}+a_{22} & a_{13}+a_{14}+a_{23}+a_{24} \\ \hline a_{31}+a_{32}+a_{41}+a_{42} & a_{33}+a_{34}+a_{43}+a_{44} \end{array}\right]$

- $HAH^T = \frac{1}{4}\left[\begin{array}{c|c} a_{11}+a_{12}+a_{21}+a_{22} & a_{13}+a_{14}+a_{23}+a_{24} \\ \hline a_{31}+a_{32}+a_{41}+a_{42} & a_{33}+a_{34}+a_{43}+a_{44} \end{array}\right]$
- Partition A in 2×2 blocks as

$$A = \left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right] = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right]$$

- $HAH^T = \frac{1}{4}\left[\begin{array}{c|c} a_{11}+a_{12}+a_{21}+a_{22} & a_{13}+a_{14}+a_{23}+a_{24} \\ \hline a_{31}+a_{32}+a_{41}+a_{42} & a_{33}+a_{34}+a_{43}+a_{44} \end{array}\right]$
- Partition A in 2×2 blocks as

$$A = \left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right] = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right]$$

- Then the (i, j) element of HAH^T is simply the average of the elements in A_{ij}!

- $HAH^T = \frac{1}{4}\left[\begin{array}{c|c} a_{11}+a_{12}+a_{21}+a_{22} & a_{13}+a_{14}+a_{23}+a_{24} \\ \hline a_{31}+a_{32}+a_{41}+a_{42} & a_{33}+a_{34}+a_{43}+a_{44} \end{array}\right]$
- Partition A in 2×2 blocks as

$$A = \left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right] = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right]$$

- Then the (i, j) element of HAH^T is simply the average of the elements in A_{ij}!
- So HAH^T is an approximation or blur of the original image. We will denote HAH^T as $\mathcal{B}$.

- The upper right hand corner is

$$HAG^T = \frac{1}{4}\begin{bmatrix} (a_{12}+a_{22})-(a_{11}+a_{21}) & (a_{14}+a_{24})-(a_{13}+a_{23}) \\ (a_{32}+a_{42})-(a_{31}+a_{41}) & (a_{34}+a_{44})-(a_{33}+a_{43}) \end{bmatrix}$$

- The upper right hand corner is

$$HAG^T = \frac{1}{4}\begin{bmatrix} (a_{12}+a_{22})-(a_{11}+a_{21}) & (a_{14}+a_{24})-(a_{13}+a_{23}) \\ (a_{32}+a_{42})-(a_{31}+a_{41}) & (a_{34}+a_{44})-(a_{33}+a_{43}) \end{bmatrix}$$

- Partition A in 2×2 blocks as

$$A = \left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right] = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right]$$

- The upper right hand corner is

$$HAG^T = \frac{1}{4}\begin{bmatrix} (a_{12}+a_{22})-(a_{11}+a_{21}) & (a_{14}+a_{24})-(a_{13}+a_{23}) \\ (a_{32}+a_{42})-(a_{31}+a_{41}) & (a_{34}+a_{44})-(a_{33}+a_{43}) \end{bmatrix}$$

- Partition A in 2×2 blocks as

$$A = \left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right] = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right]$$

- The (i,j) element of HAG^T can be viewed as a difference between columns of A_{ij}.

- The upper right hand corner is

$$HAG^T = \frac{1}{4}\begin{bmatrix} (a_{12}+a_{22})-(a_{11}+a_{21}) & (a_{14}+a_{24})-(a_{13}+a_{23}) \\ (a_{32}+a_{42})-(a_{31}+a_{41}) & (a_{34}+a_{44})-(a_{33}+a_{43}) \end{bmatrix}$$

- Partition A in 2×2 blocks as

$$A = \left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right] = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right]$$

- The (i,j) element of HAG^T can be viewed as a difference between columns of A_{ij}.
- We will denote HAG^T as $\mathcal{V}$ (for vertical differences).

- The lower left hand corner is

$$GAH^T = \frac{1}{4}\begin{bmatrix} (a_{21}+a_{22})-(a_{12}+a_{11}) & (a_{23}+a_{24})-(a_{13}+a_{14}) \\ (a_{31}+a_{32})-(a_{42}+a_{41}) & (a_{43}+a_{44})-(a_{33}+a_{34}) \end{bmatrix}$$

- The lower left hand corner is

$$GAH^T = \frac{1}{4}\begin{bmatrix} (a_{21}+a_{22})-(a_{12}+a_{11}) & (a_{23}+a_{24})-(a_{13}+a_{14}) \\ (a_{31}+a_{32})-(a_{42}+a_{41}) & (a_{43}+a_{44})-(a_{33}+a_{34}) \end{bmatrix}$$

- Partition A in 2×2 blocks as

$$A = \left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right] = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right]$$

- The lower left hand corner is

$$GAH^T = \frac{1}{4}\begin{bmatrix} (a_{21}+a_{22})-(a_{12}+a_{11}) & (a_{23}+a_{24})-(a_{13}+a_{14}) \\ (a_{31}+a_{32})-(a_{42}+a_{41}) & (a_{43}+a_{44})-(a_{33}+a_{34}) \end{bmatrix}$$

- Partition A in 2×2 blocks as

$$A = \left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right] = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right]$$

- The (i, j) element of HAG^T can be viewed as a difference between rows of A_{ij}.

- The lower left hand corner is

$$GAH^T = \frac{1}{4}\begin{bmatrix} (a_{21}+a_{22})-(a_{12}+a_{11}) & (a_{23}+a_{24})-(a_{13}+a_{14}) \\ (a_{31}+a_{32})-(a_{42}+a_{41}) & (a_{43}+a_{44})-(a_{33}+a_{34}) \end{bmatrix}$$

- Partition A in 2×2 blocks as

$$A = \left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right] = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right]$$

- The (i, j) element of HAG^T can be viewed as a difference between rows of A_{ij}.
- We will denote GAH^T as $\mathcal{H}$ (for horizontal differences).

- The lower right hand corner is

$$GAG^T = \frac{1}{4}\left[\begin{array}{cc} (a_{11}+a_{22})-(a_{12}+a_{21}) & (a_{13}+a_{24})-(a_{23}+a_{14}) \\ (a_{31}+a_{42})-(a_{32}+a_{41}) & (a_{33}+a_{44})-(a_{43}+a_{34}) \end{array}\right]$$

- The lower right hand corner is

$$GAG^T = \frac{1}{4}\begin{bmatrix} (a_{11}+a_{22})-(a_{12}+a_{21}) & (a_{13}+a_{24})-(a_{23}+a_{14}) \\ (a_{31}+a_{42})-(a_{32}+a_{41}) & (a_{33}+a_{44})-(a_{43}+a_{34}) \end{bmatrix}$$

- Partition A in 2×2 blocks as

$$A = \left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right] = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right]$$

- The lower right hand corner is

$$GAG^T = \frac{1}{4}\begin{bmatrix} (a_{11}+a_{22})-(a_{12}+a_{21}) & (a_{13}+a_{24})-(a_{23}+a_{14}) \\ (a_{31}+a_{42})-(a_{32}+a_{41}) & (a_{33}+a_{44})-(a_{43}+a_{34}) \end{bmatrix}$$

- Partition A in 2×2 blocks as

$$A = \left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right] = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right]$$

- The (i,j) element of GAG^T can be viewed as a difference between the diagonals of A_{ij}.

- The lower right hand corner is

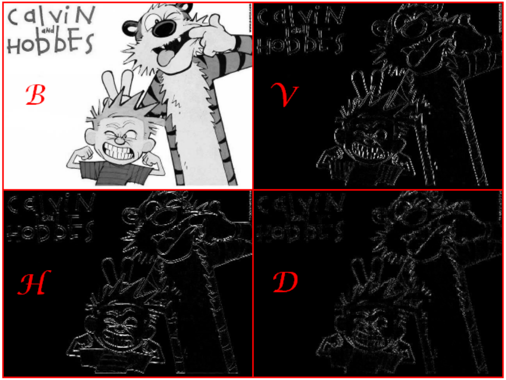
$$GAG^T = \frac{1}{4}\left[\begin{array}{cc} (a_{11}+a_{22})-(a_{12}+a_{21}) & (a_{13}+a_{24})-(a_{23}+a_{14}) \\ (a_{31}+a_{42})-(a_{32}+a_{41}) & (a_{33}+a_{44})-(a_{43}+a_{34}) \end{array}\right]$$

- Partition A in 2×2 blocks as

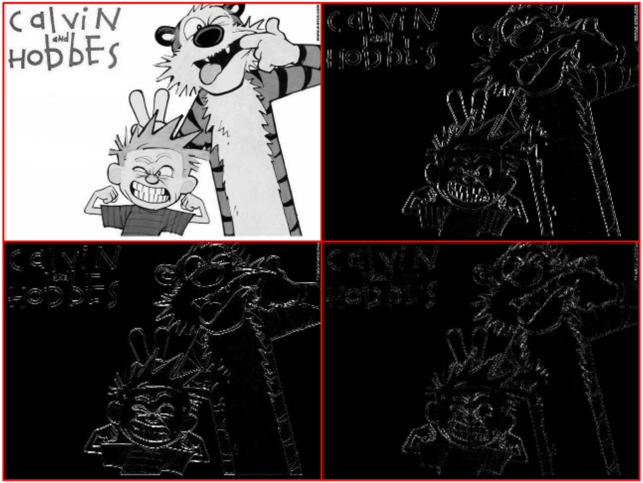
$$A = \left[\begin{array}{cc|cc} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ \hline a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{array}\right] = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right]$$

- The (i,j) element of GAG^T can be viewed as a difference between the diagonals of A_{ij}.
- We will denote GAG^T as $\mathcal{D}$ (for diagonal differences).

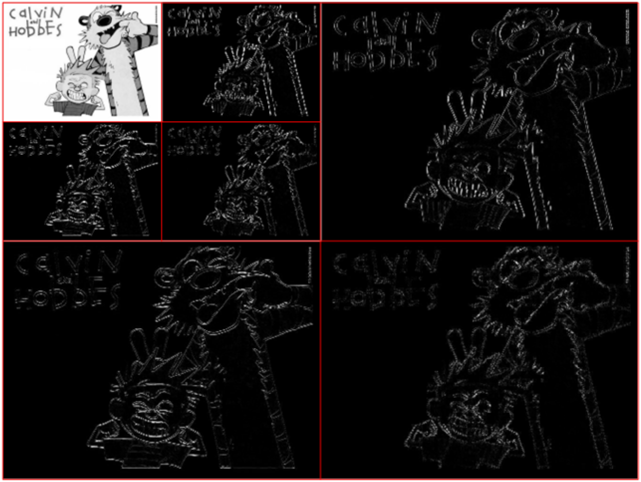
So again the transform of our image is



We can apply the HWT to the blur.



We can apply the HWT to the blur. This is called the iterated HWT.



THIS IS JUST A TASTE!

- All the examples discussed today are examples of *orthogonal* wavelets - in many applications (such as JPEG2000), wavelet transforms used are *biorthogonal*.

This is just a taste!

- All the examples discussed today are examples of *orthogonal* wavelets - in many applications (such as JPEG2000), wavelet transforms used are *biorthogonal*.
- There are variations of the wavelet algorithm - FBI fingerprint compression uses what is called *wavelet packets*.

This is just a taste!

- All the examples discussed today are examples of *orthogonal* wavelets - in many applications (such as JPEG2000), wavelet transforms used are *biorthogonal*.
- There are variations of the wavelet algorithm - FBI fingerprint compression uses what is called *wavelet packets*.
- Wavelets are a great topic for undergraduates to see some modern mathematics and connections between theory and application.

THIS IS JUST A TASTE!

- All the examples discussed today are examples of *orthogonal* wavelets - in many applications (such as JPEG2000), wavelet transforms used are *biorthogonal*.
- There are variations of the wavelet algorithm - FBI fingerprint compression uses what is called *wavelet packets*.
- Wavelets are a great topic for undergraduates to see some modern mathematics and connections between theory and application.
- Thank you for your attention! Check out the pdf file of challenge problems for the Haar wavelet transformation.