Image and video processing (mostly via Poisson equation!)

Nicolas Bonneel

Poisson Image Editing

- "Poisson Image Editing", Perez et al. 2003

Poisson Image Editing

Poisson Image Editing

Poisson Image Editing

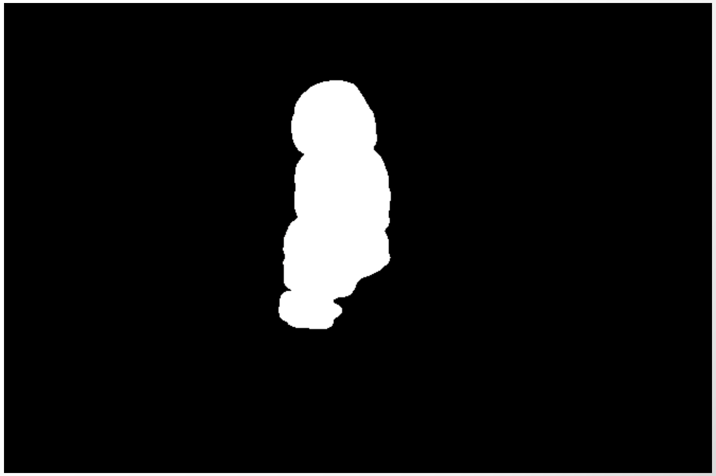
- How it works ?
  - The eye is more sensitive to color differences than absolute color values
  - We thus try to preserve color differences: $\min \int |\nabla u - \nabla g|^2 dx$
  - This leads to the equation :

$$\Delta u = \Delta g \text{ in } \Omega$$
$$u = f \text{ on } \partial\Omega$$

f

g

Ω



Poisson Image Editing

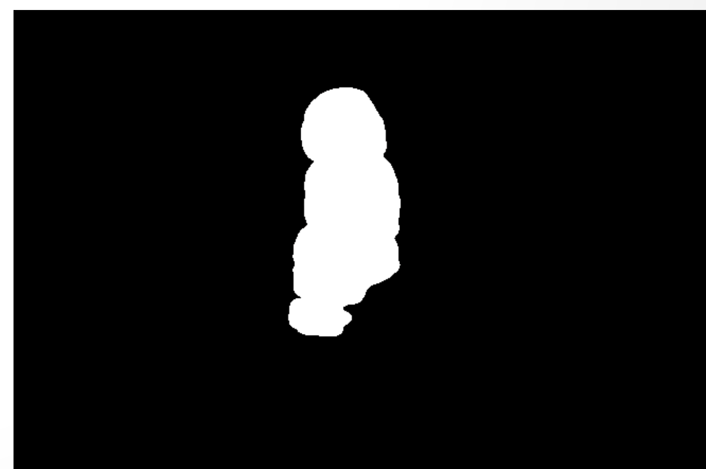
- How it works ?
  - The eye is more sensitive to color differences than absolute color values
  - We thus try to preserve color differences: $\min \int |\nabla u - \nabla g|^2 dx$
  - This leads to the equation :

$$\Delta(u - g) = 0 \text{ in } \Omega$$
$$\underbrace{u - g}_{v} = f - g \text{ on } \partial\Omega$$

f

g

Ω



Poisson Image Editing

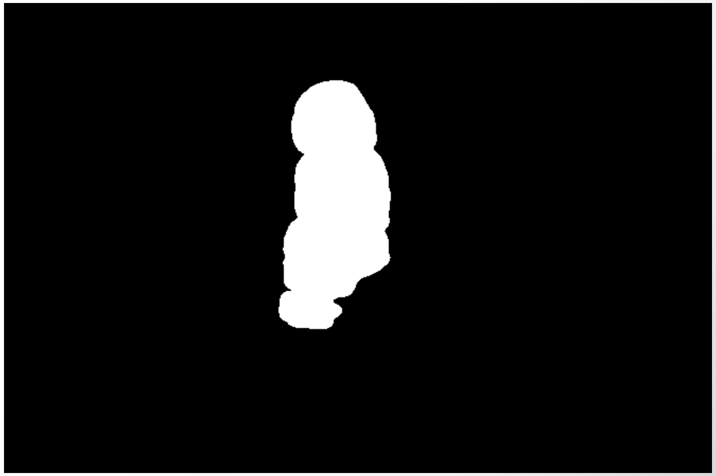
- How it works ?
  - The eye is more sensitive to color differences than absolute color values
  - We thus try to preserve color differences: min $\int |\nabla u - \nabla g|^2 dx$
  - This leads to the equation :

$$\Delta v = 0 \text{ in } \Omega$$
$$v = f - g \text{ on } \partial\Omega$$

f

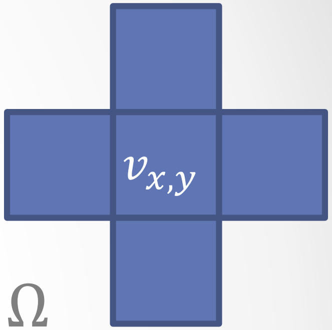
g

Ω



Poisson Image Editing

- How it works ?
  - The eye is more sensitive to color differences than absolute color values
  - We thus try to preserve color differences: $\min \int |\nabla u - \nabla g|^2 dx$
  - This leads to the equation :

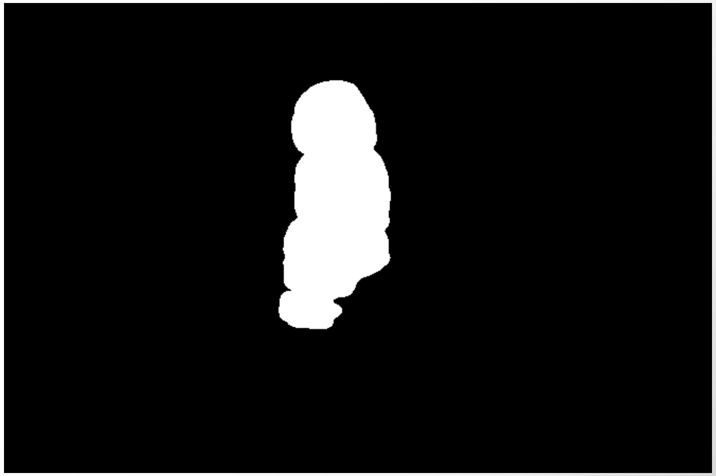


$$4v_{x,y} - v_{x+1,y} - v_{x,y+1} - v_{x-1,y} - v_{x,y-1} = 0 \text{ in } \Omega$$
$$v_{x,y} = f_{x,y} - g_{x,y} \text{ on } \partial\Omega$$

f

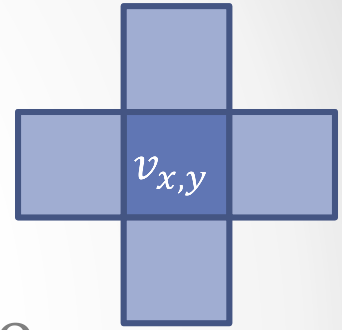
g

Ω



Poisson Image Editing

- How it works ?
  - The eye is more sensitive to color differences than absolute color values
  - We thus try to preserve color differences: $\min \int |\nabla u - \nabla g|^2 dx$
  - This leads to the equation :



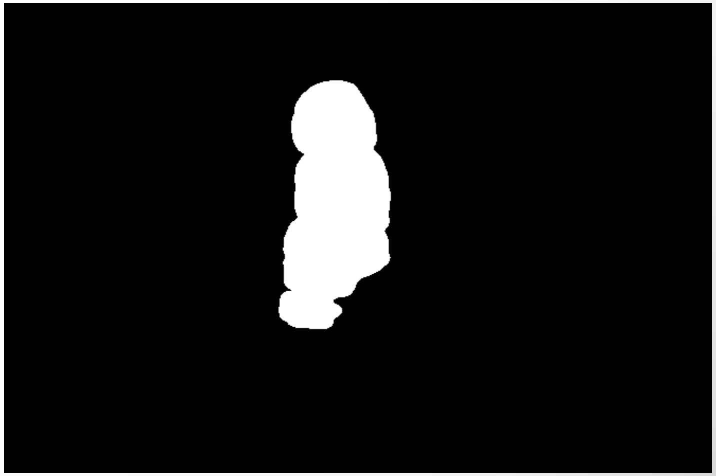
$$v_{x,y} = \frac{1}{4}(v_{x+1,y} + v_{x,y+1} + v_{x-1,y} + v_{x,y-1}) \text{ in } \Omega$$

$$v_{x,y} = f_{x,y} - g_{x,y} \text{ on } \partial\Omega$$

f

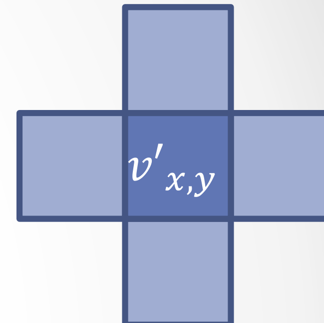
g

Ω



Poisson Image Editing

- How it works ?
  - The eye is more sensitive to color differences than absolute color values
  - We thus try to preserve color differences: $\min \int |\nabla u - \nabla g|^2 dx$
  - This leads to the equation :



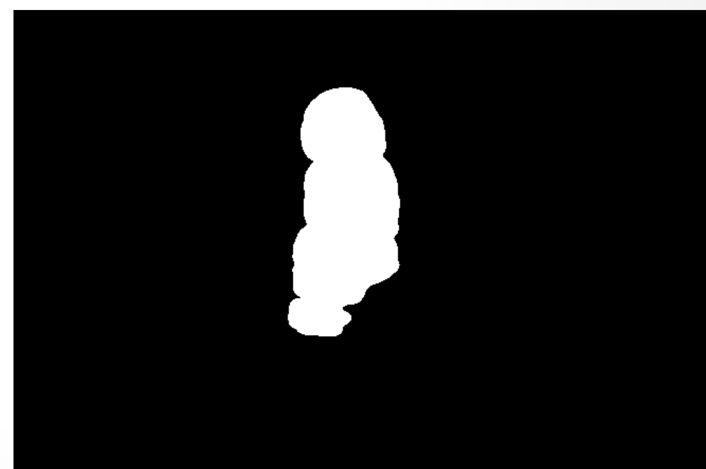
$$v'_{x,y} = \frac{1}{4}(v_{x+1,y} + v_{x,y+1} + v_{x-1,y} + v_{x,y-1}) \text{ in } \Omega$$

$$v'_{x,y} = f_{x,y} - g_{x,y} \text{ on } \partial\Omega$$

f

g

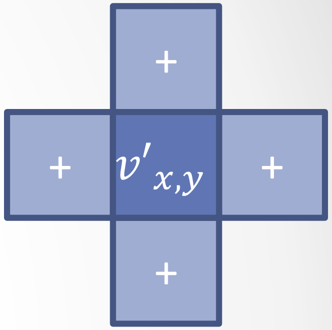
Ω



Poisson Image Editing

- How it works ?
  - The eye is more sensitive to color differences than absolute color values
  - We thus try to preserve color differences: $\min \int |\nabla u - \nabla g|^2 dx$
  - This leads to the equation :

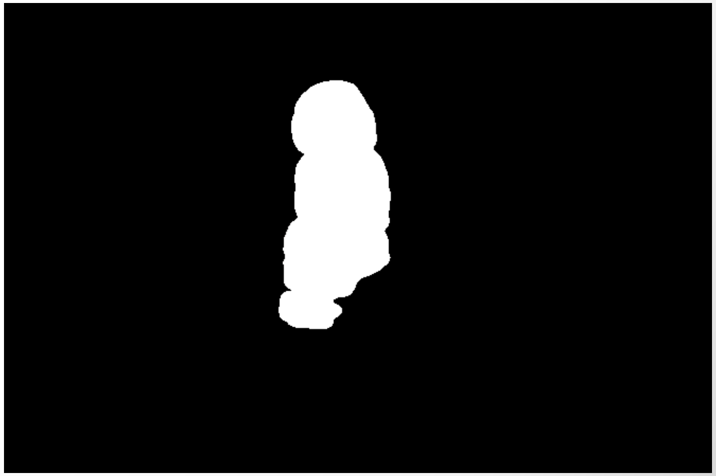
$$v'_{x,y} = \text{"}blur\text{"} \text{ in } \Omega$$
$$v'_{x,y} = f_{x,y} - g_{x,y} \text{ on } \partial\Omega$$



f

g

Ω



Poisson Image Editing

- Blur result :

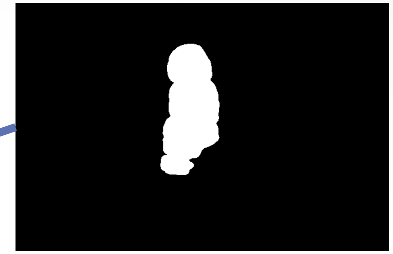
```
__global__ void relax(float4 *out, int w, int h) {
   int x = blockIdx.x*blockDim.x + threadIdx.x;
   int y = blockIdx.y*blockDim.y + threadIdx.y;
   if (x >= w || y >= h) { return; }

  float4 mask_up = tex2D(mask, x, y-1), ;
  float4 mask_dwn = tex2D(mask, x, y+1);
  float4 mask_left = tex2D(mask x-1, y);
  float4 mask_rght = tex2D(mask, x+1, y);
  float4 baby_up = tex2D(baby, x, y-1);
  float4 baby_dwn = tex2D(baby, x, y+1);
  float4 baby_left = tex2D(baby, x-1, y);
  float4 baby_rght = tex2D(baby, x+1, y);

  float4 u_up    = (tex2D(statue, x, y-1) - baby_up)    * (1.- mask_up)    + tex2D(prev_iter, x, y-1)  * mask_up;
  float4 u_dwn = (tex2D(statue, x, y+1)- baby_dwn) *  (1.- mask_dwn) + tex2D(prev_iter, x, y+1)  * mask_dwn;
  float4 u_left   = (tex2D(statue, x-1, y) - baby_left)  *  (1.- mask_left)   + tex2D(prev_iter, x-1, y)  * mask_left;
  float4 u_rght = (tex2D(statue, x+1, y) - baby_rght) *  (1.- mask_rght) + tex2D(prev_iter, x+1, y)  * mask_rght;

  float4 val = (u_up + u_dwn + u_left + u_rght)/4.;

  float4 mask_center = tex2D(mask, x, y);
  out[y * w + x] = val*mask_center + (1,-mask_center)*tex2D(statue, x, y);
}
```

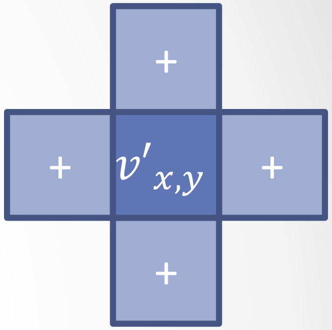


Poisson Image Editing

- How it works ?
  - The eye is more sensitive to color differences than absolute color values
  - We thus try to preserve color differences
  - Once v is known,

$$u = g + v$$

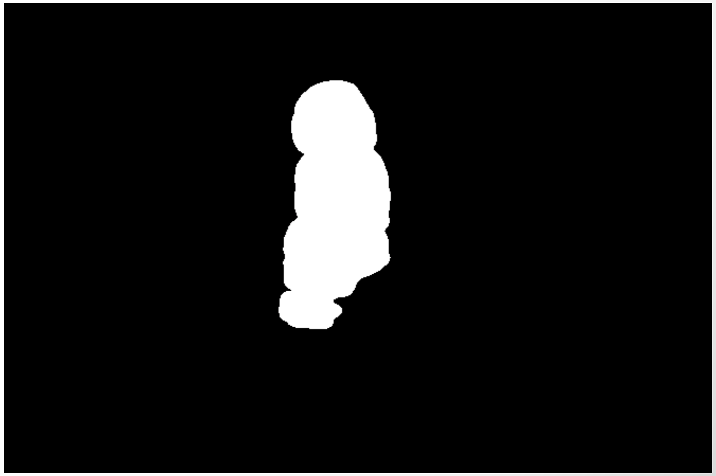
Quality highly depends on boundary (→ boundary optimization techniques)



f

g

Ω

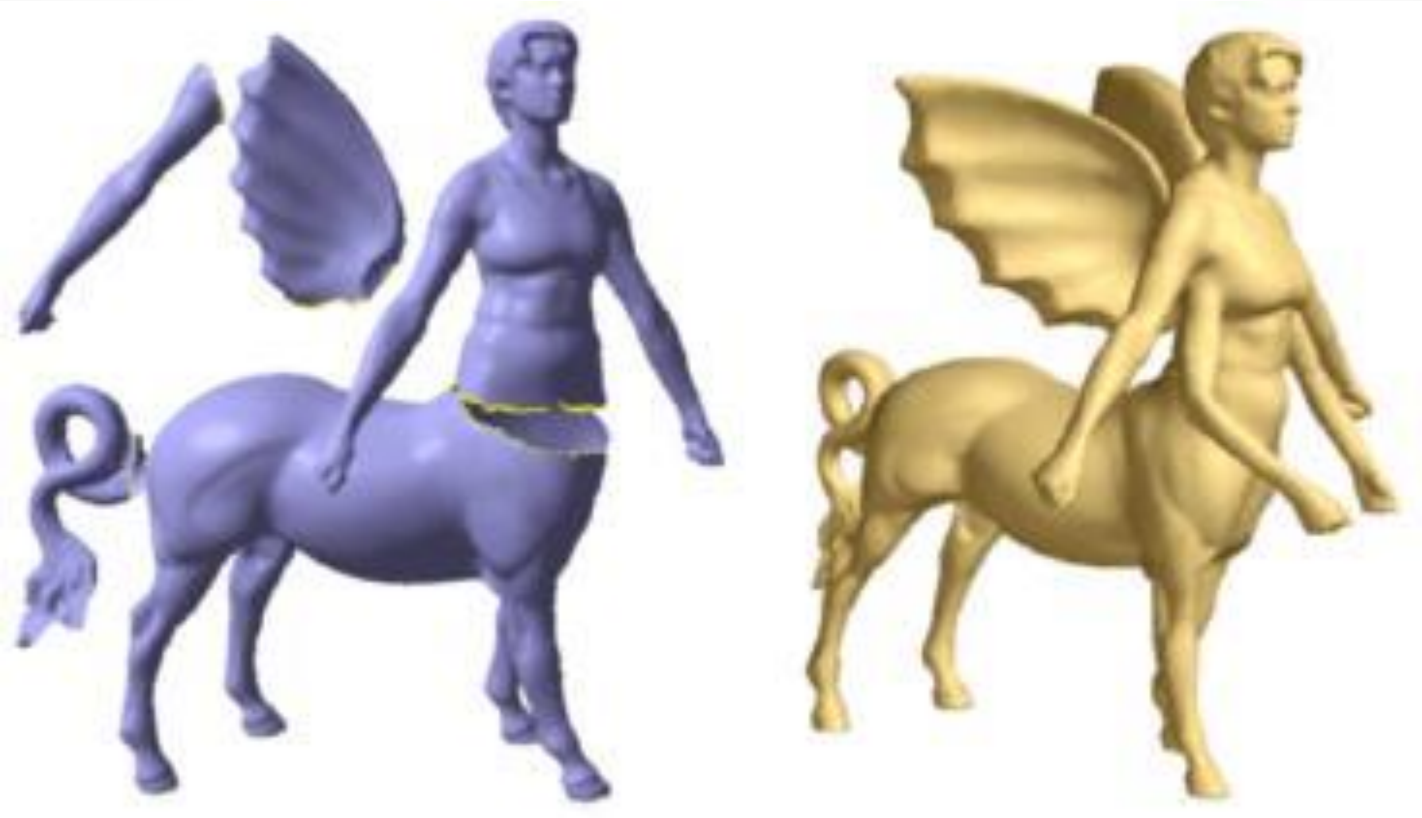


Poisson Image Editing

Poisson Image Editing

- Problem : Slow to converge + numerical precision issues
- Solution : Multigrid

Extended to meshes

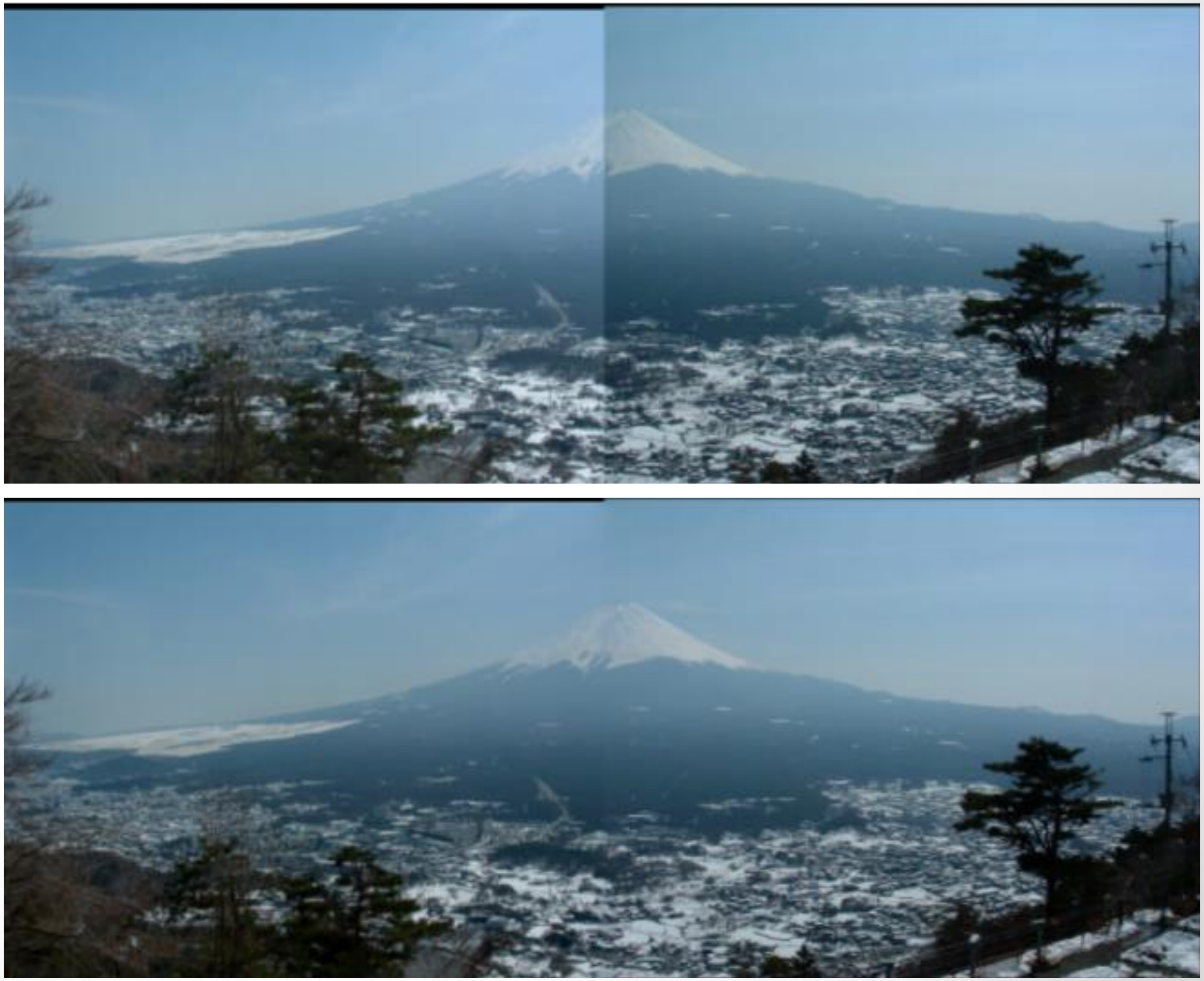


- “Mesh Editing with Poisson-Based Gradient Field Manipulation”, Yu et al. 2004

Generalizations

- L1 reconstruction
  - Use $\min \int |\nabla u - \nabla g|^1 dx$ instead of $\min \int |\nabla u - \nabla g|^2 dx$
  - Yields local formulation: $\text{div}\left(\frac{\nabla u - \nabla g}{|\nabla u - \nabla g|}\right) = 0$
  - More complex to minimize (nonlinear)
- Adding a spatial weighting term
  - $\min \int w(x)|\nabla u - \nabla g|^2 dx$
  - Yields local formulation: $\text{div}\big(w(x)(\nabla u - \nabla \text{g})\big) = 0$
- General form:
  - $\min \int w(|\nabla u - \nabla g|)dx$
  - Yields local formulation: $\text{div}\left(\frac{w'(|\nabla u - \nabla g|)}{|\nabla u - \nabla g|}(\nabla u - \nabla g)\right) = 0$
  - Most often non linear ; recovers linear isotropic diffusion with $w(u) = u^2$

Application to stitching



"Seamless Image Stitching in the Gradient Domain" Levin et al. 2002

Intrinsic decompositions

- "User-Assisted Intrinsic Images", Bousseau et al. 2009

Intrinsic decompositions

$$I = S.R$$

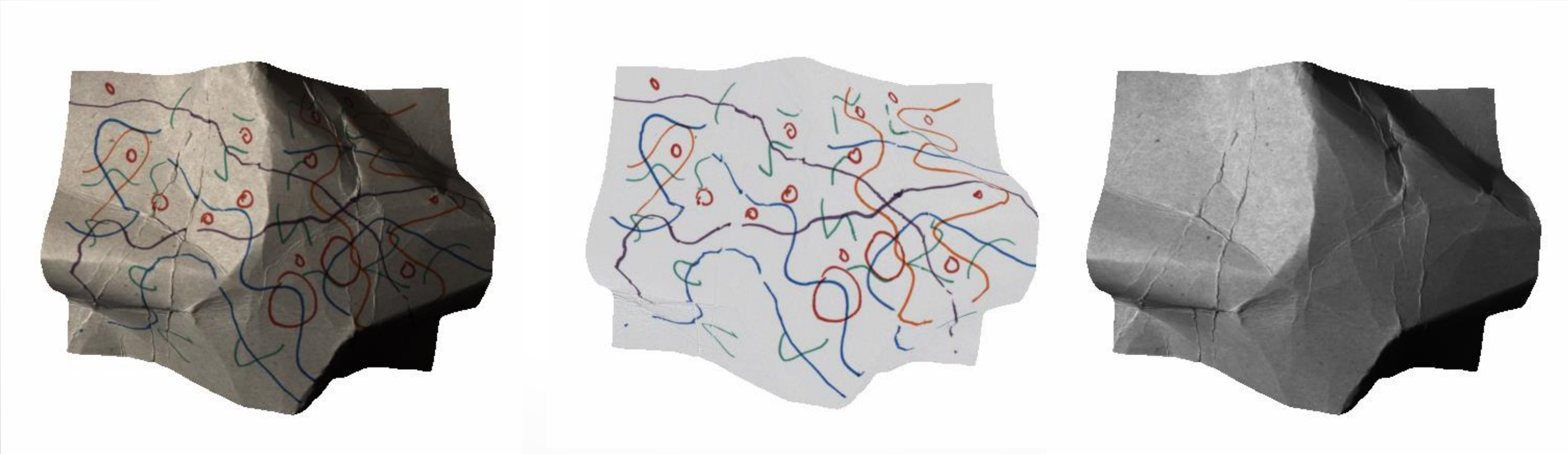
Image Shading Reflectance

- The Retinex assumption:
  - Shading layer smoother than reflectance layer
  - So, shading gradients are smaller
  - Color Retinex: if a gradient is colored, most likely comes from reflectance
- Ideas:
  - Work in log-domain : $\log I = \log S + \log R$
  - Work with gradients : $\nabla \log I = \nabla \log S + \nabla \log R$
  - Now, identify gradients belonging to either (log) S or R

"Ground truth dataset and baseline evaluations for intrinsic image algorithms", Grosse et al. 09

Intrinsic decompositions

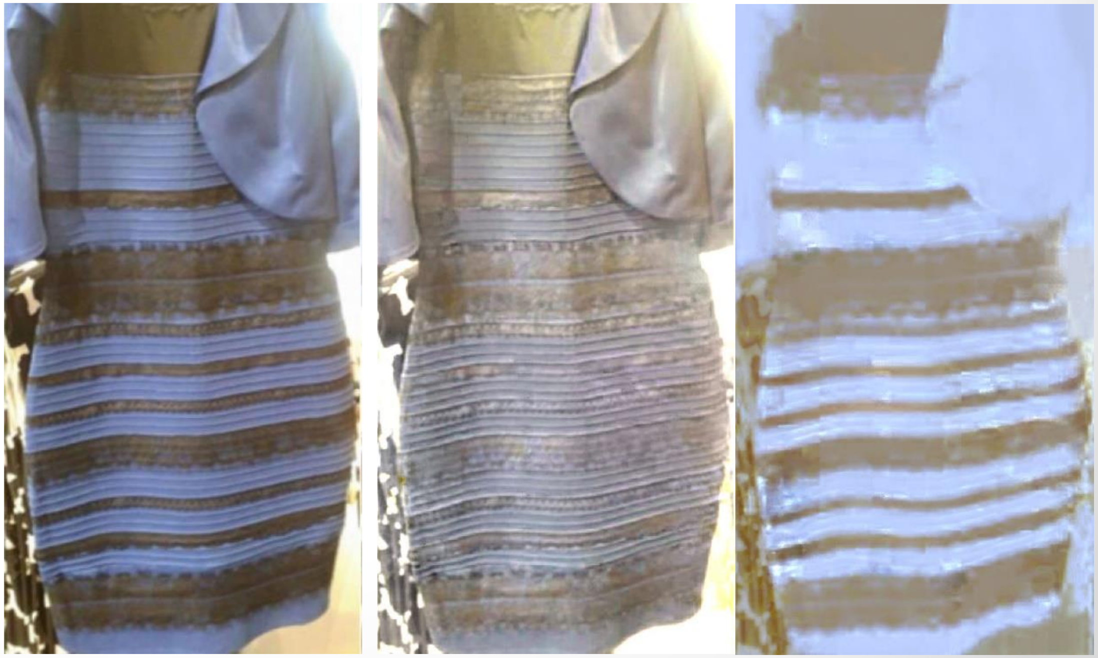
- Denote r_x the horizontal gradient of $\log R$ (same for y, I and S)
- Color Retinex algorithm:
  - If $|i_x^{br}| > T^{br}$ *and* $|i_x^{chr}| > T^{chr}$, $r_x = i_x^{br}$ *else* $r_x = 0$
  - Reconstruct R by solving a Poisson equation $\Delta \log(R) = \Delta r$
    - Can alternatively use an L1 reconstruction $\min \int |\nabla \log(R) - \nabla r| \, dx$
  - Obtain shading: $S = I/R$



Intrinsic decompositions

- Extensions:
  - Add non-local constraints on reflectance
  - Constrain reflectance colors to be sparse
  - Add reflectance constraints in time (for videos)
  - Add user constraints

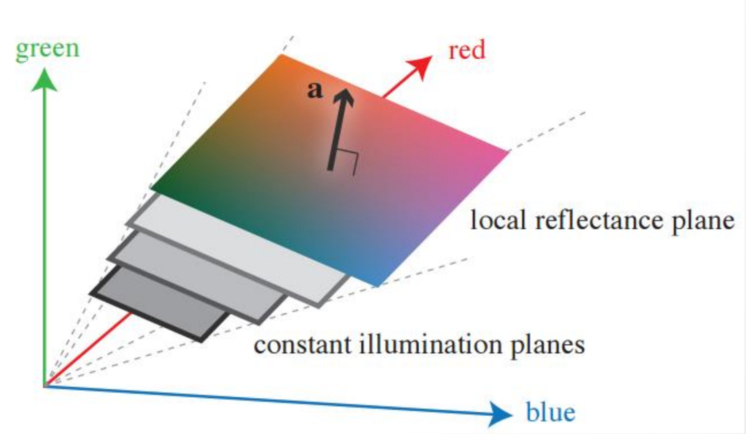
- Applications:
  - Re-texturing ; re-lighting
  - Image compositing
  - Better optical flows
  - Image segmentation
  - Scene understanding...



•

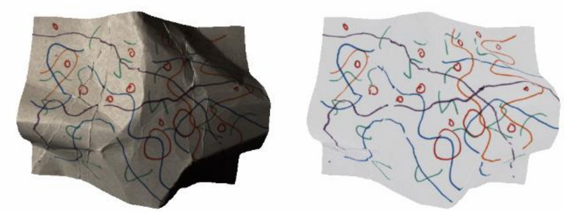
Intrinsic decompositions

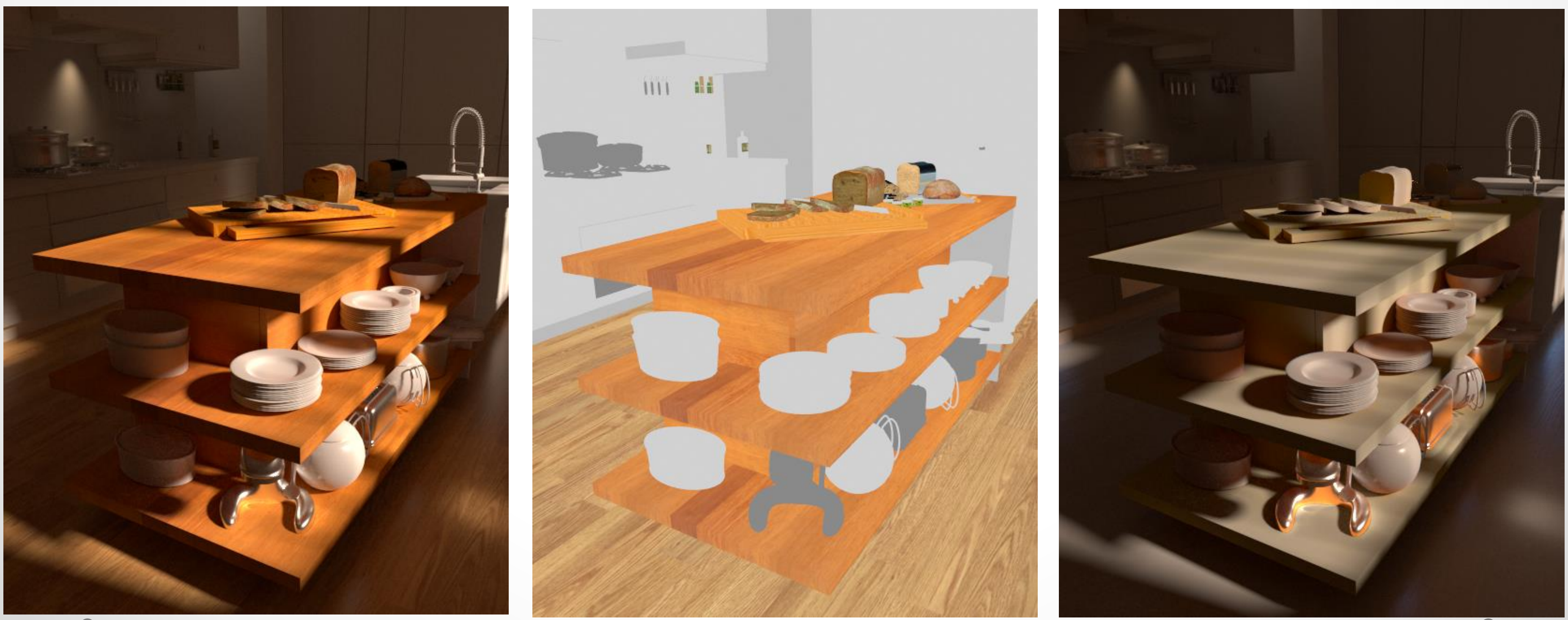
- Other approaches:
  - Based on machine learning: Neural networks, Conditional Random Fields ...
    - Difficulty: finding ground truth decompositions to learn from
  - Approaches estimating jointly shape, environment map and intrinsic decomposition
  - Algorithms based on other assumptions
    - Line model: Under skylight, pixels of same reflectance on same log-RGB line
    - Locally linear reflectance model



Intrinsic decomposition

- Ground truth images
  - Realistic scenes difficult to obtain
  - No good definition for specular scenes





Blind Video Temporal Consistency

Nicolas Bonneel[1] James Tompkin[2] Kalyan Sunkavalli[3]

Deqing Sun[2] Sylvain Paris[3] Hanspeter Pfister[2]

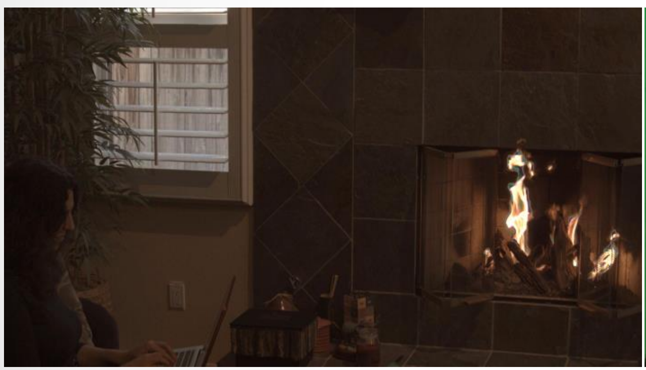
[1]CNRS / LIRIS

[2] HARVARD John A. Paulson School of Engineering and Applied Sciences

[3]Adobe Research

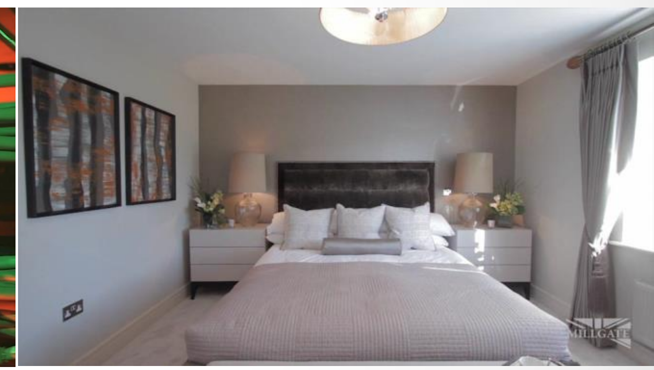
Observation

Image processing algorithms are often temporally unstable



HDR Tone Mapping

Spatial White Balance



Intrinsic Decomposition

Observation

Image processing algorithms are often temporally unstable

HDR Tone Mapping

Spatial White Balance

Intrinsic Decomposition

each frame processed independently

Goal

Input video

+

Image processing per frame

=

Stable video processing (our result)

Method

• • •

Variational formulation

Input video (V)

Processed video (P)

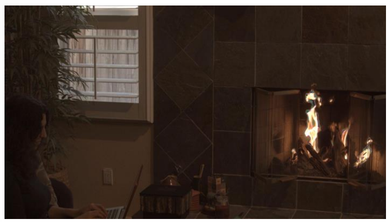
Output video (O)

$$\min \int ||\nabla O_n - \nabla P_n||^2$$

High frequency
scene dynamics

Variational formulation

Input video (V)



Processed video (P)

Output video (O)

$$\min \int \|\nabla O_n - \nabla P_n\|^2 + \quad \|O_n - warp(O_{n-1})\|^2$$

High frequency scene dynamics

Temporal consistency

Variational formulation

Input video (V)

Processed video (P)

Output video (O)

$$\min \int \|\nabla O_n - \nabla P_n\|^2 + w(x)\|O_n - warp(O_{n-1})\|^2$$

High frequency scene dynamics

Temporal consistency

$$w = \lambda \exp(-\|V_n - warp(V_{n-1})\|)$$

User parameters

$$\min \int \|\nabla O_n - \nabla P_n\|^2 + w(x)\|O_n - warp(O_{n-1})\|^2$$

- Temporal consistency strength
  - Scalar factor λ in w(x)
- "warp" operator
  - Optical flow methods
    [Sun et al. 2014] or [Wulff and Black 2015]
  - Nearest neighbor fields
    [PatchMatch, Barnes et al. 2009]

Screened Poisson Equation

Input video (V)

Processed video (P)

Output video (O)

- Energy can be minimized locally

$$-\Delta O_n + w(x)O_n = -\Delta P_n + w(x)warp(O_{n-1})$$

- Standard linear equation
  - Details in the paper

Fourier analysis (const. w)

Output (current frame) | Processed (current frame) | Output (previous frame)

$$\mathcal{F}(O_n)(\xi) = (1-\alpha)\,\mathcal{F}(P_n) + \alpha\,\mathcal{F}(warp(O_{n-1}))$$

with $\alpha = \dfrac{w}{4\pi^2\xi^2+w}$ depends on spatial frequency ξ

- Low and high spatial frequencies treated differently
  - Low frequencies more regularized
  - Unlike previous work that treats them uniformly

[Bonneel et al. - Blind Video Temporal Consistency]

Synthetic test

High frequency
noise
(except first frame)

Our result
(not suitable for
denoising)

Synthetic test

Low frequency
noise
(except first frame)

Our result

Results

Color grading

Input video

Per-frame processing

Color grading

Our result

Spatial White Balance [Hsu et al. 2008]

Input video

Per-frame processing

Spatial White Balance [Hsu et al. 2008]

Input video

Our result

Intrinsic Images [Bell et al. 2014]

Input video

Per-frame processing

Intrinsic Images [Bell et al. 2014]

Input video

Our result

Dehazing [Tang et al. 2014]

Input video

Per-frame processing

Dehazing [Tang et al. 2014]

Input video

Our result

Limitations

Processing creating edges

Matting

[Bonneel et al. - Blind Video Temporal Consistency]

Conclusion

- "Blind" approach to temporal consistency
  - Supported by Fourier analysis
- Wide range of image processing ported to videos
  - Can be applied as is to current and future algorithms
- C++ code
  http://liris.cnrs.fr/~nbonneel/consistency/

“Remove occupant…”

Texture synthesis

(also works with ex-gf/bf)

- “Parallel Controllable Texture Synthesis”, Lefebvre and Hoppe 2005

Texture synthesis

- Idea : copy pixels from the image that are coherent with an initial guess

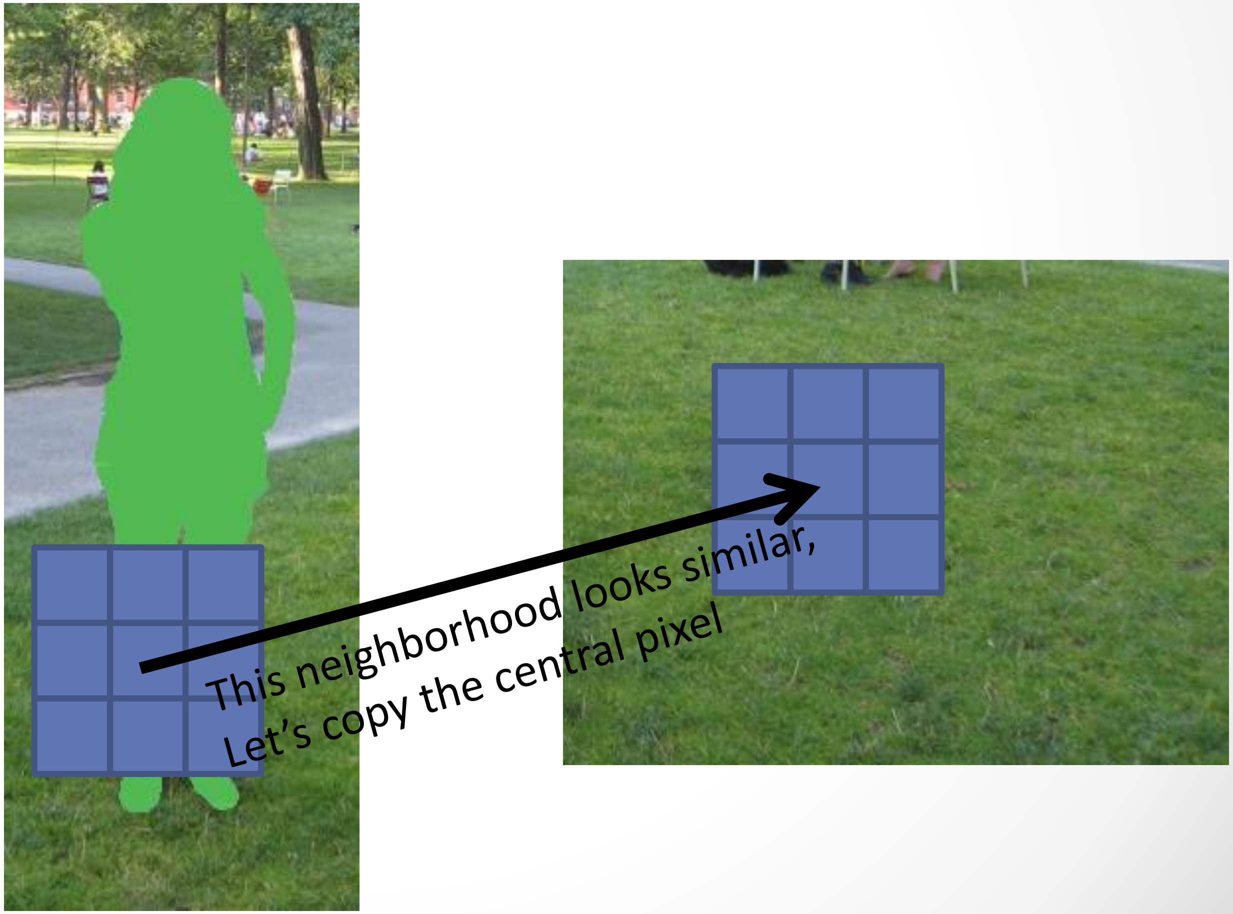
Texture synthesis

- Idea : copy pixels from the image that are coherent with an initial guess

Initial guess

Texture synthesis

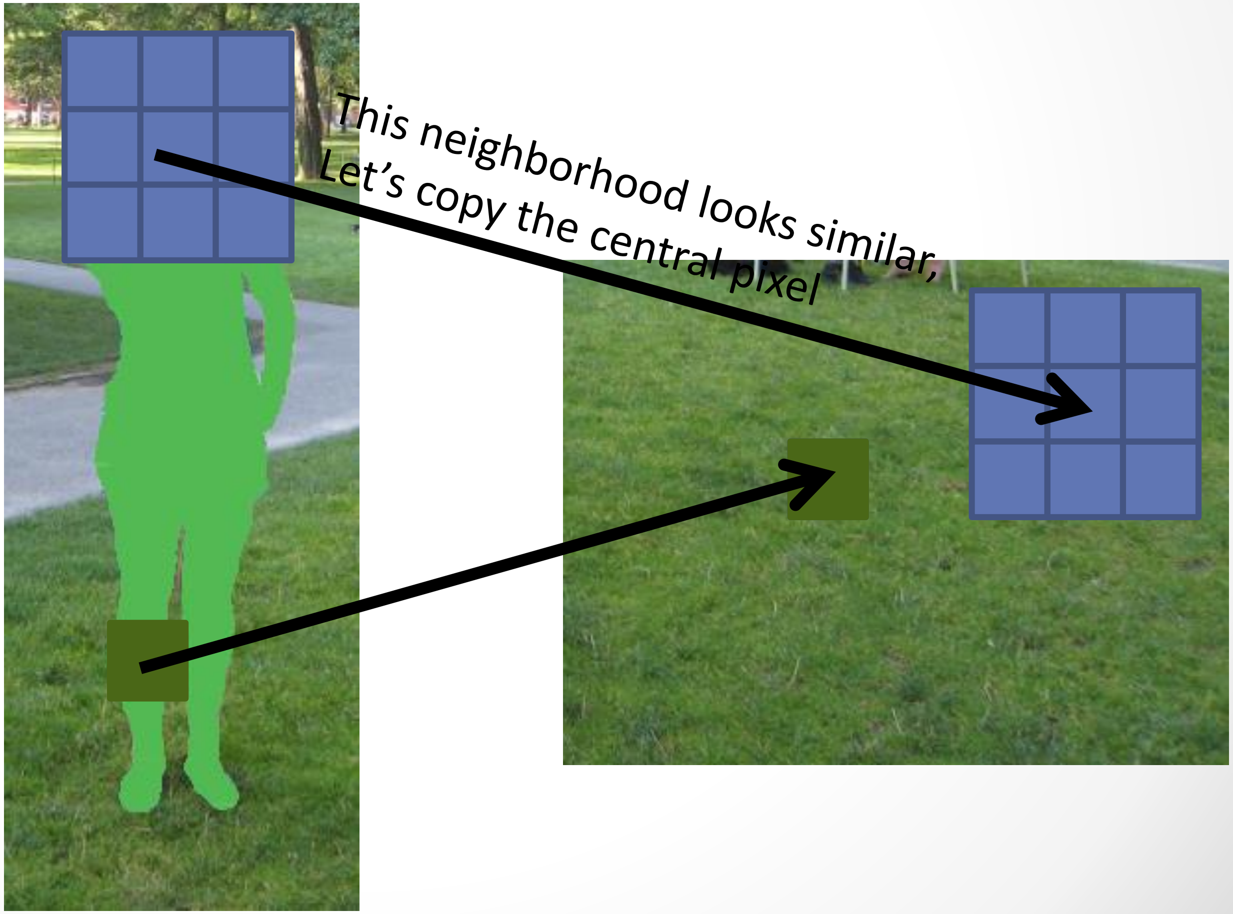
- Idea : copy pixels from the image that are coherent with an initial guess



Refinement

Texture synthesis

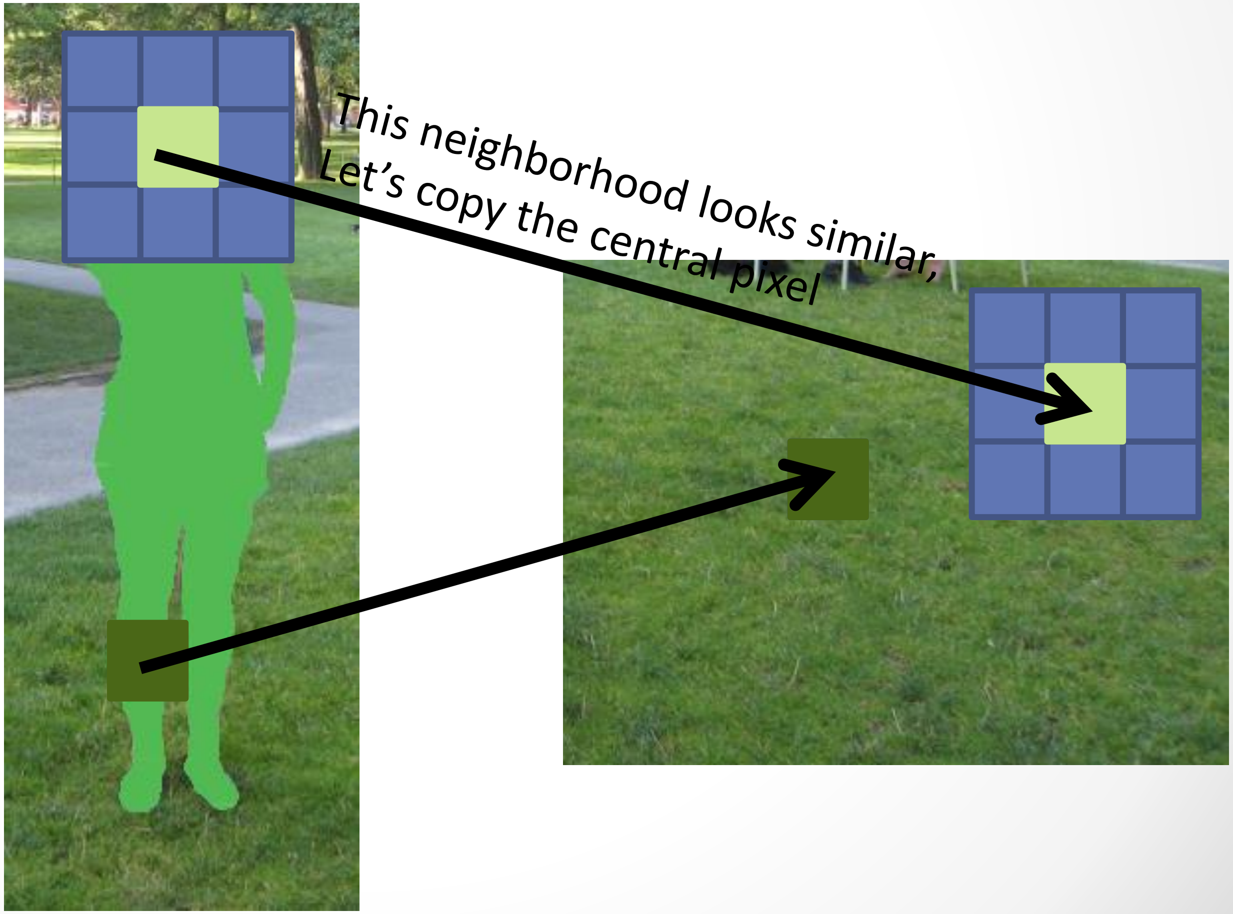
- Idea : copy pixels from the image that are coherent with an initial guess



Refinement

Texture synthesis

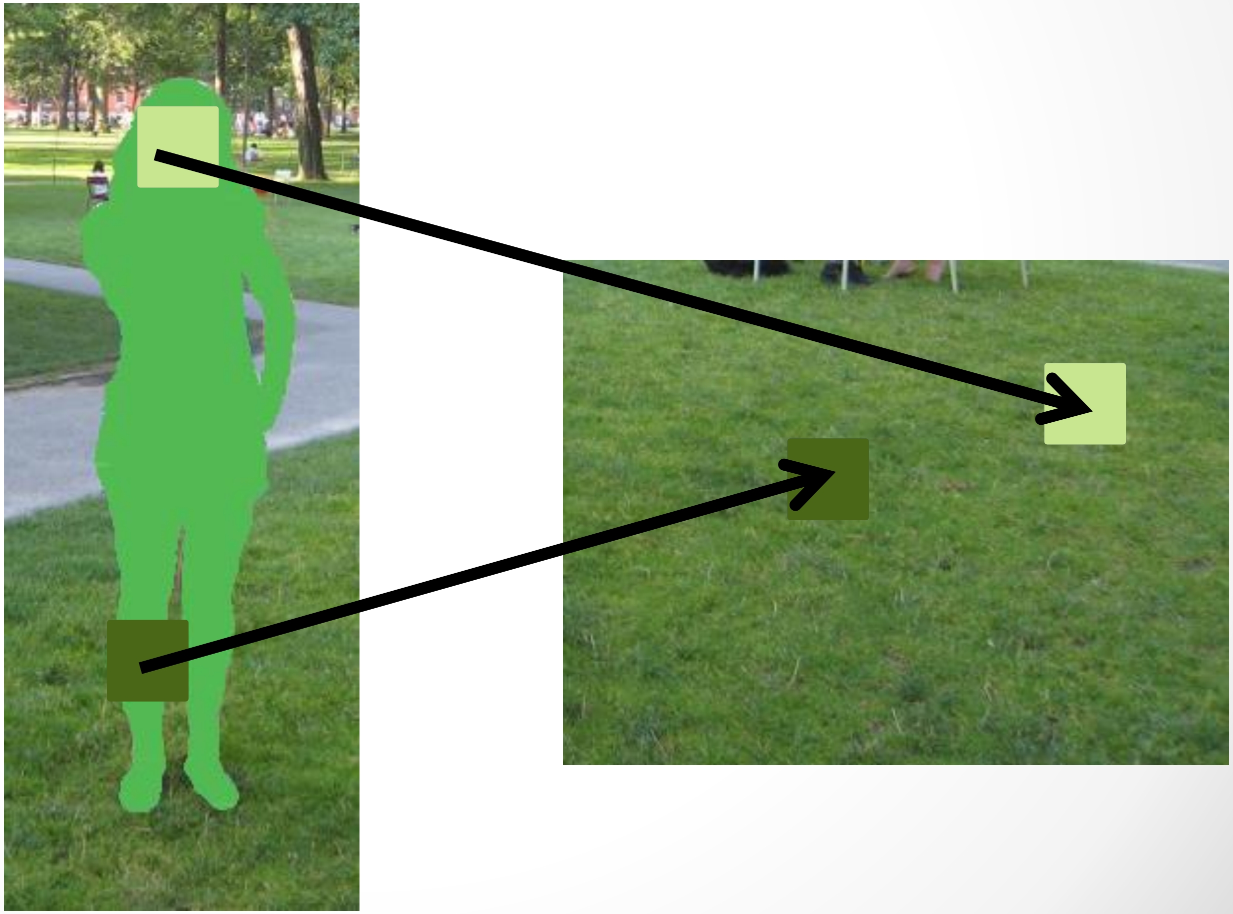
- Idea : copy pixels from the image that are coherent with an initial guess



Refinement

Texture synthesis

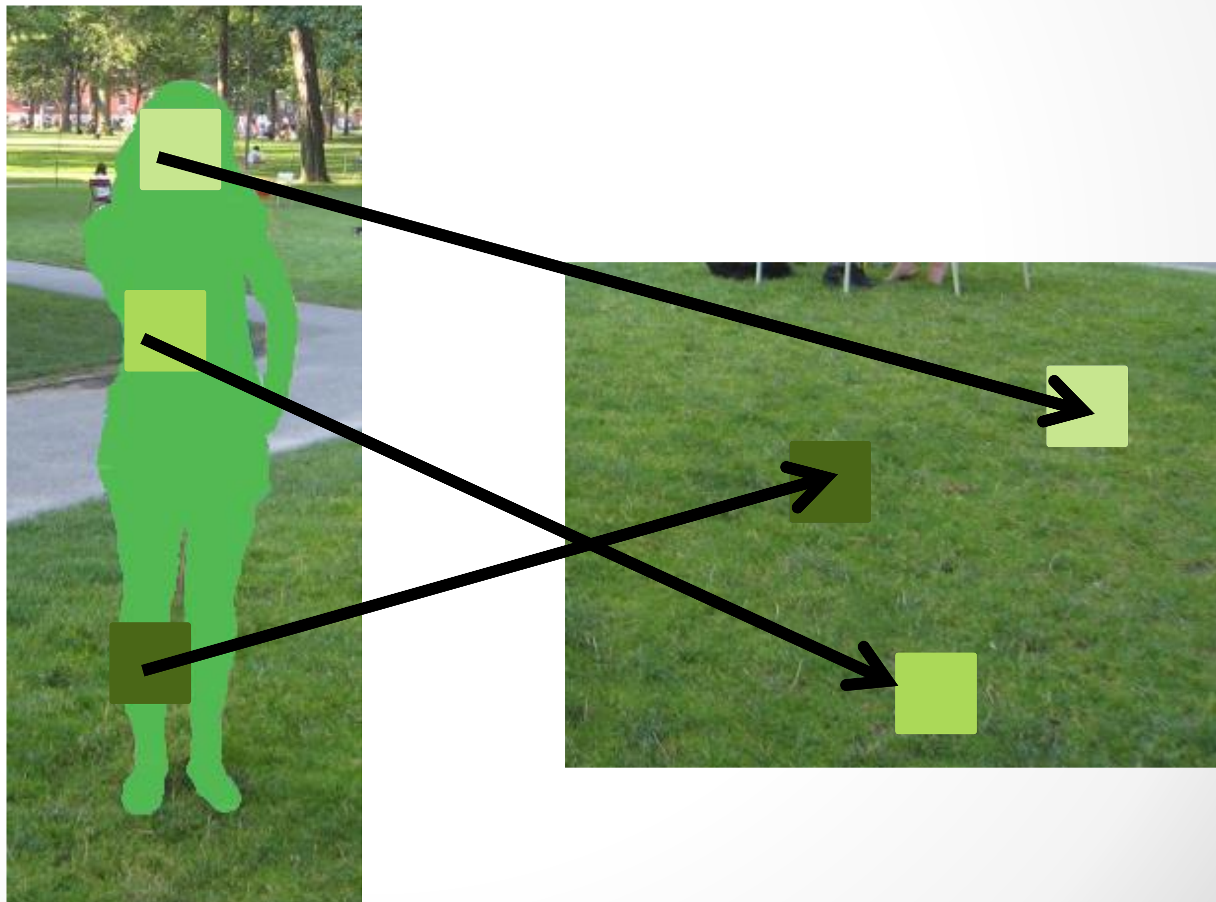
- Idea : copy pixels from the image that are coherent with an initial guess



Refinement

Texture synthesis

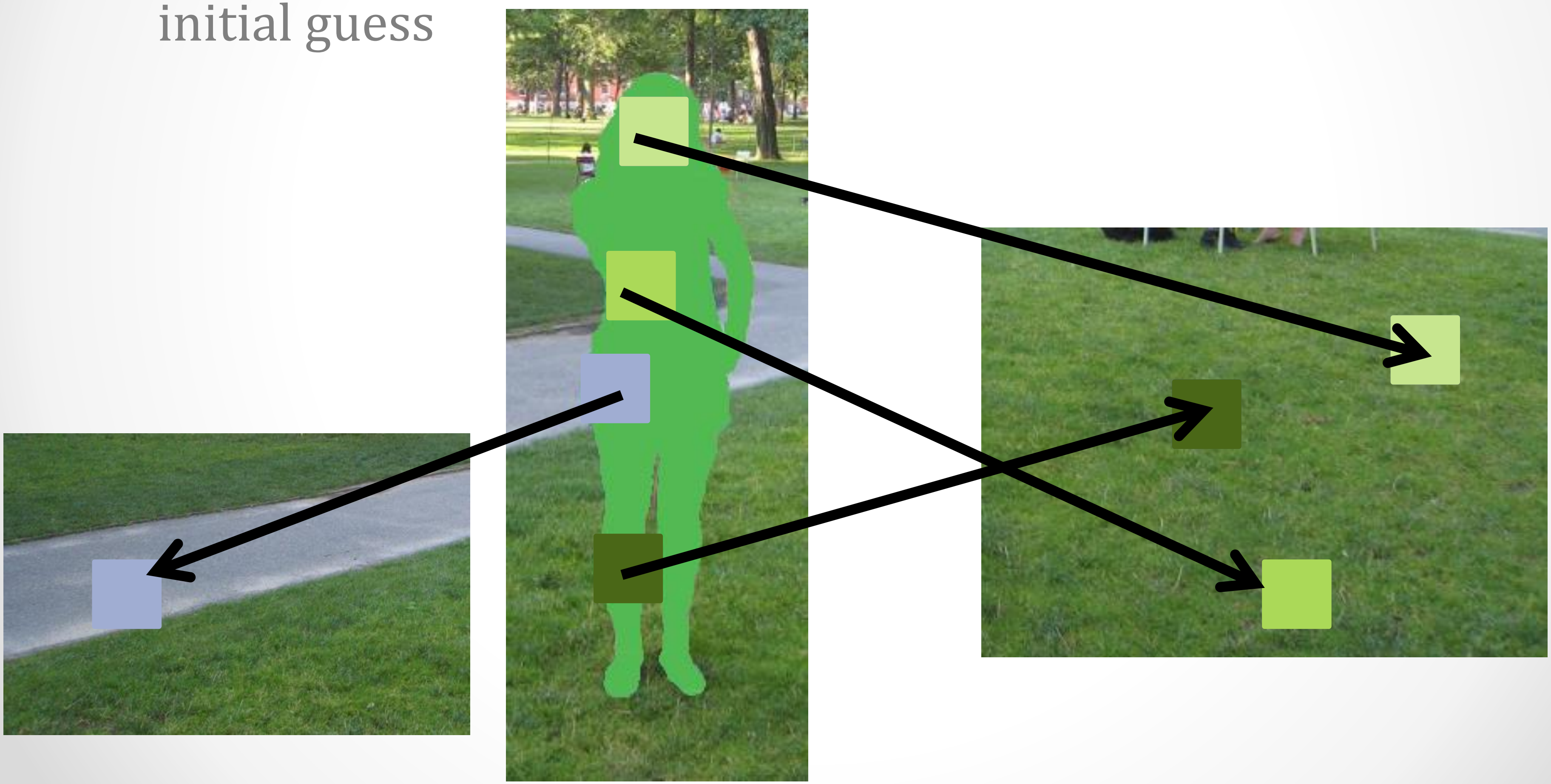
- Idea : copy pixels from the image that are coherent with an initial guess



Refinement

Texture synthesis

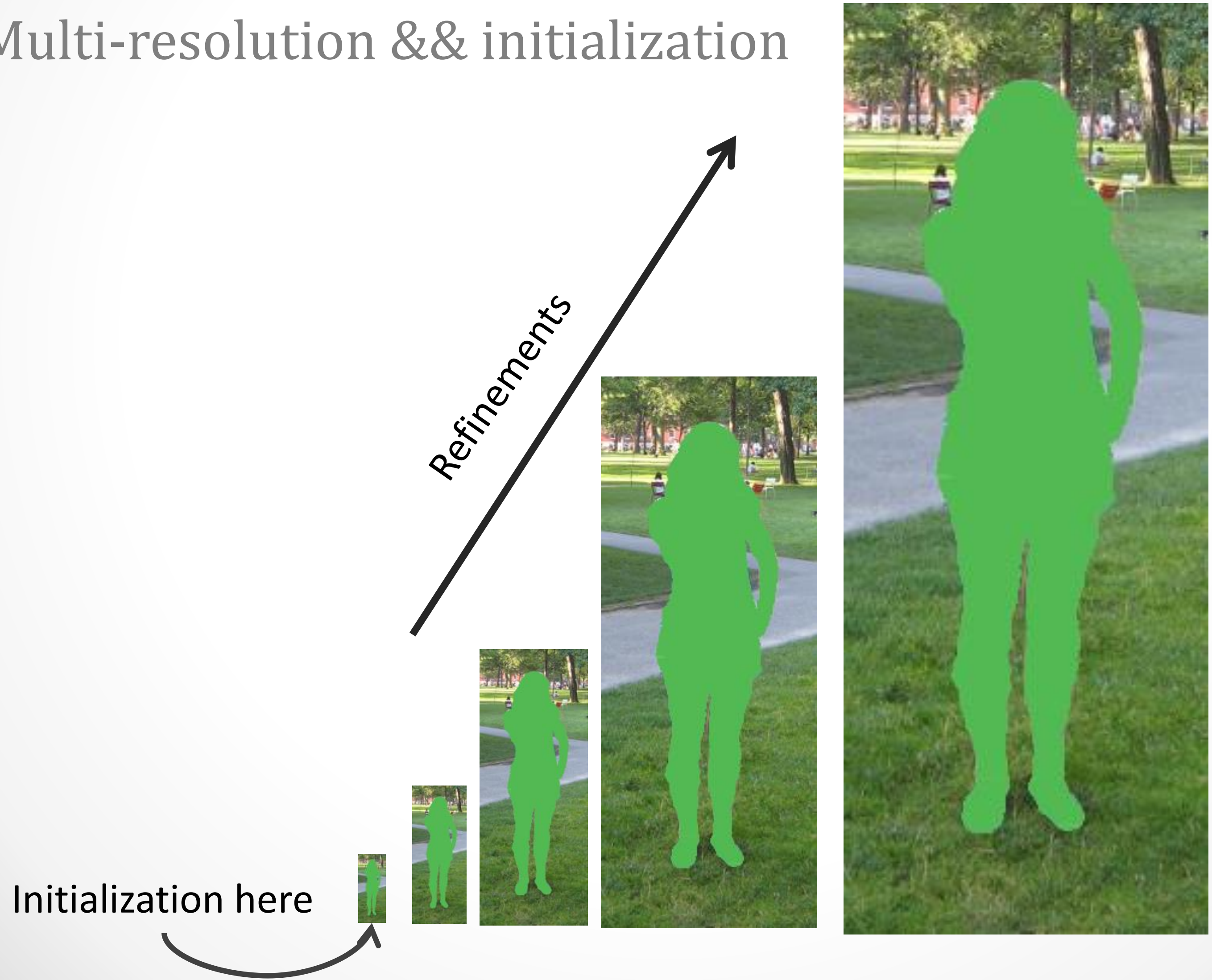
- Idea : copy pixels from the image that are coherent with an initial guess



Refinement

Texture synthesis

- Multi-resolution && initialization

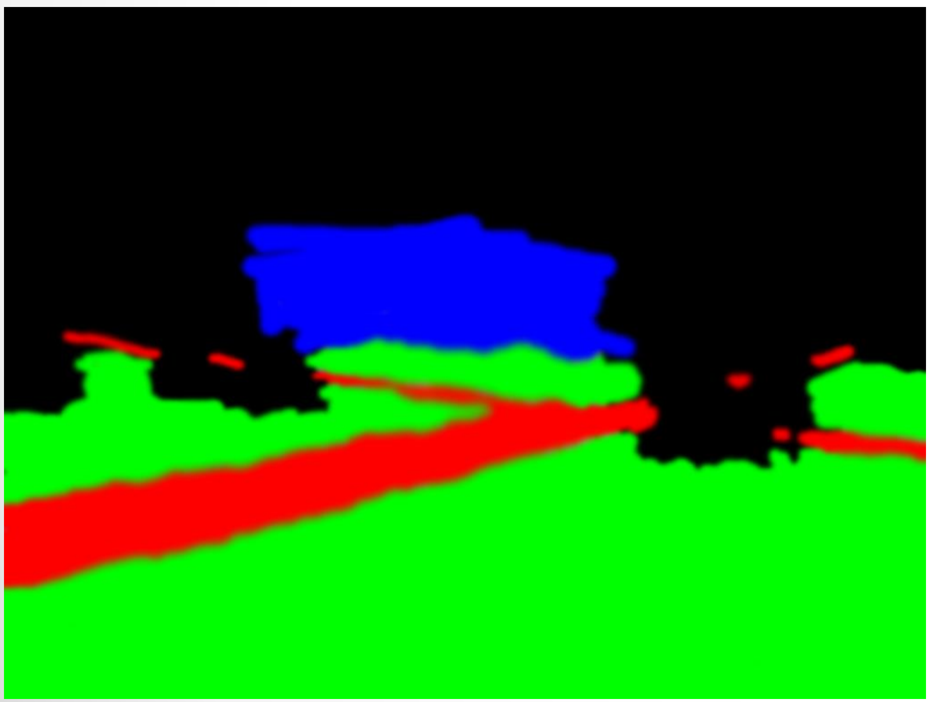


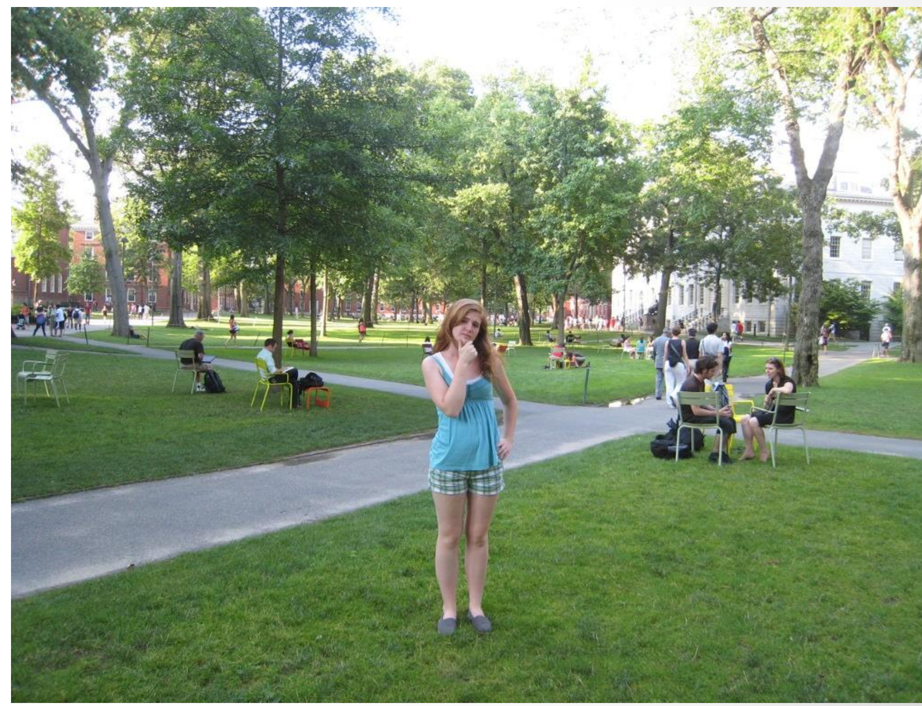
Texture synthesis

- ok – but artifacts

Texture synthesis

- Idea : use a guide

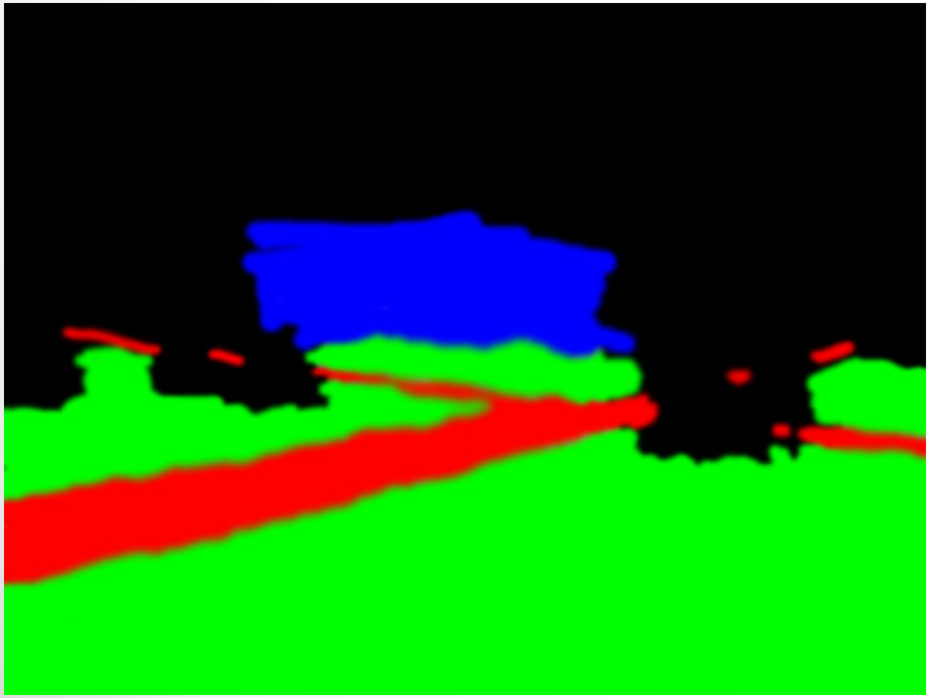




- “Image Analogies”, Hertzmann et al. 2001   (image-by-number approach)

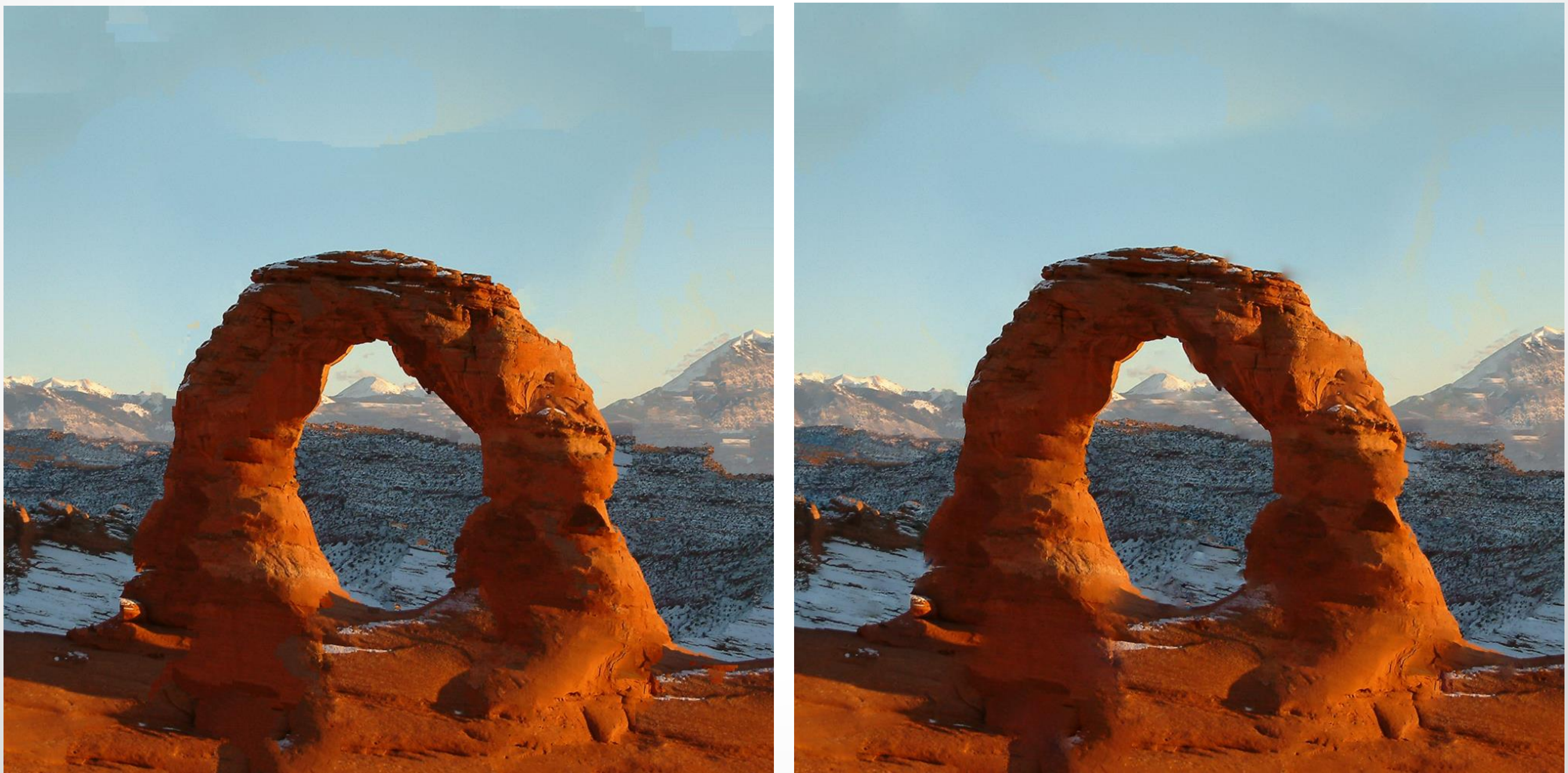
Texture synthesis

- Idea : use a guide



Texture synthesis

- Extension:
  - copy pixel gradients instead of pixels
  - Reconstruct an image with Poisson equation



Proxy-Guided Texture Synthesis

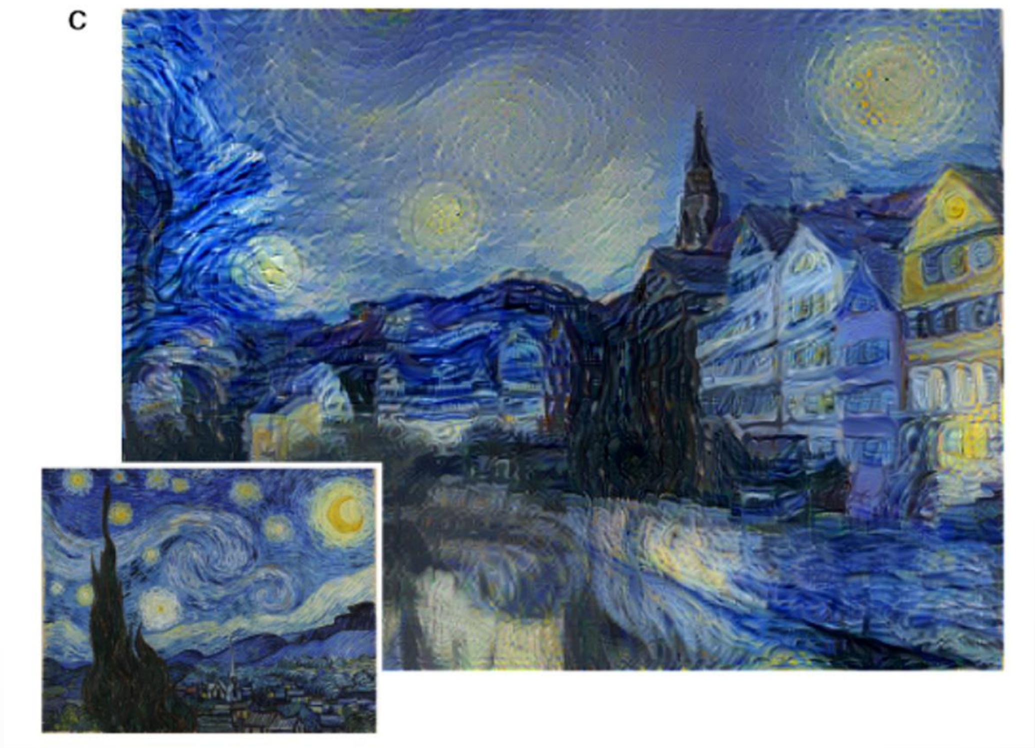
“Proxy-Guided Texture Synthesis for Rendering Natural Scenes”, Bonneel et al. 2010

Proxy-Guided Texture Synthesis

"Proxy-Guided Texture Synthesis for Rendering Natural Scenes", Bonneel et al. 2010

Texture synthesis

- That's the dinosaur version of Neural style transfer:



- "A Neural Algorithm of Artistic Style", Gatys et al. 2015

Gradient Shop

- Demoes filters achievable in the gradient domain
- Keeps a screened Poisson formulation

  $\nabla.\Big(w(x)\,\big(\nabla u - F(\nabla I)\big)\Big) + w'(x)\,\big(u - G(I)\big) = 0$


  - Sharpening: $F(\nabla I) = c.\nabla I \quad G(I) = I \quad w = 1 \;\; w' = d$
    - More robust with spatially varying w


  - NPR: $F_x(\nabla I) = \cos^2(e).\frac{\partial I}{\partial x} n \;\; F_y(\nabla I) = \sin^2(e).\frac{\partial I}{\partial y}\, n$

    $G(I) = I \;\; w' = d$

    With e an edge detector, and n another weighting term
  - etc. etc.

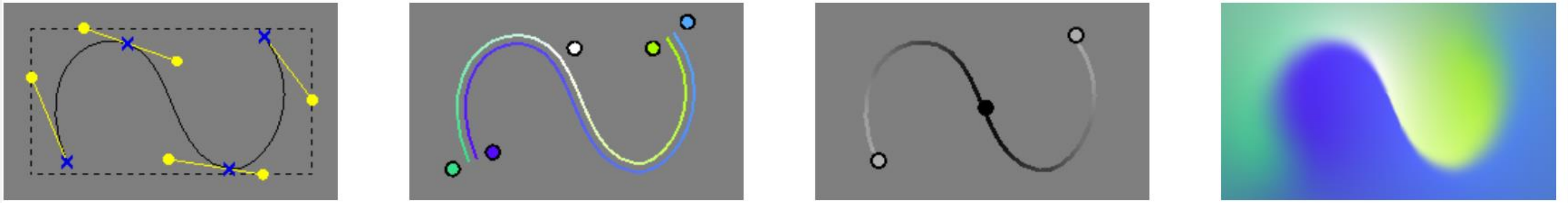
“GradientShop: A Gradient-Domain Optimization Framework for Image and Video Filtering”, Bhat et al. 2008

Depth of Field

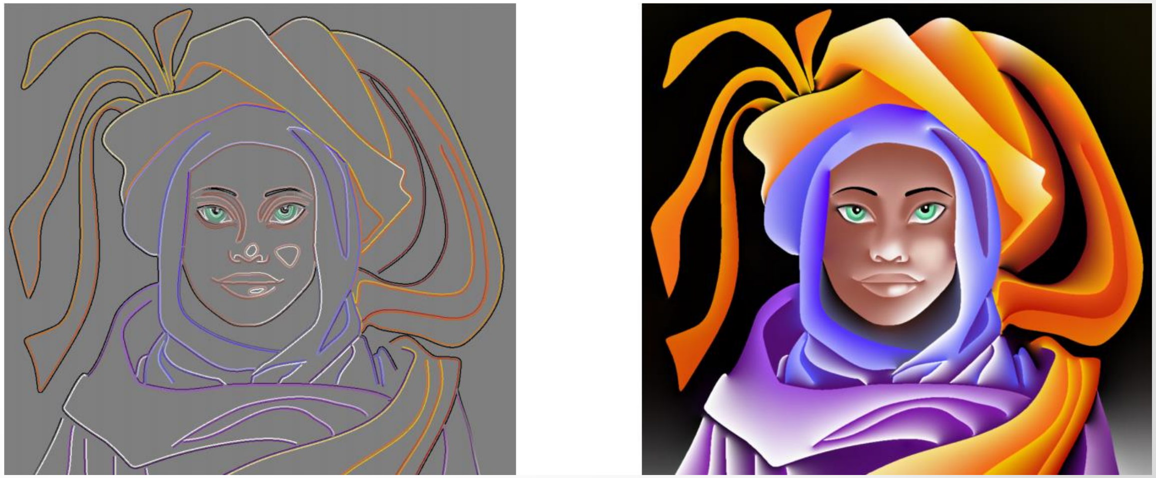
- Circle of confusion: $c = \frac{F^2}{n\,(Z_f - F)} \frac{|Z - Z_f|}{Z} = \alpha \frac{|Z - Z_f|}{Z}$
  - Z: depth ; Z_f: focal distance ; F: focal length ; n: aperture
- Consider anisotropic diffusion:
  - $\frac{\partial I}{\partial t} = \nabla.(g\,\nabla I)$ where g is the diffusivity
  - Take $g = \tilde{\alpha} \frac{|Z - Z_f|^2}{Z^2}$

“Real-time, Accurate Depth of Field using Anisotropic Diffusion and Programmable Graphics Cards”, Bertalmio et al. 2004

Diffusion curves



- Two Poisson equations:
  - $\Delta I = div\ w$ for colors (+ user constraints to I)
  - $\Delta B = 0$ for blur (+ $B = \sigma$ on curve)
- Then blur I using B with anisotropic diffusion

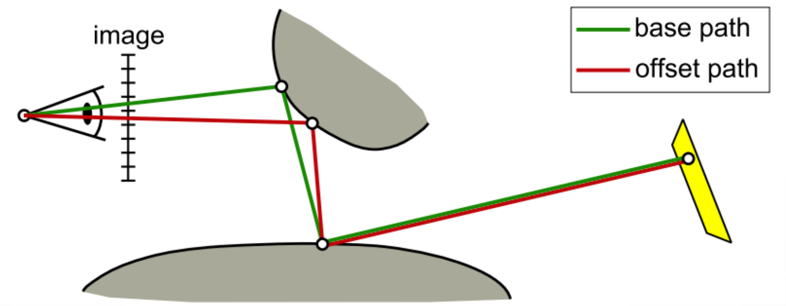


“Diffusion Curves: A Vector Representation for Smooth-Shaded Images”, Orzan et al. 2008

In rendering

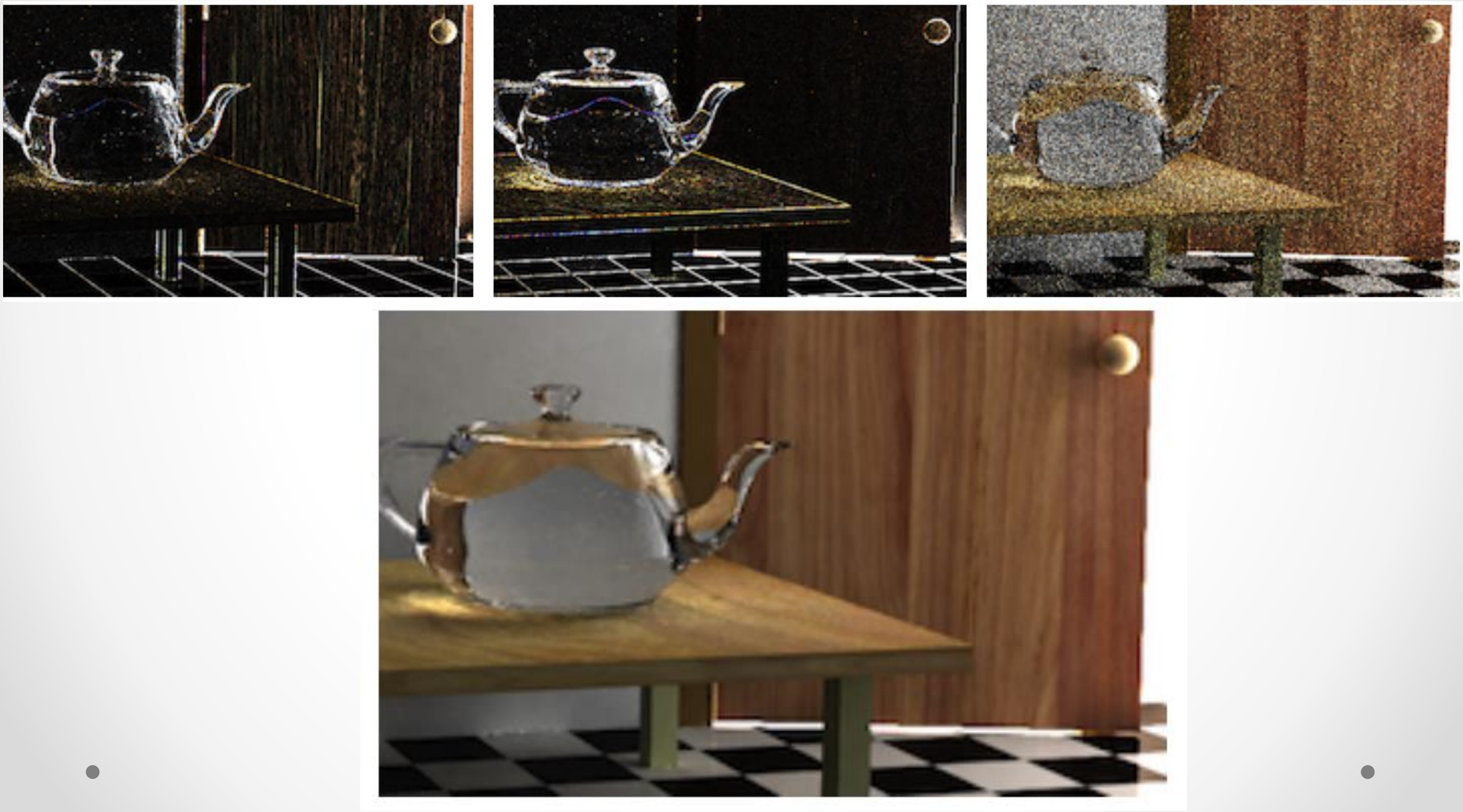
Poisson in Rendering

- Image derivatives of Metropolis light transport
  - Augment path space with vertical and horizontal image offsets
  - Estimate $I_{j+1} = \int h_{j+1}(x) f(x) d\mu(x)$
    - x in path space
    - h image filter
    - f image contribution
  - Use both $I_{j+1} - I_j$ (gradient) and I_j (actual value)
  - Drive sampler with a linear combination of both gradient norm and value
  - Solve the Screened Poisson equation (L1 works better)



- "Gradient-Domain Metropolis Light Transport", Lehtinen et al. 2013

Poisson in Rendering



Poisson in Rendering

- Simpler formulation with path tracing:

Input: Scene and camera specification, number of samples N.
Output: Path-traced image I, gradient images $\Delta_{\cdot,j}$.
for *all sampled base paths* $\bar{x} = (x, \bar{p})$ **do**
    for *all pixels* i *where* $h(x - x_i) > 0$ **do**
        // Write path contribution to primal image
        $I_i := I_i + h(x - x_i) f(\bar{x}) / p(\bar{x})$
        for *all neighbor pixels* $j \in \Phi_i$ *of* i **do**
            $\bar{y} := T_{ij}(\bar{x})$; **// offset path using shift** T_{ij}
            // gradient MIS weight $w_{ij}(\bar{x})$ **see Section 5.1**
            $\Delta_{i,j} := \Delta_{i,j} + w_{ij}(\bar{x}) h(x - x_i)(f(\bar{x}) - f(\bar{y})|T'_{ij}|)$
        end
    end
end
$I := I/N$; $\Delta_{;j} := \Delta_{\cdot,j}/N$, for all j
Reconstruct(I, $\Delta_{\cdot,\cdot}$,α)

- "Gradient-Domain Path Tracing", Kettunen et al. 2015 – reading list.

Solvers

Solving the Poisson Equation

- The Poisson equation $\Delta u = f$ can be discretized in 2d:
  - $4v_{x,y} - v_{x+1,y} - v_{x,y+1} - v_{x-1,y} - v_{x,y-1} = \mathrm{f}_{\mathrm{x,y}}$ (2nd order centered laplacian)
  - In matrix form : $M\ v = f$ with

$$M = \begin{bmatrix} & & \ddots & & & & \\ -1 & \dots & -1 & 4 & -1 & \dots & -1 \\ & -1 & \dots & -1 & 4 & -1 & \dots \\ & & -1 & \dots & -1 & 4 & -1 \\ & & & -1 & \dots & -1 & 4 \end{bmatrix}$$

(in practice, this is –Laplacian)

- We have seen one method so far
  - $v'_{x,y} = \frac{1}{4}\left(v_{x+1,y} + v_{x,y+1} + v_{x-1,y} + v_{x,y-1} + \mathrm{f}_{x,y}\right)$
  - This is the Jacobi method: $M = D - L - U$ with $D = diag(M)$
  - $M\ v = f \quad \Leftrightarrow \quad (D - L - U)v = f \Leftrightarrow \quad Dv = (L + U)v + f$
  - Build a converging sequence $Dv^{(k+1)} = (L + U)v^{(k)} + f$

Note: The solvers we'll see here also apply to other linear systems like the Radiosity linear system we saw in week 2.

Solving the Poisson Equation

- Jacobi is easy to parallelize but
  - Converges iif $D^{-1}(L+U)$ has max abs. eigenvalue < 1
    - Gershgorin argument just not enough here
    - Depends on BC (e.g., periodic BC has max eig = 1)
  - Converges if strictly diagonal dominant : not the case here
  - In practice, converges super slowly (or even not at all due to numerical precision)

- Gauss-Seidel:
  - Instead: $(D-L-U)v = f \Leftrightarrow \ (D-L)v = Uv + f$
  - This corresponds to solving a triangular system via backsubstitution:
  - $v_i^{(k+1)} = \frac{1}{M_{ii}}\left(f_i - \sum_{j=1}^{i-1} M_{ij} v_j^{(k+1)} - \sum_{j=i+1}^{n} M_{ij} v_j^{(k)}\right)$
  - For Poisson : $v'_{x,y} = \frac{1}{4}\left(v_{x+1,y} + v_{x,y+1} + v'_{x-1,y} + v'_{x,y-1} + \mathrm{f}_{x,y}\right)$

Solving the Poisson Equation

- Gauss-Seidel:
  - Converges faster
  - Converges if SPD matrix
    - Not necessarily: also depends on BC (in many cases, one degree of freedom) ; fixing them (Dirichlet) makes it ok
    - for the case of Radiosity, granted by energy conservation and reciprocity
  - Converges if strictly diagonal dominant
    - Still nope
  - In practice, converges a bit faster
  - Not parallelizable (easily) : depends on previously solved values

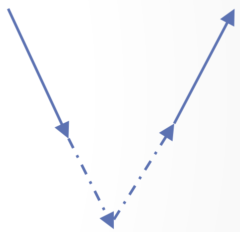
Solving the Poisson Equation

- Successive Over-Relaxation (SOR)
  - Use a weighted combination of previous and current iteration with Gauss-Seidel
  - $(D - L - U)v = f \Leftrightarrow \quad \omega(D - L)v = \omega U v + \omega f$
    $\Leftrightarrow (D - \omega L)v = (1 - \omega)D + \omega U + \omega f$
  - Leads to $v_i^{(k+1)} = (1 - \omega)v_i^{(k)} + \frac{\omega}{M_{ii}}\left(f_i - \sum_{j=1}^{i-1} M_{ij} v_j^{(k+1)} - \sum_{j=i+1}^{n} M_{ij} v_j^{(k)}\right)$
  - For Poisson : $v'_{x,y} = (1 - \omega)v_{x,y} + \frac{\omega}{4}\left(v_{x+1,y} + v_{x,y+1} + v'_{x-1,y} + v'_{x,y-1} + \mathrm{f}_{x,y}\right)$

- Convergence
  - If SPD matrix, converges for $0 < \omega < 2$
  - Expect to converge fast with $\omega > 1$ (goes further than GS)
  - For tridiagonal matrices (e.g., 1D Poisson): $\omega_{opt} = \frac{2}{1+\sqrt{1-\rho((D-L)^{-1}U)}}$
  - For 2D Poisson on an $n \times n$ grid: $\omega_{opt} = \frac{2}{1+\sin(\frac{\pi}{n})}$

Solving the Poisson Equation

- Geometric Multigrid
  - Last time we saw a multiscale approach. Good if we can build the rhs at any scale (e.g., Poisson Image Editing).
  - Otherwise:
    - Approximately solve $M_h v_h = f_h$
    - Take residual $r_h = f_h - M_h v_h$ and downsample it to r_{2h}
    - Approximately solve $M_{2h} r'_{2h} = r_{2h}$
    - ... continue...
    - Upsample r'_{2h} to r'_h by interpolation
    - Continue solving $M_h v'_h = f_h$ with $v_h + r'_h$ as starting point
  - Converges *much* faster: solves a linear system in $O(n)$
  - Still requires the matrix M_h at any scale h
    - If not, see "Algebraic multigrid"



Solving the Poisson Equation

- Conjugate Gradient
  - Example: $v^{(k+1)} = v^{(k)} + f + Mv^{(k)}$ (gradient descent for $\mathrm{F}(v) = \frac{1}{2}v^T Mv - fv$)
    - Take $v^{(1)} = f$
    - Shows $v^{(k)}$ in $\mathcal{K}_k = Span(f, Mf, M^2 f, \dots, M^{k-1} f)$: Krylov subspace
    - Can build orthogonal basis for $\mathcal{K}_k$ with Gram-Schmidt
  - We want residual $\mathrm{r}^{(\mathrm{k})} = f - Mv^{(k)}$ (which is in $\mathcal{K}_{k+1}$) to be orthogonal to $\mathcal{K}_k$
    - Squeezes the residual to smaller and smaller subspaces
    - So, $\mathrm{r}^{(\mathrm{k})}$orthogonal to $r^{(l)}$ $\forall l < k$
  - $\mathrm{r}^{(\mathrm{k})} \perp \mathcal{K}_k$ and $\mathrm{r}^{(\mathrm{k-1})} \perp \mathcal{K}_{k-1}$ so $\mathrm{r}^{(\mathrm{k})} - r^{(k-1)} \perp \mathcal{K}_{k-1}$
  - and $v^{(l)} - v^{(l-1)} \in \mathcal{K}_k$
  - So: $\left(v^{(l)} - v^{(l-1)}\right)^T \left(\mathrm{r}^{(\mathrm{k})} - r^{(k-1)}\right) = 0$ for $l < k$
  - We have $\mathrm{r}^{(\mathrm{k})} - \mathrm{r}^{(\mathrm{k-1})} = -M\left(v^{(k)} - v^{(k-1)}\right)$
  - So: $\left(v^{(l)} - v^{(l-1)}\right)^T M\left(v^{(k)} - v^{(k-1)}\right) = 0$ for $l < k$
    - The difference between iterates is M-conjugate

Solving the Poisson Equation

- Conjugate Gradient
  - $\alpha^{(k)} = \frac{r^{(k-1)\,T}\, r^{(k-1)}}{d^{(k-1)\,T}\, M\, d^{(k-1)}}$ // such that $r^{(k)} \perp r^{(k-1)}$
  - $v^{(k)} = v^{(k-1)} + \alpha^{(k)} d^{(k-1)}$
  - $r^{(k)} = r^{(k-1)} - \alpha^{(k)} M d^{(k-1)}$ // $\mathrm{r}^{(\mathrm{k})} - \mathrm{r}^{(\mathrm{k}-1)} = -M\left(v^{(k)} - v^{(k-1)}\right)$
  - $\beta^{(k)} = \frac{r^{(k)\,T}\, r^{(k)}}{r^{(k-1)\,T}\, r^{(k-1)}}$ // such that $d^{(k)}$ conjugate with $d^{(k-1)}$
  - $d^{(k)} = r^{(k)} + \beta^{(k)} d^{(k-1)}$
- Works for SPD matrices
  - Again, beware of BC for Poisson problems
- Convergence: $\|x - x_k\|_M \leq 2\left(\frac{\sqrt{\lambda_{max}} - \sqrt{\lambda_{min}}}{\sqrt{\lambda_{max}} + \sqrt{\lambda_{min}}}\right)^k \|x - x_0\|_M$

Solving the Poisson Equation

- These methods don't require building the matrix M
  - Only need applying matrix M to vector v
  - That's fortunate: even if matrices are sparse, direct solver can eat much memory

- In many cases (not Poisson), need preconditioners
  - Solver converge better when eigenvalues not too spread
  - Instead solve : $P\,Mv = Pf$ with $P \approx M^{-1}$
    - Jacobi preconditioner: $P = diag(M)^{-1}$
    - ICP: Incomplete Cholesky (e.g., a band of Cholesky)
    - Or any iterations we've seen so far (e.g., solve with CG, use multigrid preconditioner)

Solving the Poisson Equation

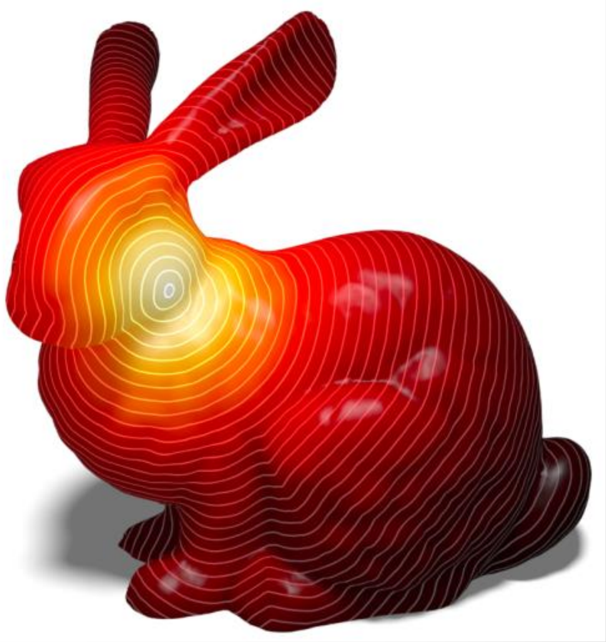
- Fourier-based approach
  - $\Delta v = f \quad \Leftrightarrow \quad \mathcal{F}(\Delta v) = \mathcal{F}(f)$
    $\Leftrightarrow 4\pi^2|\xi|^2\mathcal{F}(v) = \mathcal{F}(f)$
  - Numerically:
    - When periodic BC, use FFT (and then inverse FFT) : $\hat{v} = \frac{h^2\hat{f}}{2\left(\cos\frac{\pi m}{M} + \cos\frac{\pi n}{N} - 2\right)}$
    - When Dirichlet BC ($v = 0$), use DST : $\hat{v} = \frac{h^2\hat{f}}{2\left(\cos\frac{\pi m}{M} + \cos\frac{\pi n}{N} - 2\right)}$
    - When Neumann BC ($\nabla v = 0$), use DCT : $\hat{v} = \frac{h^2\hat{f}}{2\left(\cos\frac{\pi m}{M} + \cos\frac{\pi n}{N} - 2\right)}$

Solving the Poisson Equation

- Green's kernel approach
  - Given solution of $\Delta G = \delta$ with $G = 0$ on $\partial\Omega$,
    - Solution of $\Delta v = 0$ with $v = 0$ on $\partial\Omega$ is $v = G * f$
    - Proof: $\Delta v = \Delta(G * f) = (\Delta G) * f = \delta * f = f$
  - Green's kernel for Δ (in 2D) : $G(\rho) = \frac{1}{2\pi} \ln \rho$
  - Green's kernel for 2d diffusion: $\frac{\partial}{\partial t} - k\Delta$: $G(t, \rho) = H(t) \frac{1}{4\pi\,\mathrm{k}\,\mathrm{t}} \exp\left(-\frac{\rho^2}{4\,k\,t}\right)$
    - Gaussian convolutions

Application

- Geodesic computation
  - Varadhan's formula: $d(x,y) = \lim_{t\to 0}\sqrt{-4t\log\left(k_{t,x}(y)\right)}$
    - $k_{t,x}(y)$ heat kernel: heat transferred from x to y after time t
    - Too sensitive to errors
  - We know that $|\nabla d| = 1$ (Eikonal equation)
  - Instead only consider ∇v of correct direction
    - $v - t\,\Delta v = 0$ on $M \setminus \gamma$
    - $v(0,x) = 1$ on γ
    - Take just one Euler step to obtain $v(\epsilon, x)$
    - Consider vector field $X = \frac{\nabla v}{|\nabla v|}$
    - Solve Poisson eq. $\Delta d = \nabla . X$



"Geodesics in Heat: A New Approach to Computing Distance Based on Heat Flow", [Crane et al. 2013]

Application

- Discretization of Δ on meshes – see with Julie next time
- Solver:
  - Iterative ?
  - Advocate for Cholesky factorization: eats memory and slow but can be reused
    - Only depends on mesh (no BC!)
    - Gives $\Delta = LL^T$, solved via backsubstitution

Poisson Equation for Fluid simulation

The Navier-Stokes equations

$$\frac{\partial \vec{u}}{\partial t} + \vec{u}.\nabla\vec{u} + \frac{1}{\rho}\nabla \mathrm{p} = \vec{\mathrm{g}} + \nu\Delta\vec{u}$$

$$\nabla.\vec{u} = 0$$

$\frac{D\vec{u}}{Dt}$

(e.g., consider $\frac{d}{dt}f(\vec{x}(t), t)$ and use chain rule)

~ acceleration

Incompressibility

$m\,\dot{v} = \sum forces$

Forces: ∇p , $\rho\vec{g}, \rho\nu\Delta\vec{u}$

Viscosity:
deviation of $\vec{u}$
from average

- "Fluid Simulation for Computer Graphics", Bridson 2008 (book)

Simple Fluid Solver

- First $\overrightarrow{u'}$ = advect $(\vec{u}) + \Delta t\ \vec{g}$ based on interpolation
- Then $\overrightarrow{u''} = project(\overrightarrow{u'})$
  - $\overrightarrow{u''} = \overrightarrow{u'} - \Delta t \frac{1}{\rho} \nabla p$
  - Find p such that $\overrightarrow{u''}$ incompressible: $\nabla . \overrightarrow{u''} = \nabla . \overrightarrow{u'} - \frac{\Delta t}{\rho} \Delta p = 0$
  - i.e., solve the Poisson equation $\Delta p = \frac{\rho}{\Delta t} \nabla . \overrightarrow{u'}$

(we dropped viscosity: this actually the inviscid Euler equations ; Though numerical errors will lead to some viscosity anyway ; could other add a timestep or implicit solve of viscous term)

Bonus

Cool Image and Video processing without Poisson

Bonus: Seam carving

Resize

(no Poisson here!)

Seam carving

Crop

Seam carving

Seam Carving

Seam Carving

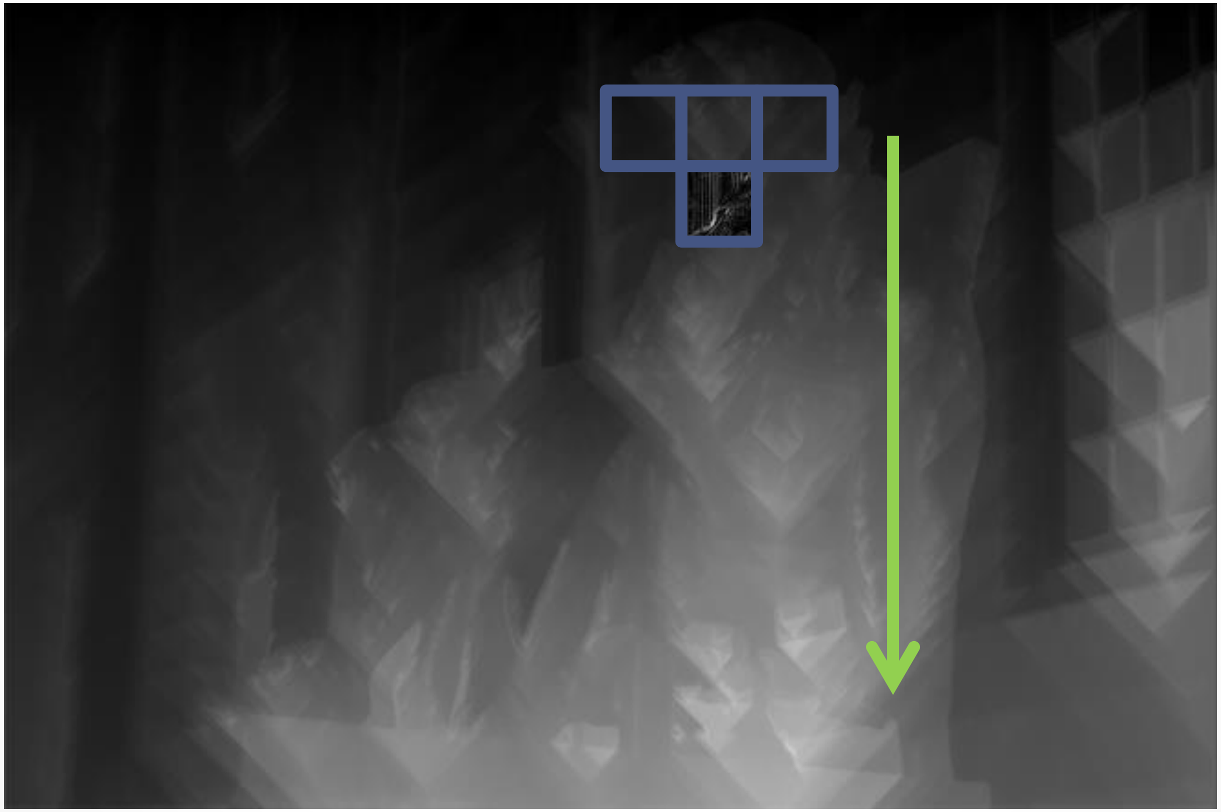
- "Seam Carving for Content-Aware Image Resizing", Avidan and Shamir 2007

Seam Carving

Seam Carving

E(x,y)

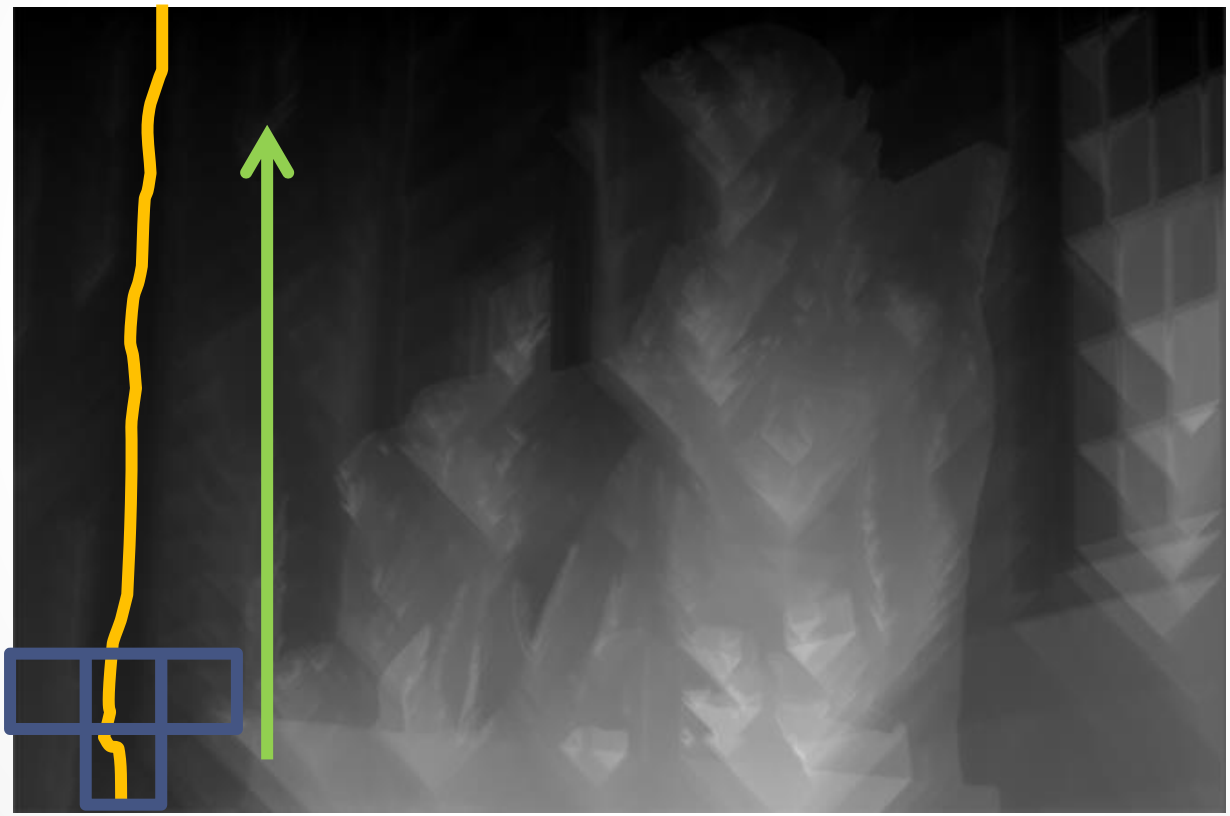
Seam Carving



Dynamic programming:

$V(x,y) = \min(V(x-1,y-1), V(x,y-1), V(x+1,y-1))$+E(x,y)

Seam Carving



Backtracking

Seam Carving

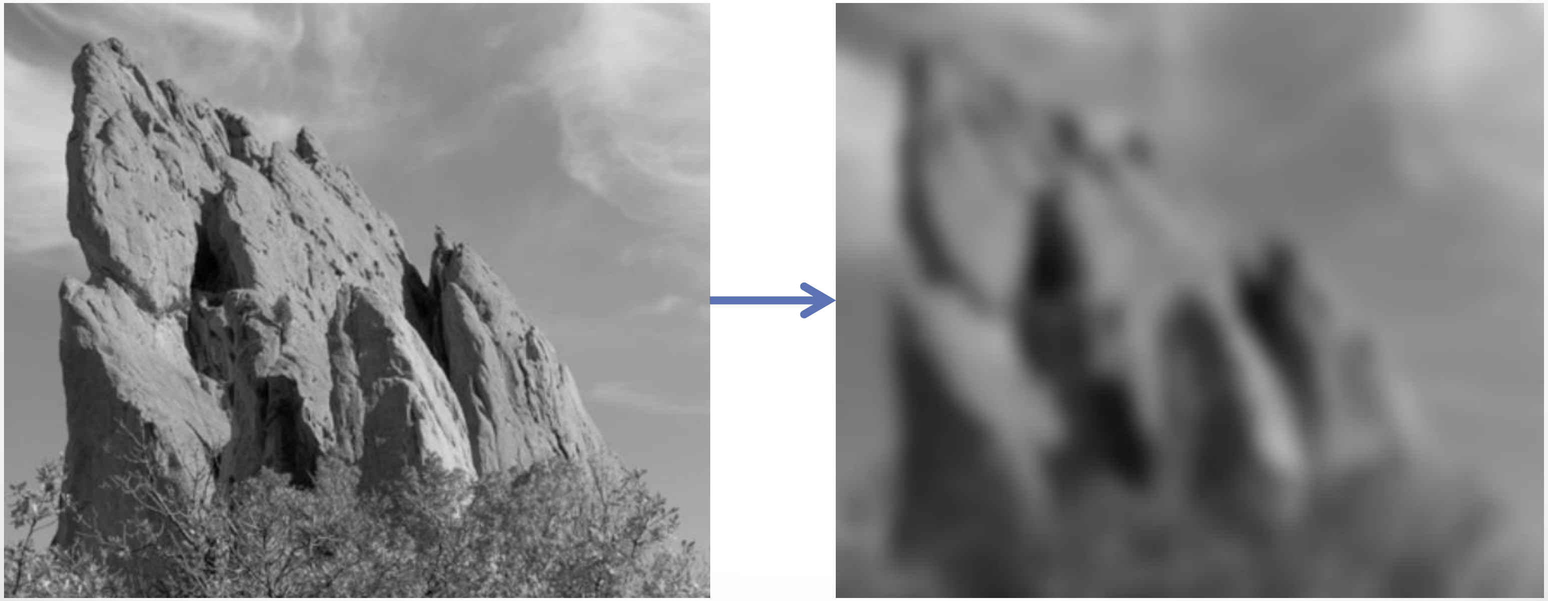
Bonus:
Bilateral Filter

- "A Gentle Introduction to Bilateral Filtering and its Applications" Paris et al. 2008 [course]

ie. Blur.

- Blur : Each pixel is a weighted average of its neighbors:

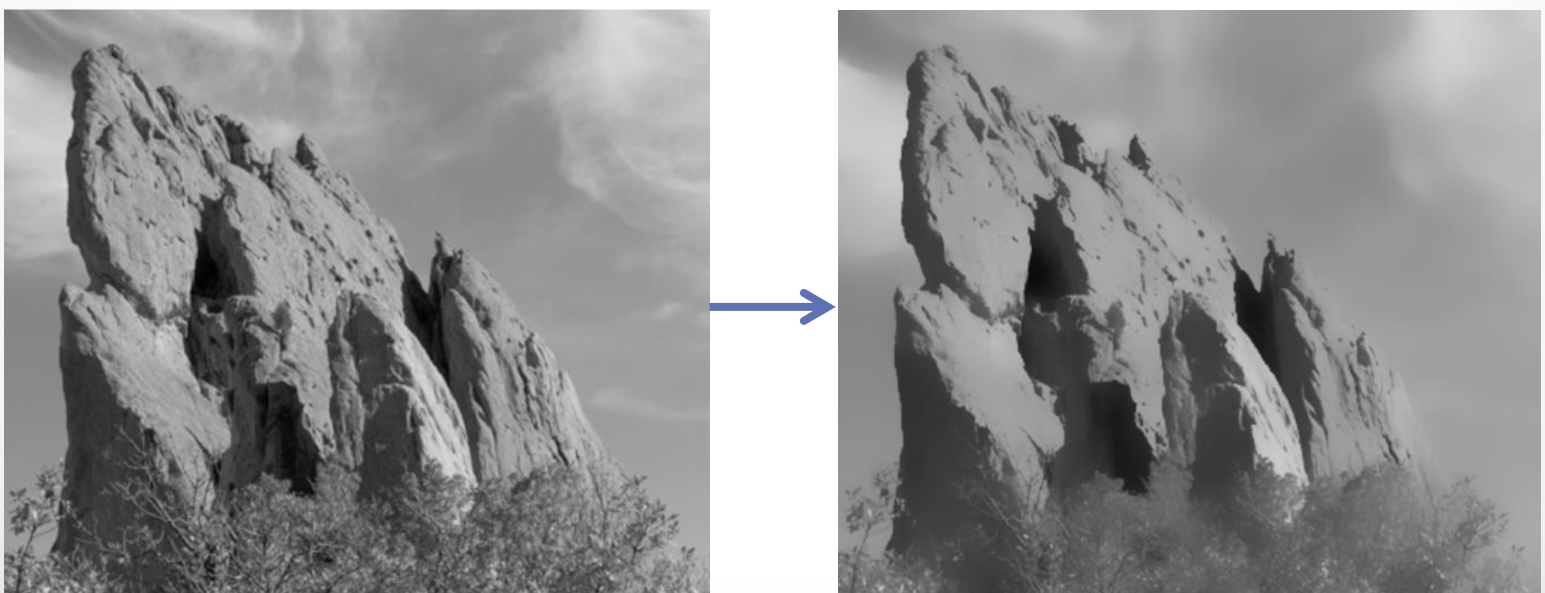
$$I(x,y) = \sum_{i=-K}^{K} \sum_{j=-K}^{K} w(i,j).I_{x+i,y+j}$$



ie. (more clever) Blur.

- Bilateral filter : weights account for intensity

$$I(x,y) = \frac{1}{W_{x,y}} \sum_{i=-K}^{K} \sum_{j=-K}^{K} w(i,j).w'(|I_{x+i,y+j} - I_{x,y}|).I_{x+i,y+j}$$



Bonus:Motion Magnification

Following slides from “Phase-Based Video Motion Processing”, [Wadhwa et al. 2013]

Goal

- Magnify motion:

(That's using a previous approach)

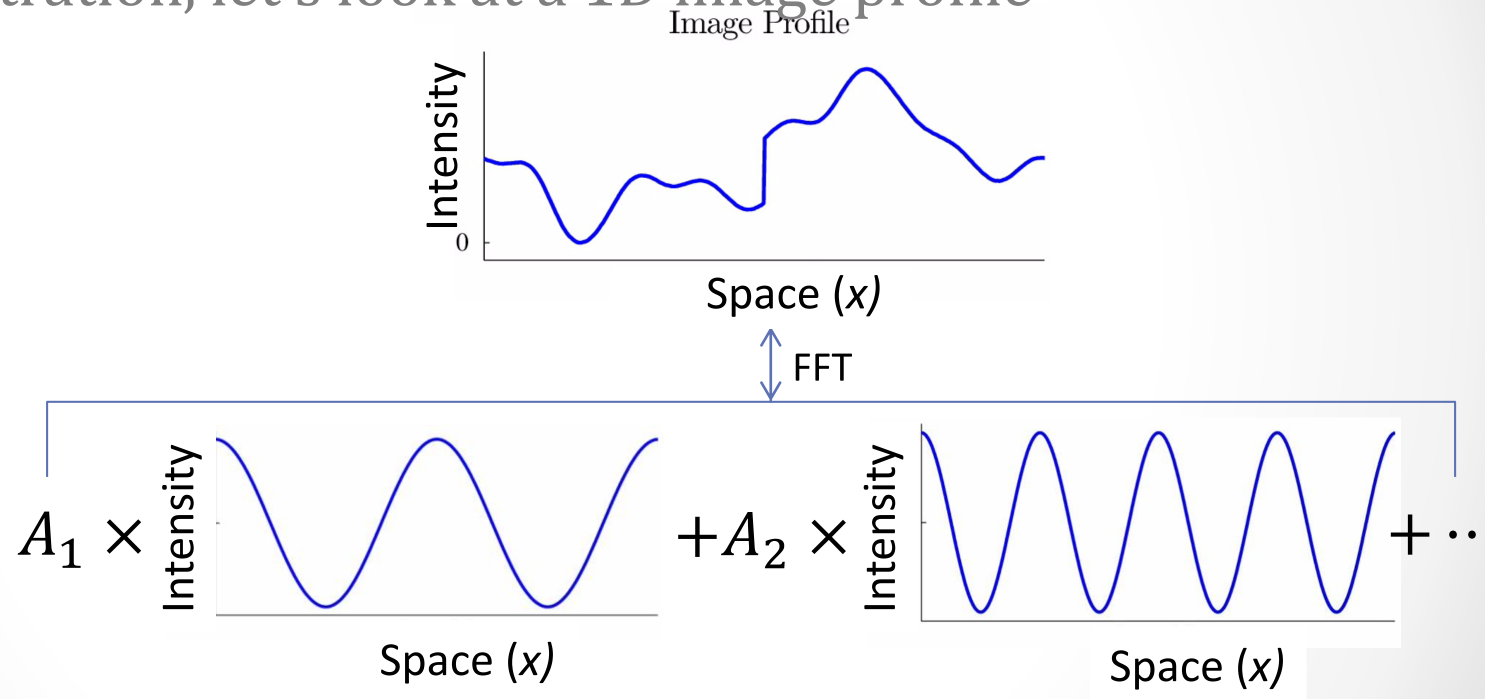
Fourier Decomposition

- For illustration, let's look at a 1D image profile

$$f(x)$$

FFT

$$\sum_{\omega=-\infty}^{\infty} A_\omega e^{i\omega x}$$

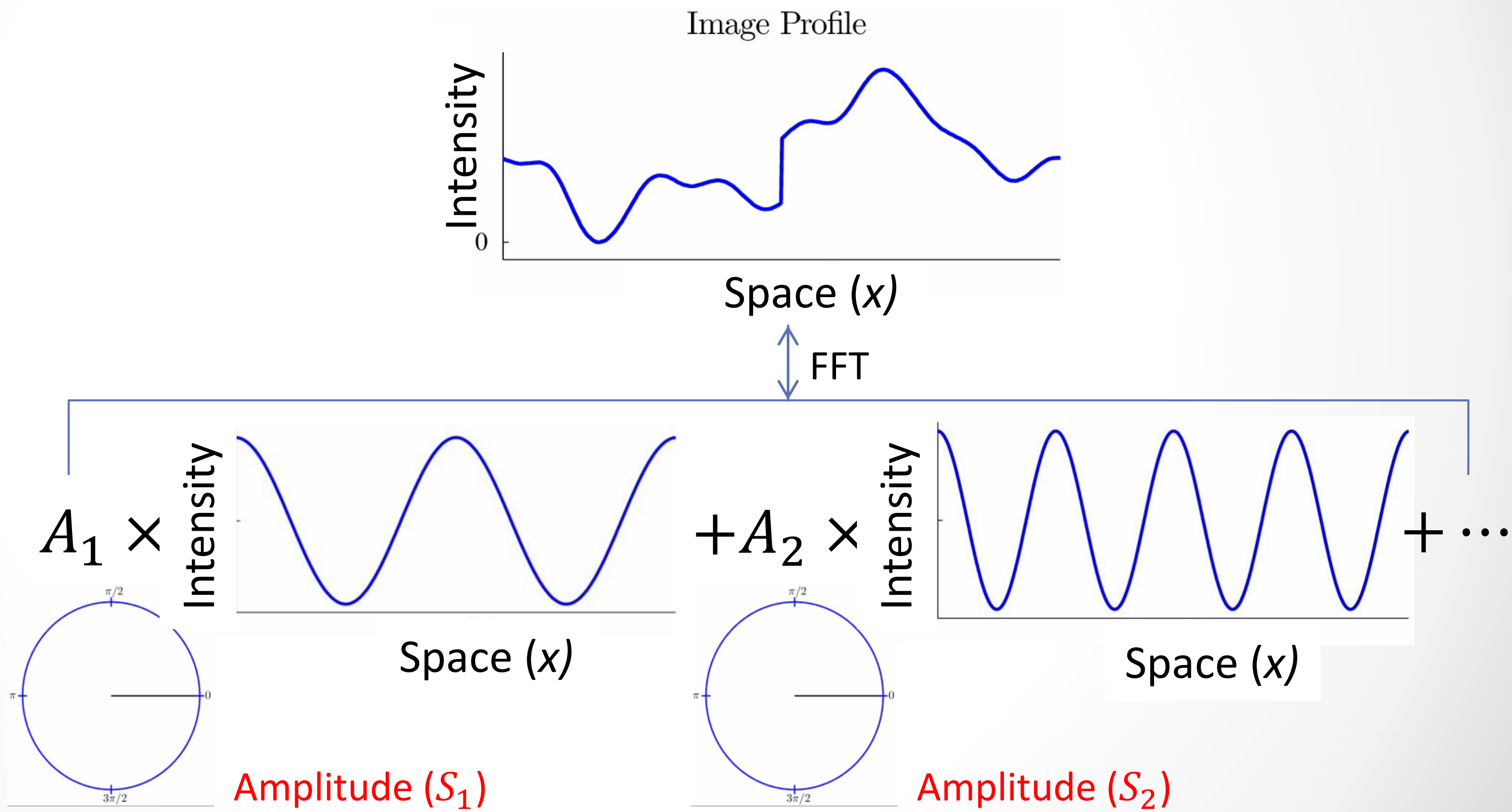


Amplitude of Basis Function

$f(x)$

FFT

$\sum_{\omega=-\infty}^{\infty} A_\omega e^{i\omega x} S_\omega$



Fourier Shift Theorem

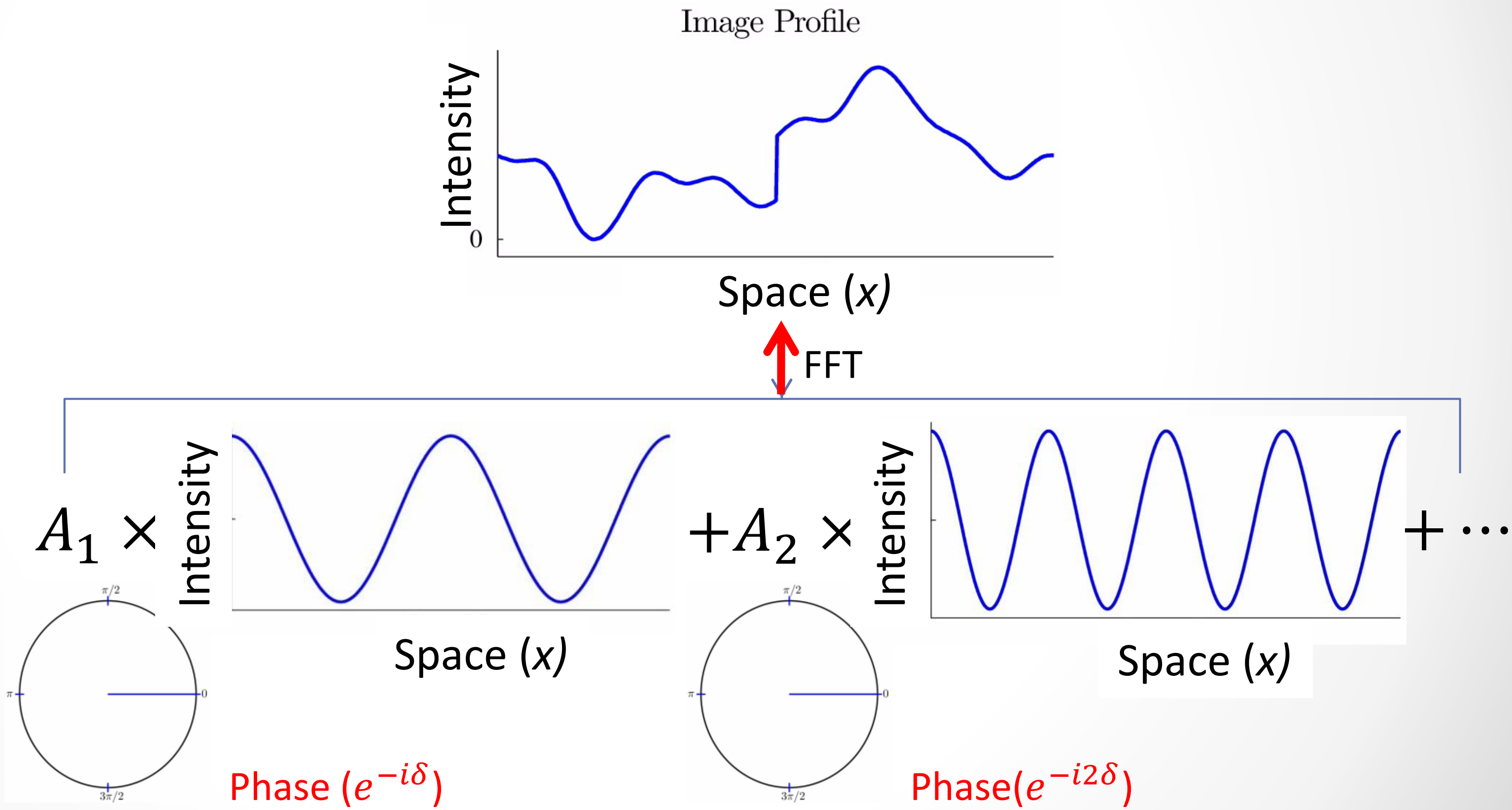
Phase Shift ↔ Translation

- Phase controls location of sinusoid

$f(x - \delta)$

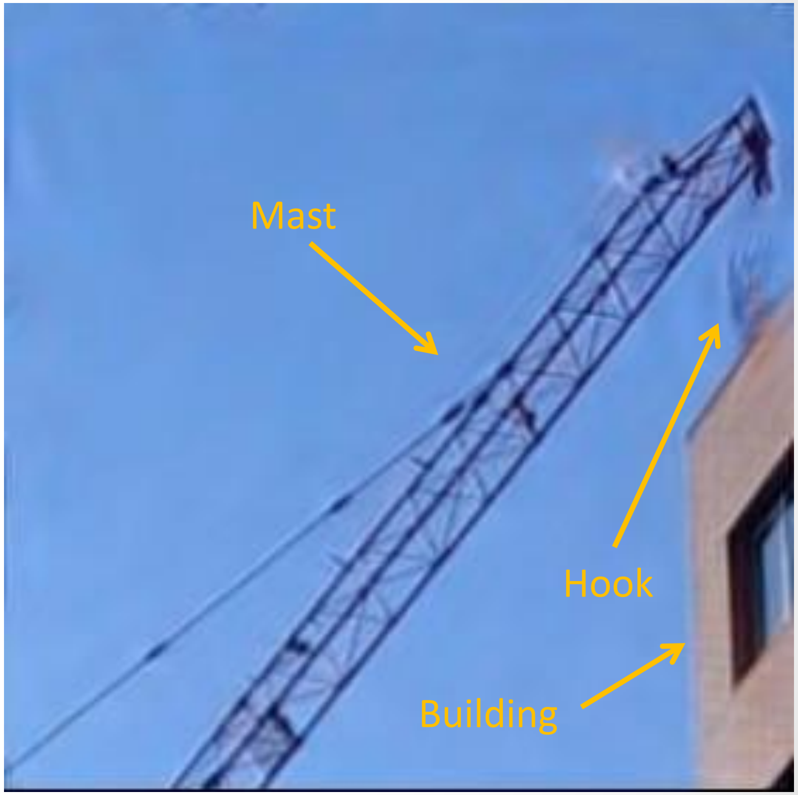
FFT

$$\sum_{\omega=-\infty}^{\infty} A_\omega e^{i\omega x} e^{-i\omega\delta}$$



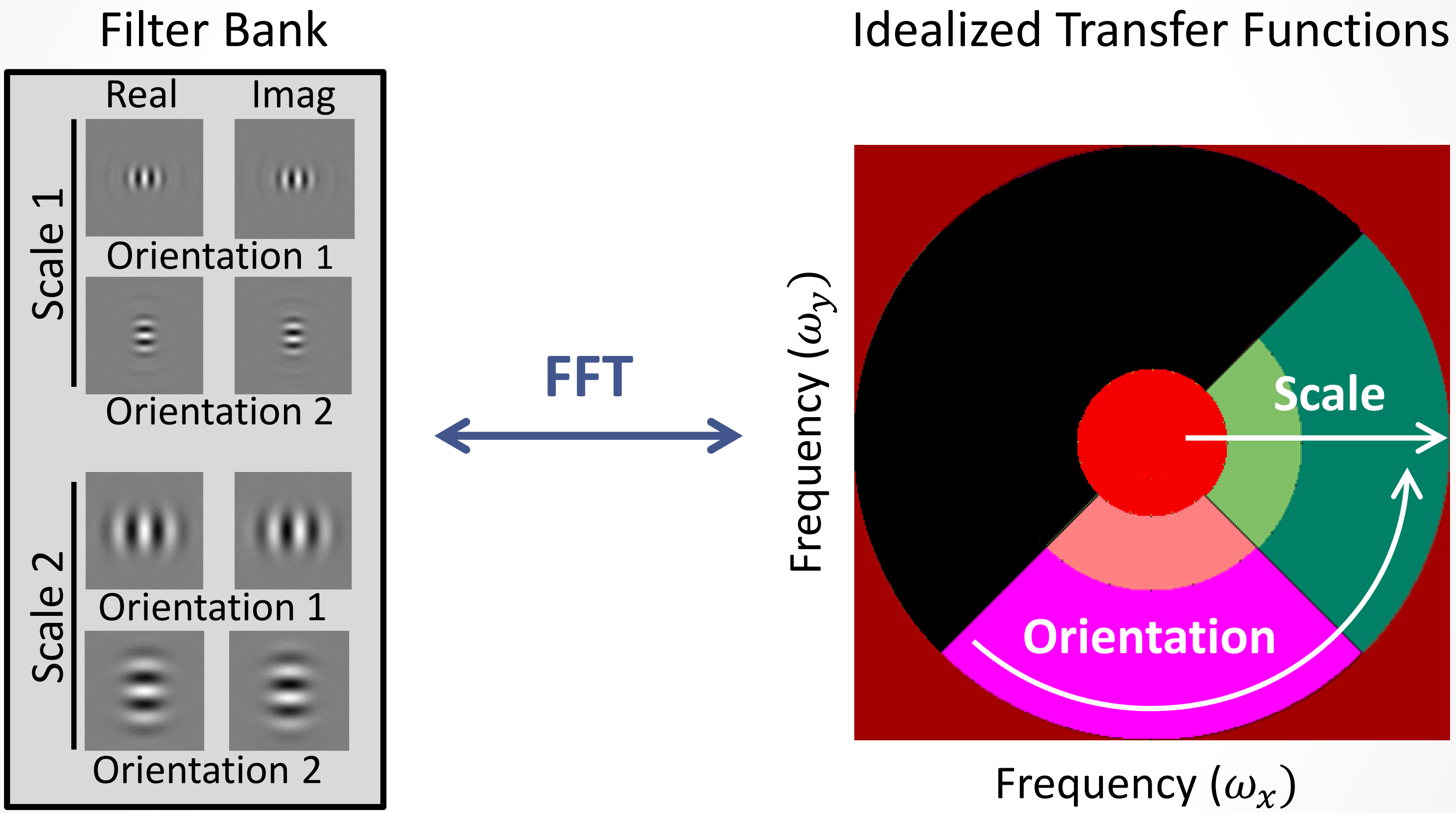
Local Motions

- Fourier shift theorem only lets us handle global motion
- But, videos have many local motions
- Need a localized Fourier Series for local motion

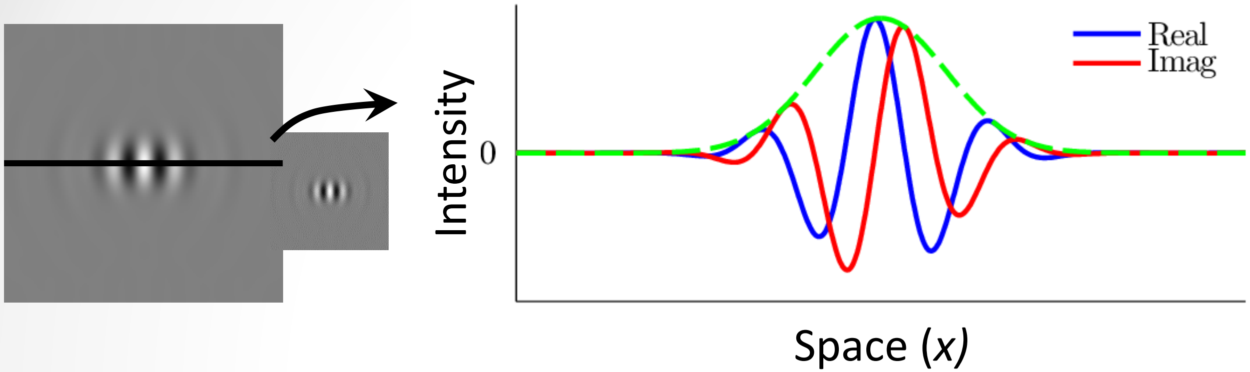


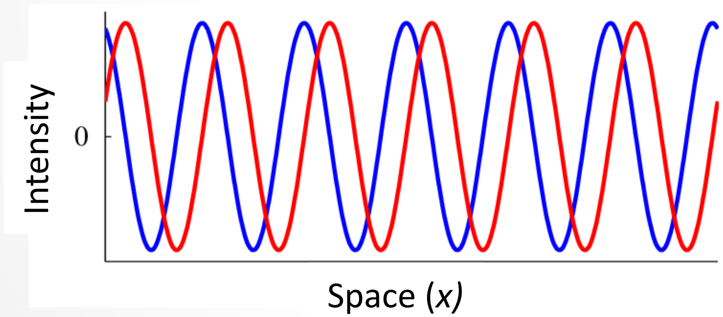
Complex Steerable Pyramid

[Simoncelli et al. 1992]



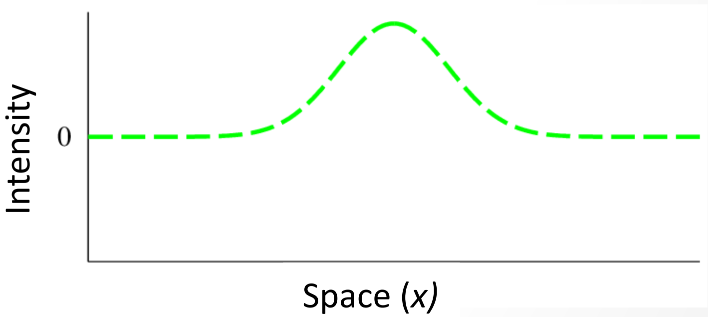
Complex Steerable Pyramid Basis Functions





Complex Sinusoid (Global)

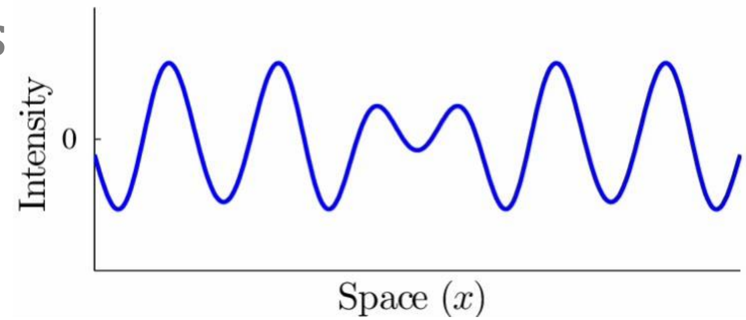
×



Window

Single Sub-Band (Scale)

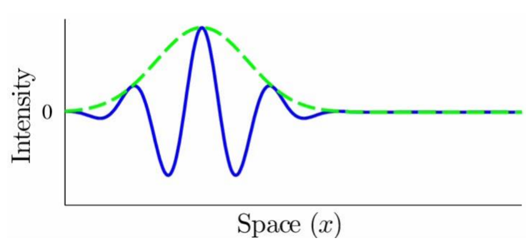
- In single scale, image is coefficients times translated copies of basis functions



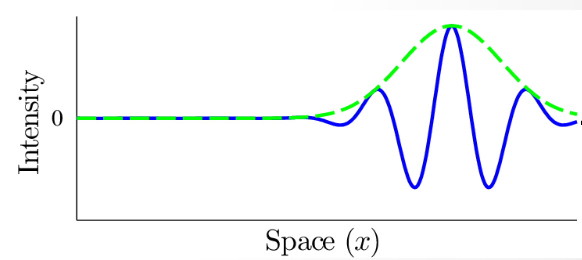
$=$

Single Sub-band of Image Profile

$A_1 \times$



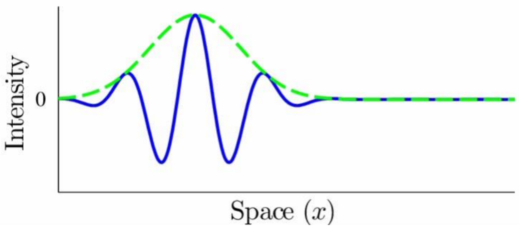
$+A_2 \times$



$+ \cdots$

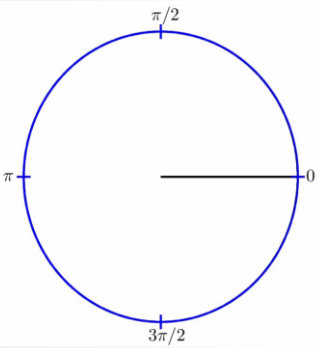
Single Sub-Band (Scale)

- In single scale, image is coefficients times translated copies of basis functions

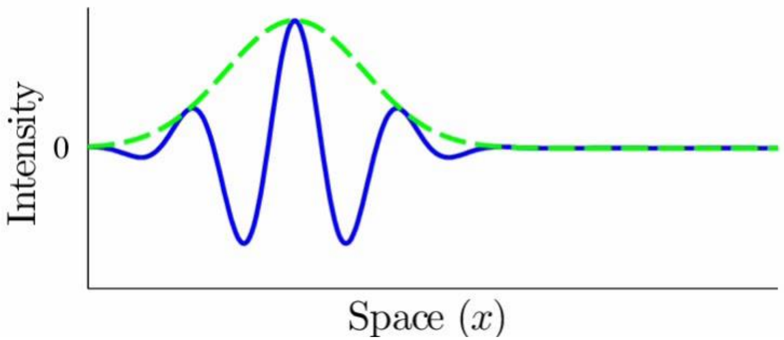
$A_1 \times$ 

Local Amplitude

- Local amplitude controls strength of basis function



$A_1 \times$

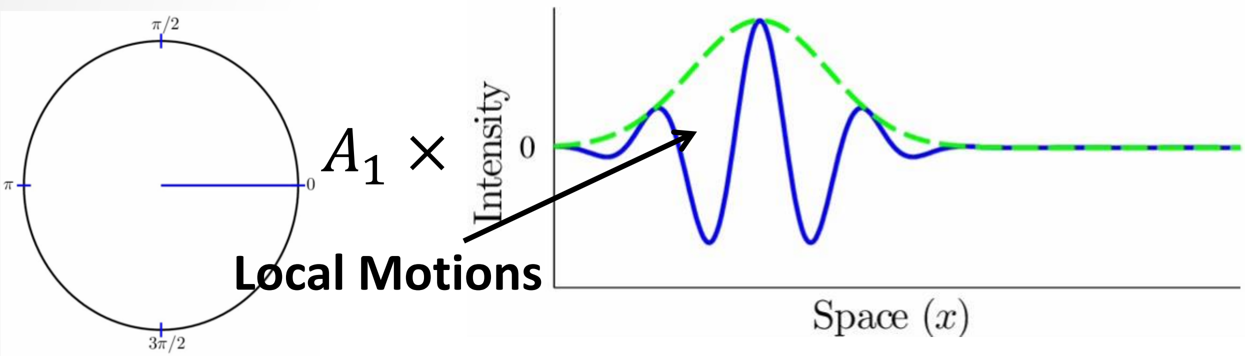


Local Amplitude

Local Phase

Local Phase Shift ↔ Local Translation

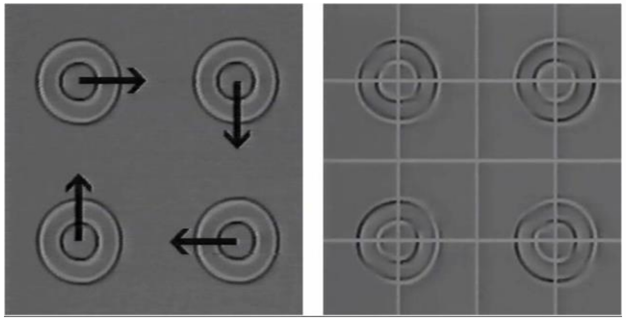
- Local phase controls location of sinusoid under window, approximates local translation



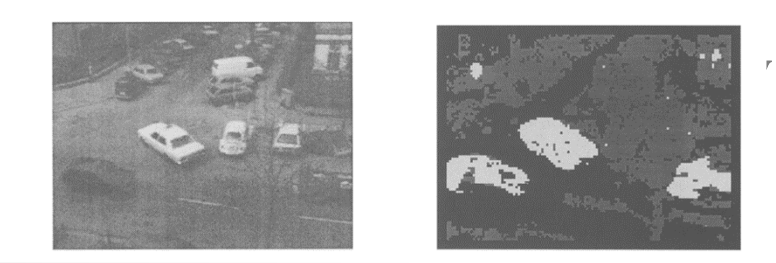
Local Phase

Phase and Motion

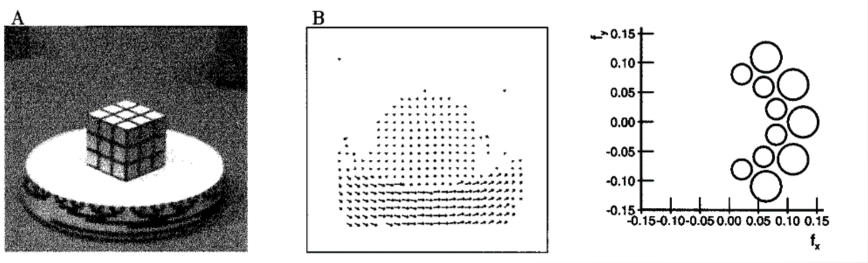
- Phase-based motion synthesis



Motion without Movement [Freeman et al. 1991]

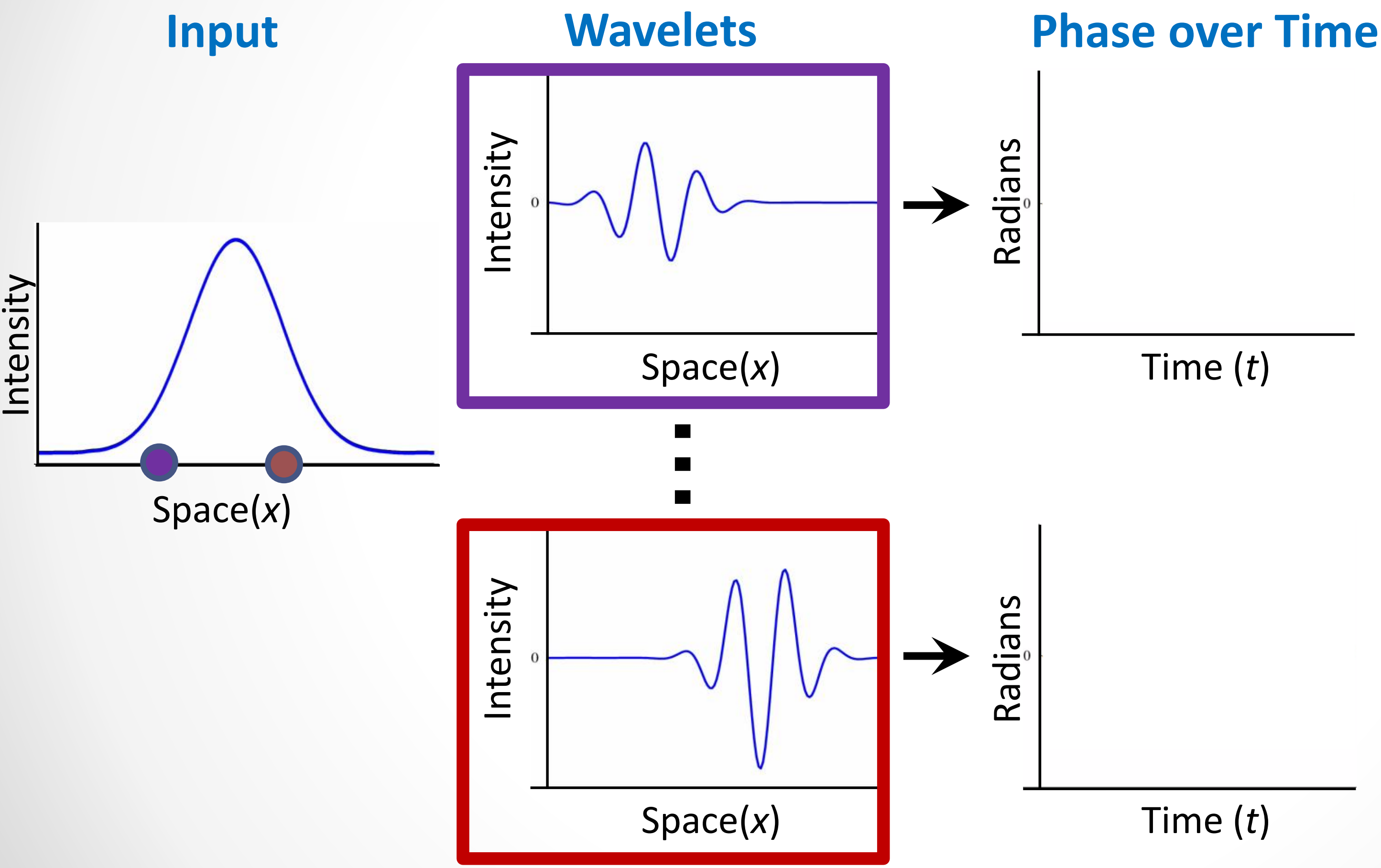


[Fleet and Jepson 1990]

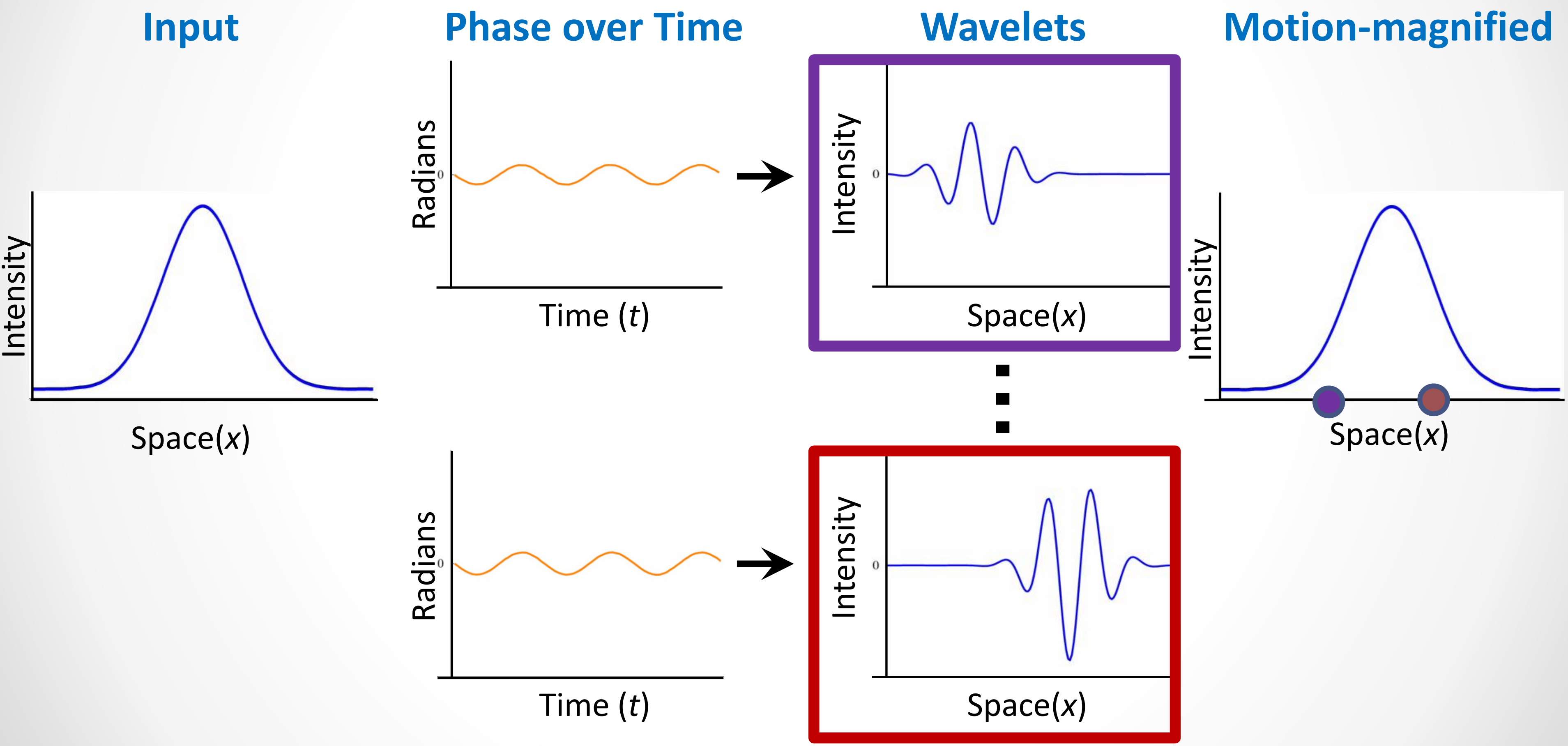


[Gautama and Van Hulle 2002]

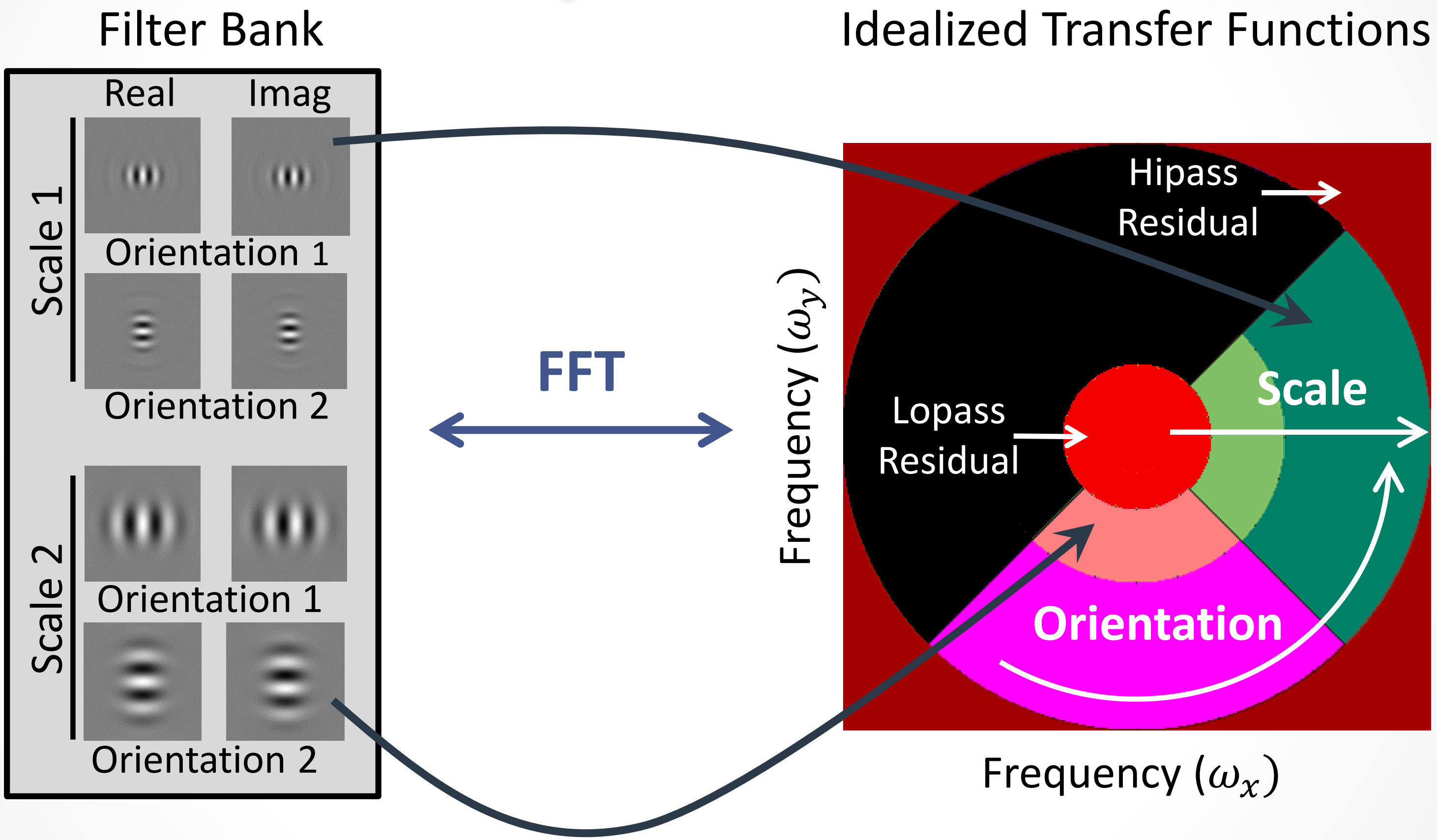
Phase over Time



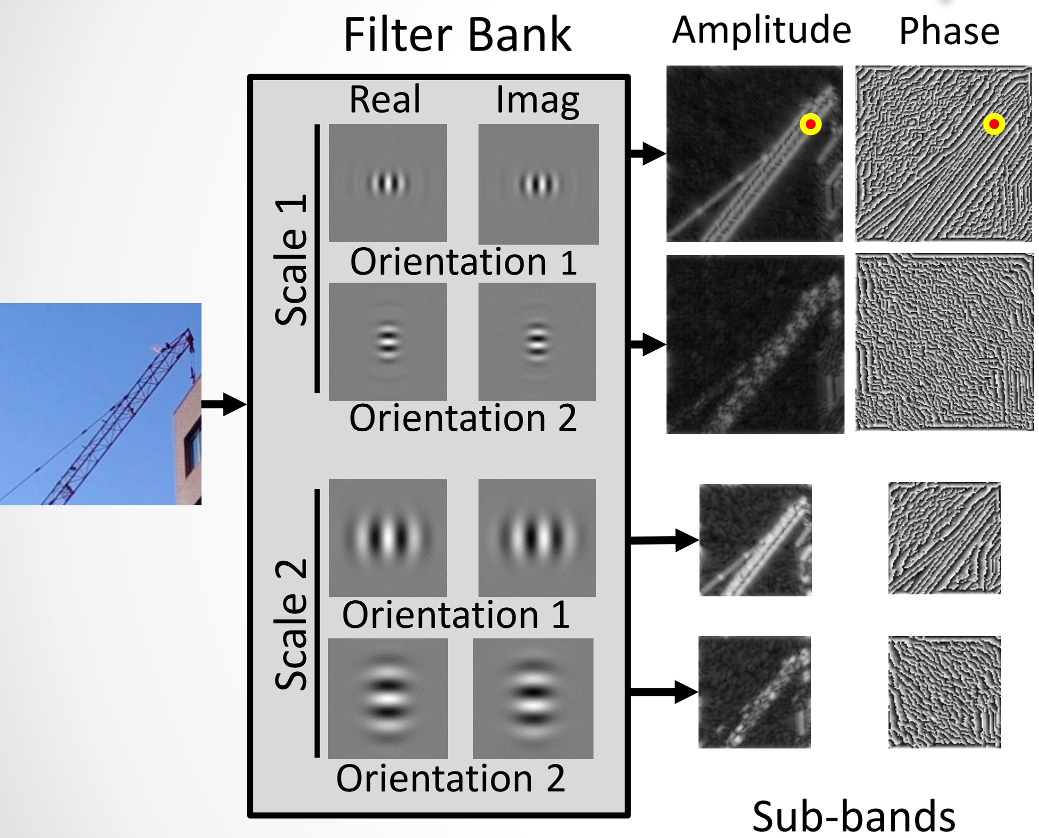
Phase over Time



2D Complex Steerable Pyramid

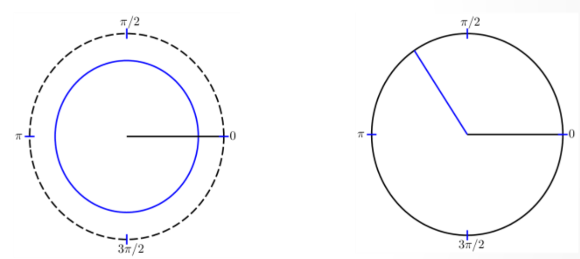


Complex Steerable Pyramid Decomposition

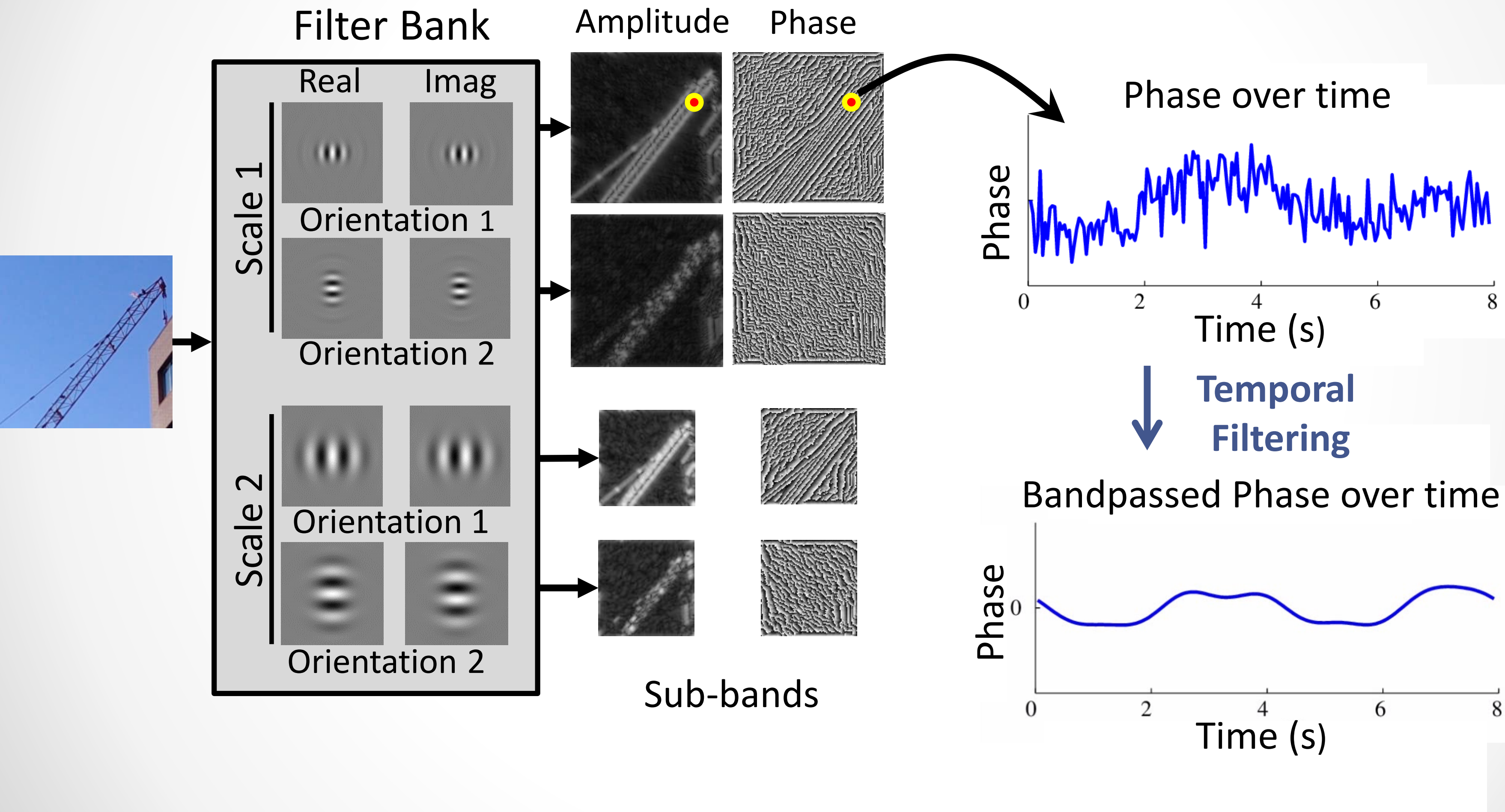


Amplitude Phase

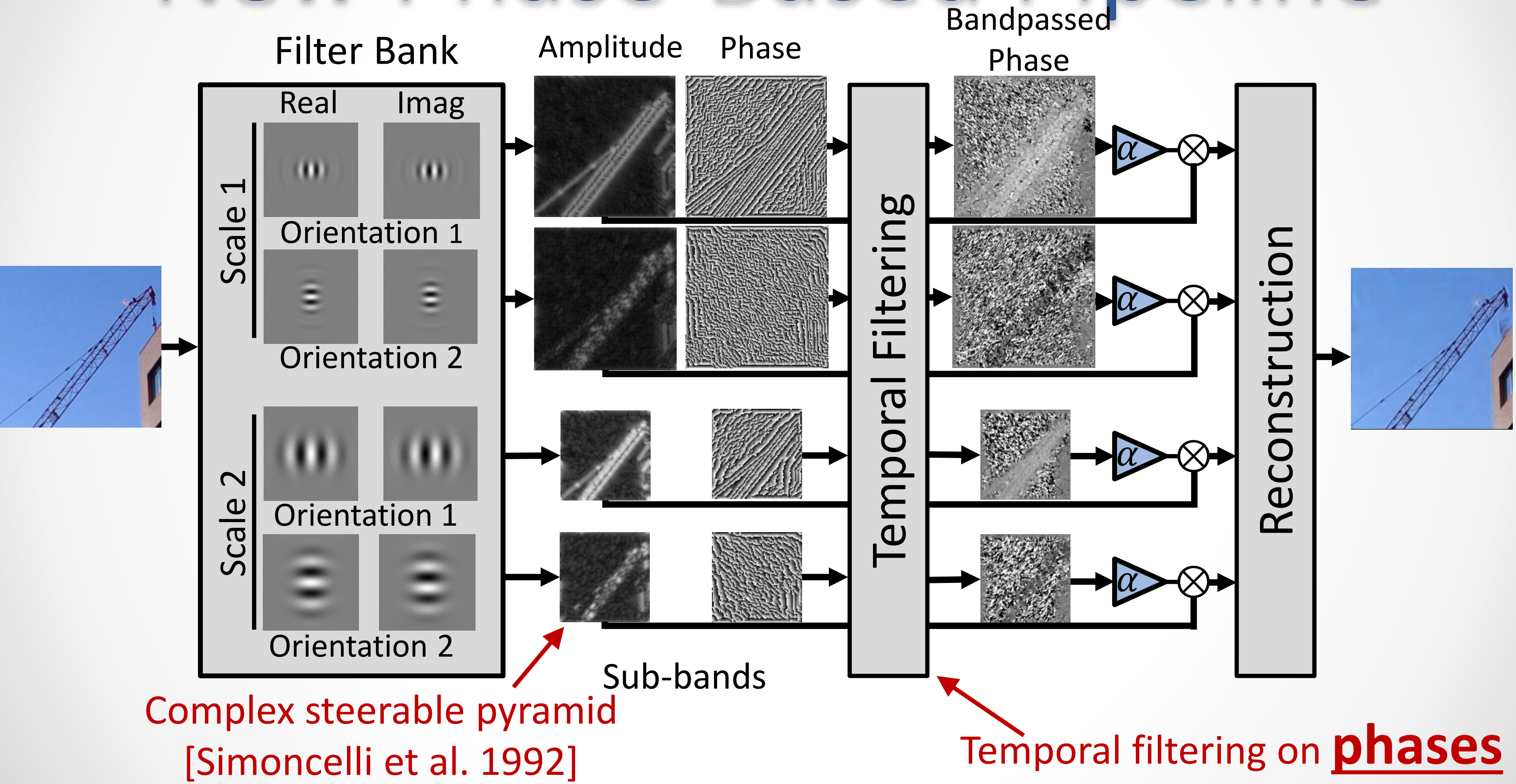
$$A \times e^{i\phi}$$



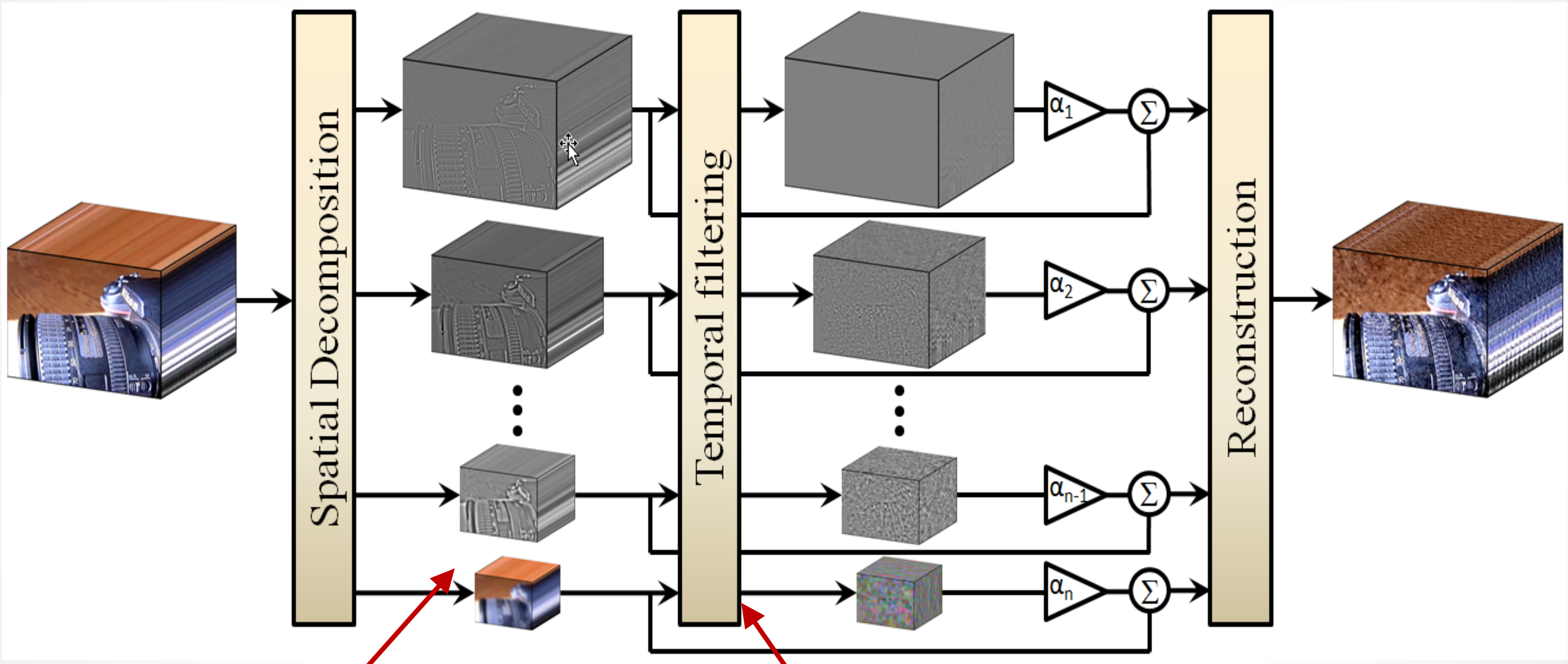
Phase over Time



New Phase-Based Pipeline



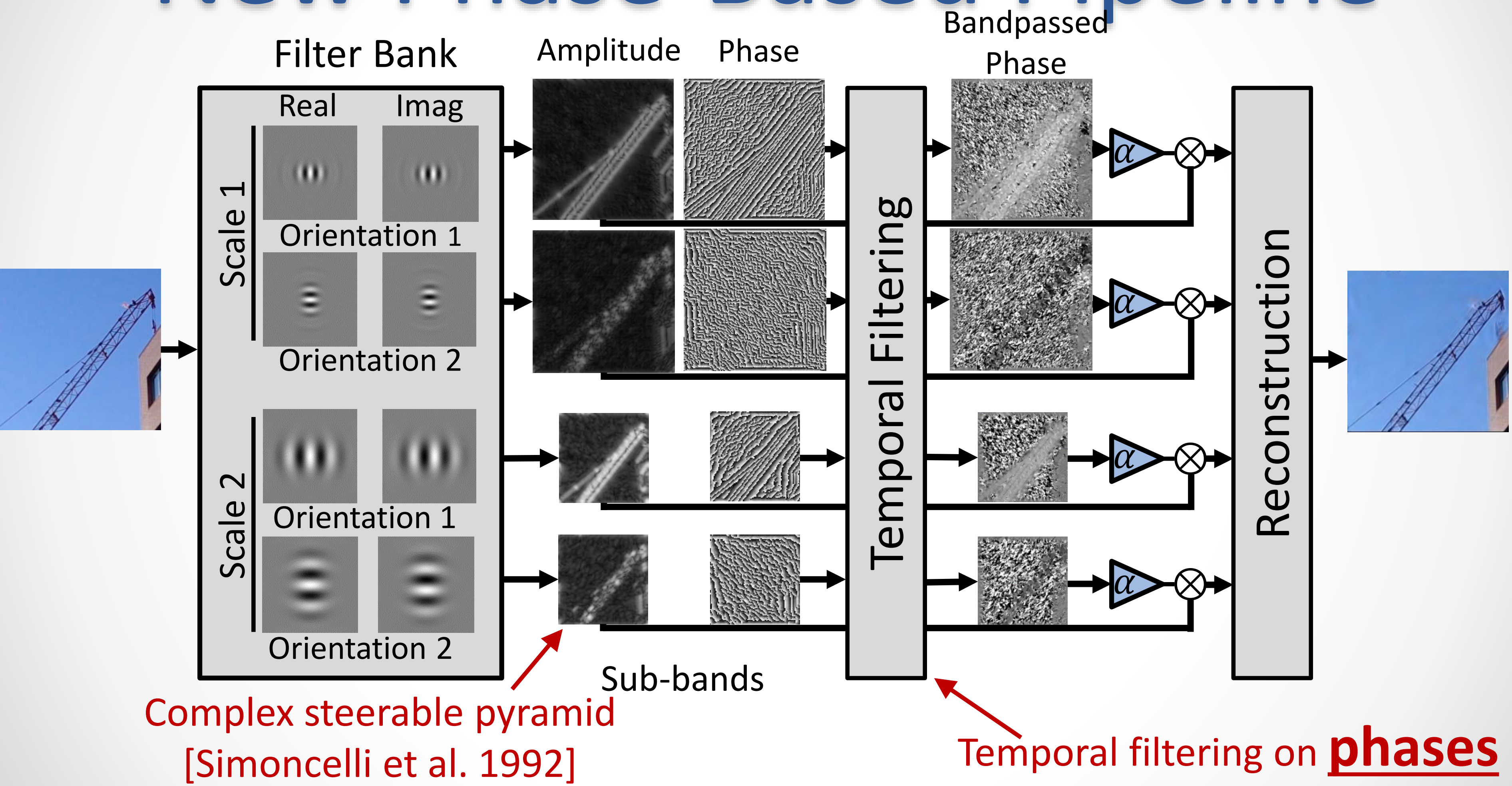
Linear Pipeline (Wu et al. 2012)



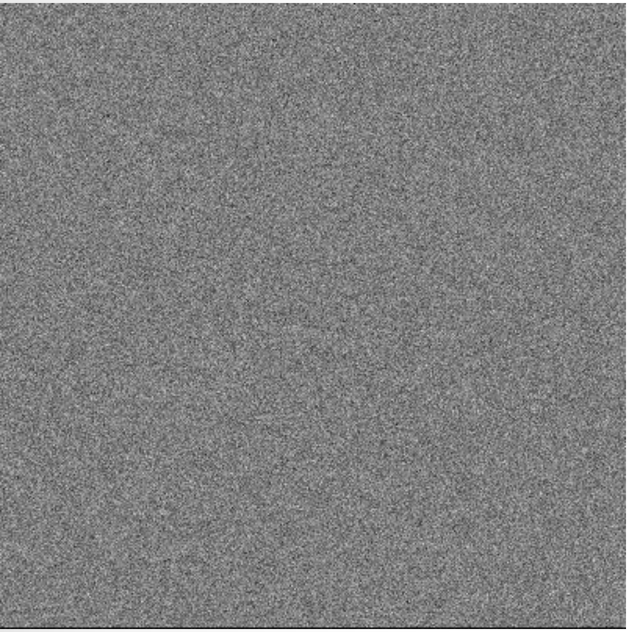
Laplacian pyramid
[Burt and Adelson 1983]

Temporal filtering on **intensities**

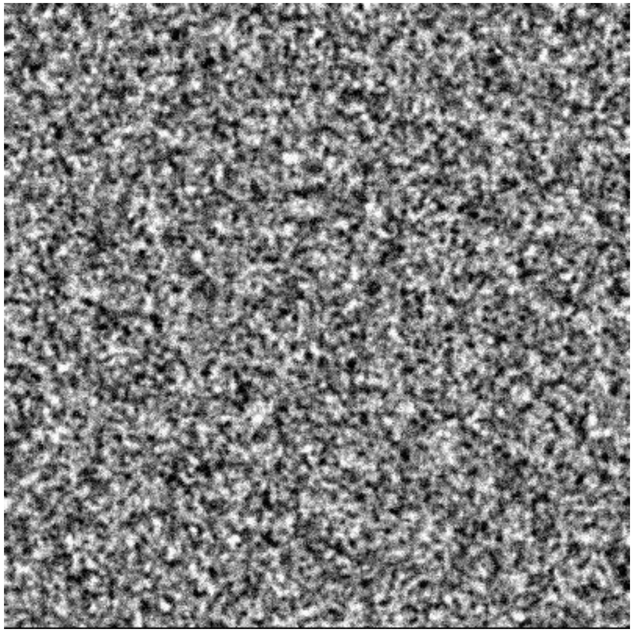
New Phase-Based Pipeline



Improvement #1: Less Noise

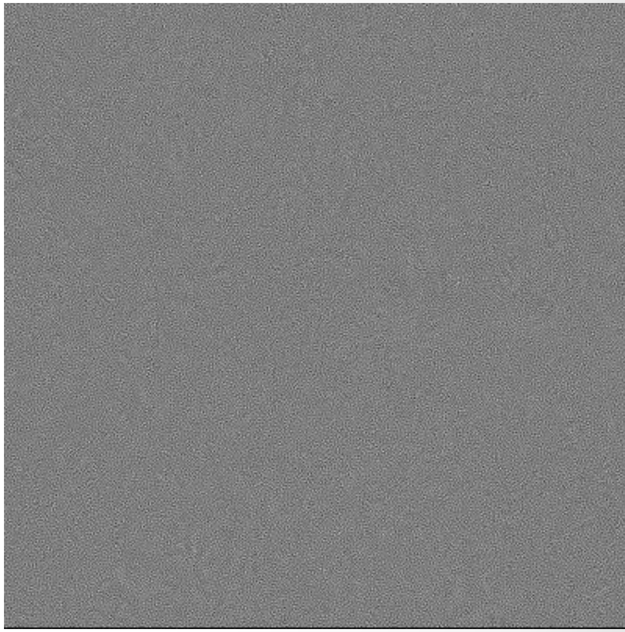


Source (IID Noise, std=0.1)



Linear [Wu et al. 2012] (x50)

Noise **amplified**

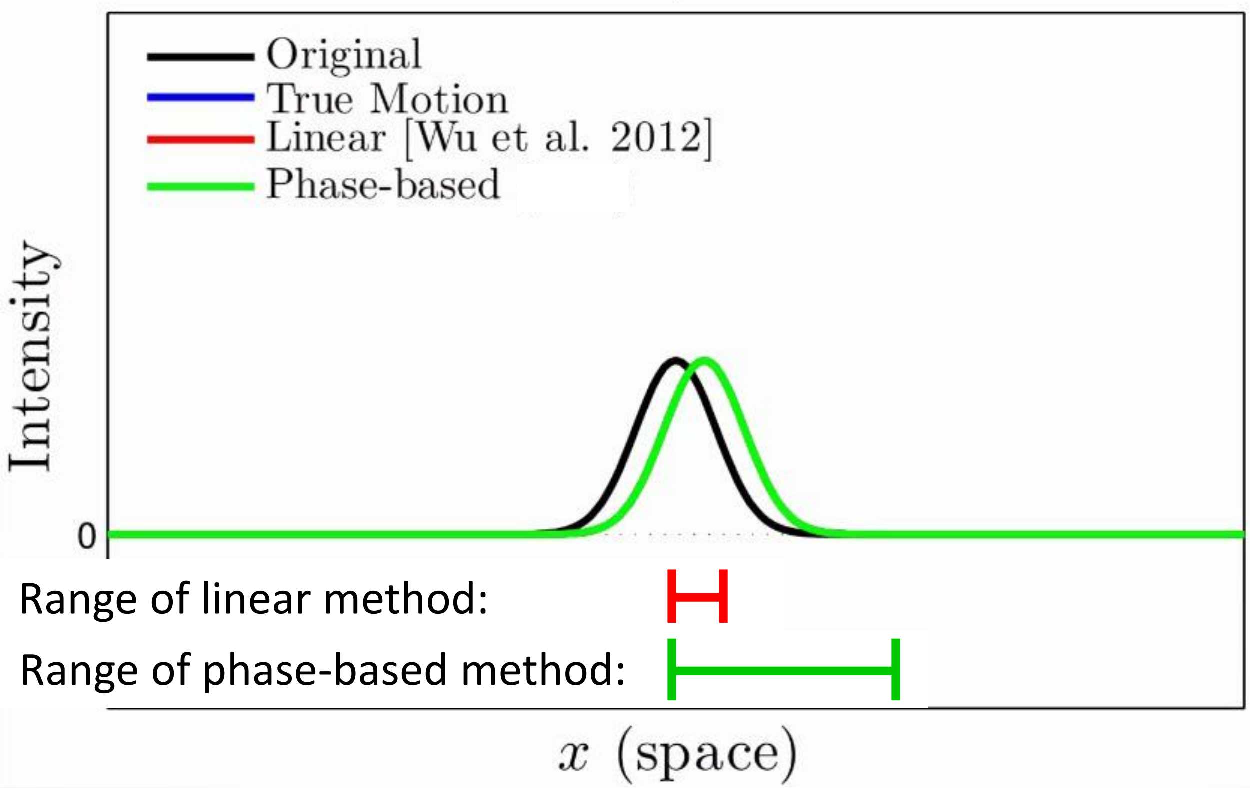


Phase-based (x50)

Noise **translated**

Improvement #2: More Amplification

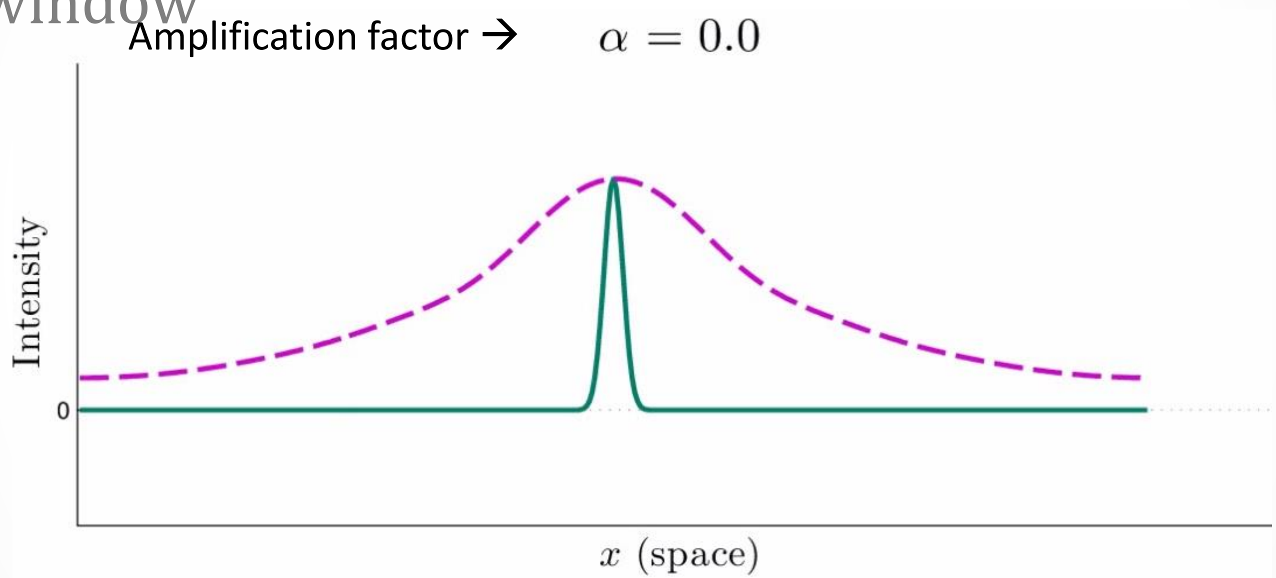
Amplification factor → $\alpha = 0.0,\ \delta = 0.1$ ← Motion in the sequence



4 times the amplification!

Limits of Phase Based Magnification

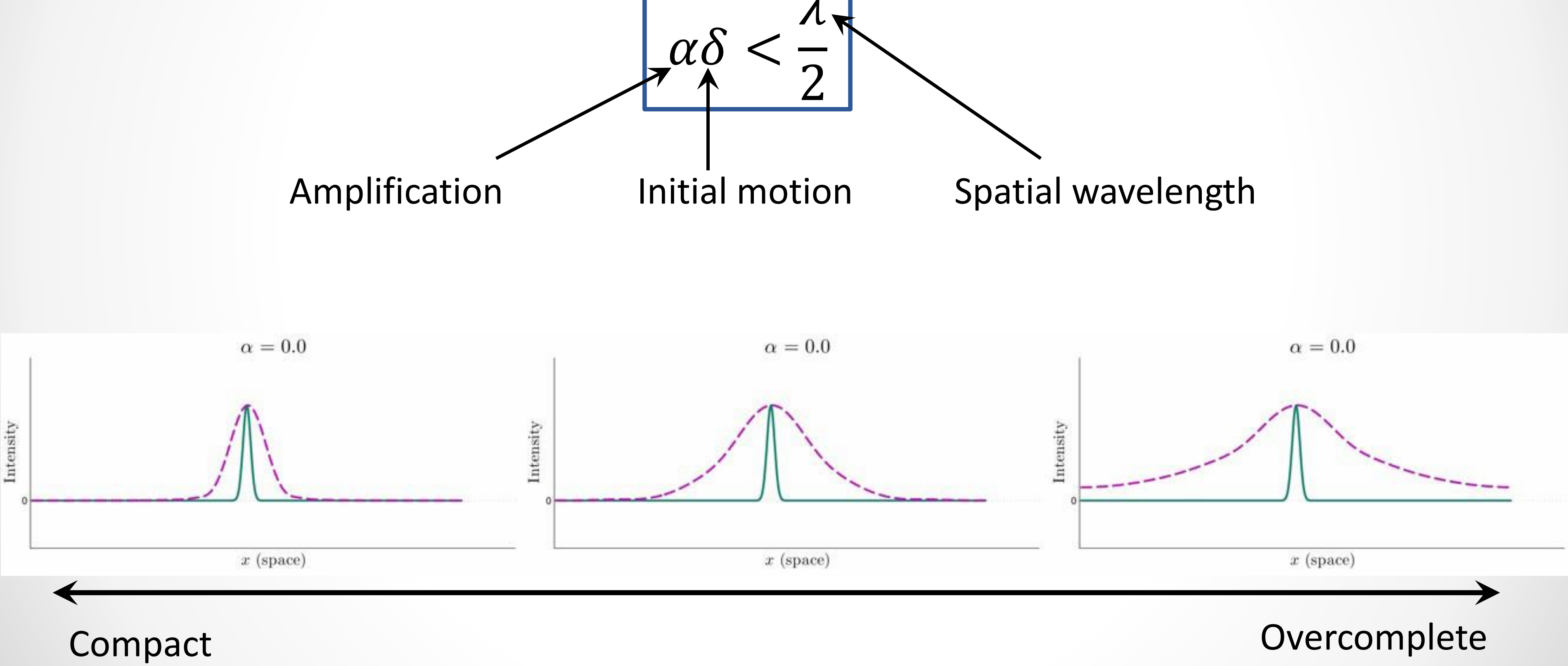
- Local phase can move image features, but only within the filter window



See Paper For…

- The bound on amplification

$$\alpha\delta < \frac{\lambda}{2}$$



Comparison with [Wu et al. 2012]

Wu et al. 2012

Vibration due to Camera's Mirror

Source (300 FPS)

Wu et al. 2012

Phase-based (this paper)

Comparison with [Wu et al. 2012] and Video Denoising


Wu et al.


Wu et al. with VBM3D


Wu et al. + Liu and Freeman 2010

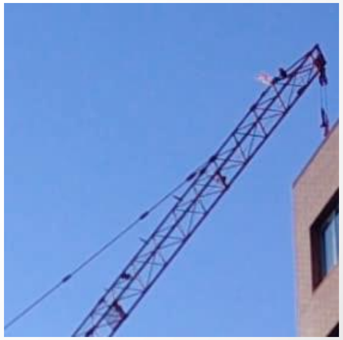

Phase-based (this paper)

Talk Overview

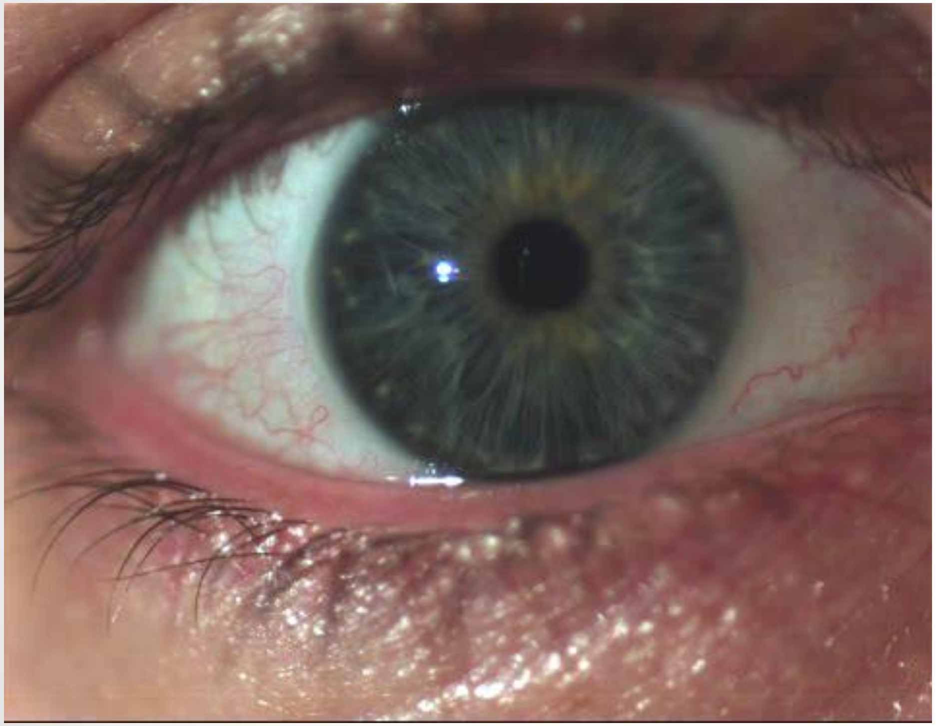
- Eulerian Video Magnification **[Wu et al. SIGGRAPH'12]**
  - Hao-yu Wu, Michael Rubinstein, Eugene Shih, John Guttag, Frédo Durand, William T. Freeman

- Phase-Based Video Motion Processing **[this paper]**
  - Neal Wadhwa, Michael Rubinstein, Frédo Durand, William T. Freeman

- Results, new applications, controlled sequences



Eye Movements



Source (500FPS)

Expressions

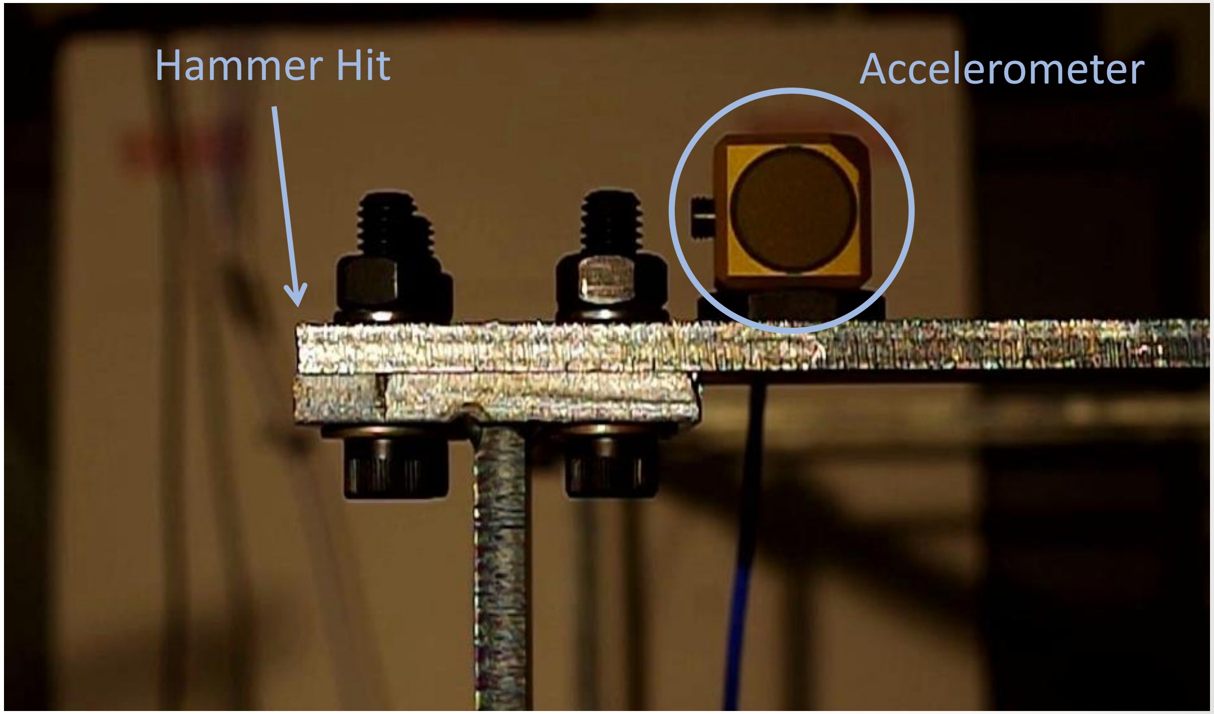
Source

Low frequency motions

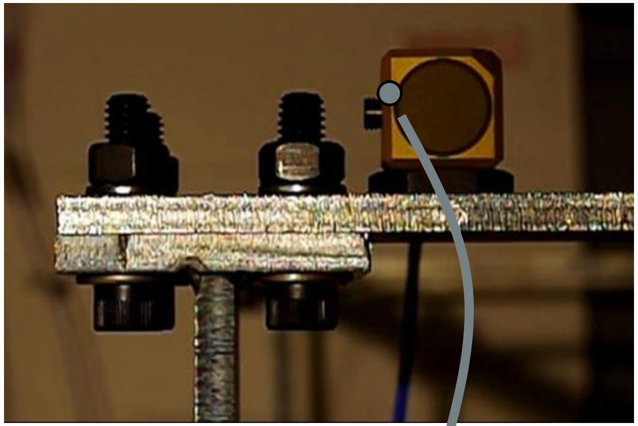
Mid-range frequency motions

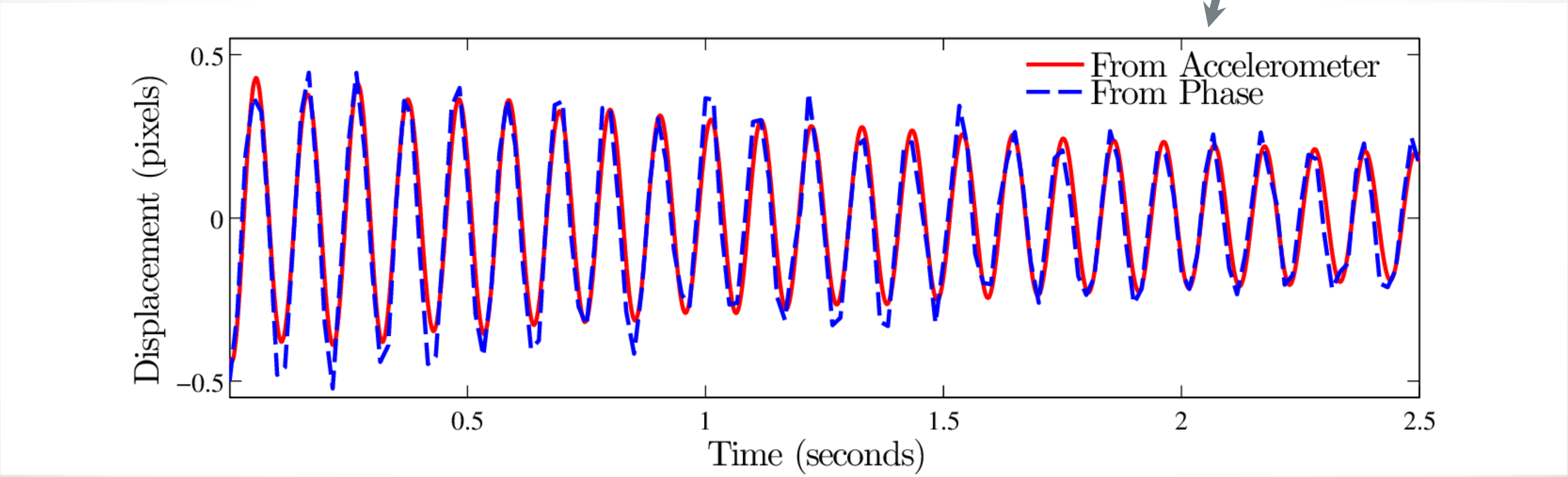
Ground Truth Validation

- Induce motion (with hammer)
- Record with accelerometer

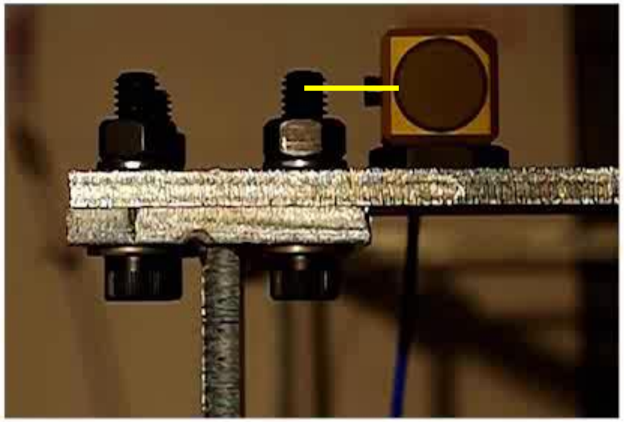


Ground Truth Validation





Qualitative Comparison



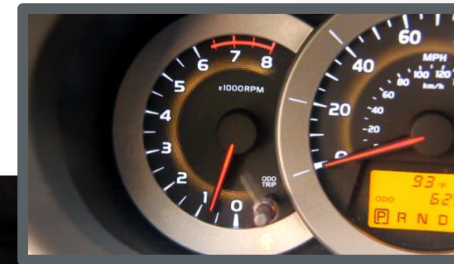
Input
(motion of 0.1 px)

Motion Attenuation

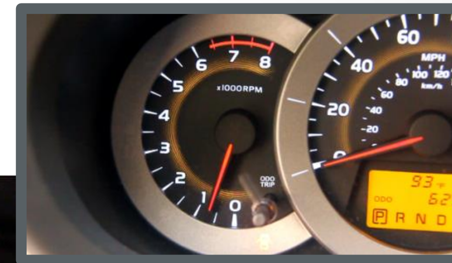
Source

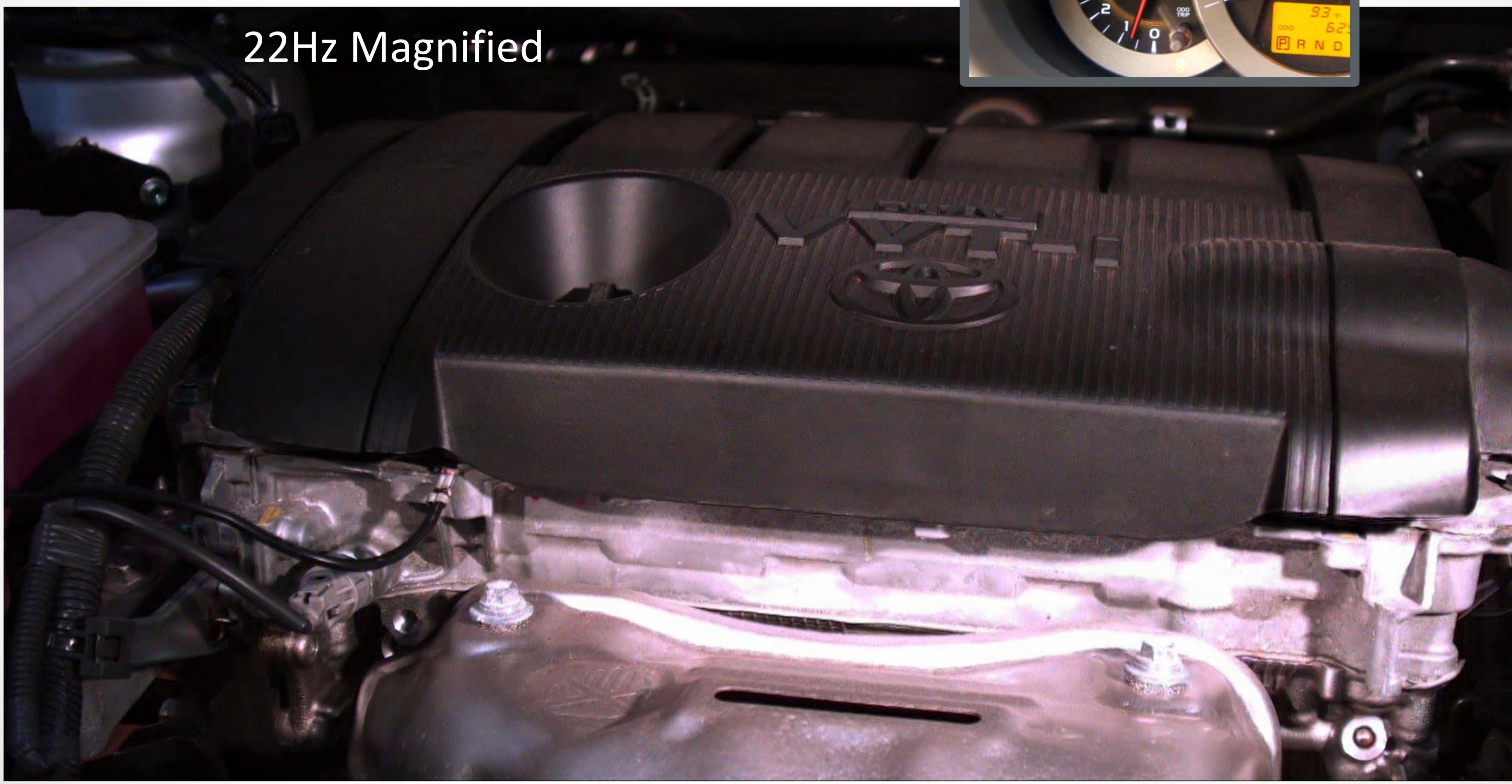
Sequence courtesy Vimeo user Vincent Laforet

Car Engine

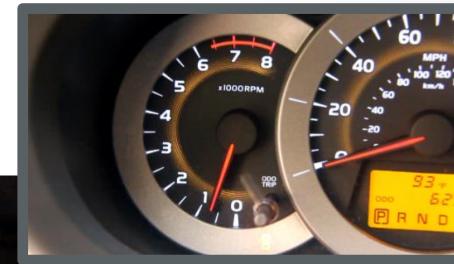


Car Engine

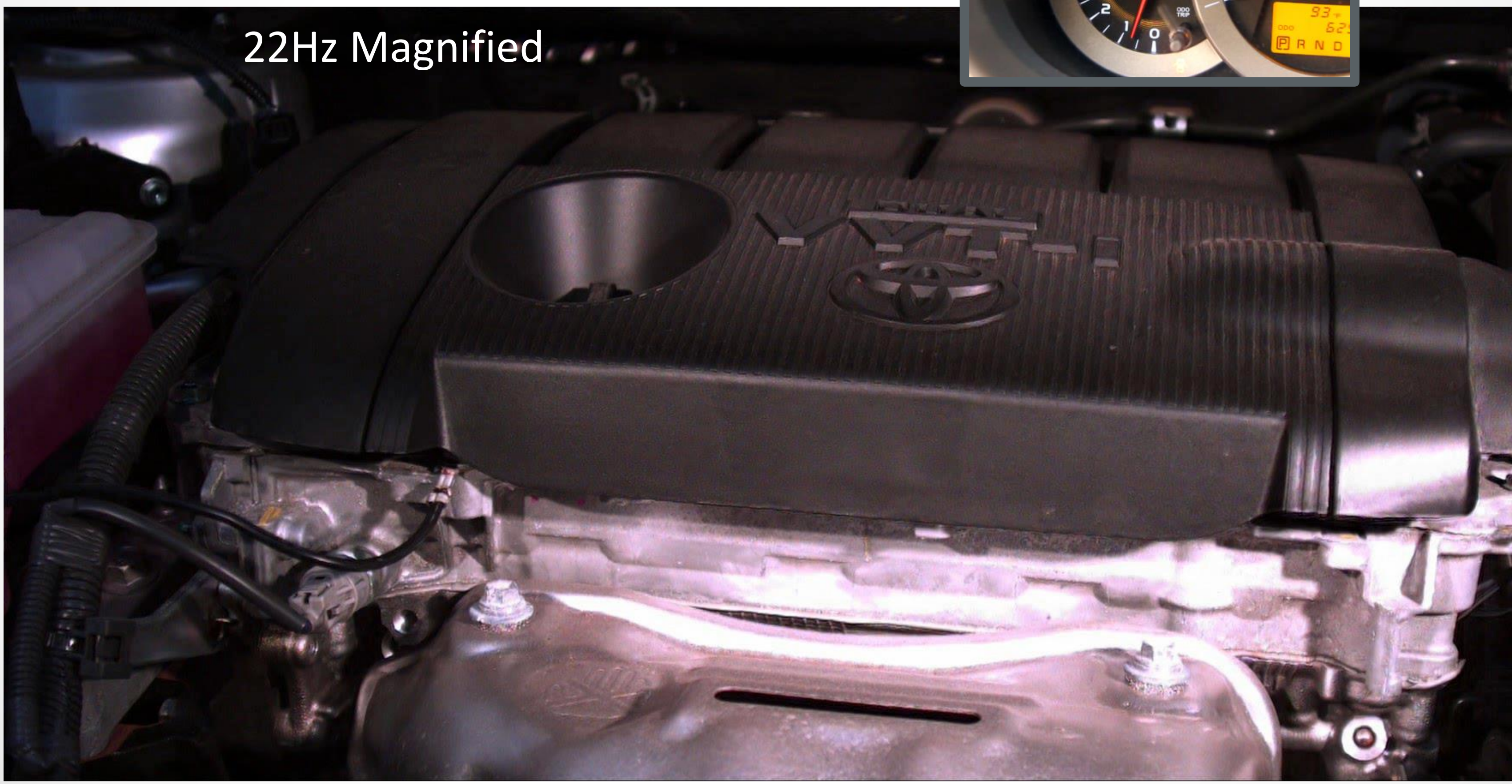


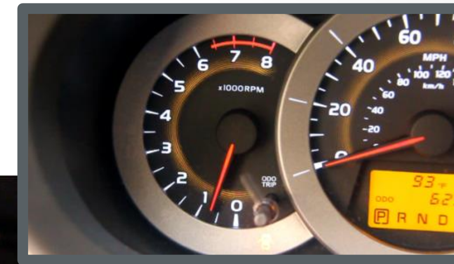


Car Engine

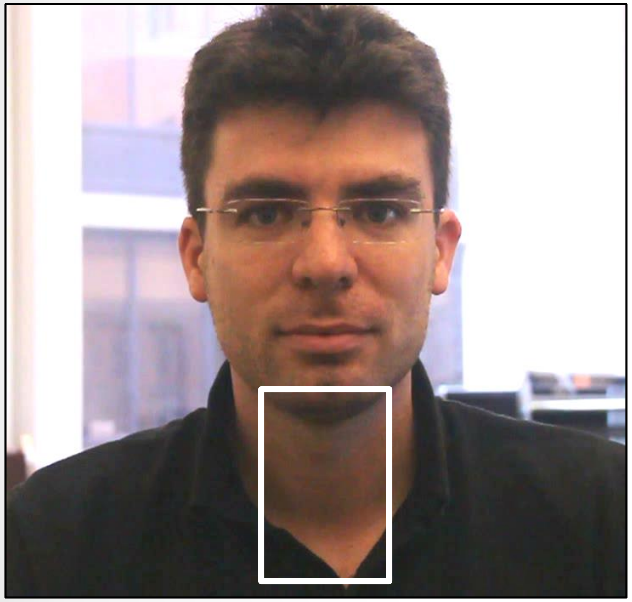


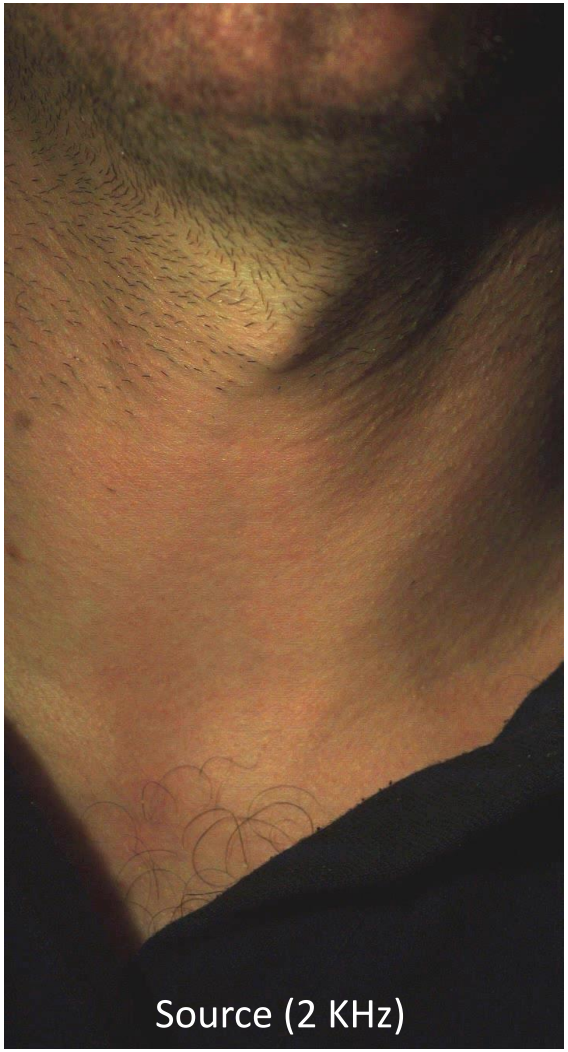
Car Engine

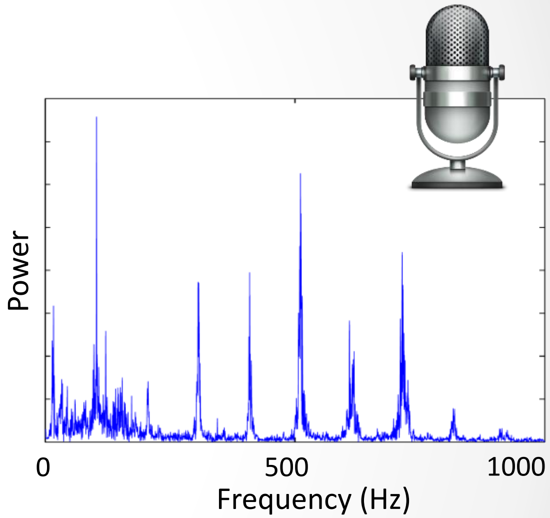


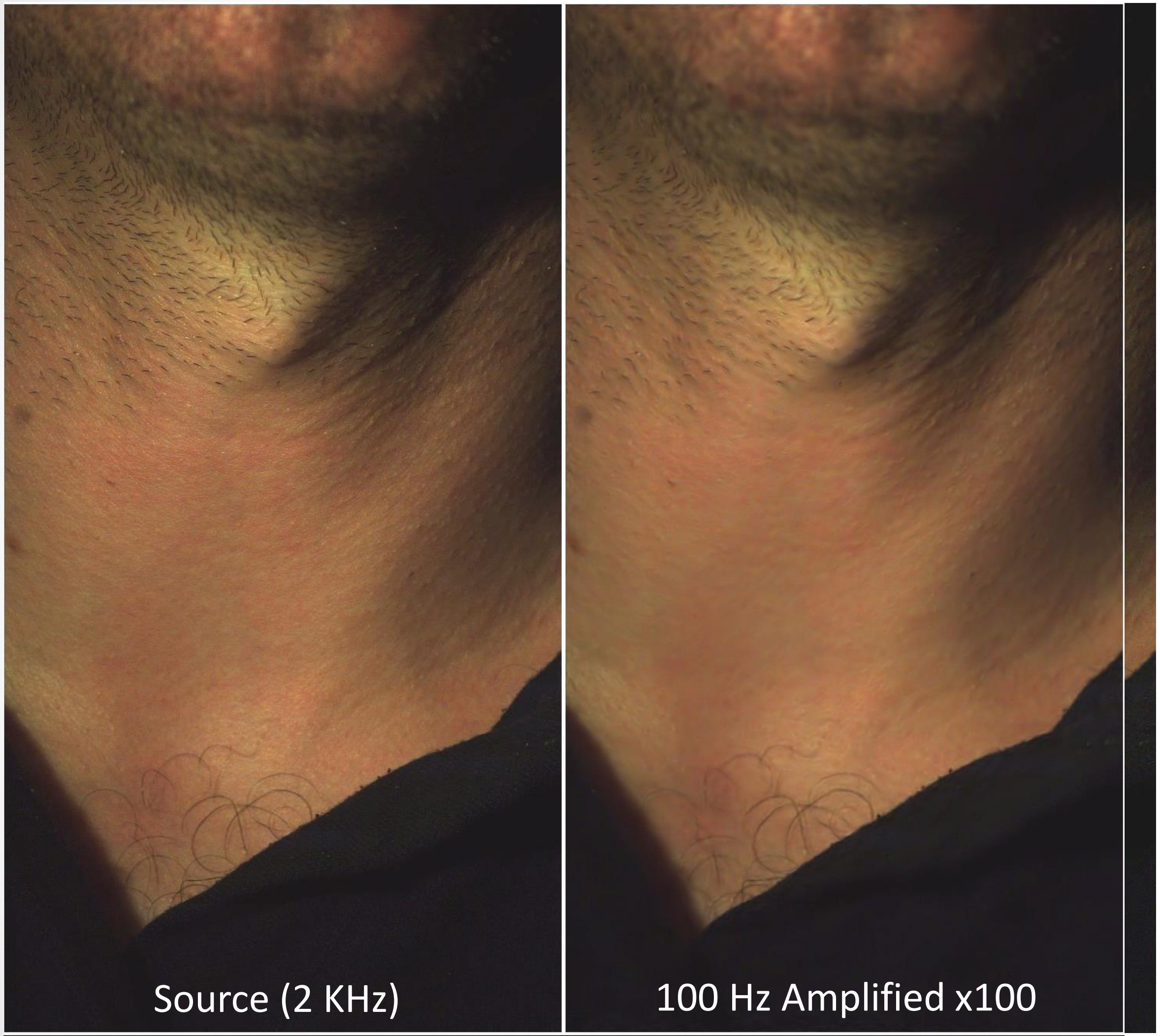


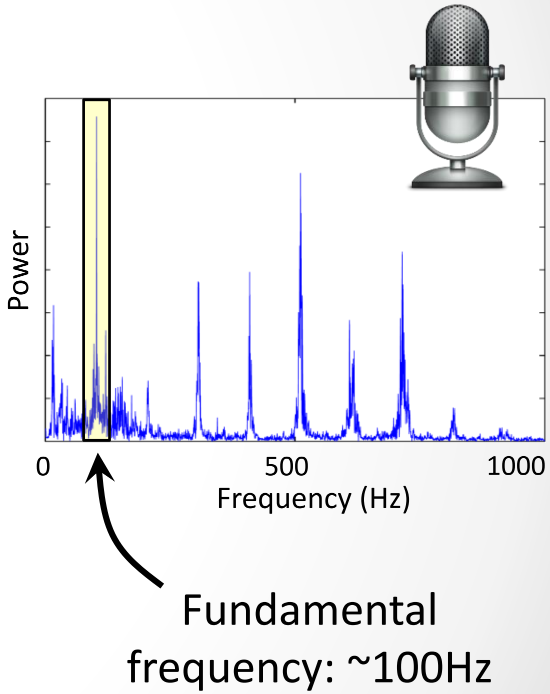
Neck Skin Vibrations

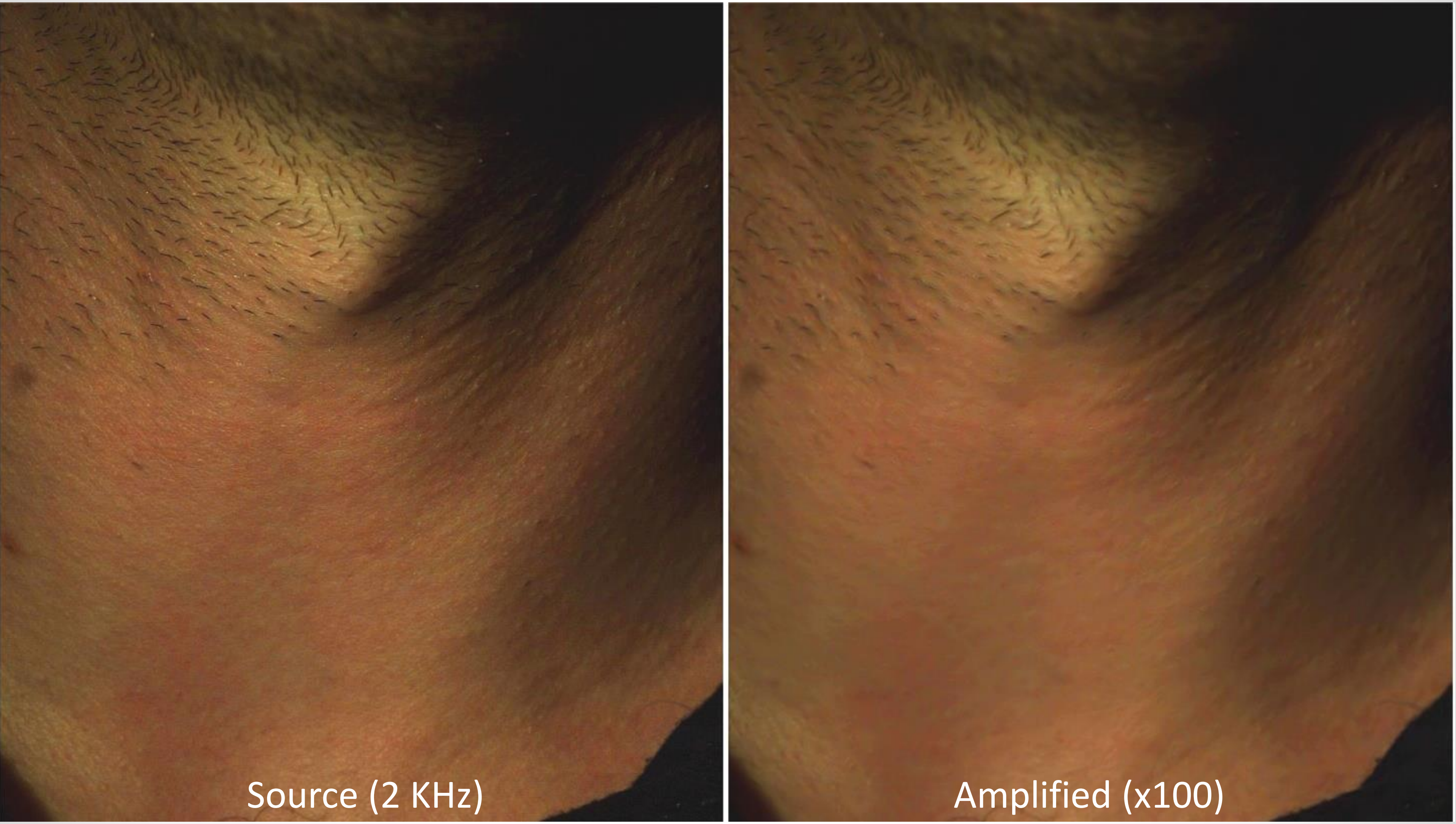






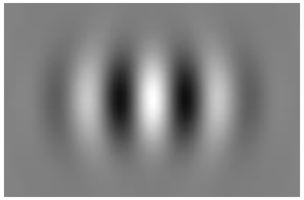

Source (2 KHz)
100 Hz Amplified x100

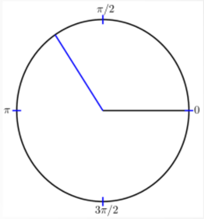

Power
0
500
1000
Frequency (Hz)
Fundamental
frequency: ~100Hz


Source (2 KHz)
Amplified (x100)

Conclusions

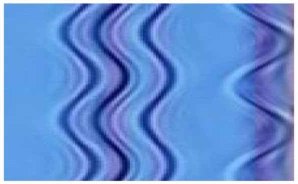
- New representation for analyzing and editing small motions





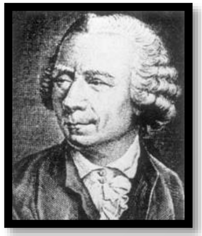
- Much better than linear EVM [Wu et al. 2012]
  - Less noise
  - More amplification

Linear
SIGGRAPH'12

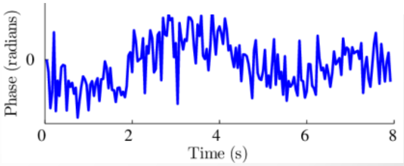


Phase-based
SIGGRAPH'13

- Still "Eulerian" (no optical flow), but more explicit representation of motion
  - New capabilities (e.g. attenuating distracting motions)

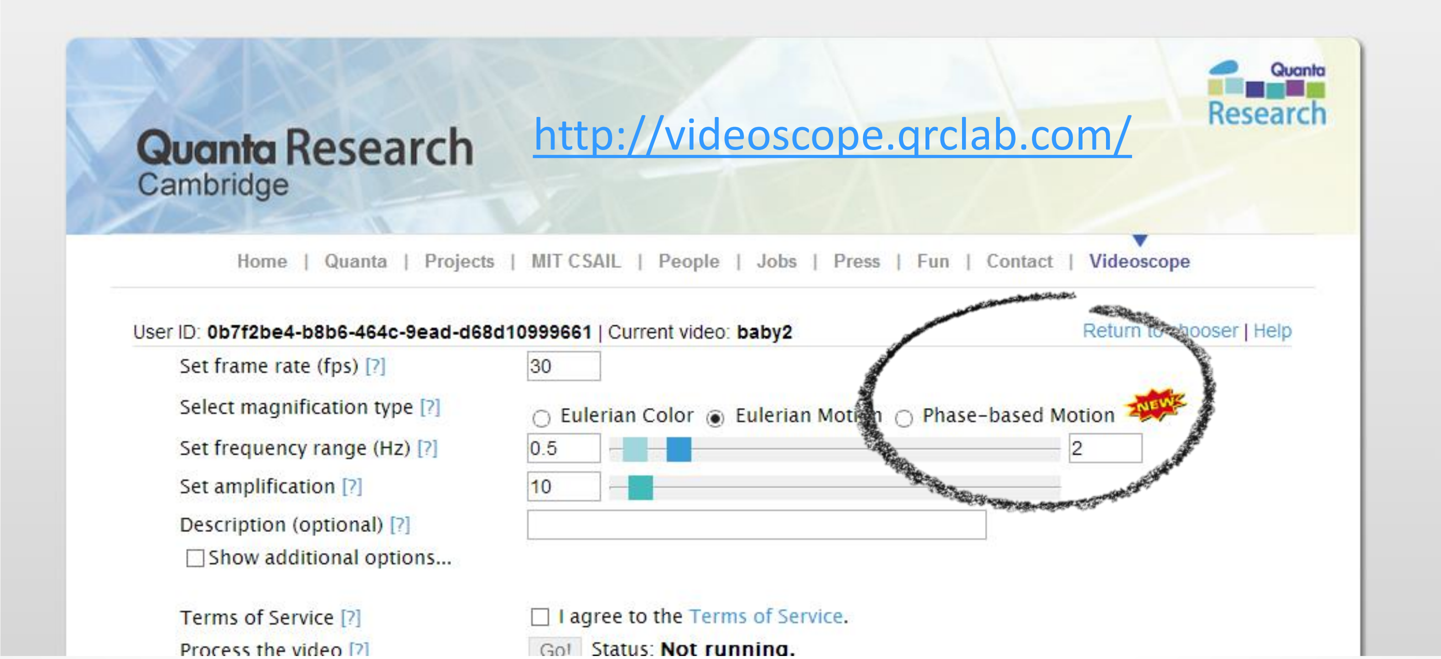


Phase over time



Phase-Based Motion Processing: Code and Web App

- Code available soon: http://people.csail.mit.edu/nwadhwa/phase-video/



Overall Conclusion

- Many problems involve solving for Poisson equations
  - For image editing
  - For video processing
  - For rendering
  - For geometry processing (more with Julie)
- Many solvers exist
  - Iterative solvers (Krylov or not...)
  - Direct solvers (Cholesky)
  - Fourier, FFT or Green's function-based
- We have seen other cool image/video applications
  - ... though not with Poisson!