Extra Hadamard Lecture

A = hah, a is the image, h is the sequency ordered hadamard matrix.

So if you just do H @ a , you've only done the first pass (transforms along the *horizontal* direction but leaves each column untouched) and haven't yet projected onto the vertical-basis patterns—hence it won't match the full 2D transform. After you have post-multiplied by h, this transforms both *rows* and *columns*, giving the true 2D Hadamard spectrum you see in your 4×4 coefficient matrix.

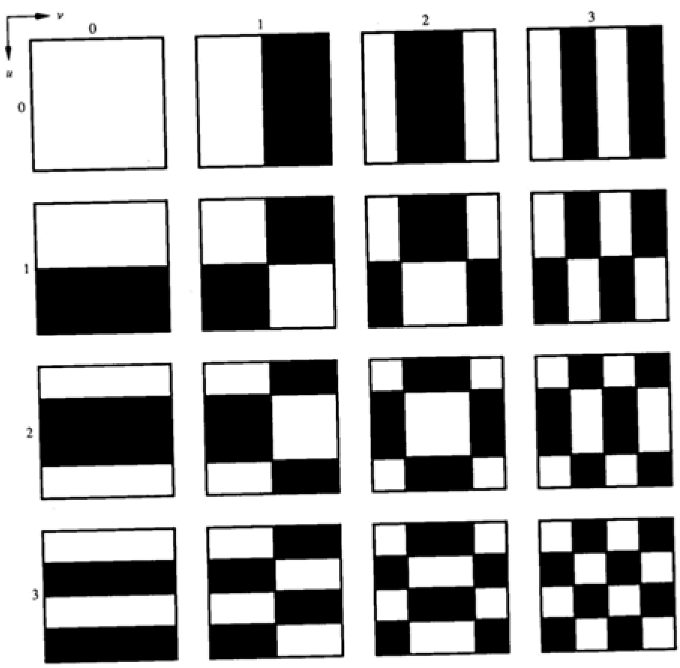
natural hadamard can do compression and coding

sequency ordered can do those plus filtering in hadamard doman

need bit reverse etc to convert natural hadamard to sequency ordered hadamard

> Any 4×4 image can be represented as a weighted sum of the following 16 4×4 square waves in the SPATIAL DOMAIN. These are the Walsh basis patterns.

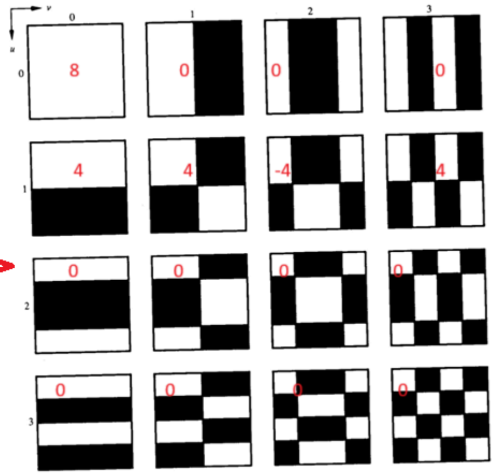
(white is 1, black is 0)



And what sequency hadamard transform actually does it calculating the magnitude of these weightages in order (low → high spatial frequency basis)

So A (4×4 matrix) = hah where a is the spatial image has these coefficients

$$A = \begin{bmatrix} 8. & 0. & 0. & 0. \\ 4. & 4. & -4. & 4. \\ 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. \end{bmatrix}$$



Notice the u and v axis in the A matrix.

u represents the vertical frequency and v represents horizontal frequency.

Eg A[0,0] represents the wavelet with no horizontal and vertical frequency

A[0,1] represents the wavelet with low horizontal and no vertical frequency

A[0,2] represents the wavelet with higher horizontal and no vertical frequency

A[1,0] → no horizontal frequency but low vertical frequency

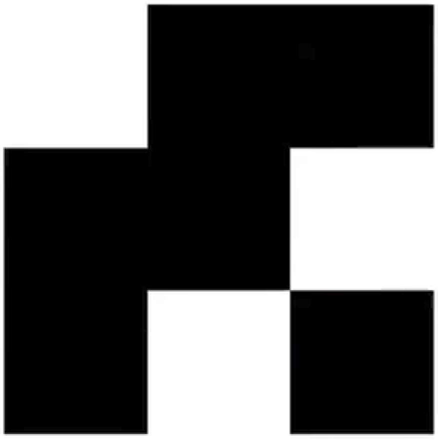
A[2,0] → no horizontal but higher vertical frequency

A[3,0] → no horizontal but more higher (lol) vertical frequency

Other A[i, j] has a combination of horizontal and vertical frequency

A[3,3] → highest horizontal and highest vertical frequency

Now, why is A[0][0] = sum of all the pixels? Look at hah and the sequency ordered hadamard matrix's first row and first column.



the (0,0) entry, A[0][0], is calculated using the first row of h on both sides. That is:

A[0][0] = (first row of h) · a · (first column of h)

So then since u and v represents frequency then you can filter out A in whatever way you like to preserve either high 2d frequencies by keeping coefficients of high frequency square waves intact or by removing them and preserving only the low frequency square waves by keeping their coefficients intact

How to convert Natural hadamard matrices to sequency ordered hadamard matrices?

(now the slide deck begins)

(by the way, take a look at the recursive formulation of the hadamard matrices)

Here, you only need to rearrange the rows, and the columns will be rearranged automatically.

1) Bit-by-bit GRAY 2 Bin algorithm

Let your Gray code be an nnn-bit string gn−1gn−2...g1g0g_{n-1}g_{n-2}\dots g_1g_0gn−1gn−2...g1g0. You build the binary bits bn−1...b0b_{n-1}\dots b_0bn−1...b0 as follows:

1. **MSB copies directly:**bn−1=gn−1.

   bn−1 = gn−1. b_{n-1} \;=\; g_{n-1}.

2. **Each lower bit is the XOR of the Gray bit with the previously computed binary bit:**bi=bi+1⊕gifor i=n−2,n−3,...,0.

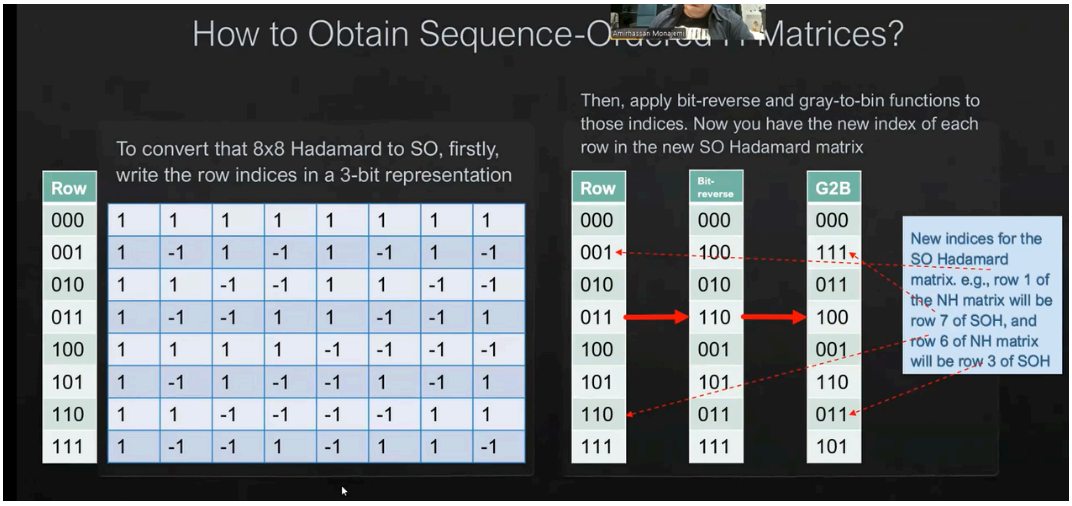
   bi = bi+1 ⊕ gifor i=n−2,n−3,...,0. b_i \;=\; b_{i+1}\;\oplus\;g_i
    \quad
    \text{for }i = n-2,n-3,\dots,0.

Example

Gray = `1101` (4-bit)

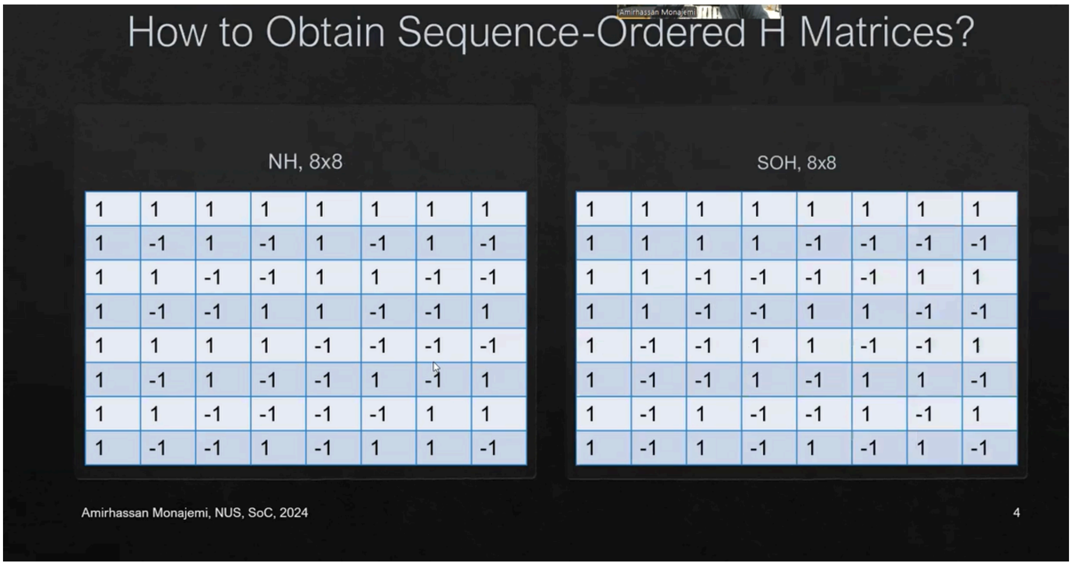
1. b3=g3=1b_3 = g_3 = 1b3=g3=1
2. b2=b3⊕g2=1⊕1=0b_2 = b_3\oplus g_2 = 1\oplus1 = 0b2=b3⊕g2=1⊕1=0
3. b1=b2⊕g1=0⊕0=0b_1 = b_2\oplus g_1 = 0\oplus0 = 0b1=b2⊕g1=0⊕0=0
4. b0=b1⊕g0=0⊕1=1b_0 = b_1\oplus g_0 = 0\oplus1 = 1b0=b1⊕g0=0⊕1=1

So `1101` (Gray) → `1001` (binary).



Row								
000	1	1	1	1	1	1	1	1
001	1	-1	1	-1	1	-1	1	-1
010	1	1	-1	-1	1	1	-1	-1
011	1	-1	-1	1	1	-1	-1	1
100	1	1	1	1	-1	-1	-1	-1
101	1	-1	1	-1	-1	1	-1	1
110	1	1	-1	-1	-1	-1	1	1
111	1	-1	-1	1	-1	1	1	-1

Row	Bit-reverse	G2B
000	000	000
001	100	111
010	010	011
011	110	100
100	001	001
101	101	110
110	011	011
111	111	101



NH, 8x8

1	1	1	1	1	1	1	1
1	-1	1	-1	1	-1	1	-1
1	1	-1	-1	1	1	-1	-1
1	-1	-1	1	1	-1	-1	1
1	1	1	1	-1	-1	-1	-1
1	-1	1	-1	-1	1	-1	1
1	1	-1	-1	-1	-1	1	1
1	-1	-1	1	-1	1	1	-1

SOH, 8x8

1	1	1	1	1	1	1	1
1	1	1	1	-1	-1	-1	-1
1	1	-1	-1	-1	-1	1	1
1	1	-1	-1	1	1	-1	-1
1	-1	-1	1	1	-1	-1	1
1	-1	-1	1	-1	1	1	-1
1	-1	1	-1	-1	1	-1	1
1	-1	1	-1	1	-1	1	-1

Butterworth low pass

For one 1:

$$H(f) = \frac{1}{1+\left(\frac{f}{f_c}\right)^{2n}}$$

This is a low pass filter

[redacted]

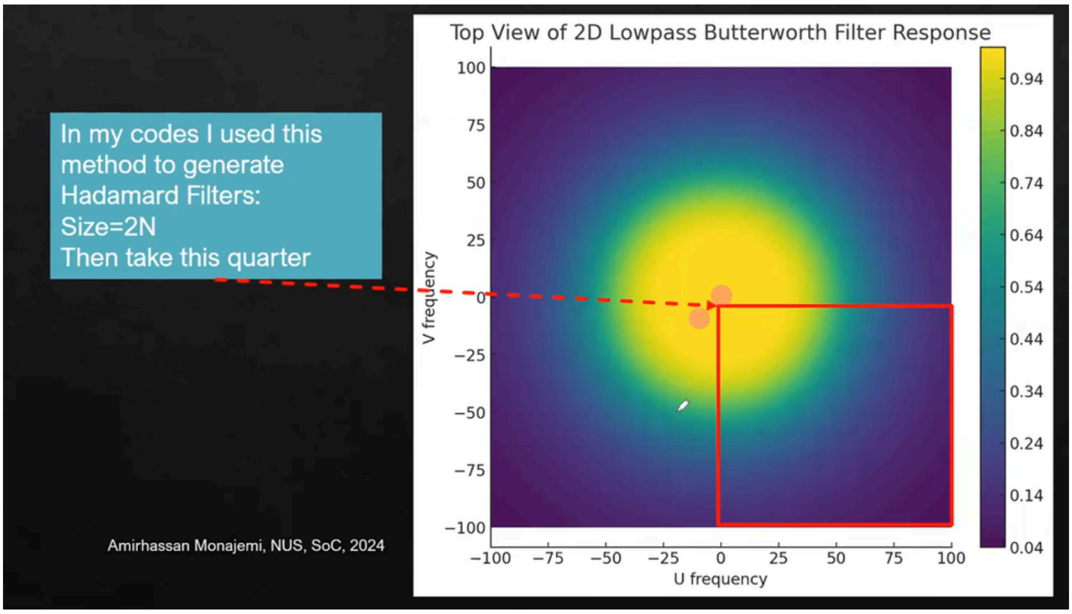
$$H(u,v) = \frac{1}{1+\left(\frac{D(u,v)}{D_0}\right)^{2n}}$$

Here's what each term in this equation represents:

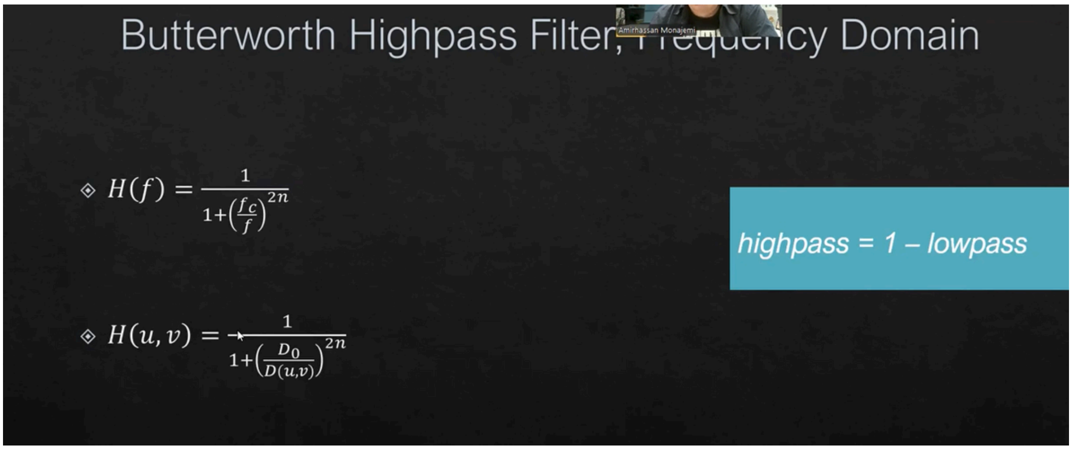
- $H(u, v)$ is the filter's transfer function at frequencies u and v, corresponding to the two spatial frequency dimensions in an image.
- $D(u, v)$ is the distance from the origin in the frequency domain to the point (u, v), typically calculated as $D(u, v) = \sqrt{u^2 + v^2}$.
- D_0 is the cutoff frequency, similar to f_c in the 1D case.
- n is the order of the filter, controlling the steepness of the roll-off at the cutoff frequency.

Look at how if D(u, v) < Do then high value.

But as soon as D(u, v) > Do, then quickly denominator explodes. Explosion is controlled by roll-off number n (order). Higher the order, higher the fall.



Butterworth highpass



But note that for ideal low pass butterworth, there is no gradual decay of magnitude, but a SHARP cutoff.

Why divide by 256^2?

For the *unnormalized* Hadamard matrix H of size N:

1. $HH = N\,I$.
2. Forward 2D transform:

$$A = H\,a\,H.$$

3. Inverse then is

$$a = \frac{1}{N^2}\,H\,A\,H,$$

since

$$H\,A\,H = H\,(H\,a\,H)\,H = (H\,H)\,a\,(H\,H) = N\,I\,a\,N\,I = N^2\,a.$$

So in code you divide by N^2. For a 256×256 image, $N = 256$, hence `…/256**2`.

Inverse via the matrix inverse

1. **General inverse formula**
   If your forward was

$$A = H\,a\,H$$

   then the inverse is

$$a = H^{-1}\,A\,H^{-1}.$$

2. **Hadamard's special inverse**
   For the (unnormalized) Hadamard matrix $H \in \{+1,-1\}^{N\times N}$, one can show

$$H\,H = N\,I \quad\Longrightarrow\quad H^{-1} = \frac{1}{N}\,H.$$

   Moreover H is symmetric so $H^T = H$.

3. **Putting it together**

$$a = H^{-1}\,A\,H^{-1} = \left(\tfrac{1}{N}H\right)A\left(\tfrac{1}{N}H\right) = \frac{1}{N^2}\,H\,A\,H.$$

   Hence your code's division by `256**2` is exactly the two factors of $1/N$ coming from each inverse.

What is a normalized hadamard matrix

A **normalized** Hadamard matrix is just the usual $\{\pm 1\}$ Hadamard matrix H rescaled so that its rows (and columns) become orthonormal rather than orthogonal. Concretely, if H is an $N \times N$ Hadamard (so that $H\,H^T = N\,I$), then you define

$$H_n \;=\; \frac{1}{\sqrt{N}}\,H.$$

Key properties of H_n

- **Orthonormality:**

$$H_n\,H_n^T \;=\; \frac{1}{N}\,H\,H^T \;=\; I.$$

- **Symmetric and involutory:**

$$H_n^T = H_n, \quad H_n^{-1} = H_n.$$

Why normalize?

1. **Simplifies inversion.**
   With the un-normalized H you need

$$A = H\,a\,H \quad\Longrightarrow\quad a = \frac{1}{N^2}\,H\,A\,H.$$

   With H_n you get

$$A = H_n\,a\,H_n \quad\Longrightarrow\quad a = H_n\,A\,H_n$$

   with no extra $1/N^2$ factor.
2. **Unit-energy basis.**
   Each row of H_n has Euclidean norm 1, so your transform becomes an orthonormal change of basis—very handy for energy-preserving signal analysis.
3. **Interpretability.**
   The DC coefficient $A_{0,0}$ now gives the *average* pixel value (rather than the raw sum), because projecting onto an all-ones unit vector divides by $\sqrt{N}$.

What are possible applications of representing an image using the first few walsh basis images?

- **Image Compression**
  - **Low-complexity codecs**

    The Walsh–Hadamard transform (WHT) uses only additions/subtractions, so it's extremely fast in hardware. Truncating to the first KKK low-sequency coefficients yields a coarse but compact representation—ideal for very low-power or real-time systems (e.g. embedded vision).
  - **Progressive coding**

    Send only the DC term and a handful of first-sequency basis images to give a quick "preview" of the frame; refine gradually by adding higher-sequency terms as bandwidth allows.
- **Denoising and Smoothing**
  - Noise in images tends to manifest in high-sequency (rapid sign-changes) coefficients. By zeroing out all but the first few Walsh coefficients, you perform a form of low-pass filtering that suppresses speckle or salt-and-pepper noise while keeping the large-scale structure intact.
- **Watermarking & Steganography**
  - Embedding a watermark in mid- or low-sequency Walsh coefficients makes it robust to JPEG-style compression (which also retains low frequencies) but less perceptible to the human eye. You can add or modify a small offset in a few chosen basis coefficients to carry hidden data.
- **Feature Extraction for Recognition/Classification**

- Projecting image patches onto the first few Walsh functions gives you a low-dimensional feature vector that captures bulk texture/brightness patterns. These “Walsh features” have been used successfully for face recognition, fingerprint matching, and texture classification because they’re rotation- and scale-invariant to a degree, and very fast to compute.

- **Template Matching & Fast Correlation**
  - Because convolution in the Walsh domain reduces to element-wise products, you can rapidly correlate a template against an image by transforming both to sequency order, multiplying only the first few coefficients (for a coarse match), and inversely transforming. This accelerates object detection or motion tracking in video.
- **Compressed Sensing & Sparse Sampling**
  - The Walsh basis is incoherent with many natural-image bases. If you know an image is sparse in the Walsh domain, you can randomly sample a small set of pixel measurements and reconstruct it by solving a sparse-recovery problem—keeping only the largest few Walsh coefficients.
- **Optical & Holographic Processing**
  - In optical computing, lenses can implement the WHT natively. Truncating to low-sequency patterns lets you perform rapid, analog smoothing, correlation, or pattern recognition directly in hardware without digital post-processing.
- **Progressive Rendering in Graphics**
  - Rather than rasterizing every pixel at full resolution, a renderer can project frames onto low-sequency Walsh patterns first to quickly display a blurred version, then refine it. This improves interactivity in remote or VR streaming scenarios.