Implementing Image Pyramids Efficiently in Software

Polymorphic Technologies

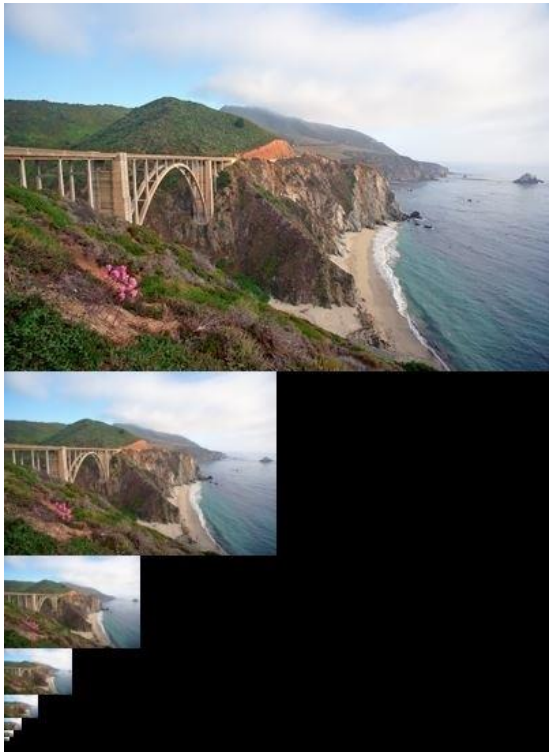
Michael Stewart

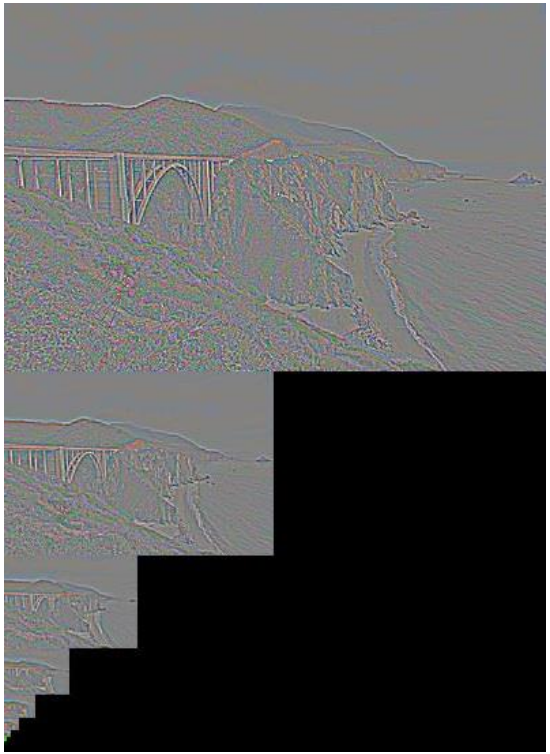
May 23, 2018

What Are Image Pyramids?

What Are Image Pyramids?

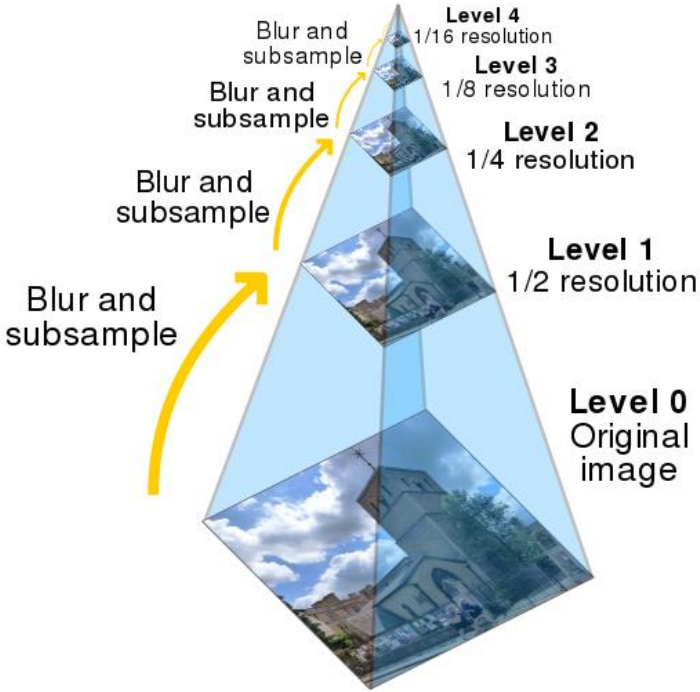
- Multi-scale image or signal representation
- Simplest is the Gaussian Pyramid (left)
- Most widely used is the Laplacian Pyramid (right)





Gaussian Image Pyramid

- Multi-scale copies of image
- Widely used as MipMaps starting with early GPUs for generating lower resolution views of images without aliased sampling artifacts
- Can be used as low pass filters for image blurring



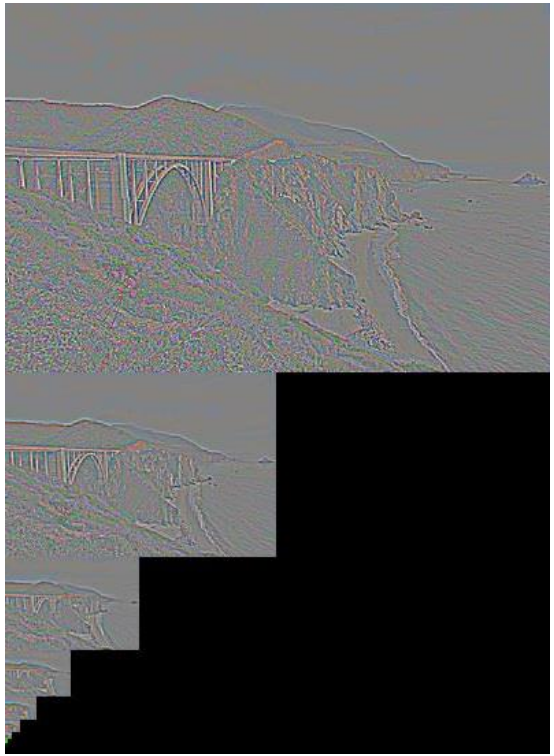
Wikipedia

Laplacian Image Pyramid

- Start with a Gaussian Pyramid
- At each resolution upsample the lower resolution and subtract from the current resolution and save the differences (differences plane).
- Lowest resolution is unsubtracted
  - If downsampled all the way to a single element (or pixel) then that represents the DC component
- Original image can be reconstructed from the Laplacian Pyramid
  - Starting from next to lowest resolution, upsample from lower resolution and add to differences all the way to the maximum resolution

Laplacian Image Pyramid Example

- Example computed in YUV color space
- Zero differences mapped to mid gray



Where Can They Be Used in Computer Vision?

Where Can They Be Used in Computer Vision?

- Noise reduction, especially to remove chroma mottle at high ISO
- Feature and object detection, pattern recognition
- Image segmentation
- Scale-space segmentation
- Optical Flow
- Stereo depth calculation
- Watershed transformation
- In-painting
- Focus estimation
- Interface

How Do They Work?

How Do They Work?

- Laplacian Pyramid performs frequency isolation in each difference plane
- Operations on a specific difference plane can be applied at that plane's spatial frequency only
- Allows operations to be performed differentially at different spatial frequencies
- Can be compared to a Guided Filter except with better, multi-frequency control
- Since the data is reduced by a factor of four at each lower resolution the amount of data to be processed is greatly reduced at lower spatial frequencies

How Are They Implemented?

How Are They Implemented?

- Matching downsamplers and upsamplers
- Floating point versus Fixed Point Fraction (FPF)
- Computational environments:
  - Interleaved versus planar data layout
  - CPU – Vectorization and Parallelization
  - GPU – Can achieve real time, often more power efficient than CPU
  - DSP – such as Qualcomm Hexagon (HVX) for simpler FPF pipelines
  - Halide – Platform independence
- Bilaterals or even non-local means sometimes used for noise reduction

Downsamplers

- 3x3 or 5x5?
- Usually uses biased sampling at edges for performance
- Want to remove Nyquist frequency as completely as possible to avoid aliasing
- Want to retain lower frequencies as much as possible
- What coefficients are best?

Coefficients

3x3 Coefficient Weighting

1	2	1
2	4	2
1	2	1
Sum:	16	

5x5 Coefficient Weighting

1	4	6	4	1
4	16	24	16	4
6	24	36	24	6
4	16	24	16	4
1	4	6	4	1
Sum:	256			

- Divide relative weight by sum to get coefficients summing to unity
- Why are they chosen?

High Frequency Removal

5x5 Coefficients (divided by sum)						High Frequency						Resulting Image				
1	4	6	4	1		1	0	1	0	1		½	½	½	½	½
4	16	24	16	4		0	1	0	1	0		½	½	½	½	½
6	24	36	24	6	x	1	0	1	0	1	=	½	½	½	½	½
4	16	24	16	4		0	1	0	1	0		½	½	½	½	½
1	4	6	4	1		1	0	1	0	1		½	½	½	½	½

Sum: 256

- Regardless of high frequency phase (alignment)
- 3x3 similar

Upsamplers

- Should match downsampler
- Usually uses biased sampling at edges for performance
- 3x3 is a bilinear interpolator
- What is a “5x5 upsampler” in this context?

“5X5 Upsampler”

- Imagine lower res image resampled to higher res sparse matrix
- Multiply by same coefficients as downsampler

5x5 Coefficient Weights						Sparse Matrix				
1	4	6	4	1		A	.	B	.	C
4	16	24	16	4		.	.	.	.	.
6	24	36	24	6	x	D	.	E	.	F
4	16	24	16	4		.	.	.	.	.
1	4	6	4	1		G	.	H	.	I
Sum:	256									

“5X5 Upsampler” Output

- The four upsampling equations:
  - 0, 0 = (A + 6B + C + 6D + 36E + 6F + G + 6H + I) / 64
  - 1, 0 = (4B + 4C + 24E + 24F + 4H + 4I) / 64
  - 0, 1 = (4D + 24E + 4C + 4G + 24H + 4I) / 64
  - 1, 1 = (16E + 16F + 16H + 16I) / 64
- With only four equations the weights can be normalized at compile time to avoid division at run time
- Applying the same method using 3x3 kernel yields a bilinear interpolator

Memory Allocation

- Mind memory alignment issues (use height, width, stride design)
- Mind cache aliasing issues (image starting addresses)
- Round lower image dimensions up to keep even
- Avoid heap thrashing

Small Bilateral Filters for Noise Reduction

- Can be applied in different places
  - Before downsampling each level during Gaussian generation to assure only filtered data is used
  - On the merged upsampled and difference planes
    - Beware that multiplying the difference plane for contrast manipulation will change your thresholds
  - On the differences plane directly
    - Variance thresholding issues when noise varies with brightness
    - Most precise frequency control and most analogous to a Guided Filter
    - Not dependent on later contrast manipulation
- Thresholds reduce as resolution reduces (about 2x each level)

Variance Stabilization vs. Dynamic Thresholds

- Variance Stabilization transforms image values so that noise sigma is constant across dynamic range
  - Enables simpler, invariant thresholds for bilateral filtering
  - Requires two more transform passes, in and out
- More commonly pixel data is available only after gamma transform
  - Noise varies non-linearly with image brightness
  - Threshold variation can be approximated with cubic polynomial
  - Use center pixel to approximate brightness to avoid increasing threshold calculations with increased bilateral size
  - Must retain original brightness if filtering Laplacian plane

Implementation on CPU

- Lends itself to both vectorization and parallelization
- Interleaved RGBA or YUVA pixels fit naturally into vectorization architecture
- Planar data can be more problematic to implement efficiently
  - Intel: Modern unaligned load/store vector instructions are now very efficient, alleviating earlier design constraints, and shuffle instruction can be very handy
  - ARM: Alignment issues are more problematic
- LUTs can be problematic, use simple calculation approximation

Memory and Cache Considerations

- Older caches: 3-4 input addresses, 1-2 output addresses
- Newer caches less restrictive
- Tiling of processing useful for neighborhood operations even with pitch-linear data layout
- Separable algorithms can be problematic in the vertical pass due to cache thrashing
- Beware cache aliasing in image allocations
- Honor cache line sizes for image line alignments
- Minimize number of passes in and out of memory
- On mobile embedded systems better perf usually means lower power

Parallelization

- Tiles are a useful unit of work, even if data is pitch linear
- In C++11 a Lambda function can be passed from thread farm manager to individual threads
  - Makes multi-threading on image processing easy, just write it as a Lambda function
  - Allows easy access to local context around Lambda function
  - Use an atomic integer for work unit (tile) index
  - Must handle recursion

Vectorization

- CPU – SIMD (Single Instruction Multiple Data)
  - Intel – SSE – AVX
  - ARM – Neon
  - More advanced instructions often require 64-bit operating mode
  - Compilers are getting better at auto-vectorization
- GPU
  - Inherently vectorized and parallelized
  - OpenGL, OpenCV, OpenCL, Vulcan, CUDA, etc.
  - Not so adept at sequential processing algorithms
- DSP
  - Qualcomm Hexagon only vectorizes integer instructions, not floating point

Fixed Point Fraction (FPF)

- 16-bit number represents a signed fraction between 1.0 and -1.0
- Technique revolves around the existence of FPF Multiply instruction
  - Multiplies two 16-bit integers for intermediate 32-bit result, shifts right 14 bits with rounding and returns low 16 bits
  - Usually only available as a vectorized instruction
    - SSE3: _mm_mulhrs_epi16 (and AVX variants)
    - Neon: vqdmulhq_s16 (and variants)
    - Also available on Qualcomm Hexagon embedded DSP
  - No division, vectorize Newton-Raphson with FPF output
- Degrading precision contraindicates for long calculation pipelines

Implementation on GPU

- Interleaved RGBA or YUVK pixels fit naturally into GPU architecture
  - Bilinear upsampler can be directly incorporated in subtraction to reduce memory passes
- Planar data can be more problematic depending on GPU hardware support
- Can be implemented in OpenGL ES for portability
- Real time video possibilities
- There are platform memory design implications

GPU Tricks

- Avoid allocation thrashing
- Be sure to cache and re-use handles for textures and uniforms
  - Heavy overhead for creation of objects and assigning handles
  - Uniform blocks to reduce handle count
- Look for ways to use dot product to fuse multiplies with addition
- Can use texture LUTs for arbitrary function
  - Use sampling interpolator to interpolate between LUT values
  - Can interpolate to crossfade between two LUTs
  - Can pack four output values into single LUT
- Cubic polynomials quickly computed for function approximation such as varying thresholds

Platform Memory Design Implications

- PC GPUs have separate memories for CPU and GPU
  - Must transfer data back and forth between memories for combined CPU & GPU processing
- Most mobile SOCs use shared memory model
  - Physical data transfer not needed
  - May need different or special memory mappings
    - GPU usually “HW” requiring contiguous memory
    - CPU simulates contiguous memory by mapping unit
    - Requires cache maintenance for coherency

Planar Data

- Newer GPUs might allow declaration and use as single element pixels without performance hit
- Older ones will eat you alive if you try (¾ processing power wasted)
- Can lie to GPU telling it to use 4 element “pixels” that are actually 4 discrete pixel values
  - Remember the 4 values are not geometrically co-located
  - Can’t use built-in interpolator
  - Must do manual edge value replication
  - Neighborhood operations more complicated to code
  - Need to lie about width since processing 4 pixels at once

Temporal and Multi-Dimensional Considerations

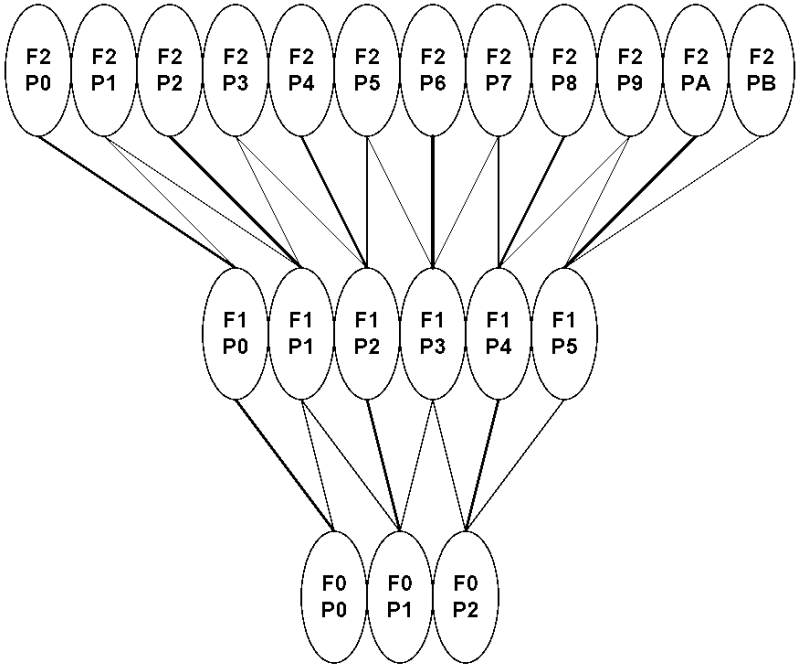
- Consider temporally interleaved data format at input if you have such control
  - Consider cache coherence restrictions
  - Might need to interleave on a cache line basis
  - Might be practical only on SOCs with HW cache coherency

Low Latency Designs

- “Just-in-Time” (JIT) processing
- Requires temporal knowledge of input image data stream progress such as a readable line counter or interrupts
- Process bands of lines at a time

Dependency Chart – 1D Example
3 Element Downsampler and Bilinear Upsampler

- Downsample value can be calculated when upscale dependencies have been input or computed
- Upsample value can be calculated when all downscale dependencies have been computed
- 2D example is analogous except with rows instead of pixels
- Latency varies across frame but can achieve sub-frame, especially if pyramid is not too deep



Conclusions

Pyramids for Efficient Spatial Processing Methods and Interchanges

- Efficient processing for neighborhood operators in the spatial frequency domain
- Consider sharing pyramids between processing modules
- Useful for image streams that serve both computer vision and human vision clients
- Choose implementation platform carefully

Resources

Resources

- *Laplacian Pyramid as a Compact Image Code*, Burt and Adelson, 1983
- *Pyramid methods in image processing*, E.H. Adelson and C.H. Anderson and J.R. Bergen and P.J. Burt and J.M. Ogden
- *Fast Feature Pyramids for Object Detection*, Piotr Dollar, Ron Appel, Serge Belongie, and Pietro Perona
- *Fast Pattern Recognition using Normalized Grey-Scale Correlation in a Pyramid Image Representation*, W. James MacLean, John K. Tsotsos
- *Optical Flow Estimation using a Spatial Pyramid Network*, Anurag Ranjan, Michael J. Black
- *Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition*, Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun

Resources (Continued)

- *Guided Image Filtering*, Kaiming He, Jian Sun, Xiaoou Tang
- *The Shadow Meets the Mask: Pyramid-Based Shadow Removal*, Yael Shor, Dani Lischinski
- *Image Inpainting based on Pyramids*, M. Shahid Farid, Hassan Khan, Arif Mahmood
- *The Steerable Pyramid. Pyramid methods in image processing*, Eero Simoncelli
- *Local Laplacian Filters: Edge-aware Image Processing with a Laplacian Pyramid*, Sylvain Paris, Samuel W. Hasinoff, Jan Kautz
- *Edge-Avoiding Wavelets and their Applications*, Raanan Fattal
- *Image Pyramids with Python and OpenCV*, Adrian Rosebrock, https://www.pyimagesearch.com/2015/03/16/image-pyramids-with-python-and-opencv/

Backup Slides

Where Are Pyramids Used in Image Enhancement?

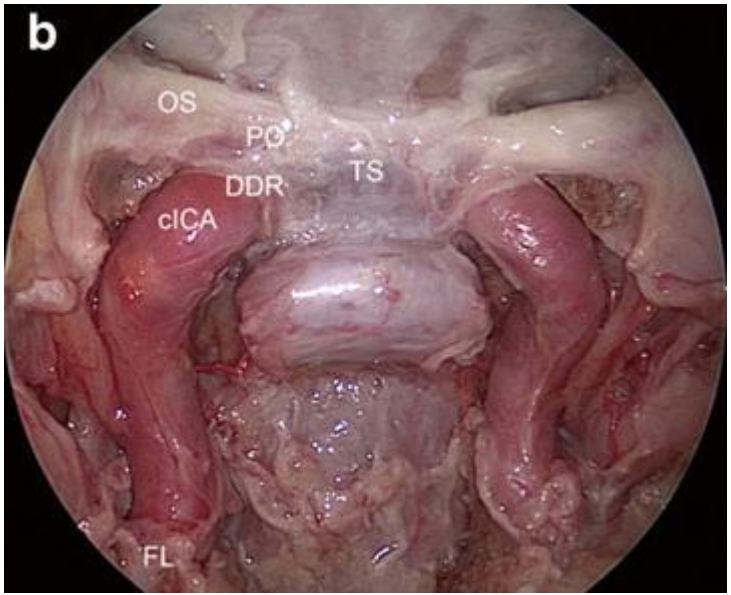
- Image blurring
- Noise reduction
- Spatial domain operations such as local tonemapping, contrast manipulation and enhancement
- Multi-resolution panoramic stitching
- Multi-focus blending
- Morphing

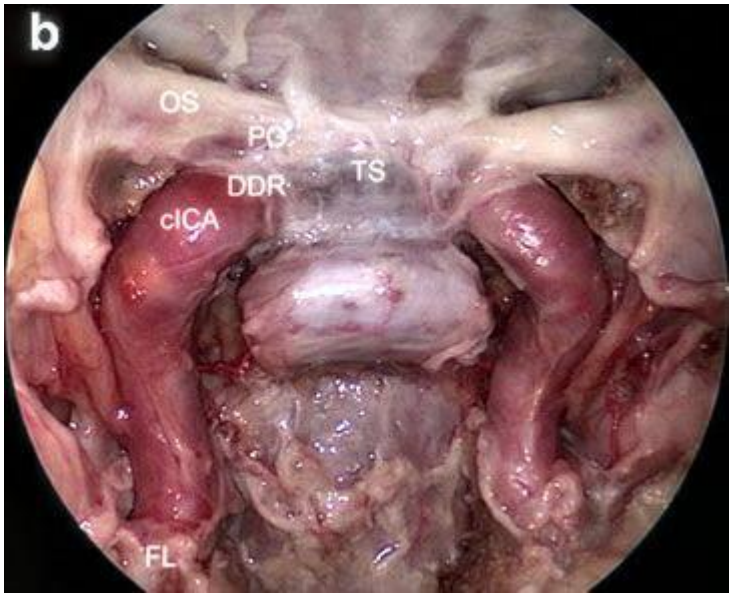
Local Tonemapping

- Before and after enhancement

Depth, Contour, and Shape Perception

- No high frequency sharpening





Guided Filter Comparison

- Guided Filter
  - Generates a low-pass filter (often a box filter on CPU)
  - Subtracts low-pass from original signal to create high pass
  - Operates on high-pass, such as noise filtering or contrast amplification
  - Adds low and high pass back together for result.
- Laplacian Pyramid
  - Next lower resolution band can be considered the low pass
  - Can be thought of as stacked “Guided Filter” operations independently at each frequency level

Non-Local Means (NLM)

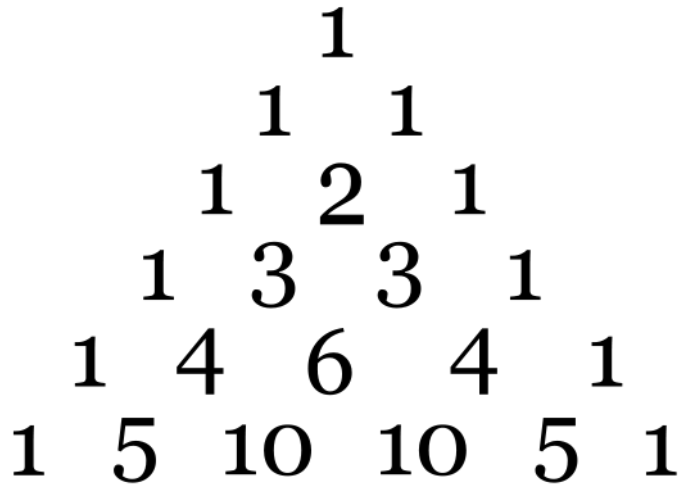
- An NLM filter can be used in place of the bilateral, especially in the differences plane
- Differences plane reduces adverse effects of gradients or illuminant on patch matching – mostly matches textures or reflectant
- Lower resolutions increase window area while decreasing patch-match comparison operations resulting in performance improvement
- Lower resolution vectors may be upscaled for use as hints in higher resolutions resulting in fewer patch-match comparisons

Wavelets

- Transforms image data into sequence of frequency coefficients
- Widespread use in image compression due to better compressibility of coefficients
- Usually implemented by separable horizontal and vertical operations
  - Especially on CPUs the vertical operation can be very cache unfriendly
- Not sure the cost/benefit ratio is very good for noise reduction applications

Binomial Coefficients

- The it (one dimension) binomial coefficients appear as the entries of Pascal's triangle where each entry is the sum of the two above



Wikipedia

Halide

- Facilitates cross platform utilization
- Separates algorithm from scheduling
- Some algorithms adapt well to Halide, others not so much
- Efficiency can be platform dependent
- Fixed Point Fraction designs not really supported
- Pyramid implementation as single Halide function available on GitHub
  - https://github.com/halide/Halide/tree/master/apps/local_laplacian

How To Convert Between Pixels and 16-Bit FPF?

- How to convert between, say 10-bit, pixels and 16-bit FPF, or between pixels of different bit depths?
- The correct way is very non-intuitive!
- From 10-bit pixels shift left 5 bits and then bit-replicate the high order 5 bits into the newly vacated low 5 bits, to get 16-bit FPF
- From 16-bit FPF clip any negative values to zero, then shift right 5 bits with truncation, not rounding!
- Converting between pixel formats of different bit depths is analogous
- The reason stems from the fact that there are an odd number of quantization intervals between 0 and 1 (all bits set to one).

Speaker Patents

- US Patent Application 16264059, Methods and Apparatus for Enhancing Optical Images and Parametric Databases
- US Patent 9,292,908 B2, System, Method, and Computer Program Product for Enhancing an Image Utilizing a Hyper-Clarity Transform
- US Patent Application 20090213211, Method and Device for Reducing the Fixed Pattern Noise of a Digital Image
- US Patent US 8197399 B2, System and Method for Producing and Improving Images
- US Patent 4,636,850, Apparatus and Method for Enhancement of Video Images (vascular landmarking and road-mapping)