A Fast Implicit Gaussian Curvature Filter

Yuanhao Gong

source code at https://github.com/YuanhaoGong

Outline

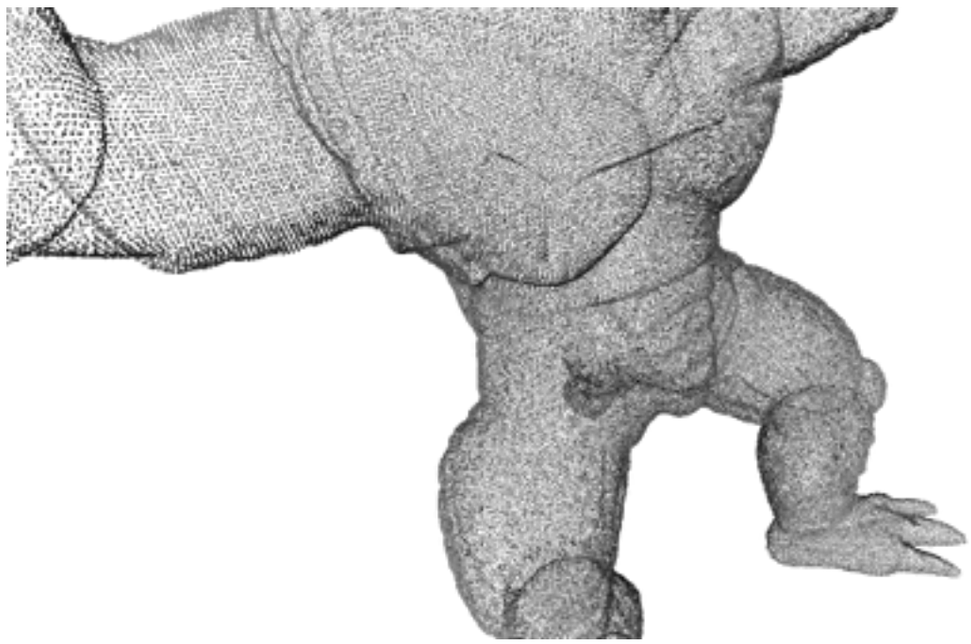
- Introduction
- Gaussian Curvature Filter
- Applications and Benchmarks
- Conclusions

Introduction

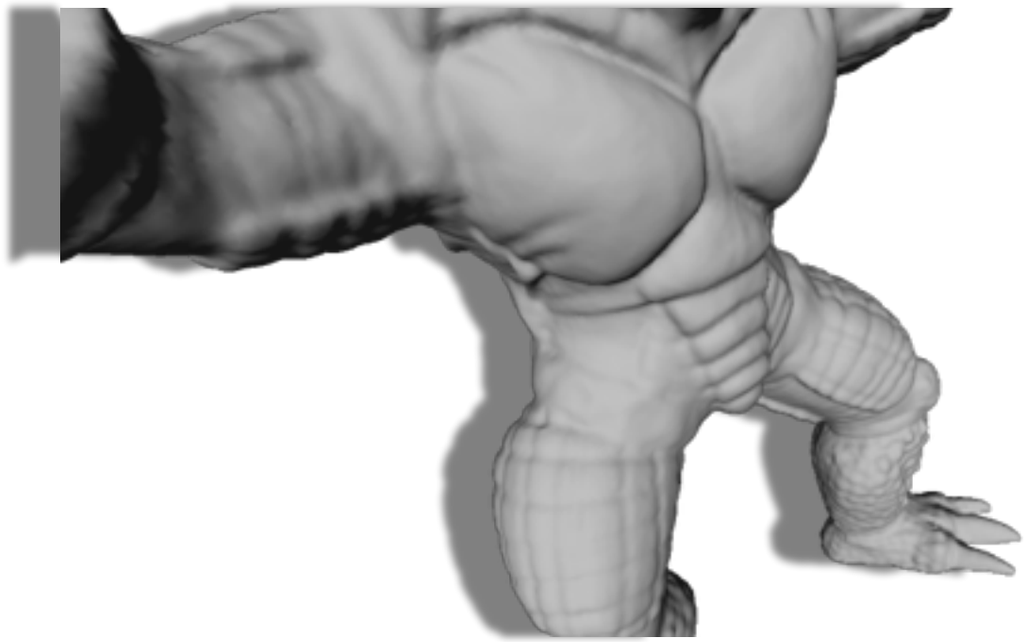
- Estimating ground truth from discrete samples is usually ill-posed.

Blurred Image

Estimated deblurring



Point Cloud



Estimated Surface

Introduction

- Ill-posed problems need regularization or a prior!

Piecewise Constant

Bilateral Filter
ROF Model
CV Model
Total Variation
CLMF0
RTV
etc…

Piecewise Linear

Guided Filter
CLMF1
etc…

Piecewise Smooth

Mumford and Shah
etc…

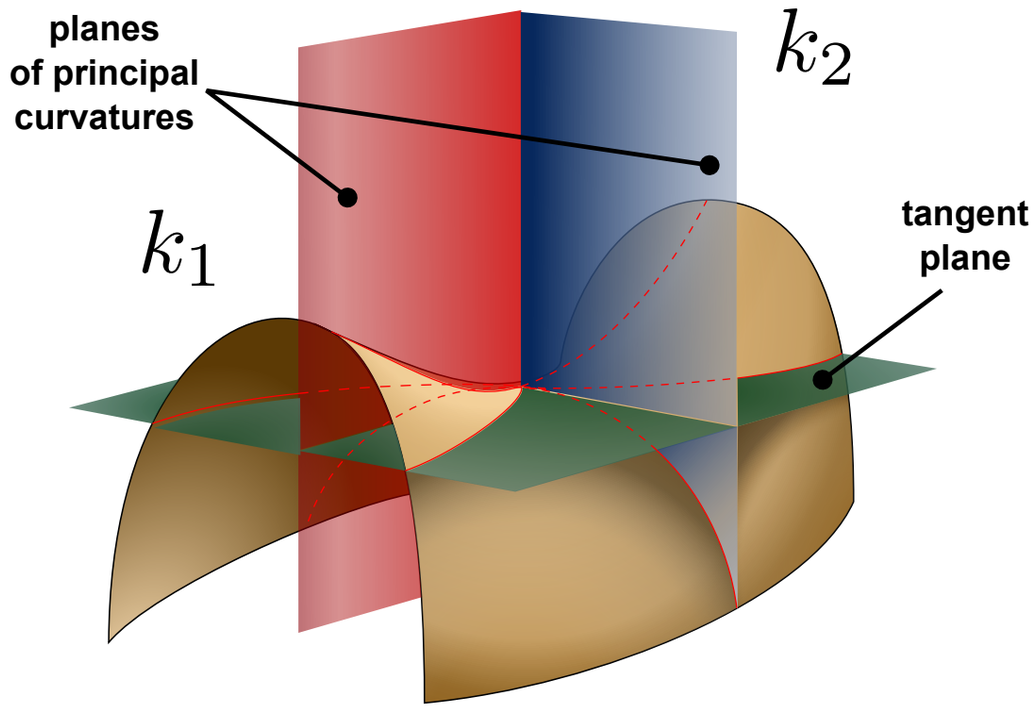
problems are getting harder to solve!

Introduction

- Ill-posed problems need regularization or a prior!

Smoothness is characterized by curvature

Piecewise Smooth



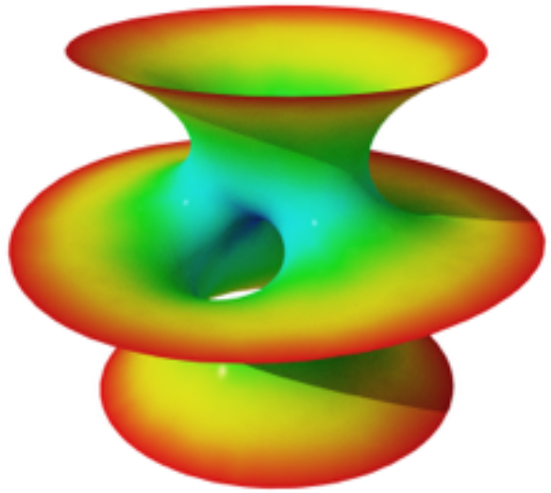
$$\text{mean curvature} = \frac{k_1 + k_2}{2}$$

$$\text{Gaussian curvature} = k_1 * k_2$$

Introduction

- Why is Gaussian curvature important?

Mean curvature: $\frac{k_1+k_2}{2}$
The external embedding matters!
Leads to a minimal surface.



minimal surface

Developable surfaces allow sharp edges while minimal surfaces do not

Gaussian curvature: $k_1 * k_2$
Intrinsic: external embedding does NOT matter!
Leads to a developable surface.

developable surface

Introduction

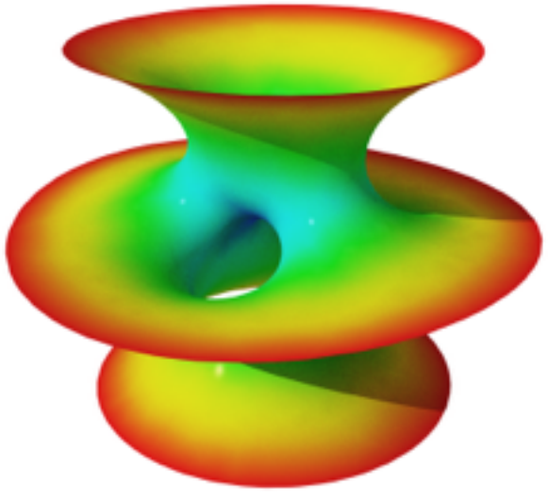
- Ill-posed problems need regularization or a prior!

Piecewise Smooth

traditional way

mean curvature flow

Geometric flow (diffusion PDE)



minimal surface

Gaussian curvature flow

developable surface

Introduction

Previous Works on Gaussian curvature:

- Diffusion based models (Lee 2005, Zhu 2007, Overgaard 2007, etc), Domain decomposition acceleration (Firsov 2006), edge weighted diffusion (Lu 2011)
- Iterative reweighted scheme (Gong 2013), etc.

Issues:

- PDE evolution (diffusion) converges slowly.
- All methods need explicit computation of Gaussian curvature, which requires the result to be at least twice differentiable (hence does not preserve edges).

Introduction

Our Goal:

Minimize Gaussian curvature efficiently without explicitly computing it.

I.e., a fast implicit filter!

Introduction

Forward process:

Traditionally

minimizing Gauss curvature by diffusion

Backward process:

Our method

approximating the image by developable surfaces

It is easy to use the property of a developable surface in the backward process, but hard in the forward process.

Theorem:

There are only three types of developable surfaces: cylinders, cones and tangent developables.

Outline

- Introduction
- Gaussian Curvature Filter
- Applications and Benchmarks
- Conclusions

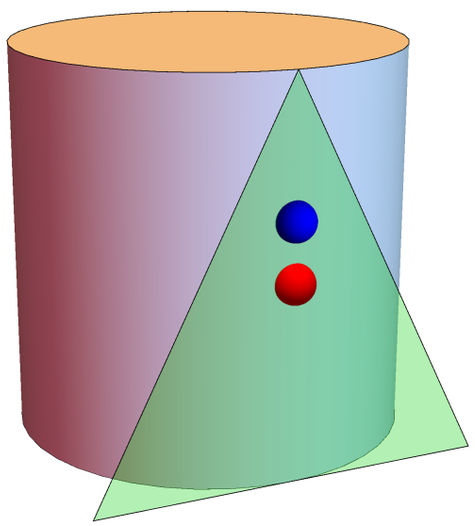
Gaussian Curvature Filter

Theorem:

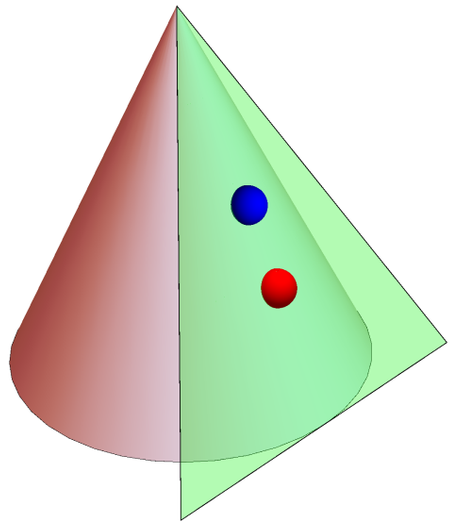
any developable surface *S* and its tangent plane *TS* have the following relationship:

Theorem:

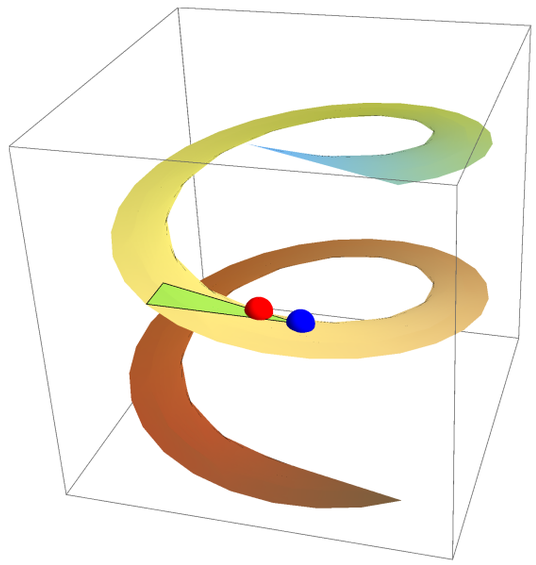
$$\forall \vec{x} \in \mathcal{S}, \forall \epsilon > 0, \exists \vec{x}_0 \neq \vec{x}, \text{s.t. } \vec{x} \in \mathcal{TS}(\vec{x}_0) \text{ and } |\vec{x} - \vec{x}_0| < \epsilon.$$



cylinder



cone



tangent developable

Gaussian Curvature Filter

Theorem:

any developable surface *S* and its tangent plane *TS* have the following relationship:

Theorem:

$$\forall \vec{x} \in \mathcal{S}, \forall \epsilon > 0, \exists \vec{x}_0 \neq \vec{x}, \text{ s.t. } \vec{x} \in \mathcal{TS}(\vec{x}_0) \text{ and } |\vec{x} - \vec{x}_0| < \epsilon.$$

This means that computing GC is not necessary. Instead, we can implicitly minimize it by just minimizing one of the principle curvatures.

Gaussian curvature: $k_1 * k_2$

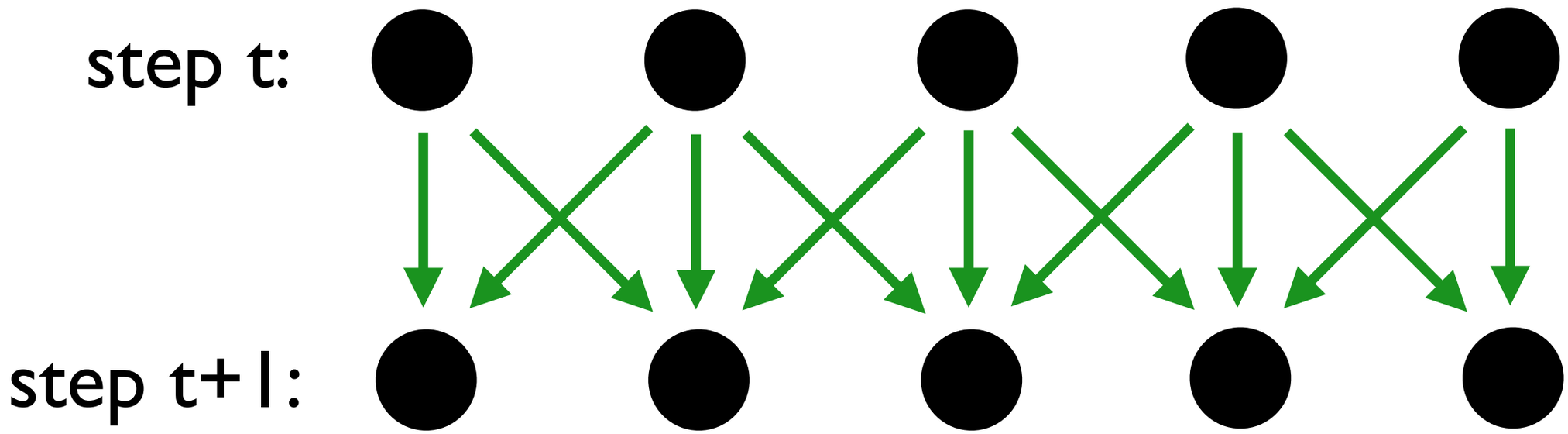
Minimize one of them!!!

Gaussian Curvature Filter

Backward process:

change $U(\vec{x})$ such that $U(\vec{x})$ falls on one tangent plane of its neighbors

However (dependency):



Neighbors have changed!!!

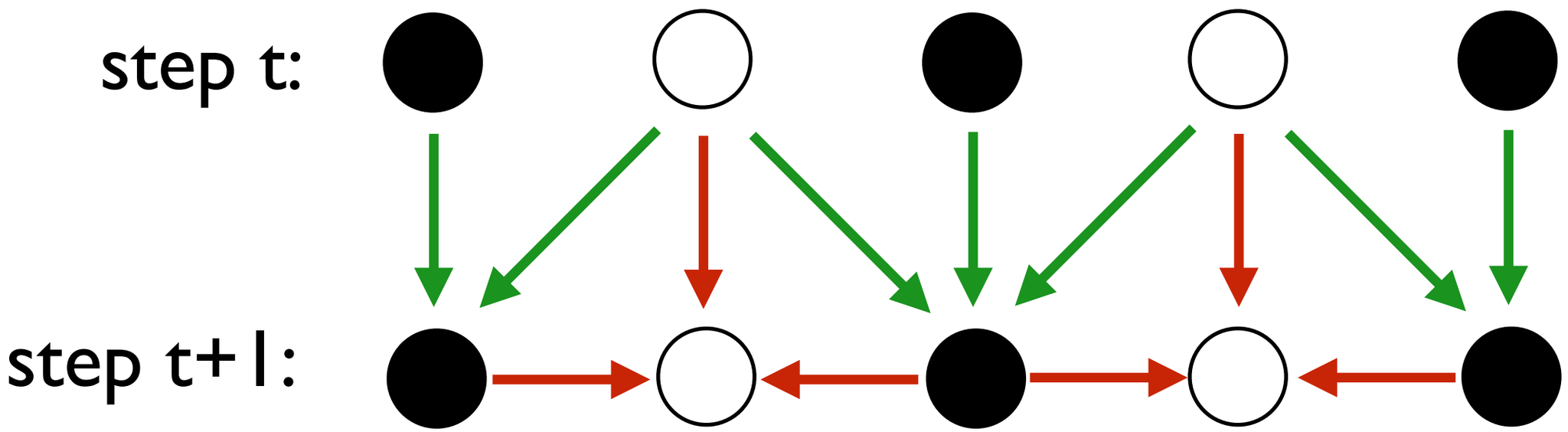
Gaussian Curvature Filter

Backward process:

change $U(\vec{x})$ such that $U(\vec{x})$ falls on one tangent plane of its neighbors

Domain Decomposition:

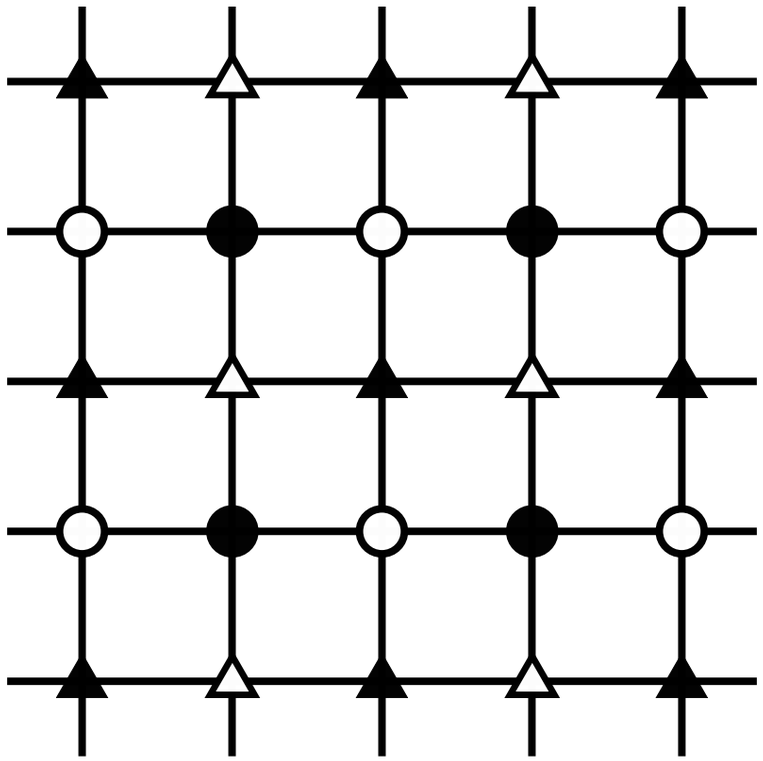
First, update black through green lines; then, update white through red lines.



Changed neighbors have been considered.

Gaussian Curvature Filter

2D domain decomposition:



B_C, B_T

W_C, W_T

Gaussian Curvature Filter

Tangent plane representation:

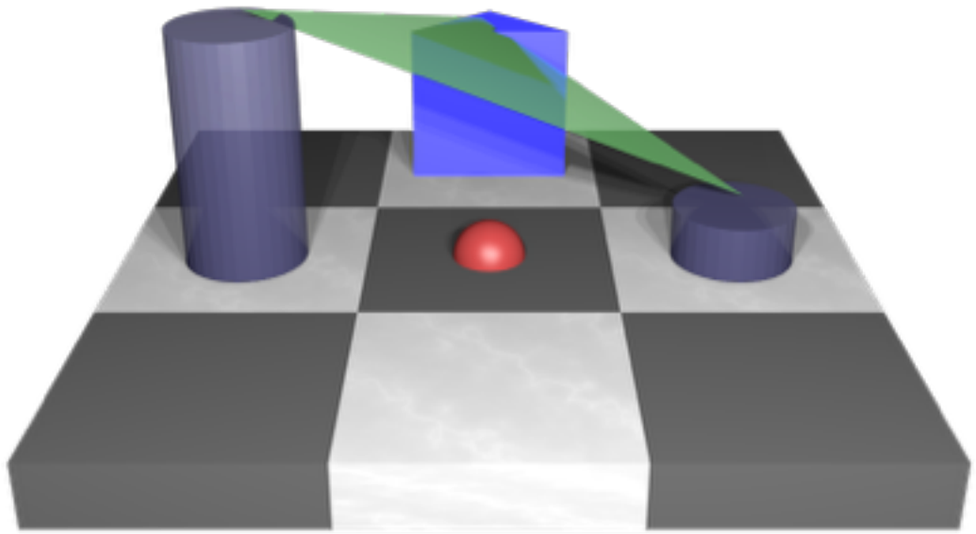
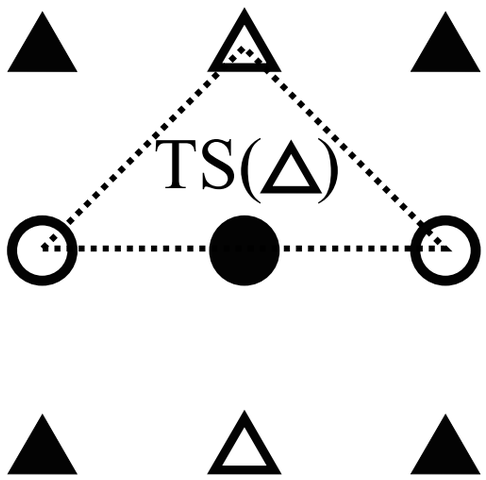
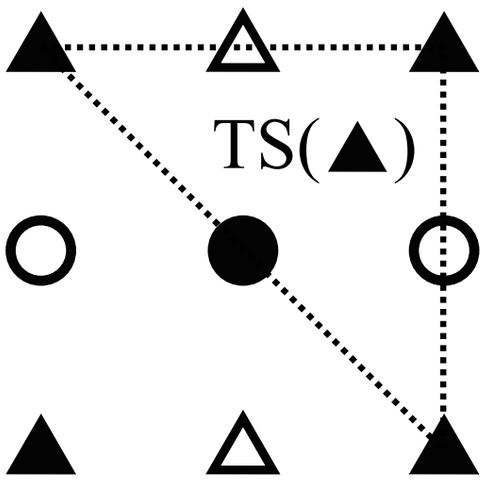


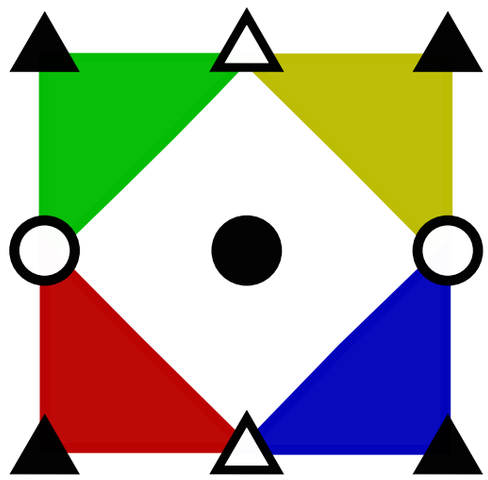
illustration of one tangent plane (green) at the blue triangle cylinder, which is neighbor of the red dot



tangent plane of the white triangle



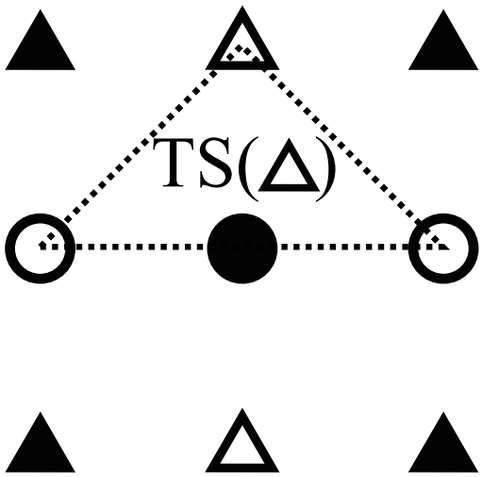
tangent plane at the upper-right corner

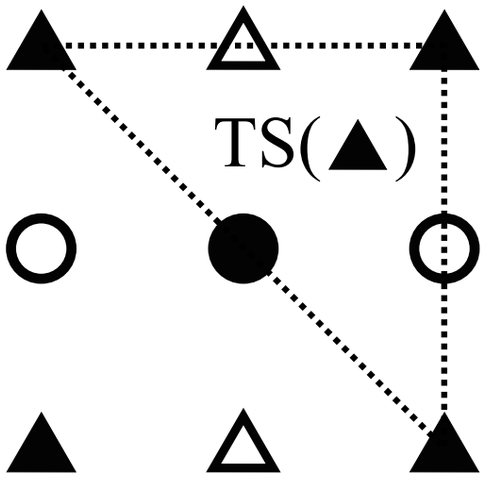


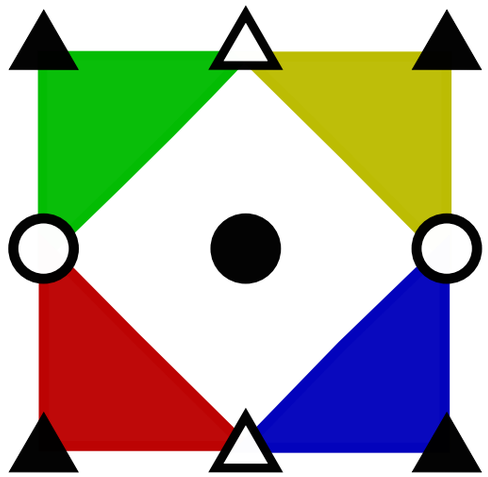
tangent planes from mixed

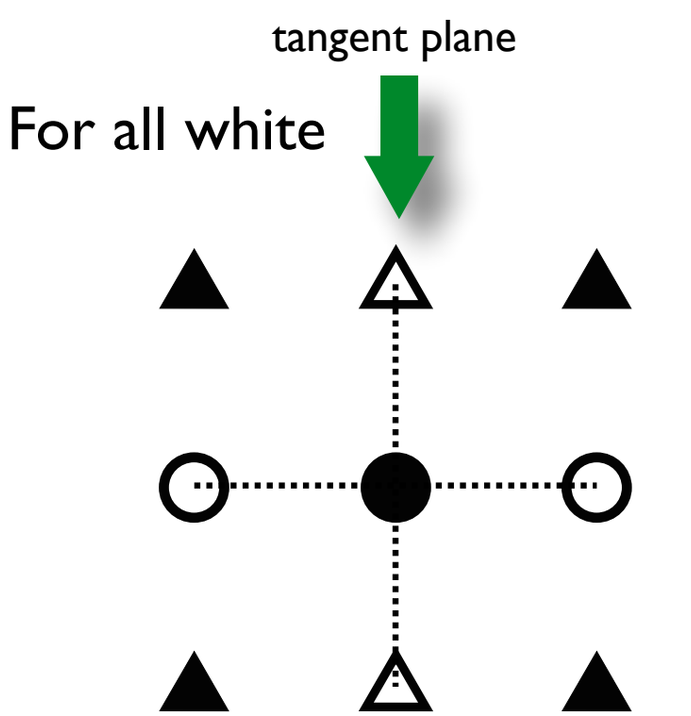
Gaussian Curvature Filter

projection to tangent plane:

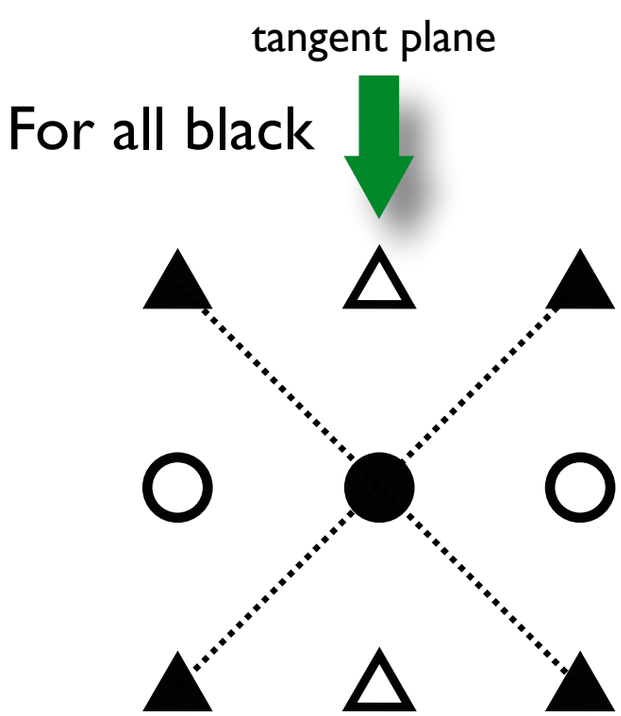




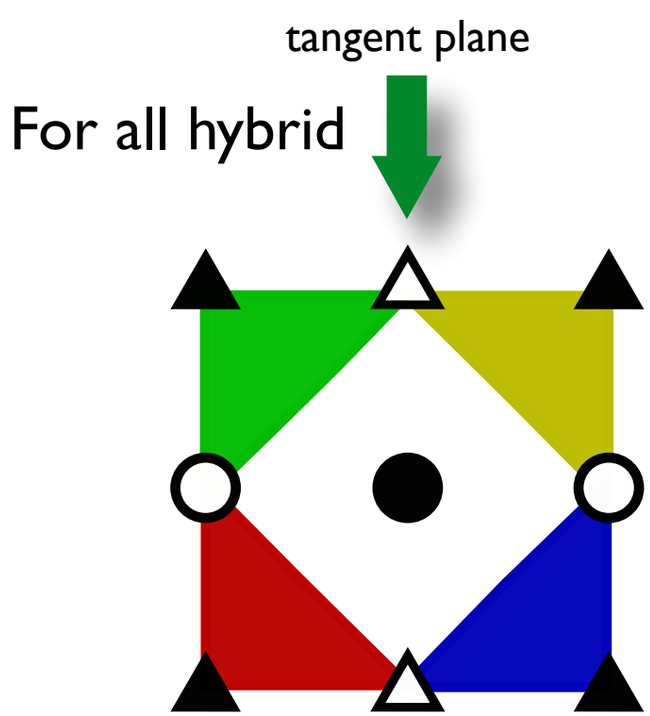




projection
(black point to the dashed lines)



projection
(black point to the dashed lines)



projection
(black point to colored planes)

Gaussian Curvature Filter

Minimal Projection Operator:

$$
\begin{aligned}
d_1 &= (U(i-1,j) + U(i+1,j))/2 - U(i,j) \\
d_2 &= (U(i,j-1) + U(i,j+1))/2 - U(i,j) \\
d_3 &= (U(i-1,j-1) + U(i+1,j+1))/2 - U(i,j) \\
d_4 &= (U(i-1,j+1) + U(i+1,j-1))/2 - U(i,j) \\
d_5 &= U(i-1,j) + U(i,j-1) - U(i-1,j-1) - U(i,j) \\
d_6 &= U(i-1,j) + U(i,j+1) - U(i-1,j+1) - U(i,j) \\
d_7 &= U(i,j-1) + U(i+1,j) - U(i+1,j-1) - U(i,j) \\
d_8 &= U(i,j+1) + U(i+1,j) - U(i+1,j+1) - U(i,j) \\
&\text{find } d_m, \text{ such that } |d_m| = \min\{|d_i|, i = 1,\ldots,8\} \\
\hat{U}(i,j) &= U(i,j) + d_m
\end{aligned}
$$

only 25 (+, -, /) ops. per pixel !!!

Gaussian Curvature Filter

Gaussian Curvature Filter:

$$\forall \vec{x} \in B_T,\ \mathcal{P}(U(\vec{x}))$$
$$\forall \vec{x} \in B_C,\ \mathcal{P}(U(\vec{x}))$$
$$\forall \vec{x} \in W_T,\ \mathcal{P}(U(\vec{x}))$$
$$\forall \vec{x} \in W_C,\ \mathcal{P}(U(\vec{x}))$$

Theorem: $\mathcal{E}(\mathcal{P}(U)) \leq \mathcal{E}(U),\ \ \forall \vec{x}$

Theorem: $\mathcal{E}(\mathcal{G}_c^{k_2}(U)) \leq \mathcal{E}(\mathcal{G}_c^{k_1}(U)) \leq \mathcal{E}(U)$ for $k_2 > k_1 > 0$.

Theorem: GC filter converges and its convergence rate is at least one.

Gaussian Curvature Filter

Properties of the present GC Filter:

- Two orders of magnitude faster than diffusion-based methods
- Converges fast (ten iterations are enough in practice)
- Minimize GC without computing it
- The resulting image is continuous, but NOT smooth (edges!!!)
- Preserves developable surfaces
- Linear computational complexity
- Easy to implement (100 lines C++) and parallelize

Outline

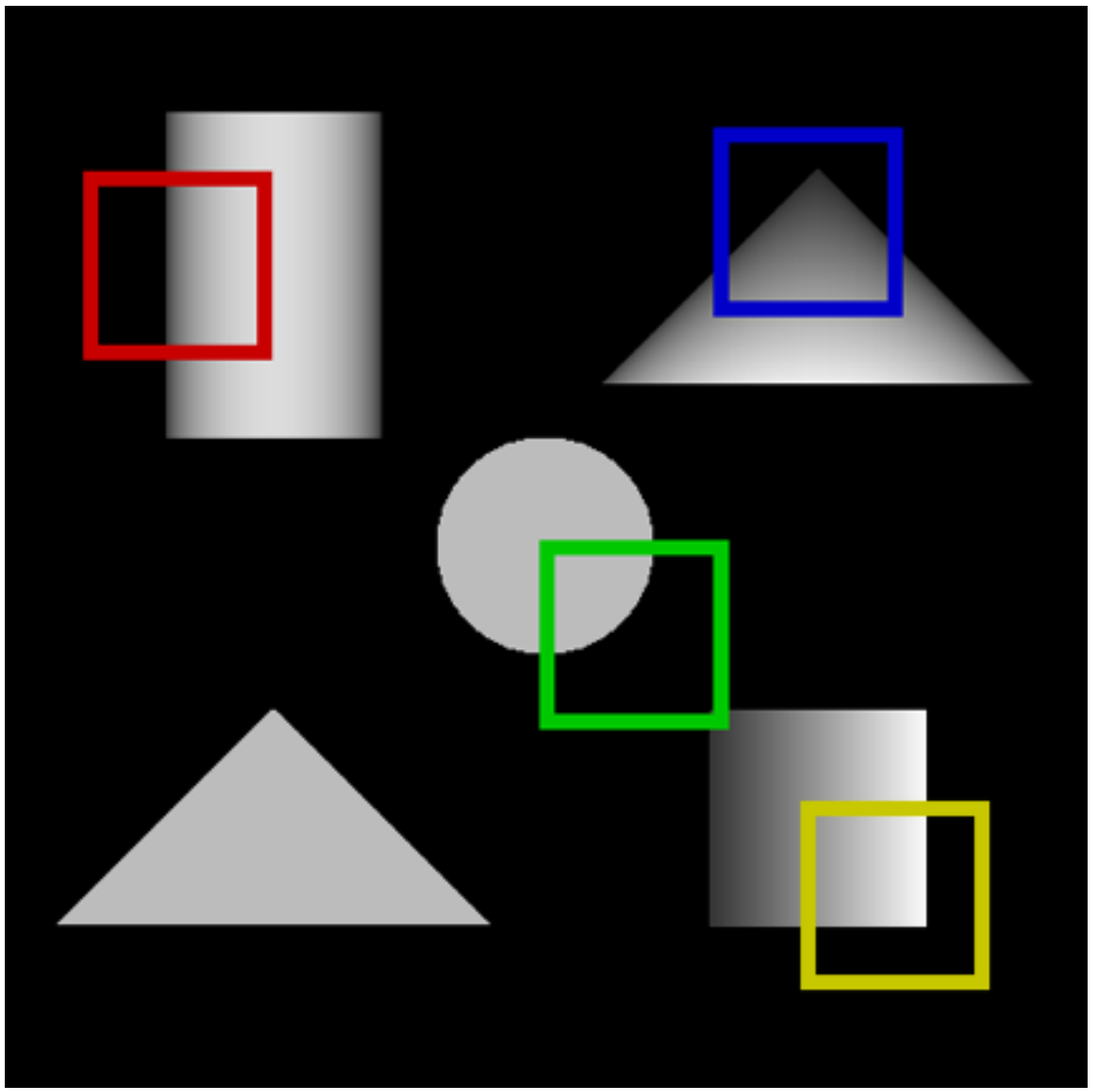
- Introduction
- Gaussian Curvature Filter
- Applications and Benchmarks
- Conclusions

Outline

- Introduction
- Gaussian Curvature Filter
- Applications and Benchmarks
  - test on developable and non-developable surfaces
  - test on image denoising
  - benchmark on natural-image dataset
  - comparison with other edge-preserving image smoothing methods
- Conclusions

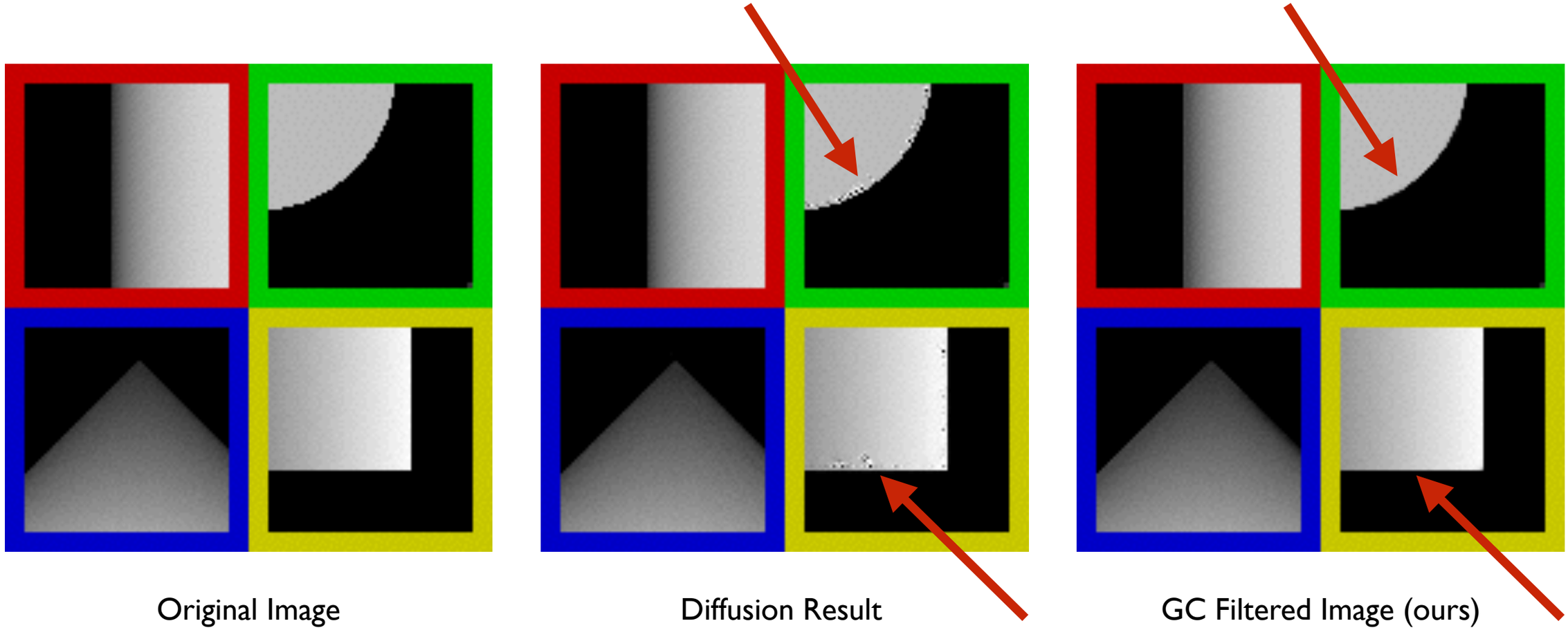
Applications and Benchmarks

Test on developable surfaces:



Applications and Benchmarks

Test on developable surfaces:



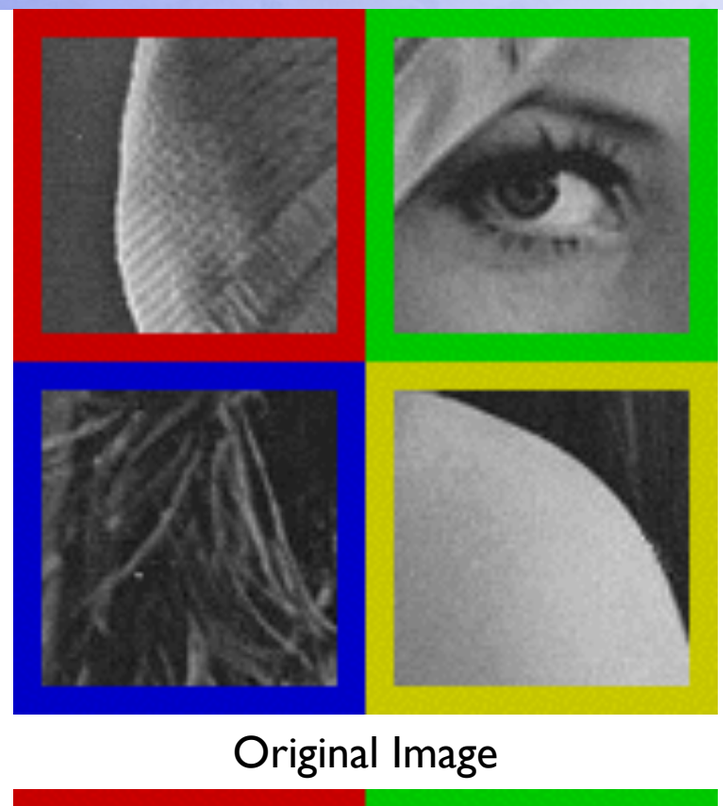
Original Image

Diffusion Result

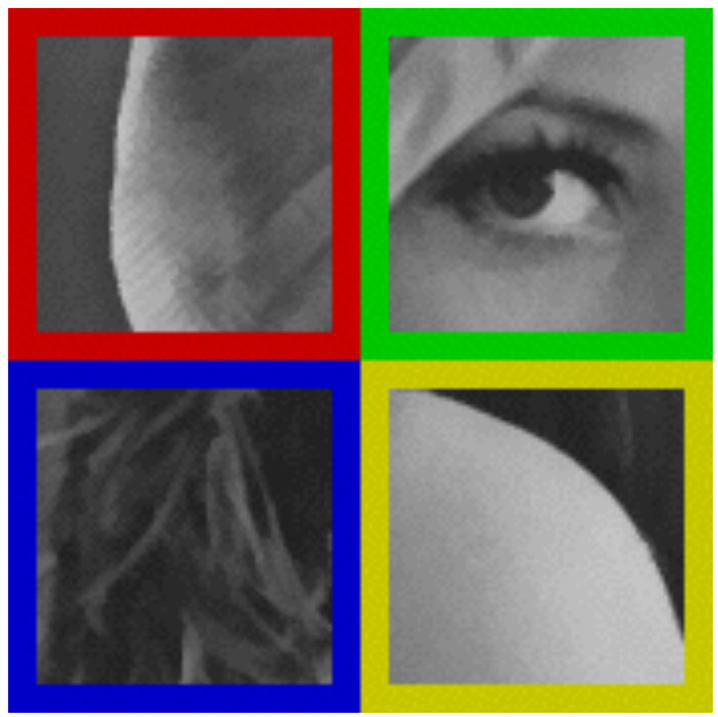
GC Filtered Image (ours)

Applications and Benchmarks

Test on a non-developable surface:



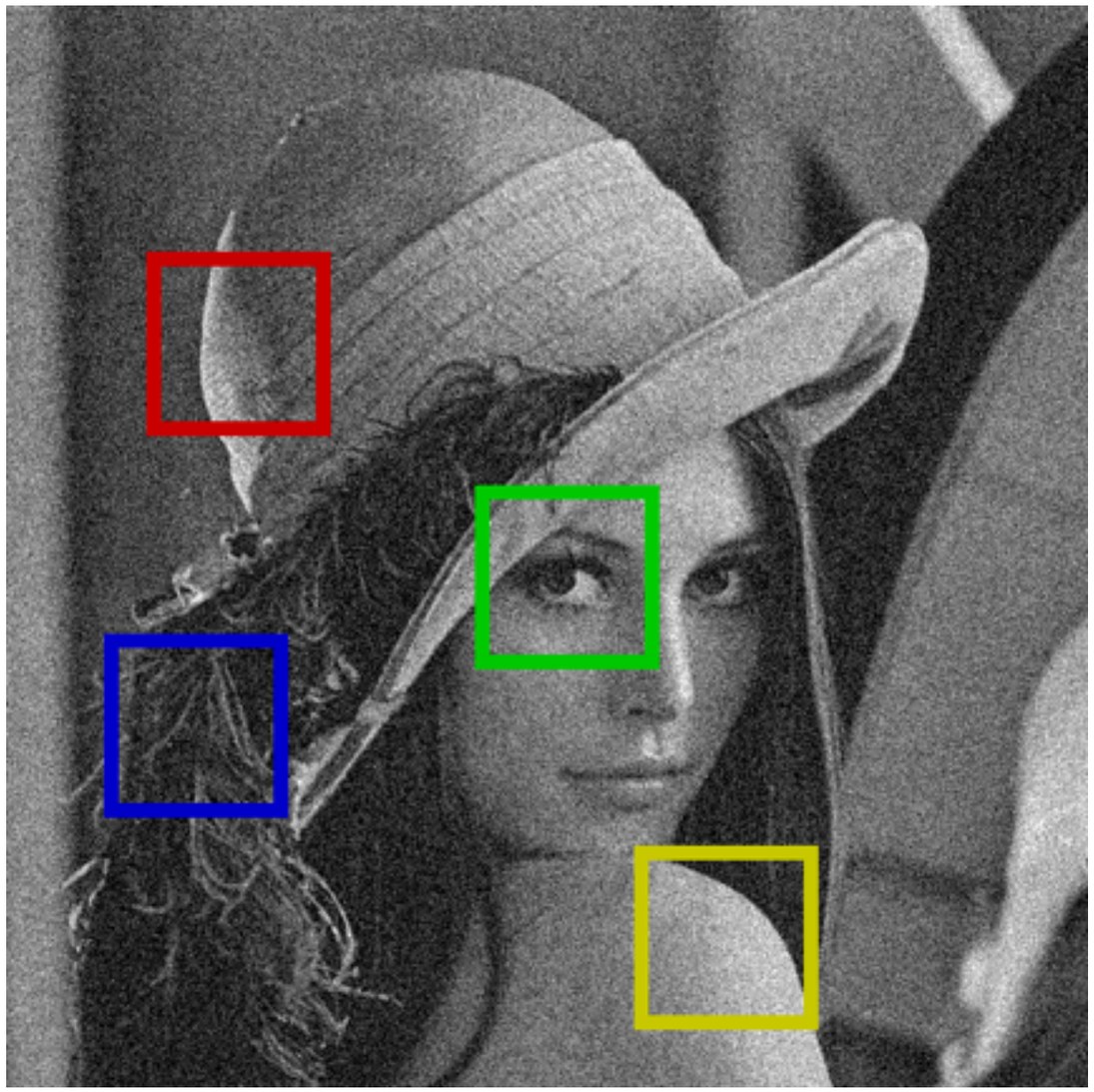
Original Image



GC Filtered Image

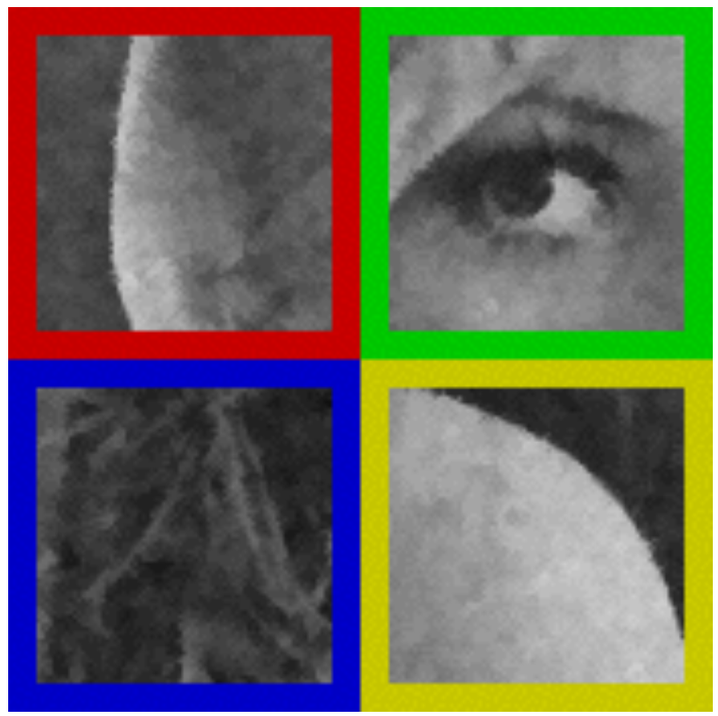
Applications and Benchmarks

Image denoising:



Gaussian noise

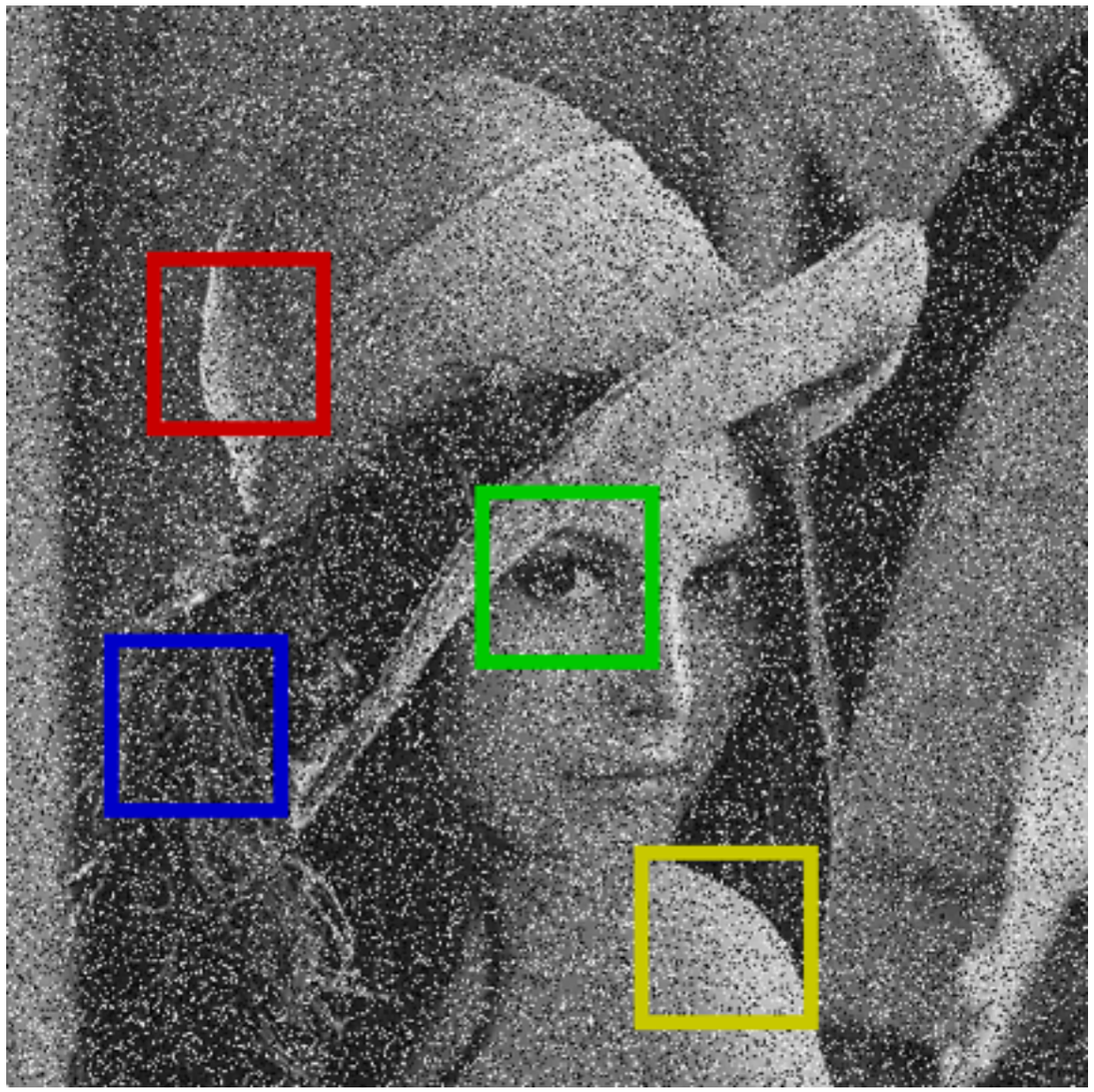
Original Image



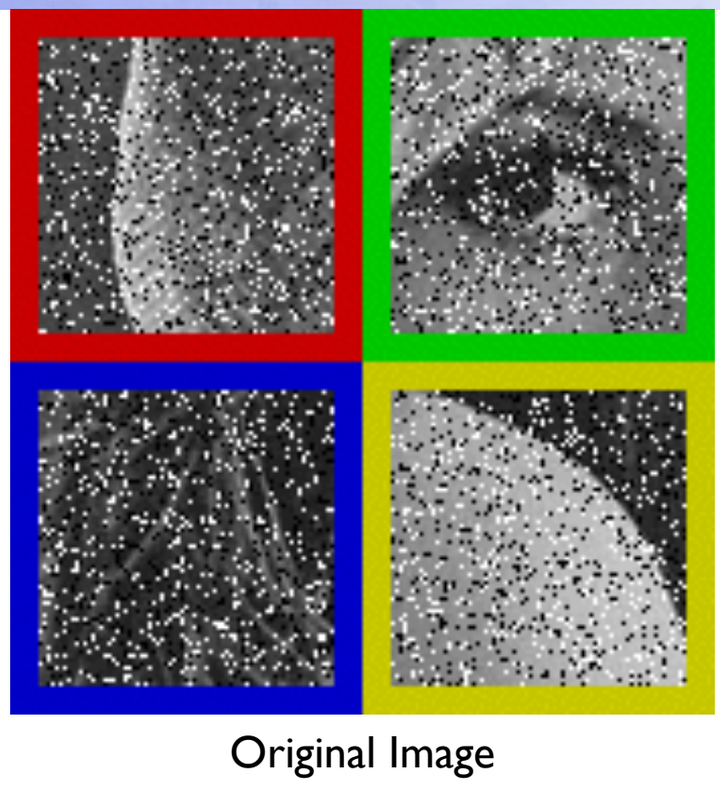
GC Filtered Image

Applications and Benchmarks

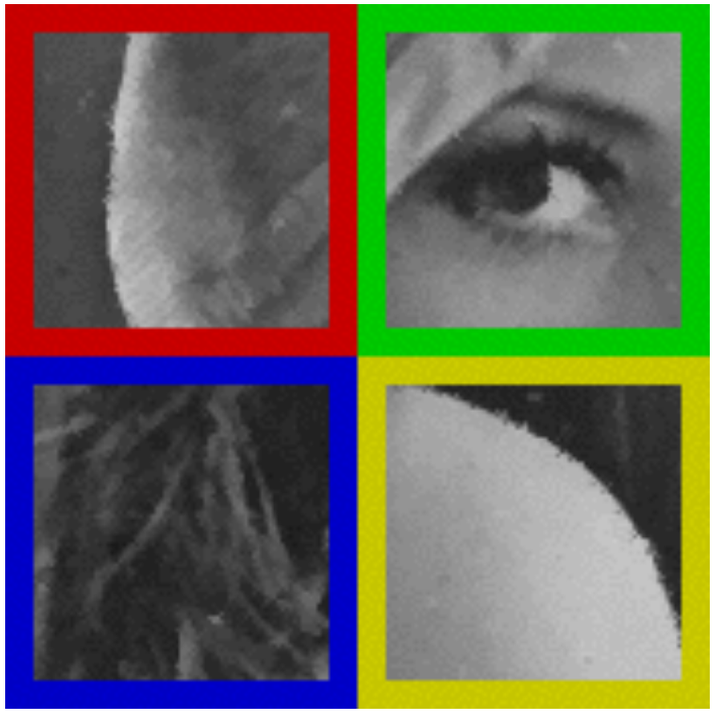
Image denoising:



Salt pepper noise



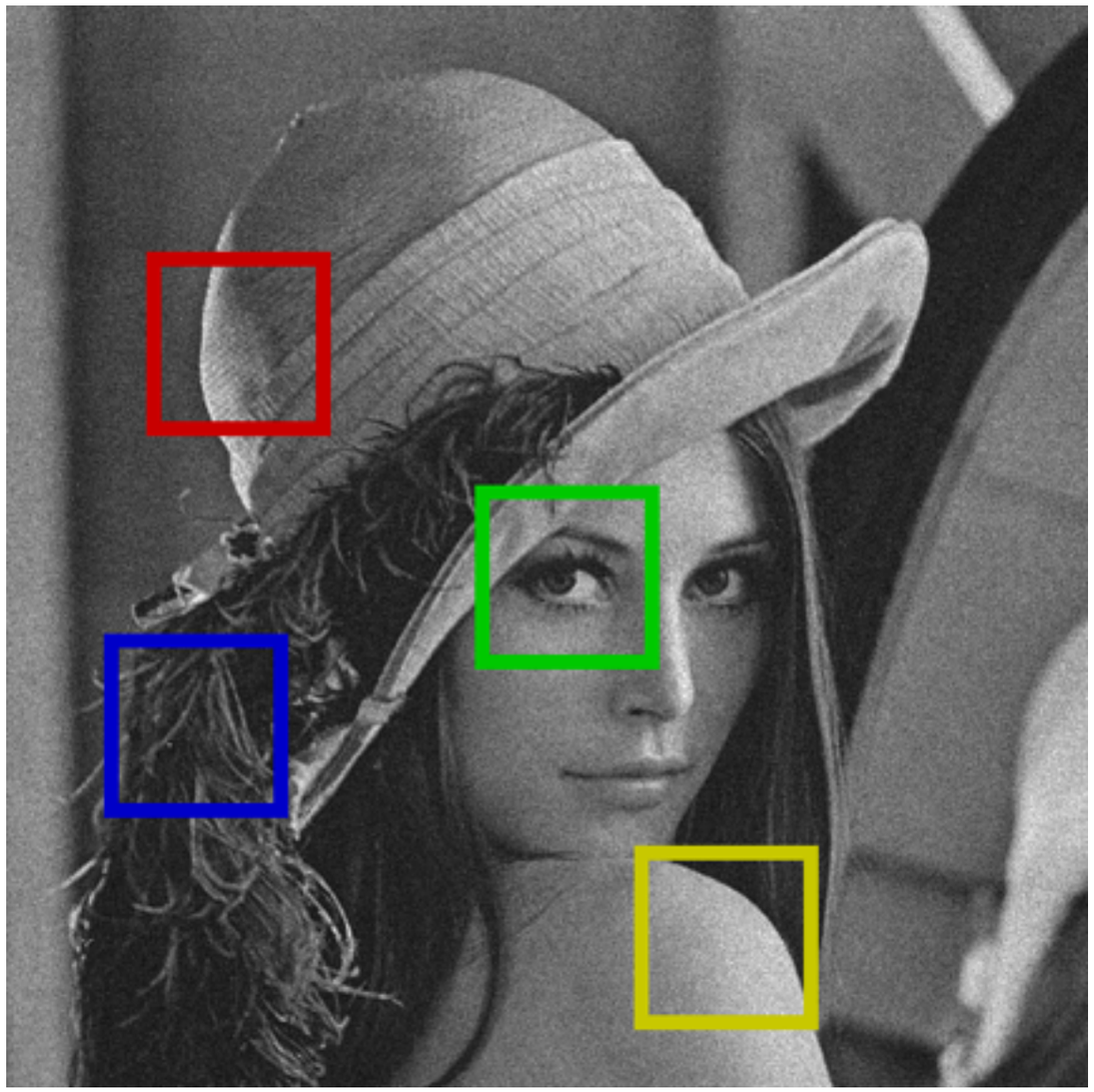
Original Image



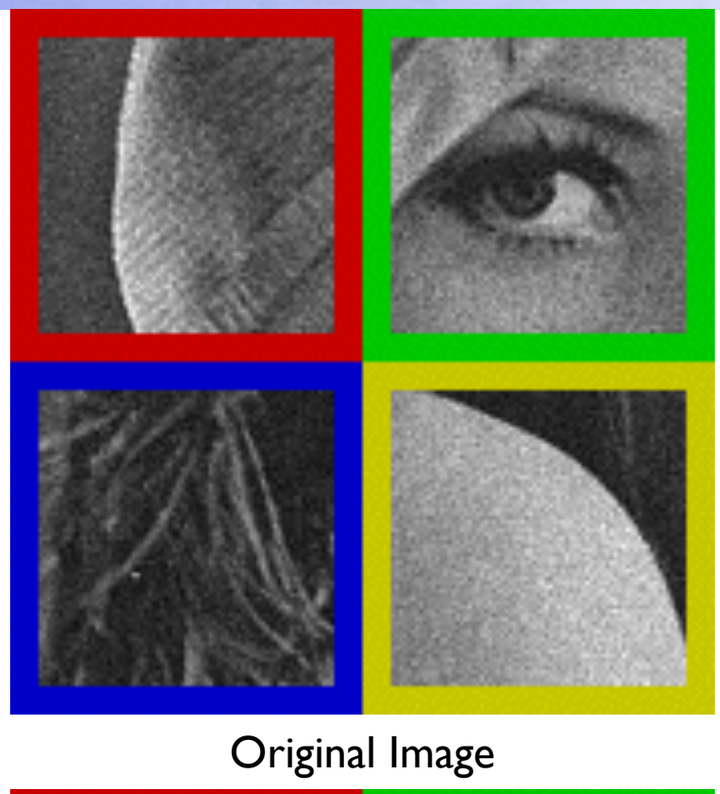
GC Filtered Image

Applications and Benchmarks

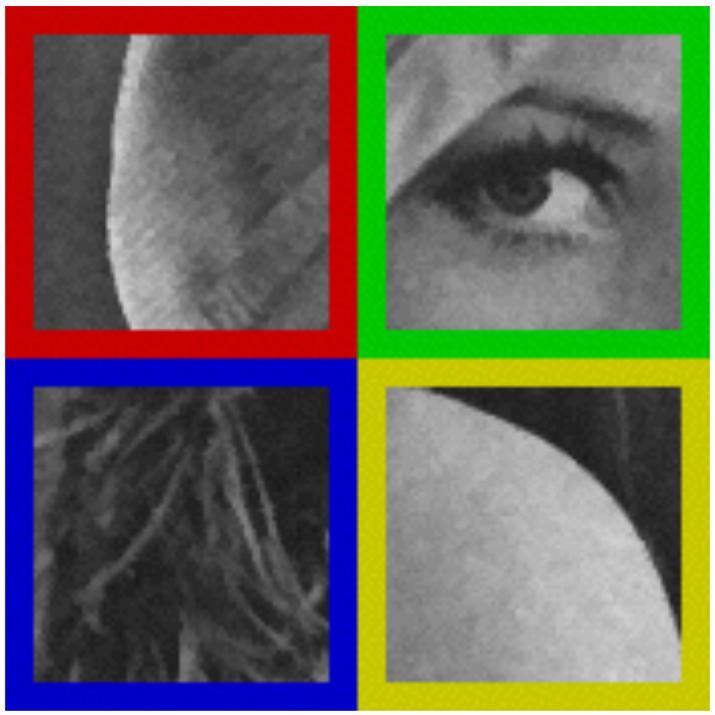
Image denoising:



Poisson noise



Original Image



GC Filtered Image

Applications and Benchmarks

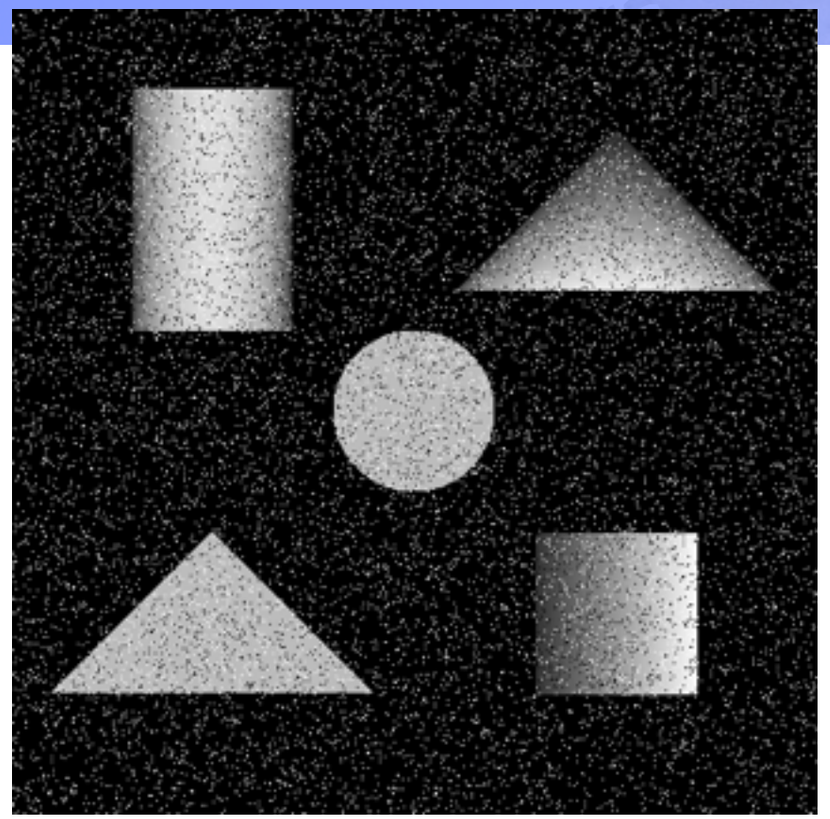
Image denoising:

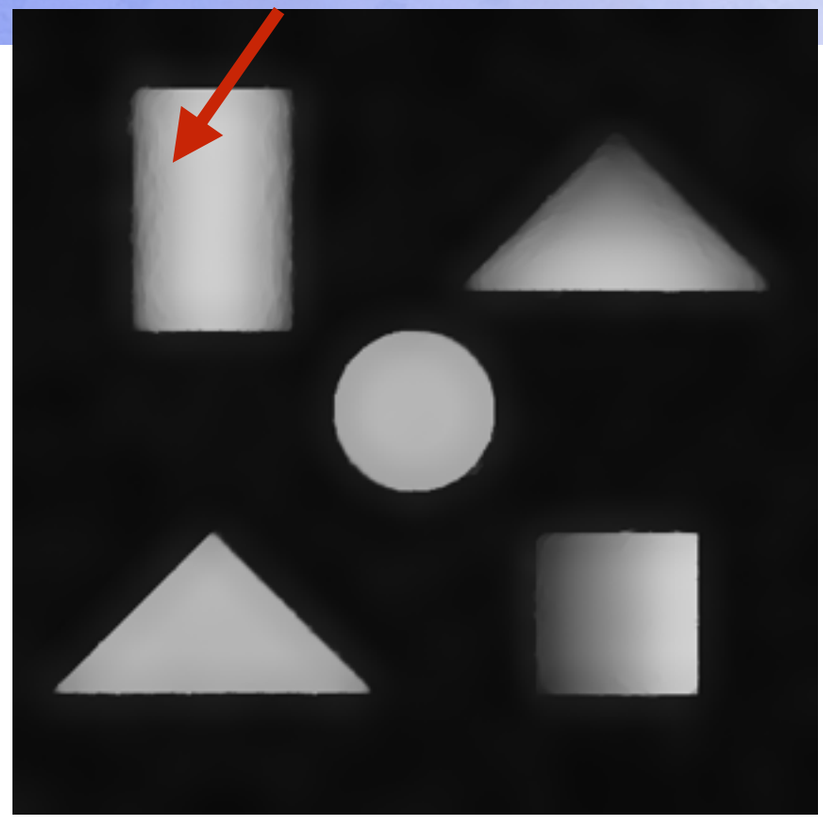
noise type	Gaussian	Salt+Pepper	Poisson
noisy image	21.60	12.26	28.31
after $\mathcal{G}_c^{10}$	29.15	31.7	32.42

Table 1: PSNR of the noisy and filtered *Lena* images.

Applications and Benchmarks

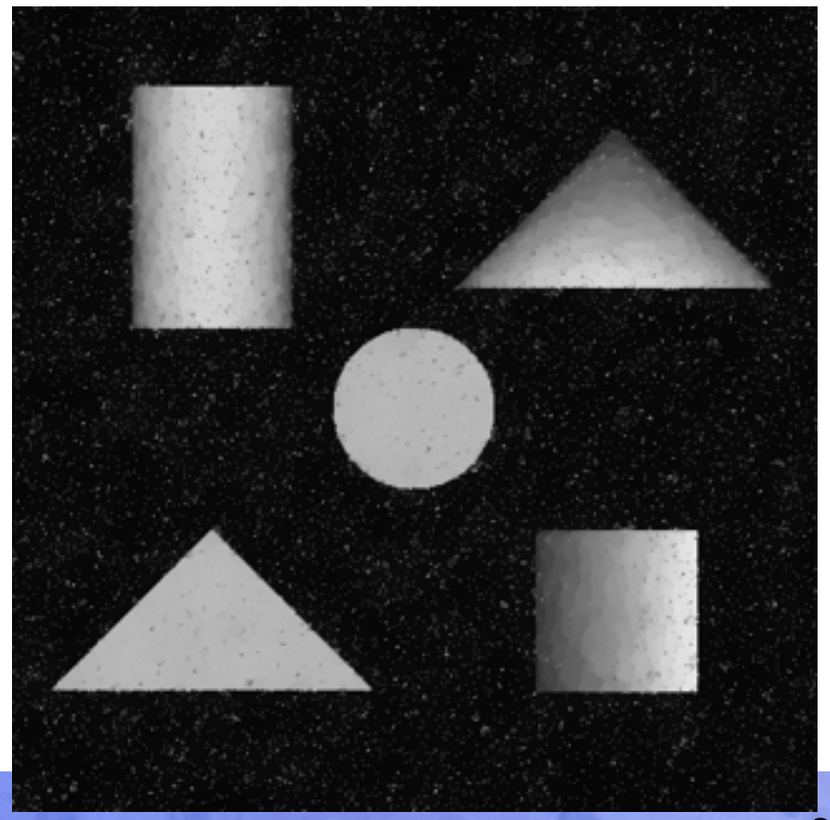
Original Image
PSNR=13.3

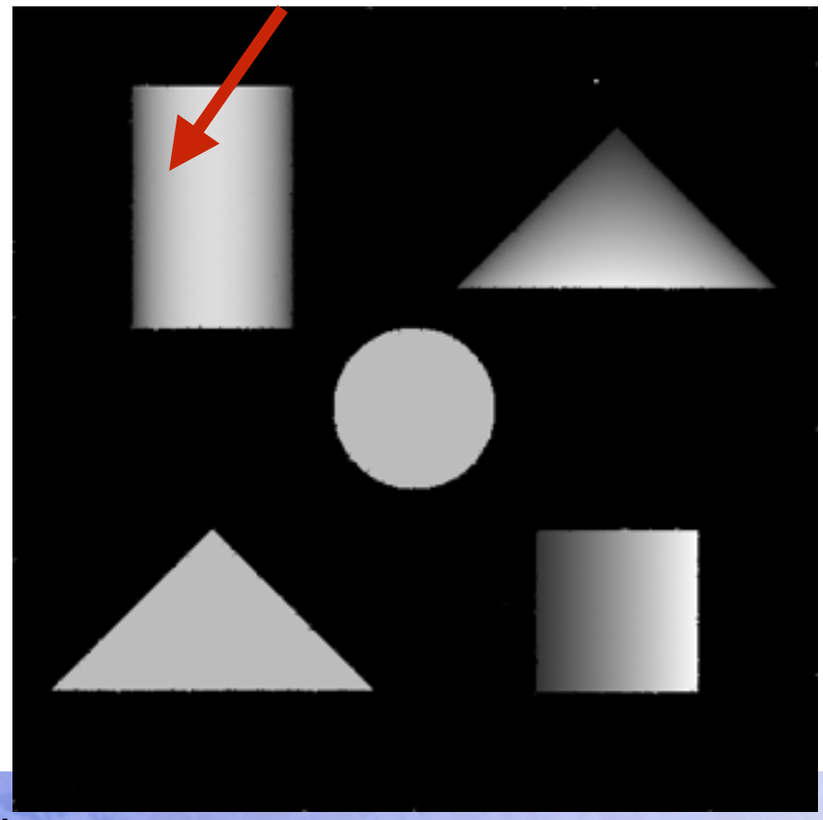




TV
large lambda
PSNR=23.9

TV
small lambda
PSNR=21.6





GC filter
PSNR=35.6

Applications and Benchmarks

Benchmark on BSDS500:

Average runtime on the whole data set

For diffusion: 3.9 seconds, dt = 0.01, iteration = 2000

For GC filter: 0.021 seconds, iteration = 10

both are in C++

Theoretically, one iteration of GC filter should be faster than one iteration of diffusion.

However: our GC filter implementation is not cache efficient because of the domain decomposition.

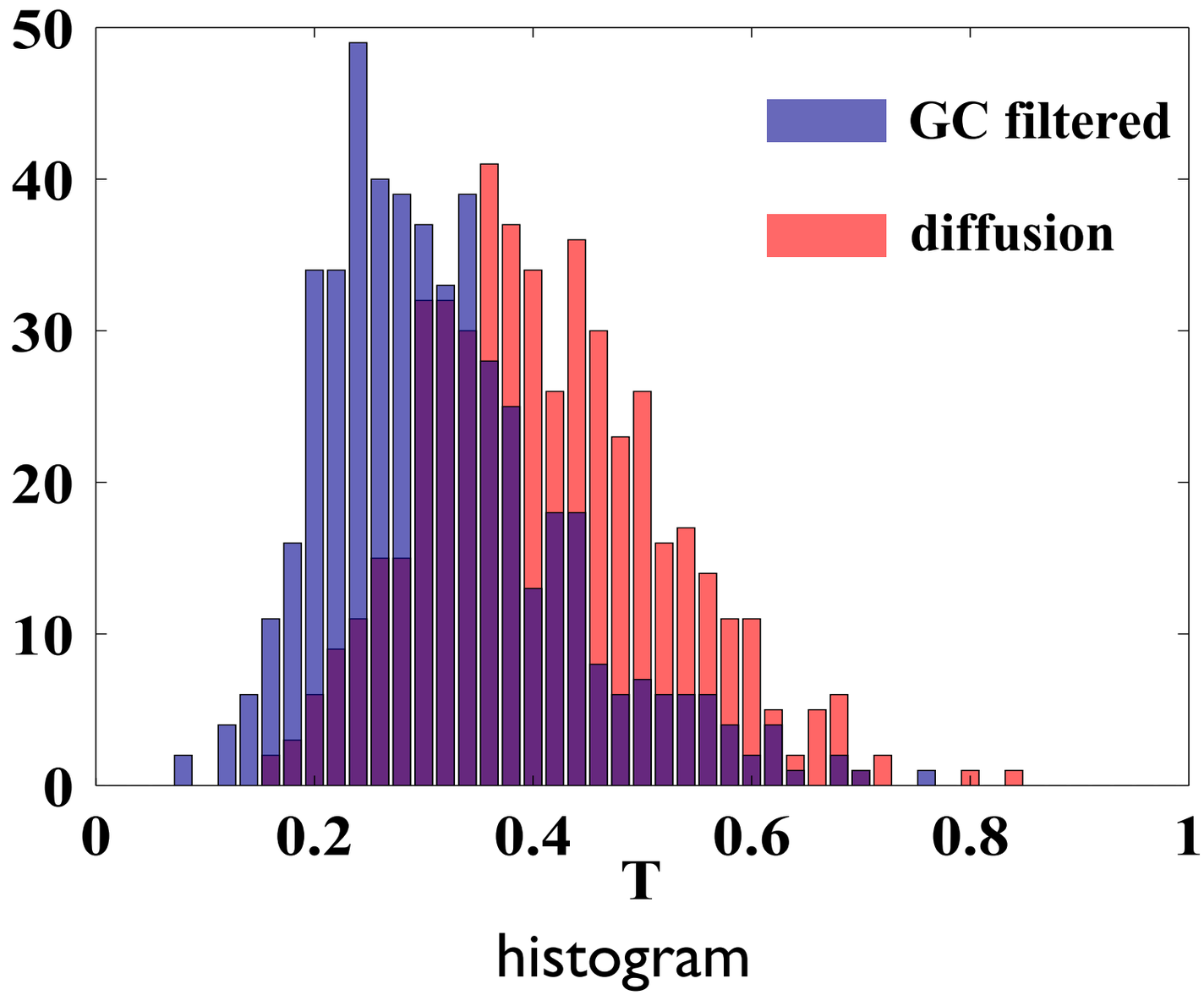
Applications and Benchmarks

Benchmark on BSDS500:

processed image

$$T = \frac{\mathcal{E}(\hat{U})}{\mathcal{E}(U)}$$

original image



histogram

not only faster, but also better results !!!

Applications and Benchmarks

One example from BSDS500:

original image

$\mathcal{E}$ =364.19

diffusion result

$\mathcal{E}$ =195.13

GC filter result

$\mathcal{E}$ =89.45

More examples are in the supplementary.

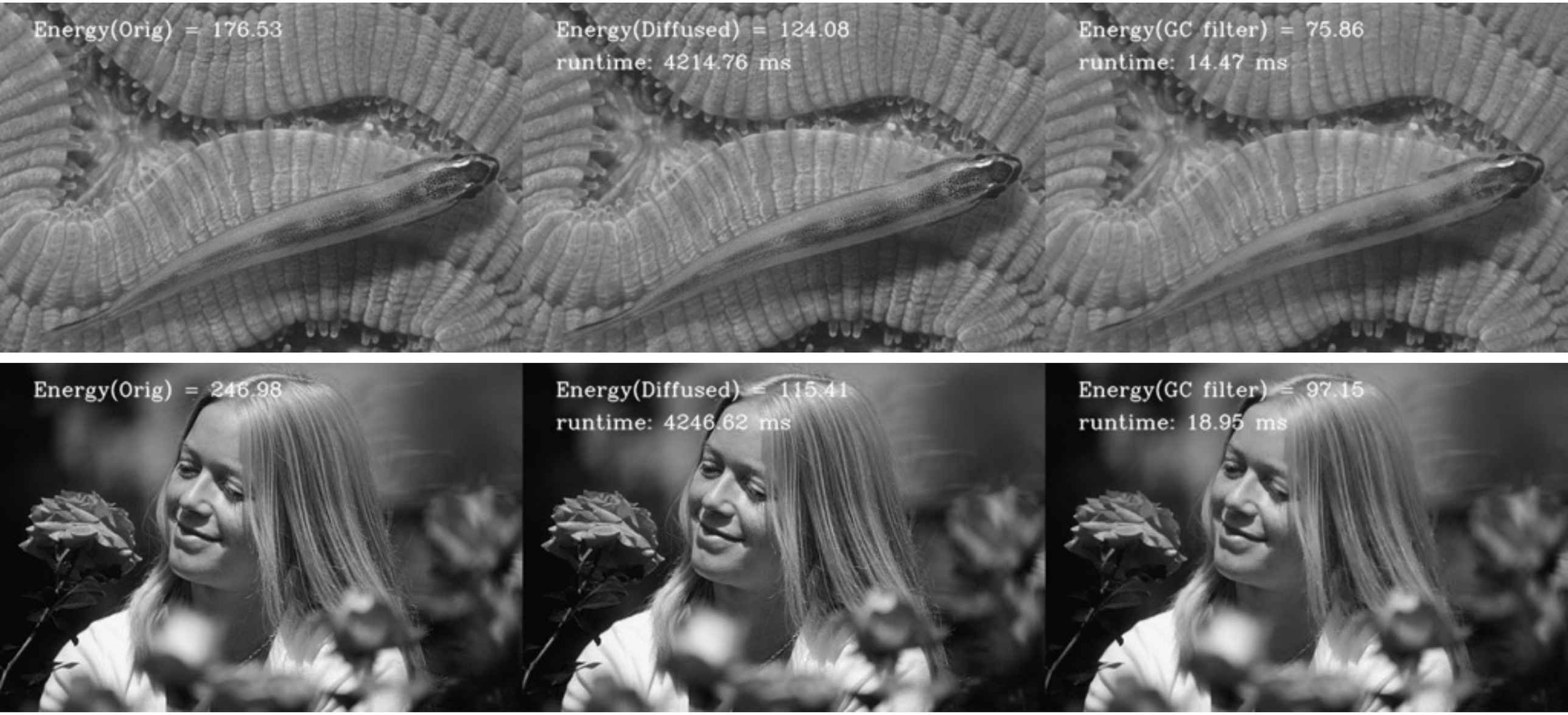
Applications and Benchmarks

More examples from BSDS500:

More examples are in the supplementary.

Applications and Benchmarks

More examples from BSDS500:



More examples are in the supplementary.

Applications and Benchmarks

comparison with other edge-preserving methods:

Original Image

AM

DT

GF

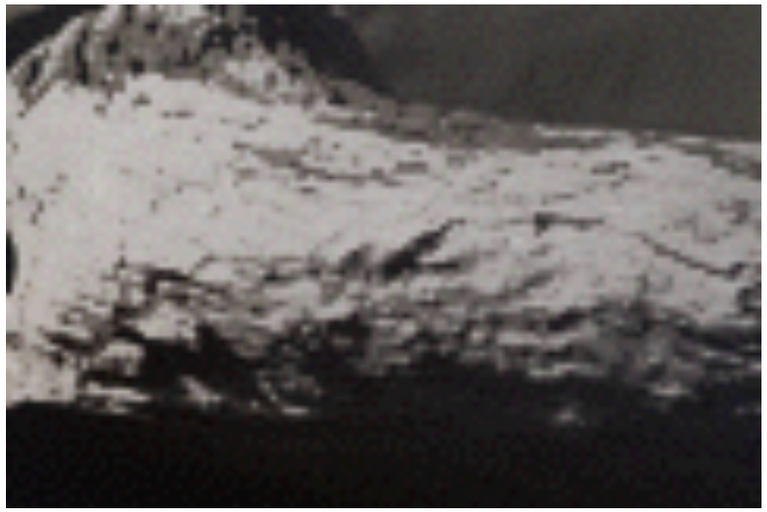
RTV

GC filtered

Applications and Benchmarks

comparison with other edge-preserving methods:

Original Image



AM

DT

GF

RTV

GC filtered

Applications and Benchmarks

comparison with other edge-preserving methods:

Original Image

AM

DT

GF

RTV

GC filtered

Applications and Benchmarks

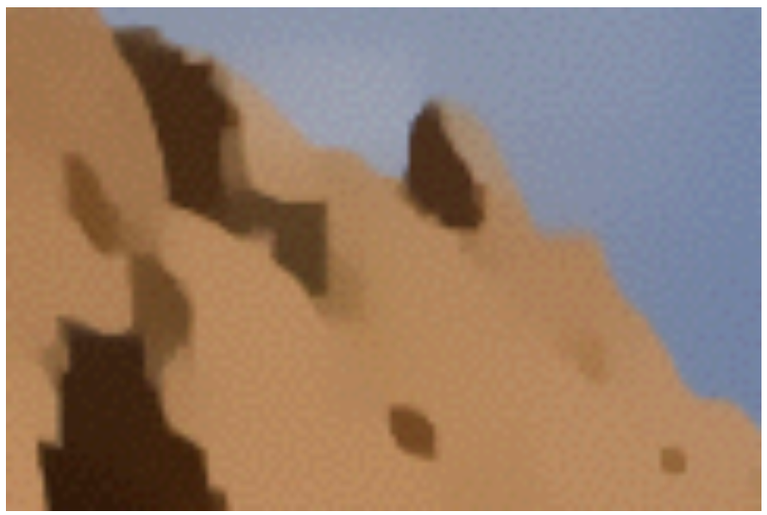
comparison with other edge-preserving methods:

Original Image

AM

DT

GF



RTV

GC filtered

Applications and Benchmarks

comparison with other edge-preserving methods:

Original Image

AM

DT

GF

RTV

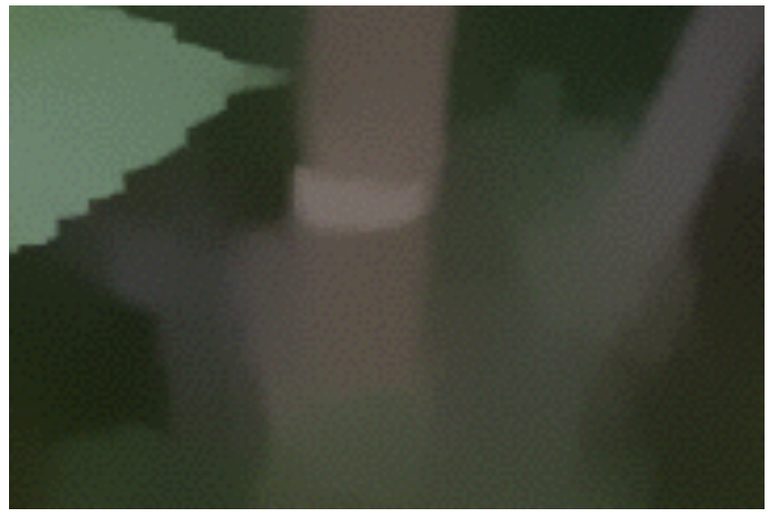
GC filtered

Applications and Benchmarks

comparison with other edge-preserving methods:

Original Image

AM



DT

GF

RTV

GC filtered

Applications and Benchmarks

comparison with other edge-preserving methods:

Image	WLS	AM	DT	GF	L0	RTV	$\mathcal{G}_c^1$
Beach	15.3	4.7	118.1	2.0	12.3	14.6	1.6
Flower	15.6	4.3	118.0	2.1	12.3	15.1	1.6
Rock	15.2	3.9	119.4	1.9	13.8	13.9	1.7
Eyes	16.4	4.6	114.7	1.8	12.0	12.3	1.5
avg.	15.7	4.4	117.5	1.9	12.2	14.1	1.7

Runtimes in seconds/MPixel, tested on a single core of a 2 GHz Intel Core i7 processor.
All methods are implemented in MATLAB.

Applications and Benchmarks

comparison with other edge-preserving filters:

Method	Modeling	Multi/Point	Preserving	limitation
Bilateral	Piecewise constant	Point wise	edge	edge flip, slow
CLMF0	Piecewise constant	Multipoint	edge	edge flip
Guided Filter	Piecewise linear	Multipoint	gradient	not spatial adaptive
CLMF1	Piecewise linear	Multipoint	gradient	local window regression
RTV	Piecewise constant	Point wise	edge	staircase artifact, slow
GC filter	Piecewise developable	Point wise	developable surface	complex in larger window

Table 1: Compare with other edge preserving filters.

New unique model of piecewise developable images!

Outline

- Introduction
- Gaussian Curvature Filter
- Applications and Benchmarks
- Conclusions

Conclusion

Our Goal:

Minimize Gaussian curvature efficiently without explicitly computing it.

Conclusion

Properties of the present GC Filter:

- Two orders of magnitude faster than diffusion-based methods
- Converges fast (ten iterations are enough in practice)
- Minimizes GC without computing it
- The resulting image is continuous, but NOT smooth (edges!!!)
- Preserves developable surfaces
- Linear computational complexity
- Easy to implement (100 lines C++) and parallelize

Thank you!

source code at https://github.com/YuanhaoGong