Packet Compressed Sensing Imaging (PCSI)

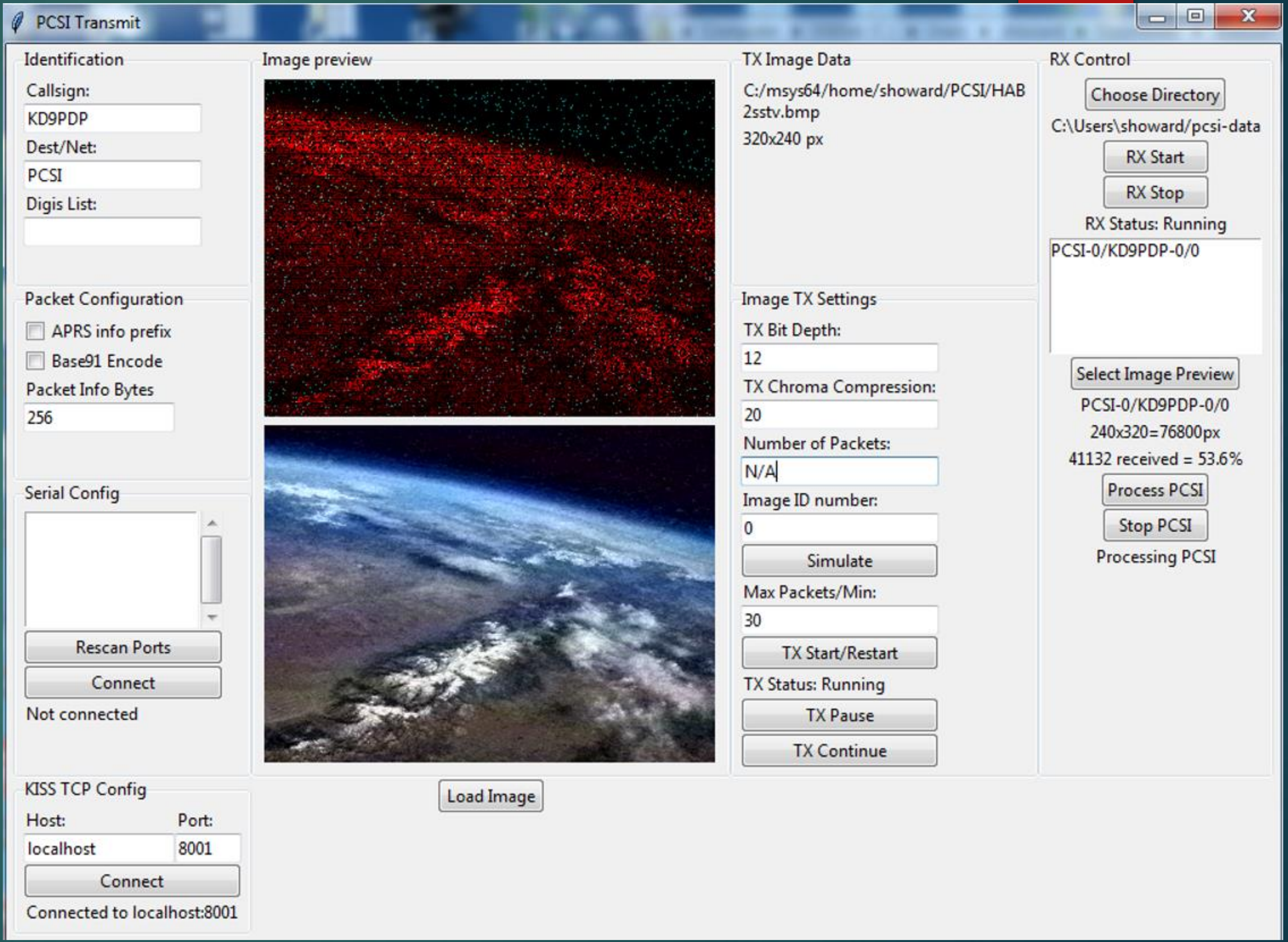
SCOTT HOWARD (KD9PDP), GRANT BARTHELMES, CARA RAVASIO, LISA HUANG, BENJAMIN POAG, & VARUN MANNAM

DEPARTMENT OF ELECTRICAL ENGINEERING
UNIVERSITY OF NOTRE DAME

THIS WORK THIS IS LICENSED CC-BY-SA 4.0

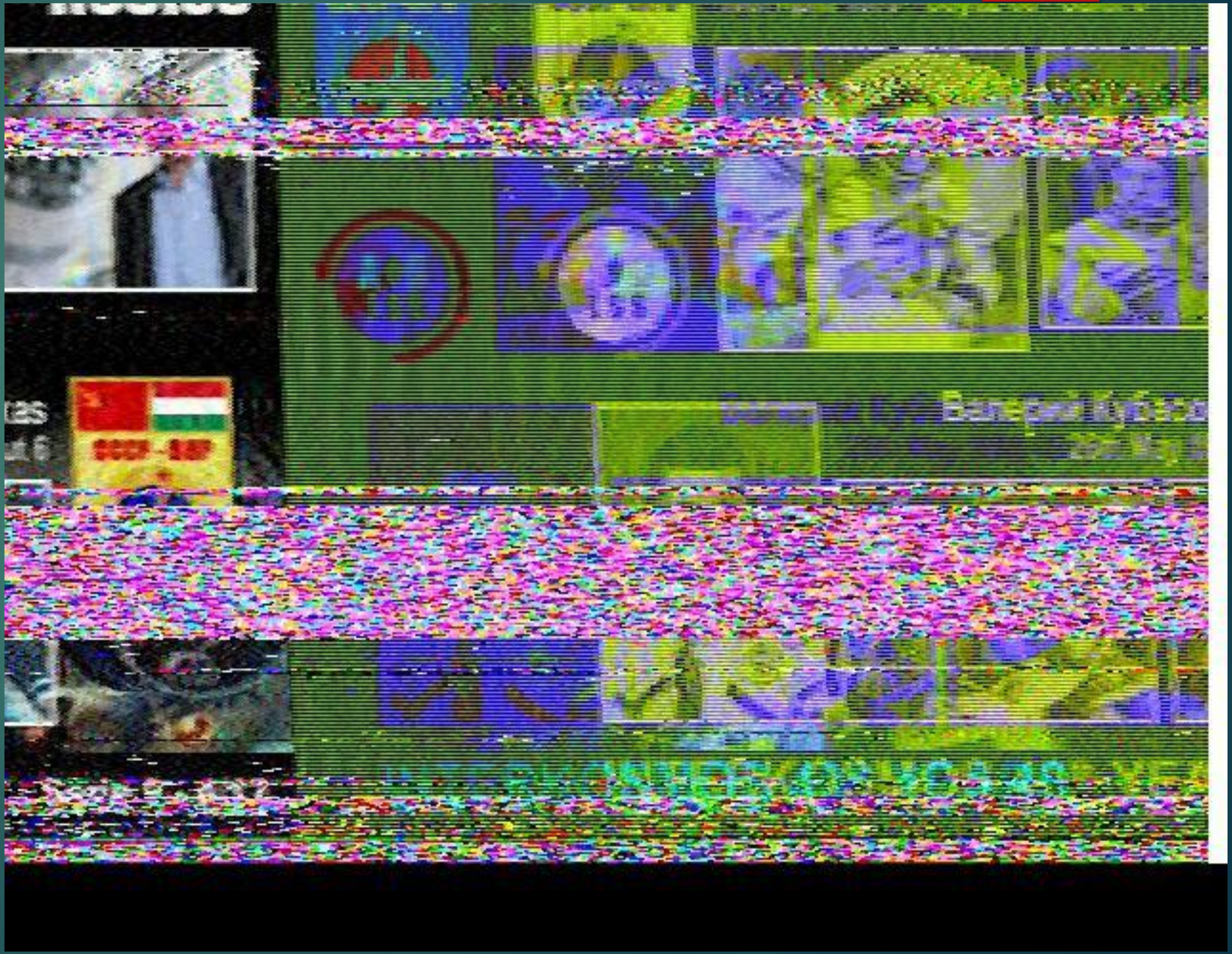
What is PCSI?

- Image Transmission
  - Digital Packets
- Unconnected Networks
  - Multiple receive at once
- Receive Full Image regardless of packet loss!
  - Images look good with 80-70% packet loss!
  - Image to right: ~45% loss!
  - Miss beginning/end – it's ok!
- Any band or mode
  - Software exists for any KISS TNC (hardware/software)
- Trivial for transmitter. The receiver does all the hard computing



Our Image Transmission Challenge

- Hard Environments:
  - Weak SNR
  - Fading/Signal Loss
  - Low compute power
  - Limited time windows
- Analog
  - e.g., SSTV
- Digital
  - e.g., SSDV



SSDV

- Encode image as JPEG
- Transmit in packets of JPEG minimum coded units (MCU)
- Optional forward error correction (FEC)
- Excellent results!
- Two issues:
  - Lost packets = blank parts of image
  - JPEG encoding + FEC hard in low-memory devices


https://ukhas.org.uk/guides:ssdv


SSDV image of Moon and Earth taken by Longjiang-2 / Lunar-OSCAR-94
Credit Cees Bassa

PCSI Examples

Original Image
(would require 914 packets)

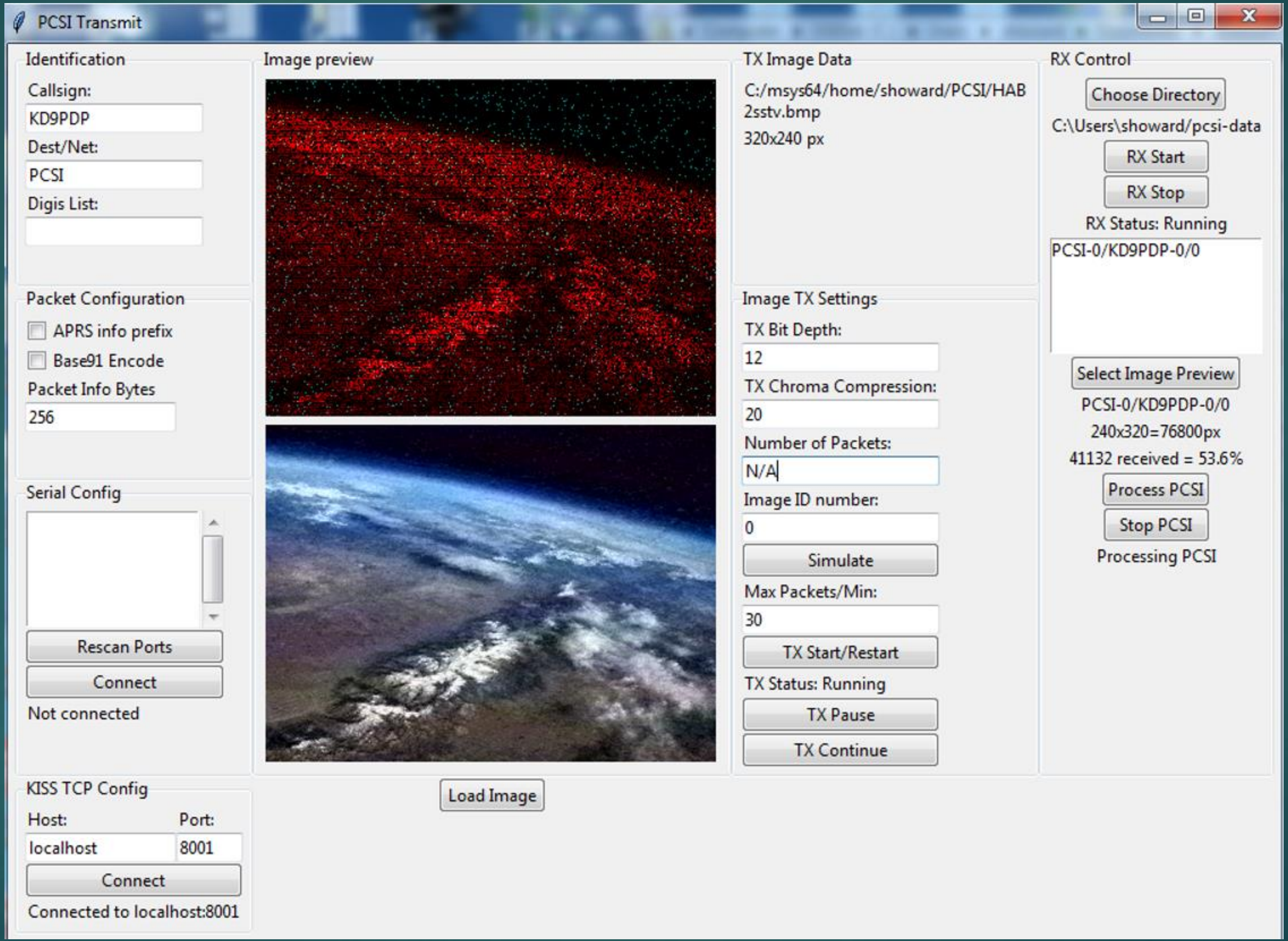
Only send 10 packets

Only send 30 packets

Only send 100 packets

Only send 300 packets

PCSI Demonstration



- https://maqifrnswa.github.io/PCSI/

How does PCSI work?

FOR MORE INFO ON “COMPRESSED SENSING IMAGING,” CHECK OUT:

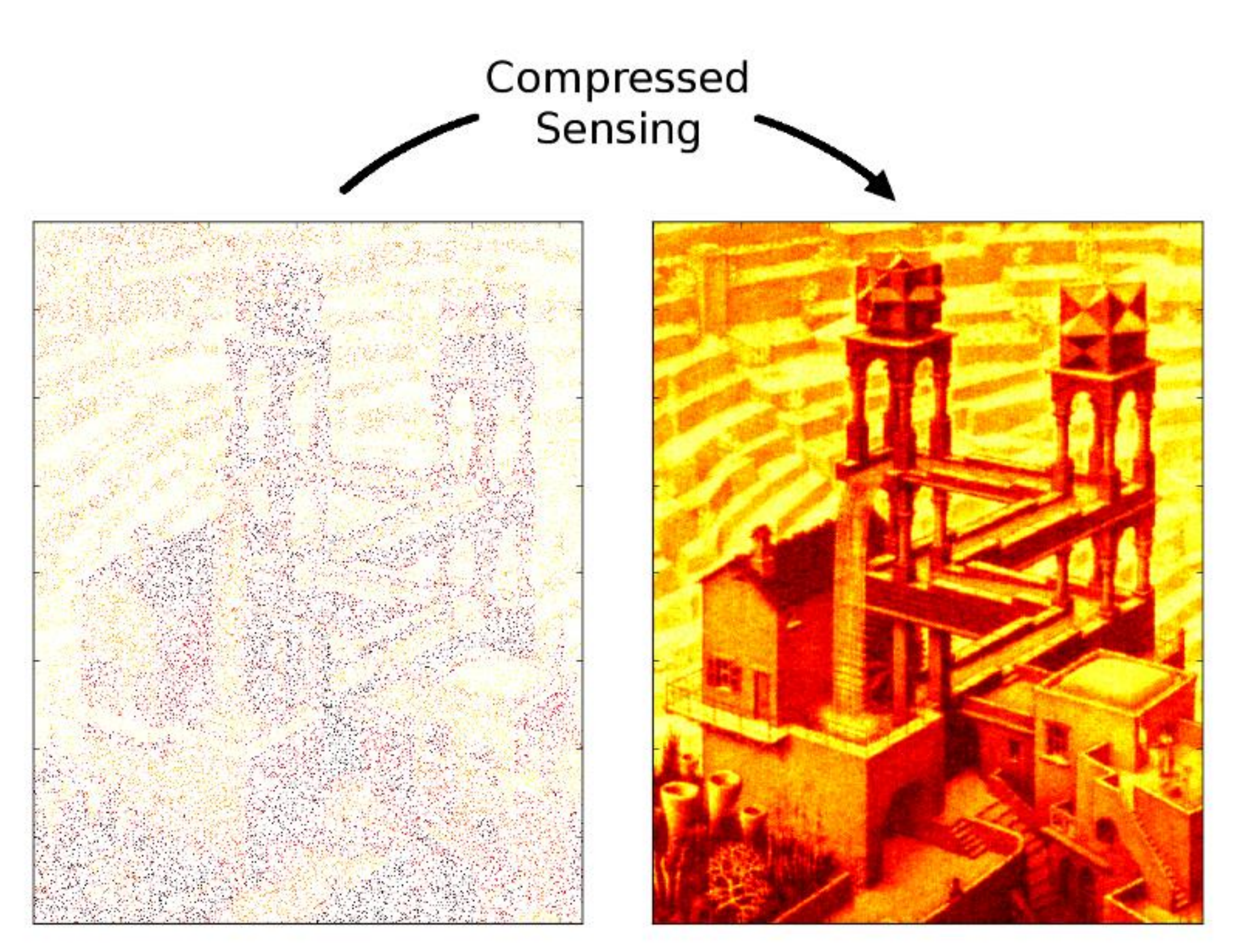
HTTP://WWW.PYRUNNER.COM/WEBLOG/2016/05/26/COMPRESSED-SENSING-PYTHON/

Compressed Sensing in Biology

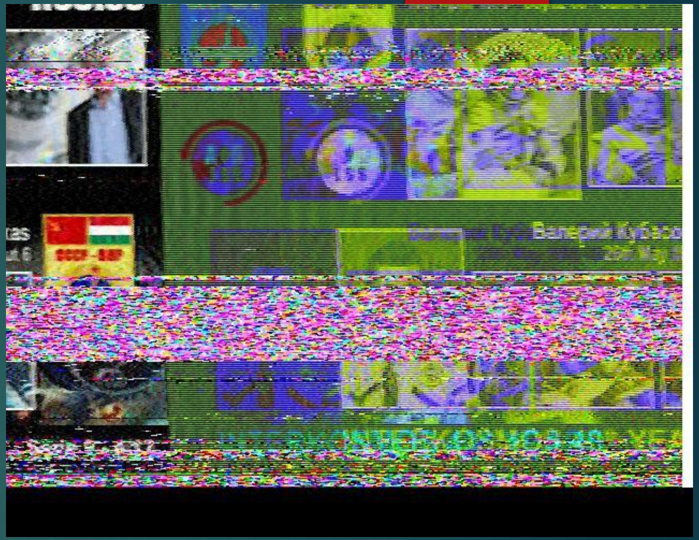
- How do our eyes work?
  - 120 million rods (greyscale pixels), 6 million cones (color pixels)
  - Humans Perceive ~24 bit color (16mil colors) and ~10bit grey (1000 levels)
  - Humans perceive ~100 frames per second
- What is the needed bandwidth?
  - (120million*24 bits + 6million*10)*100 = 294 Gb (gigabits) per second
  - Optic nerve can carry less than 10 Mb (megabit) per second (Koch, et. al., "How Much the Eye Tells the Brain" *Current Biology*, 2006)
  - **IT'S 30,000 times too slow!!! How can it be??!?**

Compressed Sensing in Biology

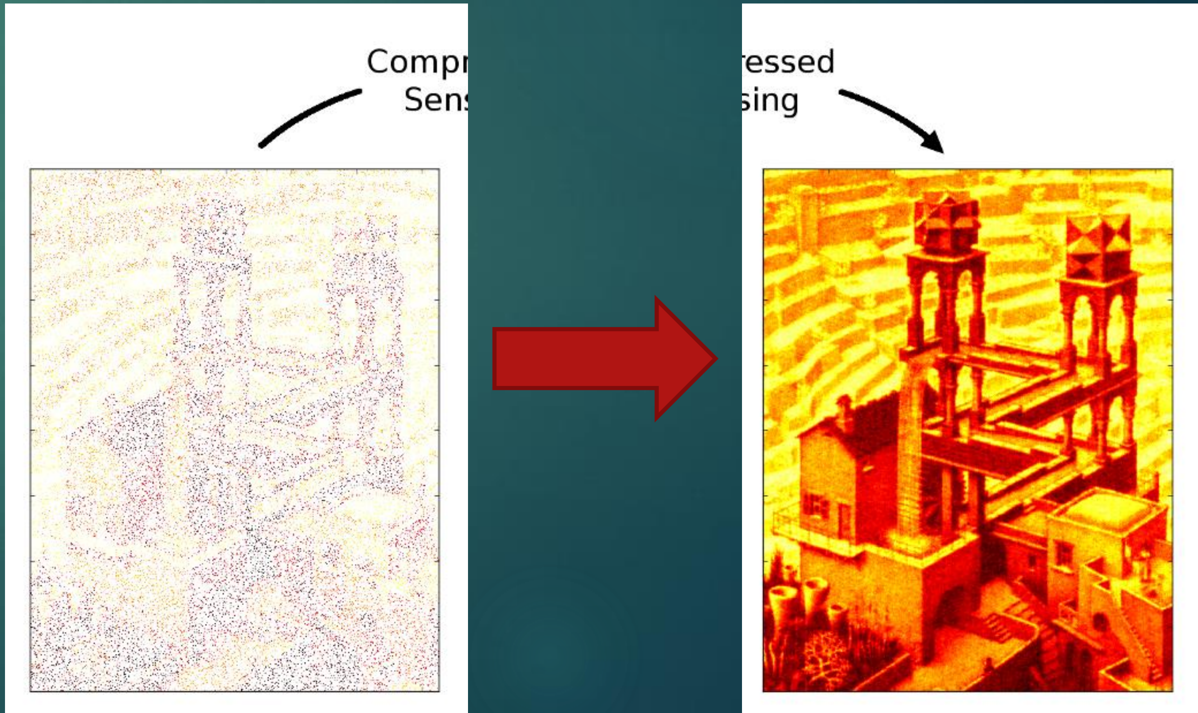
- Our eye doesn't have to transfer all the imaging data, just enough for our brain to reconstruct what is there (or perceives is there)
- Some amount of neural network processing: pattern recognition (sends "shapes" not "pixels"), only transmit changes, etc.
- Can we use similar tricks to transmit imaging data?
  - Just transmit pixels on left and reconstruct image on right



Why this matters:



- If we have imaging data in one location, and we want to broadcast it, you can:
  - Send it "analog" (e.g., SSTV from the ISS)
  - Send it digitally (in packets)
    - need two way communication to handle lost packets (request resend)
    - Compression need memory, power at transmitting end
    - If transmitting without acknowledgements, data is lost – corrupting or losing the image
  - What about digital packets that can be lost?
    - Do we even need to send everything?

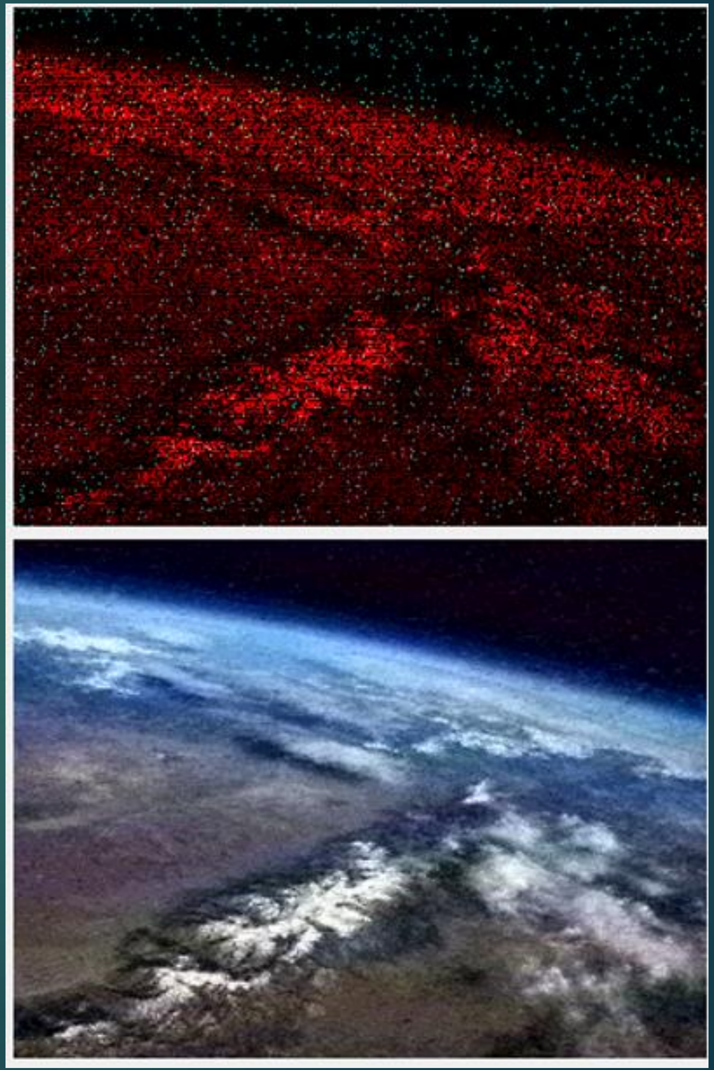


Do we even need to transmit all the data?

- If your image is "sparse," you don't need to send every pixel!
- See paper for details and references

Random Pixels in the Packet

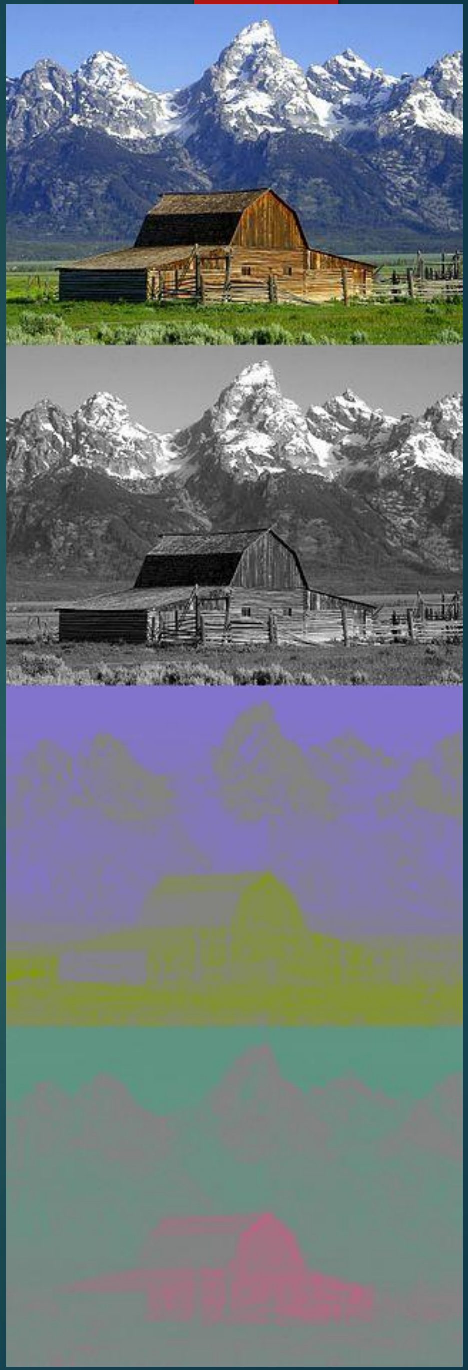
- Need to send random pixels each packet – but needs to be deterministic.
- Pseudorandom Number Generators are deterministic and sort-of-random!
- Each packet contains data from randomly selected pixels and includes the "packet id number" in the header. From the packet id number, the receiver can deterministically figure out which pixels are present in the packet!



Another Trick: Chroma Compression

- Human eye: 120 million rods (greyscale pixels), 6 million cones (color pixels)
- Do we need to send high resolution color?
- JPEG uses chroma compression (up to 4x)
- Chroma Compression in PCSI:
  - Represent image as YCbCr instead of RGB
  - Send Y data (e.g., B&W) for all pixels
  - Send only a fraction of Cb and Cr (color) pixels
  - Reconstruct each channel separately.
  - PCSI Chroma Compression of 20 works well! (only send 5% of the pixels as color, the rest as black and white!)
  - Optional down size from 24 bit color to lower (e.g., 12 bit)

https://en.wikipedia.org/wiki/YCbCr



Packet Framing

- Any framing works! All that matters in the payload format (see paper for spec)
- Currently implemented: AX.25
  - Existing KISS hardware and software
  - Compatible with APRS hardware/software – but not a good idea to spam APRS channels
  - Optional digipeating
- Future framing: SSDV-style Framing
  - Less overhead, better for higher BER environments

PCSI Transmit Steps

- Use a PRNG to select random pixels from the image
  - Chroma compression: select pixels to be YCbCr and others just Y
- Convert to RGB to YCbCr. Downscale bit depth
- Create Pseudorandom Datagram Payload (PDP) consisting of header w/ image info and pixels
  - Each PDP contains all the information you need to fully reconstruct an image. Each additional PDP received increases image quality.
- Frame it up!
  - Send it off to your TNC to handle flags, bit stuffing, CRC, etc.

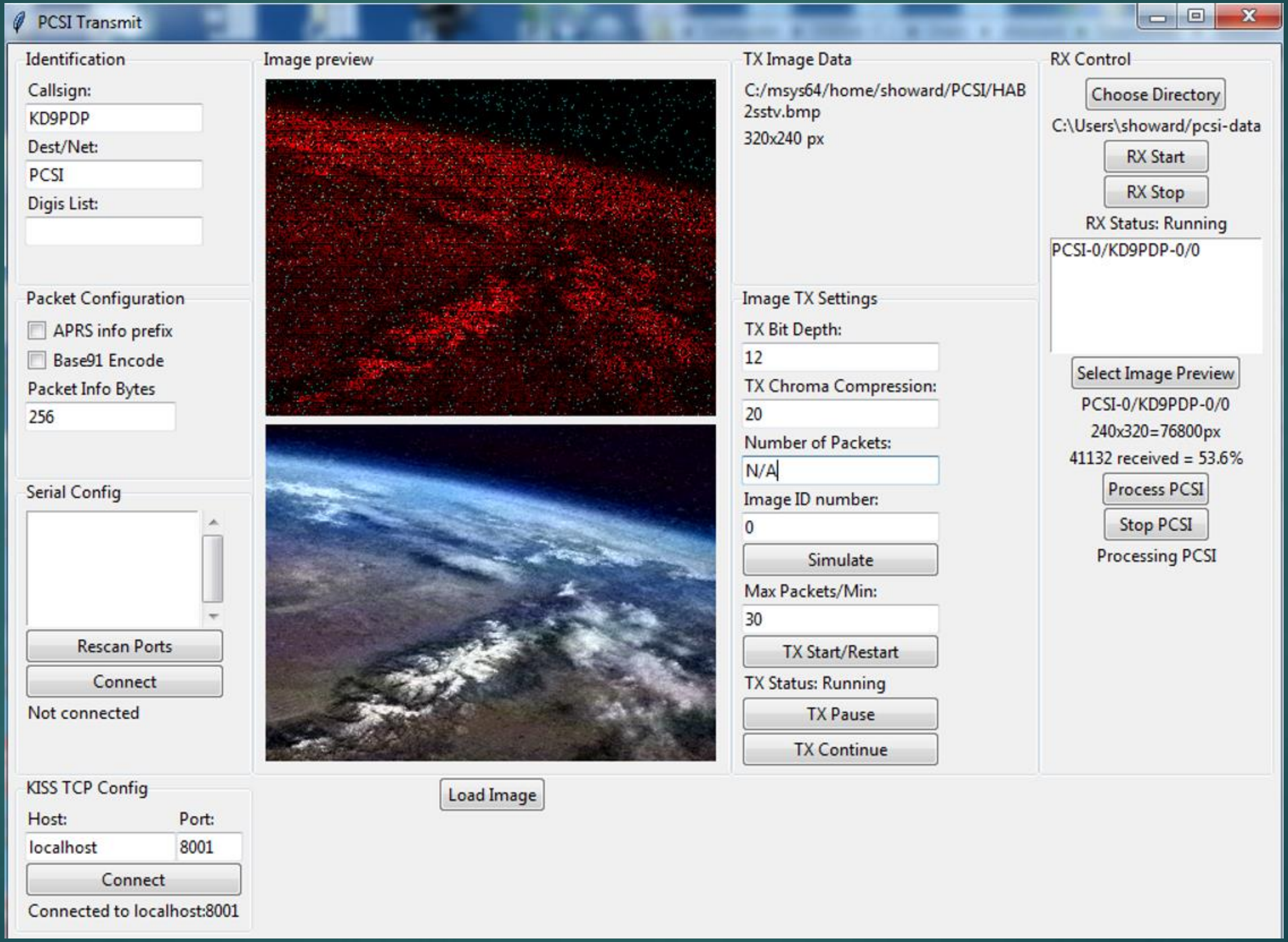
PCSI Receive Steps

- Extract header and pixel info
  - You now know image size, bit depth, etc. And you have the locations and values of the full color (YCbCr) and B&W (Y) pixel channels
- Use a compressed sensing algorithm to make a best-guess as to what each channel originally looked like, individually.
  - We used: OWL-QN variant of L-BFGS (see paper) to reconstruct a sparse DCT spectrum. This is the only computationally intense step of the process.
  - Combine reconstructed channels and convert back to RGB
  - Display the image!

Future of PCSI

- Arduino transmit client
- Centralized packet upload system (similar to SSDV's)
- APRS Frequency Objects format + protocols
- Cell phone apps!
  - Rob Riggs (WX9O) at Mobolinkd has been helping testing, we're starting a partnership developing PCSI phone apps

Thank you, any questions?



- https://maqifrnswa.github.io/PCSI/

Math of compressed sensing (more detailed)

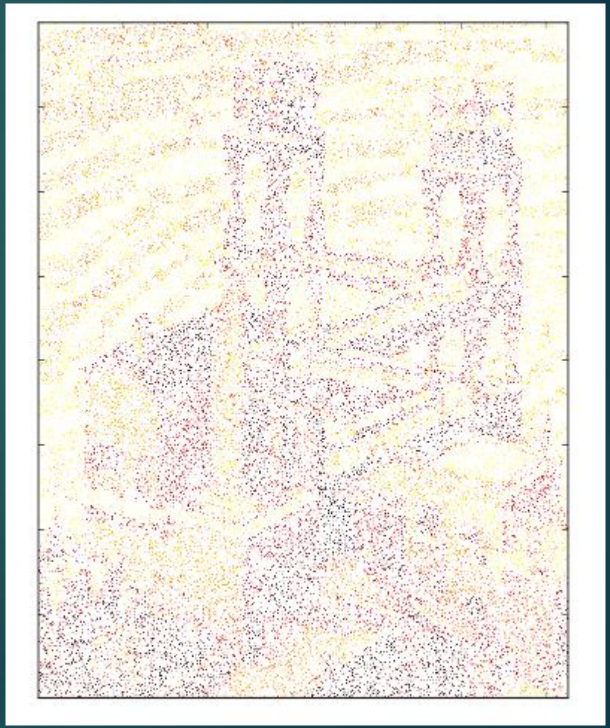
- If I have an image and only transmit a random subset of pixels, I'm doing this math:

$$\boldsymbol{Ax = b}$$

- "x" is a single vector that is a list of the value of each pixel from the original image
- "b" is a single vector that is a list of the pixel values received at the end of the transmission
- "A" is the "measurement" matrix (example next slide)

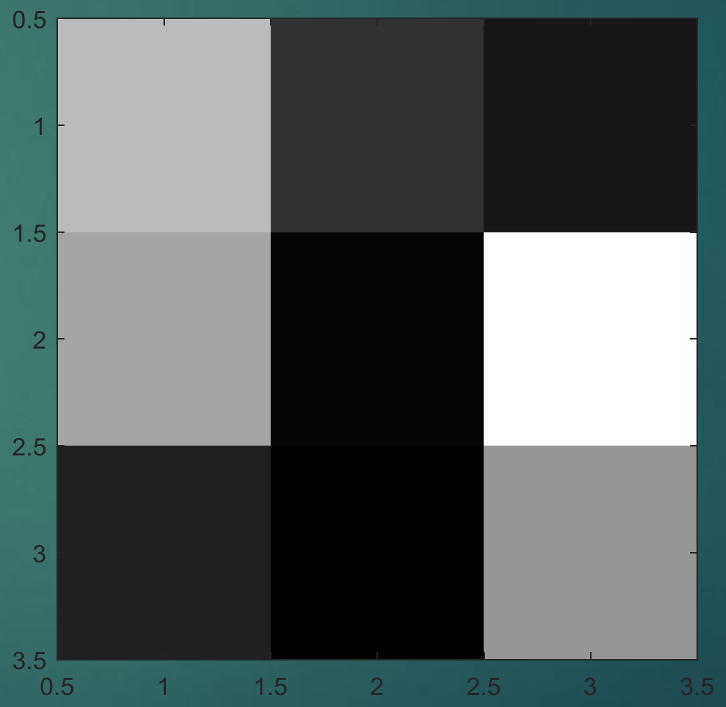
x:

b:



Math of compressed sensing (more detailed)

- Let's say we have a 3x3 image (9 pixels)
- $\boldsymbol{x_{2d}} = \begin{bmatrix} 42 & 12 & 6 \\ 37 & 2 & 57 \\ 8 & 1 & 34 \end{bmatrix}$
- Therefore
- $\boldsymbol{x} = \begin{bmatrix} 42 & 12 & 6 \\ 37 & 2 & 57 \\ 8 & 1 & 34 \end{bmatrix}_{\text{flatten}} = \begin{bmatrix} 42 \\ 12 \\ 6 \\ 37 \\ 2 \\ 57 \\ 8 \\ 1 \\ 34 \end{bmatrix}$



If we only transmit some pixels to b…

- Let's only transmit a few pixels (counting from the top, starting with 0):
  - First pixel 5 (**x**[5]), then 1 (**x**[1]), then 7 (**x**[7]) to **b**
- $$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 42 \\ 12 \\ 6 \\ 37 \\ 2 \\ 57 \\ 8 \\ 1 \\ 34 \end{bmatrix} = \begin{bmatrix} 57 \\ 12 \\ 1 \end{bmatrix}$$

"**b**" vector is what we receive!

$$\boldsymbol{Ax = b}$$

What if we are b? What do we know?

- Now let's just pretend we're **b**, and we see the pixels that were transmitted. Can we figure out what the original image is?
- $$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_{00} \\ x_{10} \\ x_{20} \\ x_{01} \\ x_{11} \\ x_{21} \\ x_{02} \\ x_{12} \\ x_{22} \end{bmatrix} = \begin{bmatrix} 57 \\ 12 \\ 1 \end{bmatrix}$$

$$\boldsymbol{Ax = b}$$

- Too many unknowns!!
  - Need more constraints!

What if we are b? What else do we know?

- Now let's just pretend we're **b**, and we see the pixels that were transmitted. Can we figure out what the original image is?
  - Too many unknowns!!
  - But we do know that, for the most part, if I take a Fourier transform of most signals (and images), most of the frequencies are zero.
  - If I know that most of values of the frequency space is zero, then I might have enough measurements to solve my problem!
  - NOTE: instead of Fourier transform, people use a "Discrete Cosine Transform" (DCT) which is very similar. Maybe we'll get to details later, for now – it's just like an FFT but it is only real (no phase)

$$\boldsymbol{Ax = b}$$

What if we are b? What else do we know?

- So let's transform our equation in to the frequency domain. "x" is the list of pixel values, "X" is the DCT spatial frequencies

$$\mathrm{dct}(\boldsymbol{x}) = \boldsymbol{X}$$

$$\mathrm{idct}(\boldsymbol{X}) = \boldsymbol{x}$$

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

$$\boldsymbol{A}\ \mathrm{idct}(\boldsymbol{X}) = \boldsymbol{b}$$

- We know A and b, so we want to find "X" in: $\boldsymbol{A}\ \mathrm{idct}(\boldsymbol{X}) - b = 0$
  - We want to solve it using the "X" that has the fewest NONZERO elements as possible (most of X are zeros)

How to find the right "X"

- "Minimizing number of non-zeros" is hard for a computer to do
  - It's called an "L0 norm"
- However, people found that if you instead minimize the "L1 norm," you pretty much get the same answer.
  - Minimizing the **L1 norm** means just minimizing the sum of the abs of each element: $\sum|X_k|$
- OK, so in the end, what we want:
  - Find ("guess") the "frequency components" **X** that minimizes the error between what we received and what we guess AND minimizes the "L1 norm"

$$\textbf{L1 norm} = \|\boldsymbol{X}\|_1 = \sum|X_k|$$

$$\textbf{error} = \sum|\boldsymbol{A}\ \operatorname{idct}(\boldsymbol{X}) - \boldsymbol{b}\ |^2$$

How we "guess" the right **X**!

$$\text{Minimize } \boldsymbol{X} = \sum |\boldsymbol{A}\ \text{idct}(\boldsymbol{X}) - \boldsymbol{b}\,|^2 + \boldsymbol{C} \sum |X_k|$$

- Lots of math, but the above is easy! Programs already written to do that problem!
  - https://en.wikipedia.org/wiki/Limited-memory_BFGS#OWL-QN
  - http://www.pyrunner.com/weblog/2016/05/26/compressed-sensing-python/
  - Python: https://bitbucket.org/rtaylor/pylbfgs
  - C: http://www.chokkan.org/software/liblbfgs/
- What do you need to do it?
  - **"b"** – a vector of the pixels we received and we know the value
  - "C" – a "weighting" coefficient for how much we want to "prioritize" finding a solution has the least non-zero values in **X**. C is typically between 3 and 5.
  - The program will just find us "**X**" that fits the above equation!

Our example:

- Find the DCT coefficients (X_k) that minimize below:
- $$\sum \left\| \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \operatorname{idct} \begin{bmatrix} X_{00} & \cdots & X_{02} \\ \vdots & \ddots & \vdots \\ X_{20} & \cdots & X_{22} \end{bmatrix}_{\text{flatten}} - \begin{bmatrix} 57 \\ 12 \\ 1 \end{bmatrix} \right\|^2 - \sum |X_k|$$
- We set this up, and let the computer guess **X** for us. The computer then asks us to find what is the value of the function above, we tell it, and I makes another guess.
- We also can tell it the derivative to make it faster.