# RAISR

Rapid and Accurate Image Super Resolution

Nearest Neighbor

Bilinear

Nearest Neighbor

Bilinear

Cubic

Lanczos4

Cubic

Lanczos4

# hqx -- Pixel Art Scaling

- Intended for simple 2D art

Nearest Neighbor

- Use patch around each pixel
- Neighbors are “close” or “distant”
- Lookup neighborhood in LUT
- Determine filter

hq3x

Not intended for photographs!

Not intended for photographs!

Nearest Neighbor

hq4x

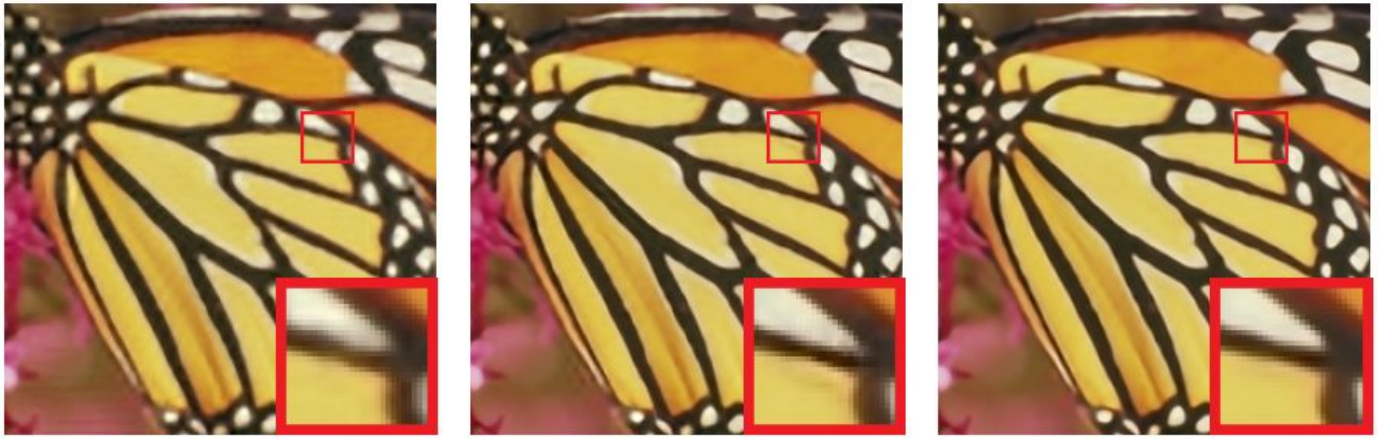
# Example-Based Approaches

- Build a dictionary from low resolution (LR) patches to high resolution (HR) patches
  - Dictionary is very large, so sparse coding is typically used
  - Usually limited to fixed upscaling factors

- Anchored Neighborhood Regression (ANR) and A+
  - Uses neighbor embedding and ridge regression
  - Somewhat fast

- SRCNN
  - End-to-end approach with a convolutional neural network (CNN)
  - Slow

(a) High-res (b) A+ (c) SRCNN



(d) JSB-NE (e) CSCN (f) DEGREE

# RAISR -- History

- Developed by Google; paper published in 2017
- Intended for mobile devices
- Lightweight, easy to parallelize, low memory requirements
- Performs roughly as well as SRCNN

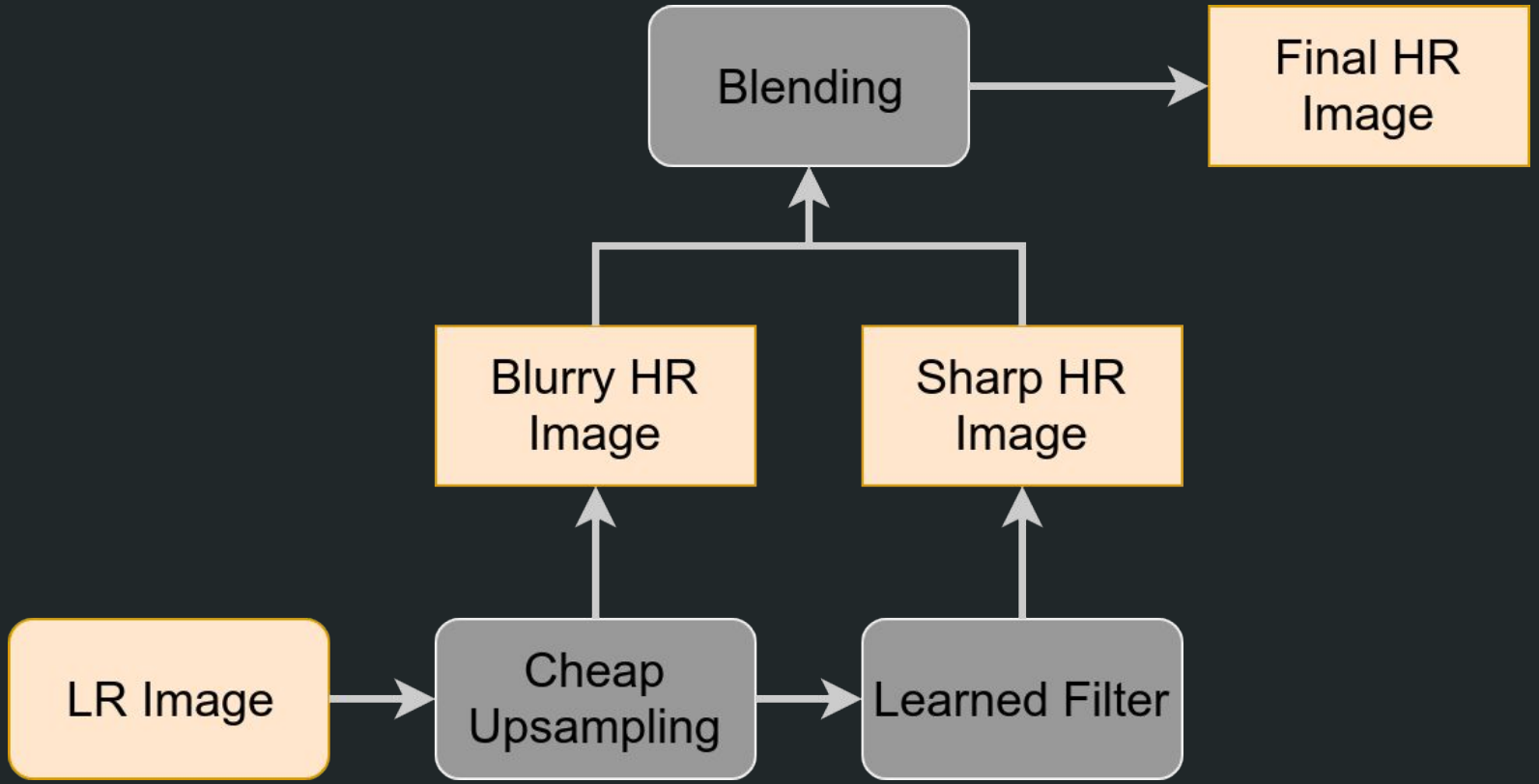
# RAISR -- Overview

- Use a cheap filter to scale the image up

- Sharpen the image using pre-learned filters
  - The filter to use depends on the neighborhood of the pixel

- Perform a blending pass to remove undesirable oversharpening effects

# RAISR -- Overview

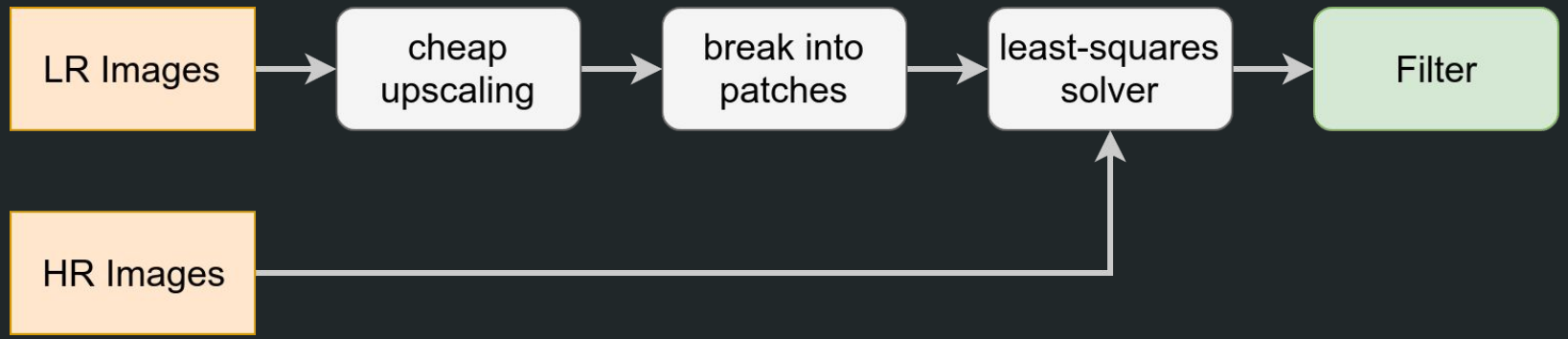


Nearest Neighbor

Bicubic Upsampling

RAISR Upsampling

# Learning Phase



# Objective Function

- Goal: A database of filters that minimizes the Euclidean distance between the **generated HR images** $x_i$ and the **training HR images** $y_i$.
- Minimize

$$\sum_{i=1}^{L} ||A_i h - b_i||_2^2$$

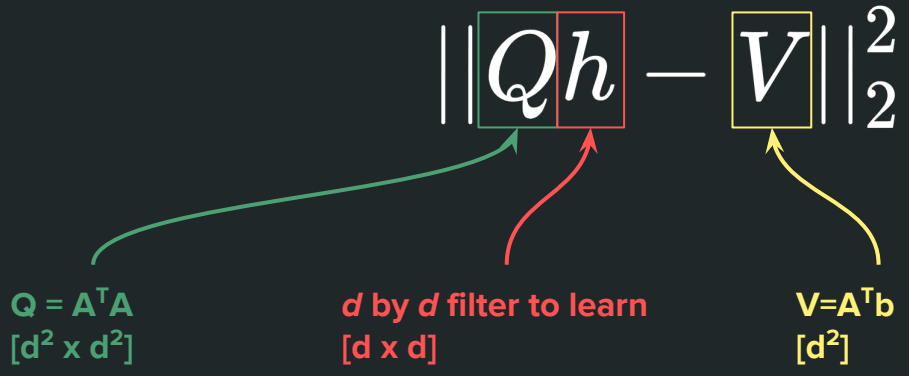
**All *d* by *d* patches in $y_i$ [M N x $d^2$]**

***d* by *d* filter to learn [d x d]**

**All pixels in $x_i$, correspond to center of $y_i$ patches [M N]**

# Actual Objective Function

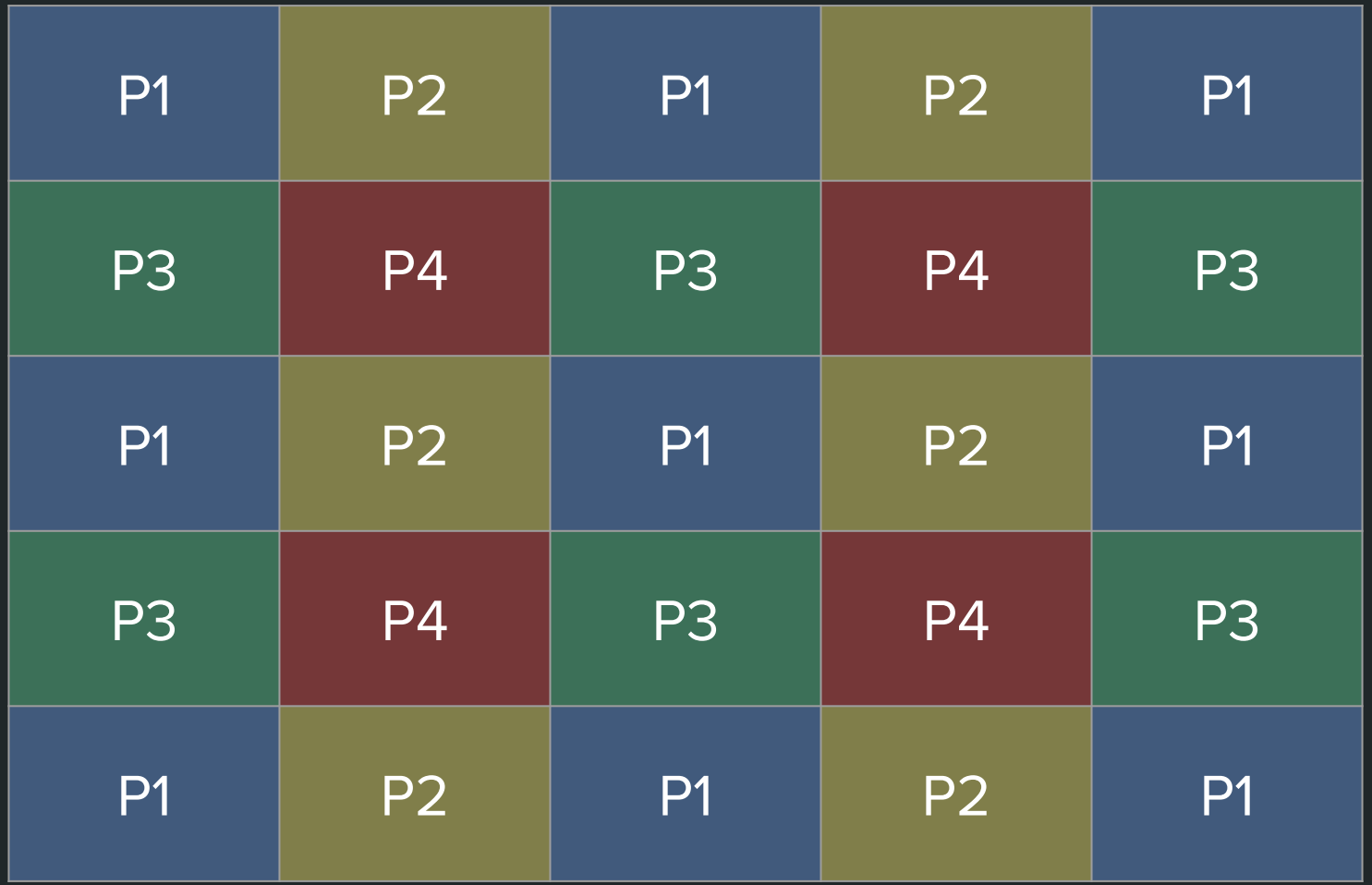
- *A* is a big matrix and we do not need all of the patches
- Much simpler computational/memory requirements in this form
- Minimize



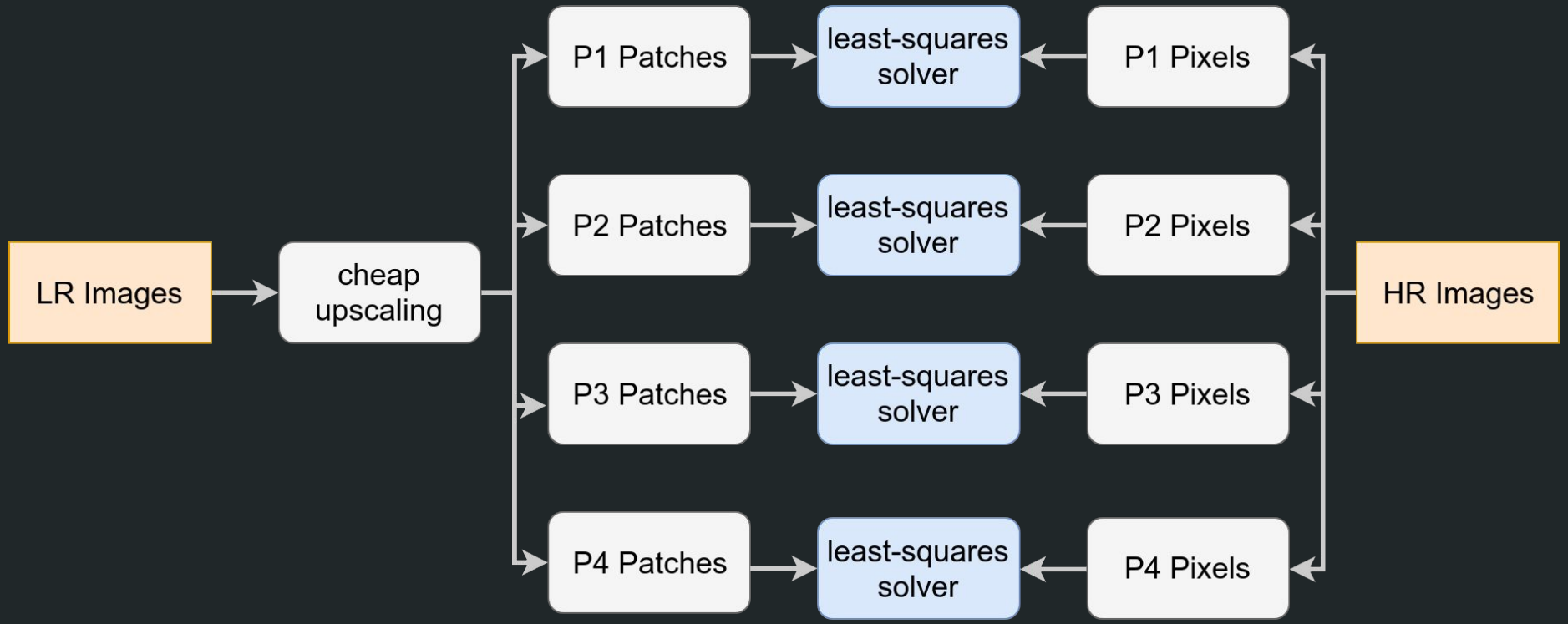

- Q is small!
- We don’t need to keep the whole image in memory

# Shift-invariance and Aliasing

- The cheap upscaler is not shift invariant
- 2x upsample mean 4 types of pixels; 3x upsample means 9 types of pixels; etc
- Separate filter for different types of pixel


P1
P2
P1
P2
P1
P3
P4
P3
P4
P3
P1
P2
P1
P2
P1
P3
P4
P3
P4
P3
P1
P2
P1
P2
P1

# Learning Phase (with Shift Invariance)



# Hashing-Based Learning

- Global image filtering is fast, but not very accurate
- Operate on the **patch** level to improve performance
- Instead of clustering, we ***hash*** the patch to get a bucket
- Now we compute each pixel filter for each bucket

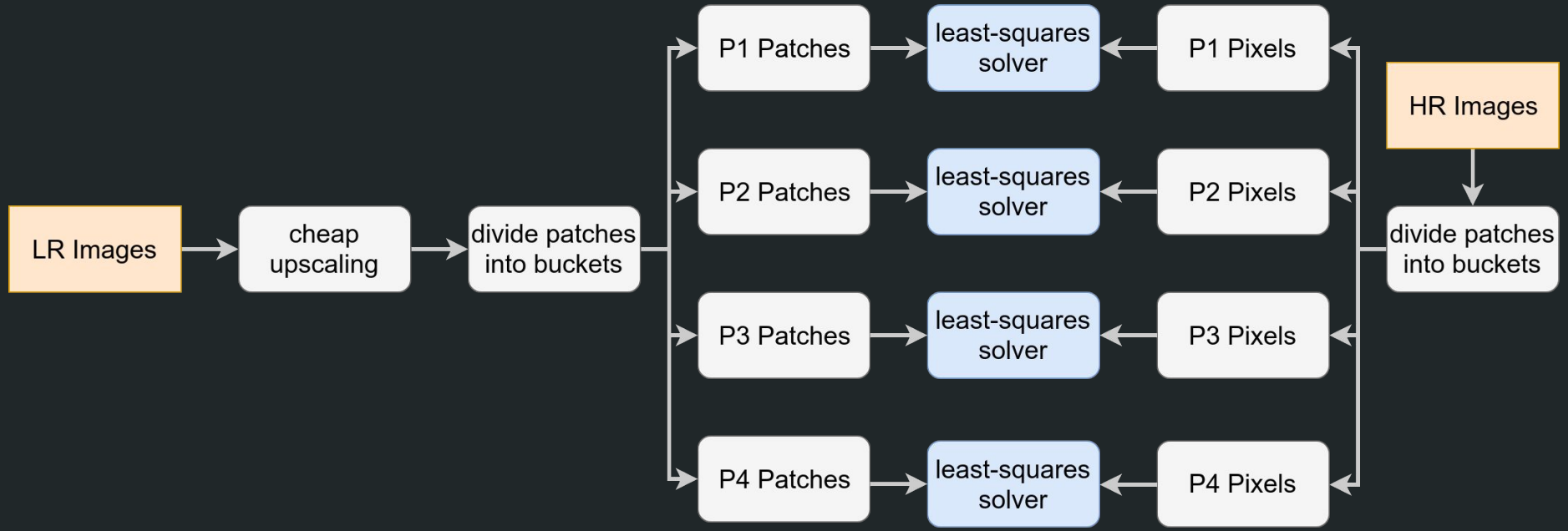
# Hashing-Based Learning

- A 9x9 neighborhood of the surrounding pixels is the hash key

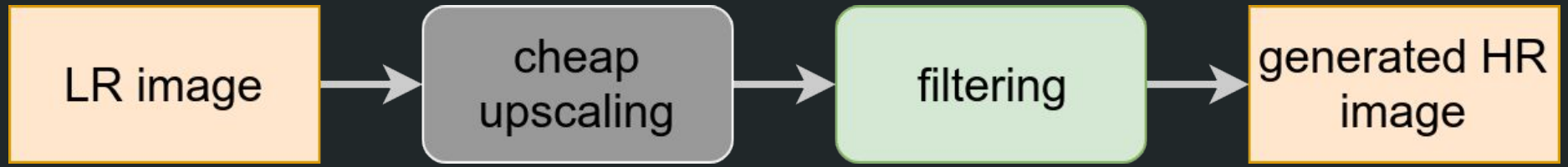
- Hash( *Neighborhood* ) = (θ, λ, μ)

- Patches are divided into buckets; solver operates on individual buckets

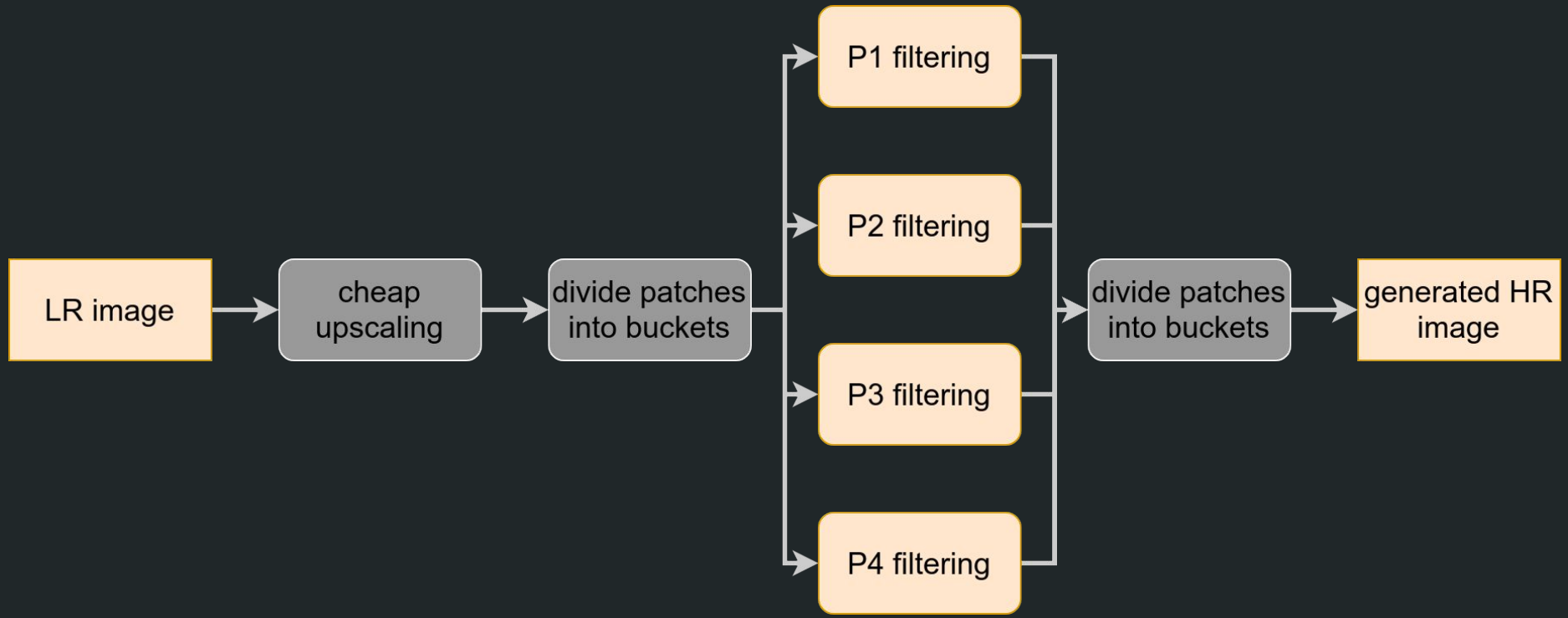
# Learning Phase (Hash-based with Shift Invariance)



# Inference Phase



# Inference Phase (Hash-based with Shift Invariance)



# Inference Phase

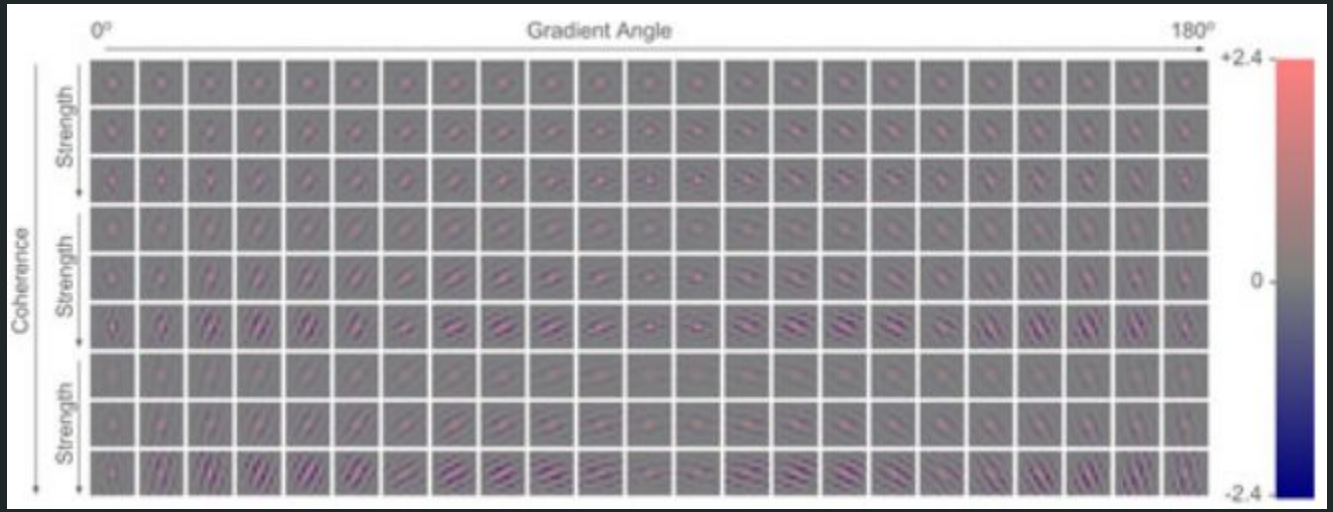
- Hash the neighborhood
- Find the bucket for that hash value
- Select the correct pixel filter from the bucket
- Apply the filter

- How does the hashing algorithm work?

# Hashing algorithm

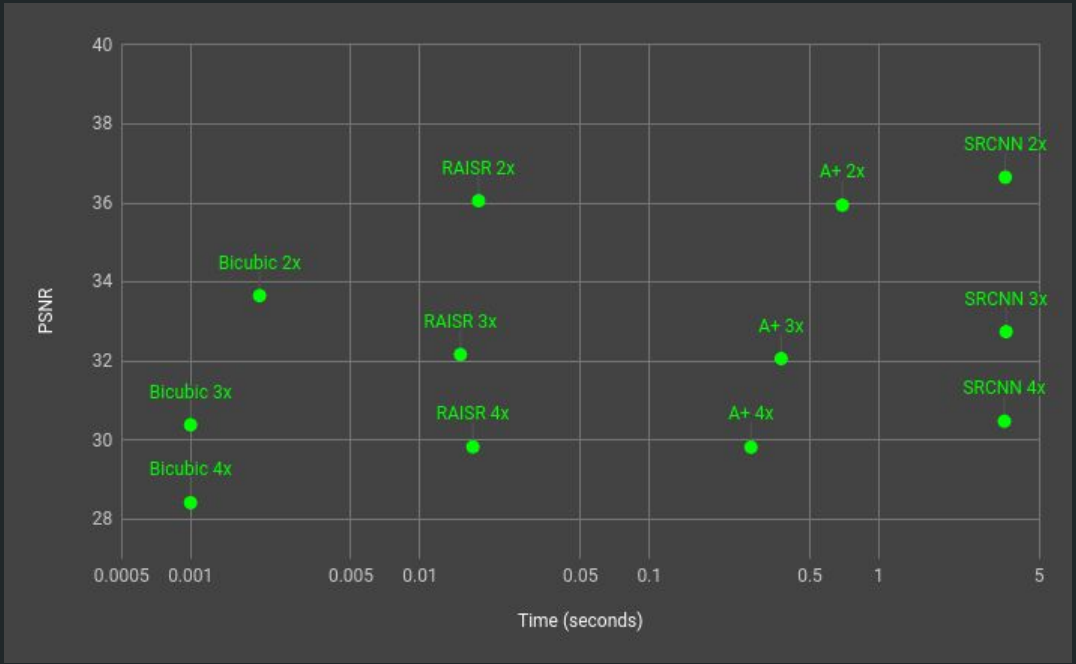
1. Collect a 9x9 neighborhood of the pixel
2. Compute the gradient of this patch
3. G is a [$n^2$ x 2] matrix where the rows are the gradients at each location
4. Compute $G^T WG$ where W is a normalized Gaussian kernel (applied separably)
    a. Note that this is a [2 x 2] matrix
5. Compute the SVD of this matrix
6. The **angle (θ)** parameter is the angle of the eigenvector corresponding to the larger singular value
7. The **strength (λ)** parameter is the square root of the larger eigenvalue.
8. The **coherence (μ)** (strength + spread) is $\frac{\sqrt{\lambda_1}-\sqrt{\lambda_2}}{\sqrt{\lambda_1}+\sqrt{\lambda_2}}$
9. Quantize gradient statistics to buckets

# Learned 2x Filters

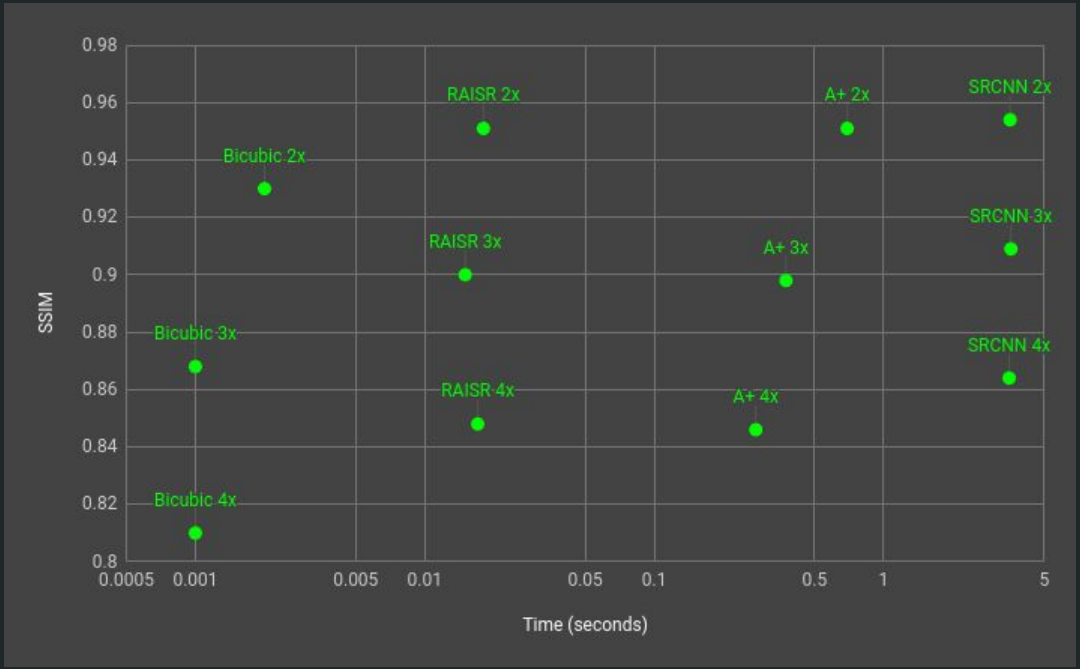


# (Almost) Final Steps

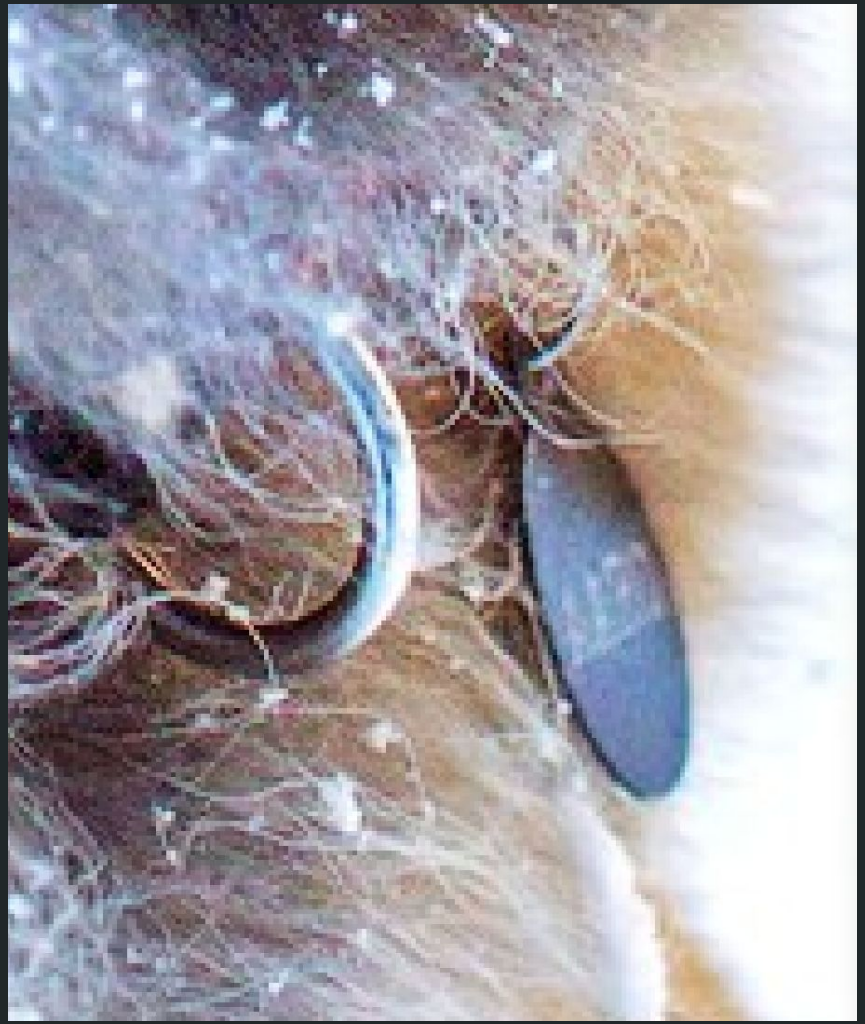
1. *FilterDatabase[θ, λ, μ, PixelType] = h*
    a. In the paper, filters were [11 x 11]
2. Apply the filter to the [11x11] neighborhood of the pixel
3. Add the (cheaply-scaled) chroma back in

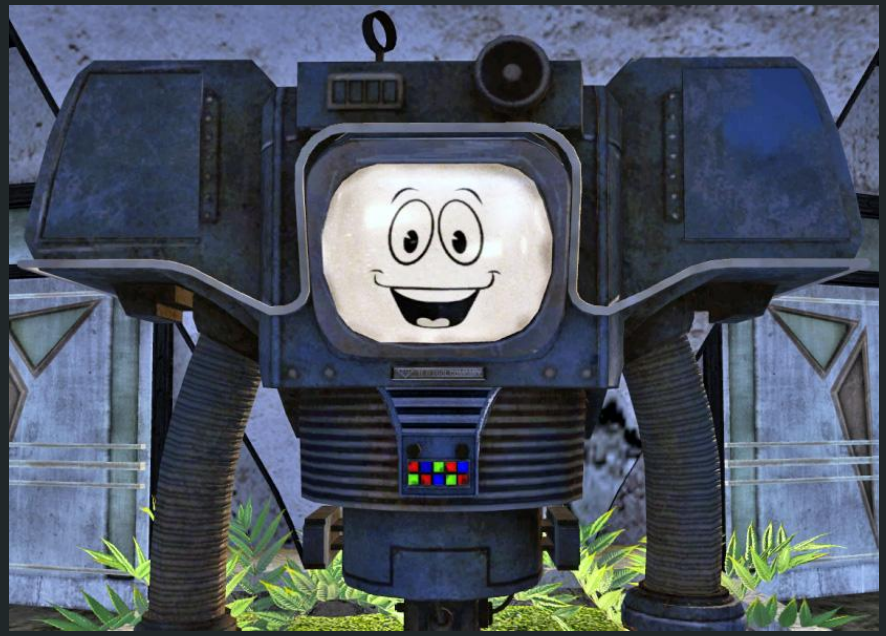


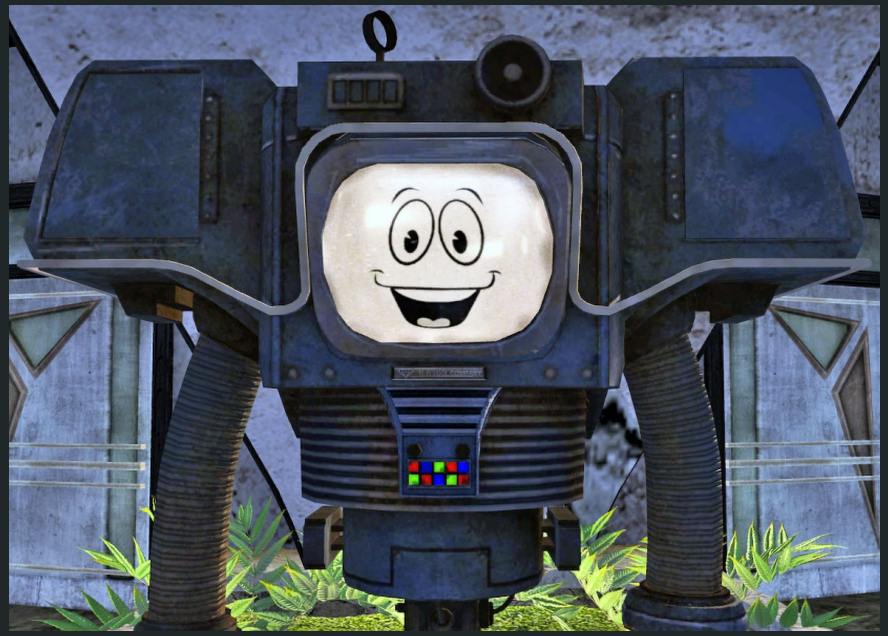

# Evaluation (PSNR) -- 3.4GHz 6-Core Xeon




# Evaluation (SSIM) -- 3.4GHz 6-Core Xeon







# Removing Artifacts

- So far RAISR worsens existing artifacts in image

- LR images are often distorted by an unknown PSF (e.g blur) or have JPEG compression artifacts

- Solution
  - To avoid oversharpening, train on LR images with ***compression artifacts*** to normal HR images
  - To avoid blurriness, train on normal LR images to ***oversharpened*** HR images.

# Removing Artifacts

Bilinear

Uncompressed LR

Compressed LR

# Avoiding Oversharpening

- Sadly this technique tends to amplify noise and create halos

- We want a cheap filter for **low**-frequency regions, learned filter for **high**-frequency regions

- Could cluster and handle areas differently, but that would be slow

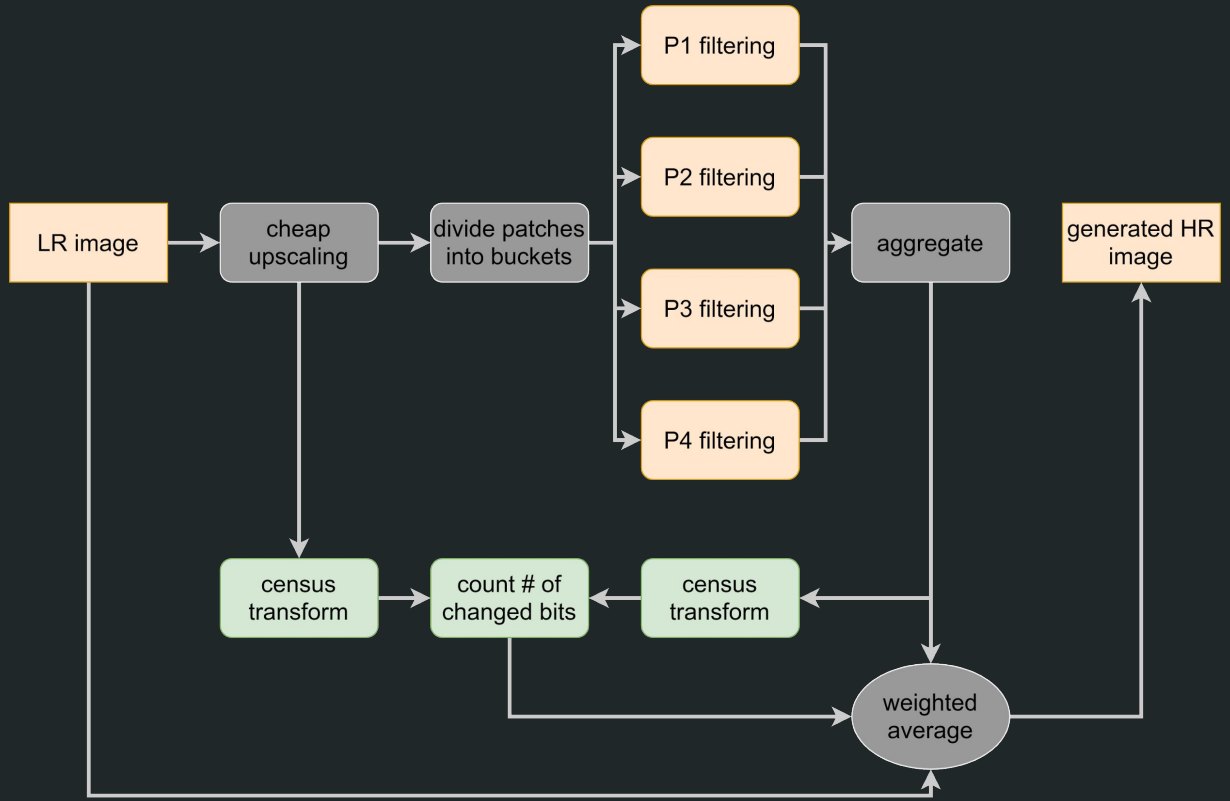
# Contrast Enhancement

- Need a measure of structure
- Least Connected Component (LCC) of the Census Transform (CT) could be used to detect high-frequency regions
- We can either selectively amplify high frequencies, or enhance a variety of frequencies

# Evaluating Local Change in Structure

- Compute the CT of the cheap upscaled image and the filtered image
- Per pixel, evaluate the Hamming distance
- Distance is proportional to the change in structure
- This weight is used for blending both images together

# Inference Phase (Everything + CE)



# Pre-processing for Contrast Enhancement

- Use a **DoG** sharpener on the HR **training** images to increase **contrast**

- Automatically enhances details (HF) and contrast (MF/LF)

Original Image (HR)

Bicubic upscaling (LR -> HR)

Filtered (**not trained** on sharpened HR)

Filtered (**trained** on sharpened HR)

Blended (with bicubic)

Original Image (once again)

# RAISR

- Lightweight
- Simpler and faster than Deep Learning techniques
- Works well for a variety of content -- games, photographs, video, etc

# Some ideas

- Applying the [11x11] filter is likely the most expensive operation (~121 MADs)
  - Why not use separable convolutions (~22 MADs)?

- 3D applications may have access to the **depth buffer**
  - Could be used in the hashing algorithm or the filter