A screen space GPGPU surface LIC algorithm for parallel interactive exploration of large composite datasets

Astronum 2014

B. Loring LBNL, H. Karimabadi and V. Roytershteyn UCSD, W. Daughton LANL

March 16, 2015

Introduction

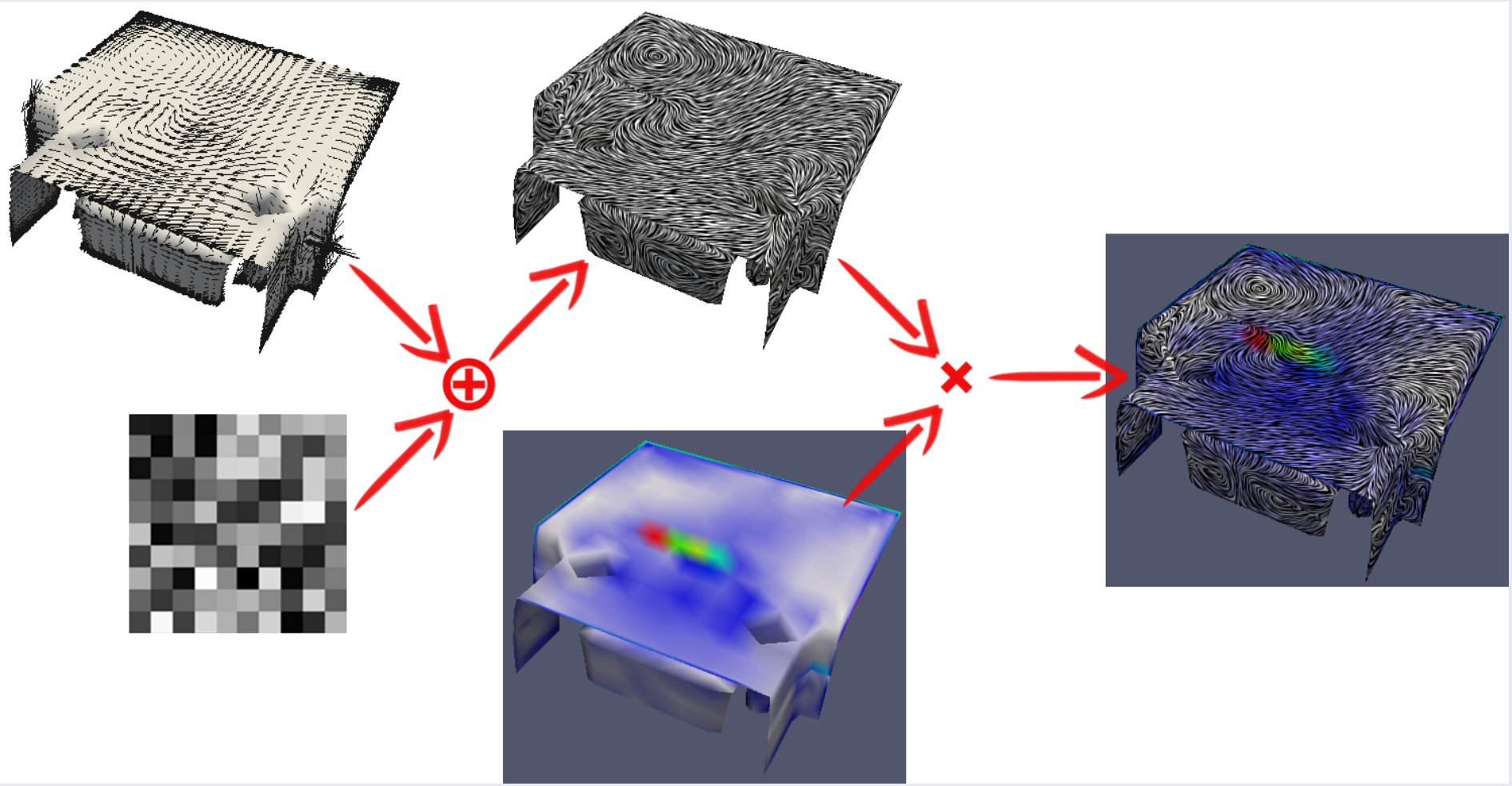
Goal of the work

We set out to design a surface LIC algorithm that:

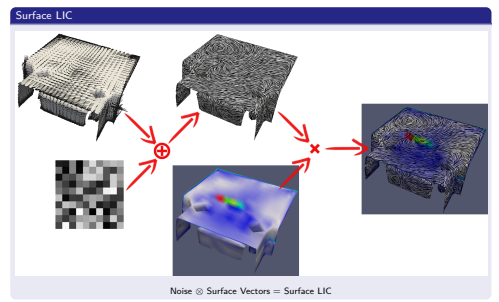
1. handles all common datasets, including AMR, multiblock etc...
2. is parallel, can process large simulation data
3. deployable in existing vis tools
4. delivers interactive performance
5. is flexible enough to work well on wide range of data
6. exposes tuning parameters to the user

Introduction

Surface LIC



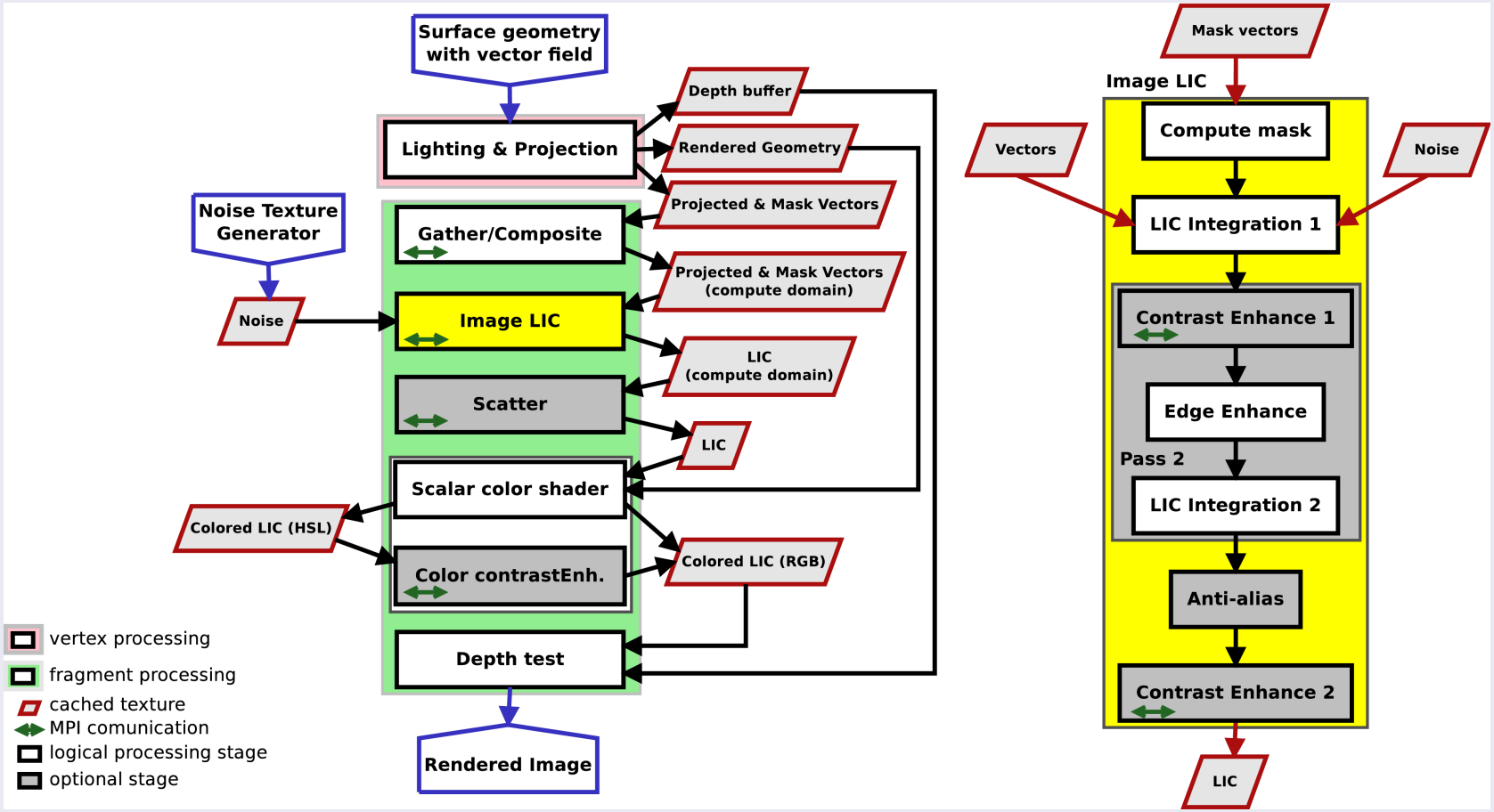
Noise $\otimes$ Surface Vectors $=$ Surface LIC

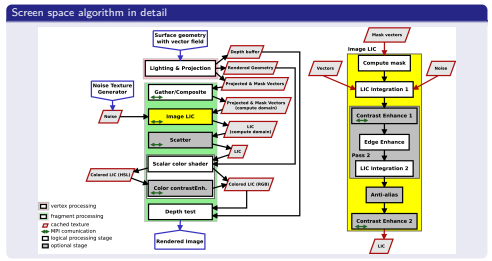


* global view of the vector field, easily identify features
* computationally costly, GPGPU implementatation to make it interactive
* working in screen space introduces some interesting complexity and challenges
* especially for the parallelization, if you expect it to work in existing parallel vis tools

Introduction

Screen space algorithm in detail

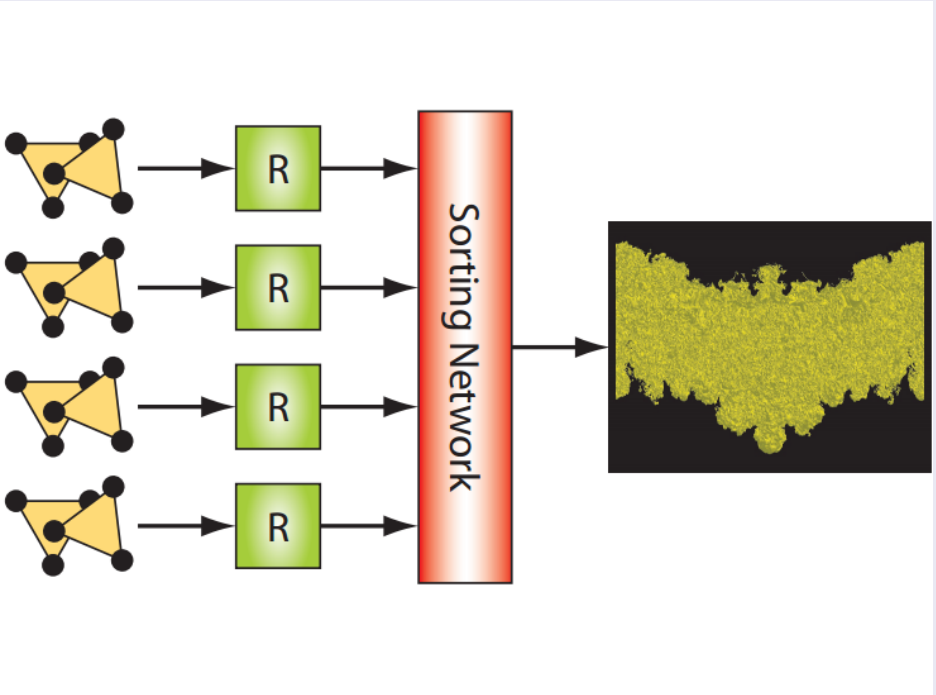


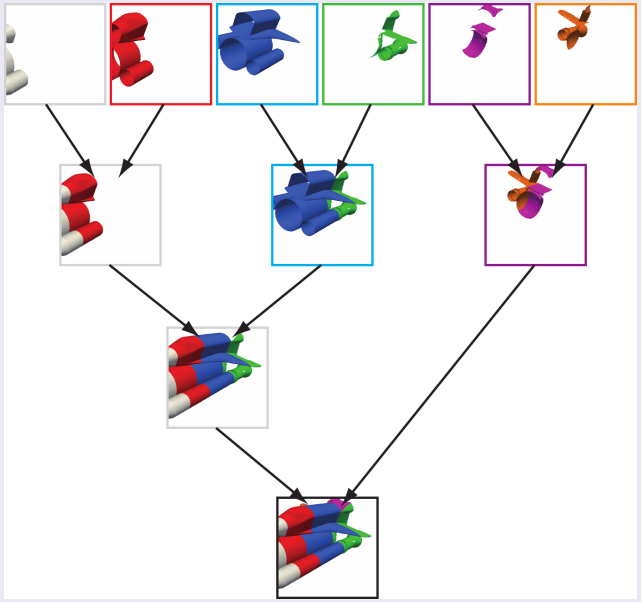


* lighting and projection shader * screen space after this * compositing stages for parallel * image LIC is broken out on the right * 2 pass image LIC, first pass convolves noise, second pass convolves first pass after image processing filters * some of the image processing stages require guard pixels for correct parallel operations * some require reduction * shaders for combining LIC and rendered geometry * stages are cached * works with IceT image compositing library used by ParaView and VisIt

Parallelization

Typical sort-last redering

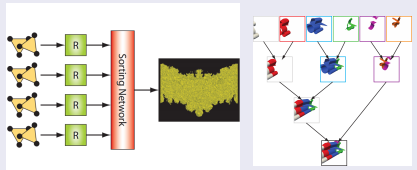




- Render geometry distributed. z-buffer compositing reduction to rank 0 selects visible data. image delivery.
- Typically simulation domain decomp is used. Pixels processed where the lie. Fine rendering is typically fast compared to I/O and processing.
- Sorting is done in screen space. communication pattern depends on view paramters.
- We're working with the IceT compsiting library used by ParaView and VisIt.

Parallelization

Typical sort-last redering

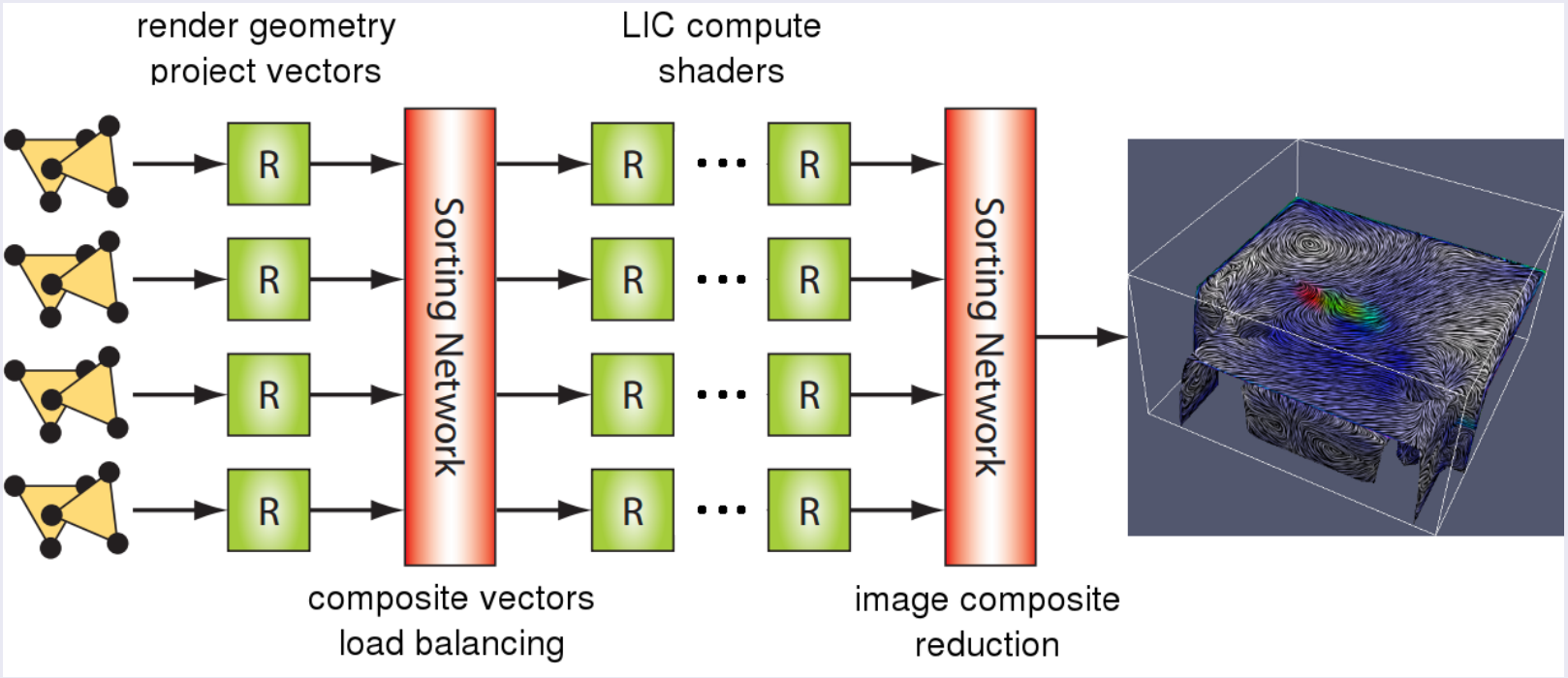


- Render geometry distributed. z-buffer compositing reduction to rank 0 selects visible data. image delivery.
- Typically simulation domain decomp is used. Pixels processed where the lie. Fine rendering is typically fast compared to I/O and processing.
- Sorting is done in screen space. communication pattern depends on view paramters.
- We're working with the IceT compsiting library used by ParaView and VisIt.

distributed geometry using simulation domain decomp
screen space domain decomposition is view dependent
cloesest pixels need to be displayed thus z-buffer compositing

Parallelization

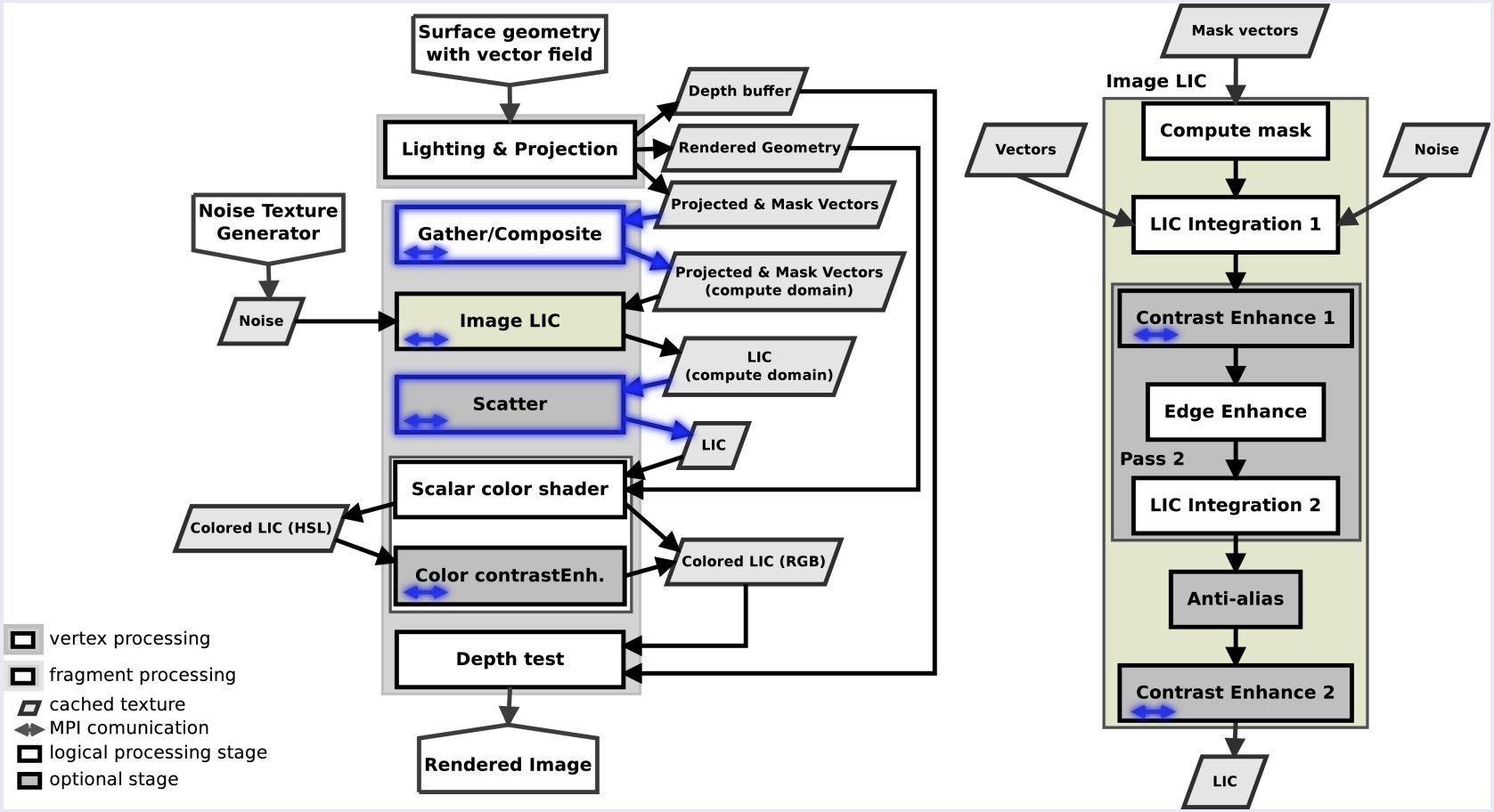
Sort last with screen space surface LIC



- prior to integration composite vectors where screen space projections overlap
- need “guard pixels” at screen space block boundaries
- shader communication, eg. to get min/max for histogram stretching

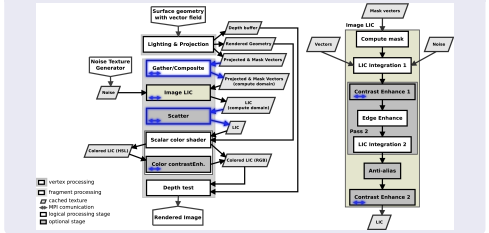
Parallelization

Communication stages



Parallelization

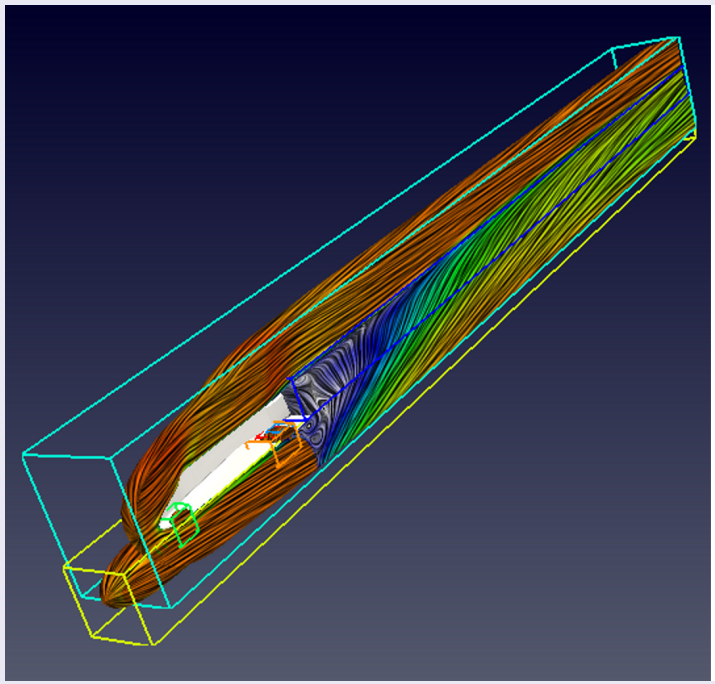
Communication stages

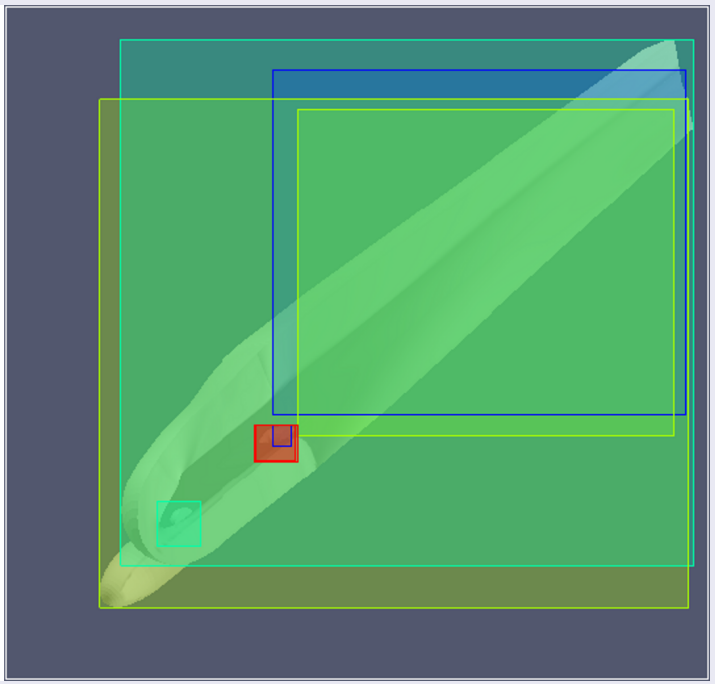


* compositing + ghost generation done in the gather stage * image processing stages need min/max * scatter is a transfer only op, no compositing * scatter stage is needed *if* we do any screen space load balancing because IceT pre-computes the image space bounds * this could be addressed so that the scatter is not needed, but then we need to composite color buffer so its actually not that bad

Parallelization

Working in screen space



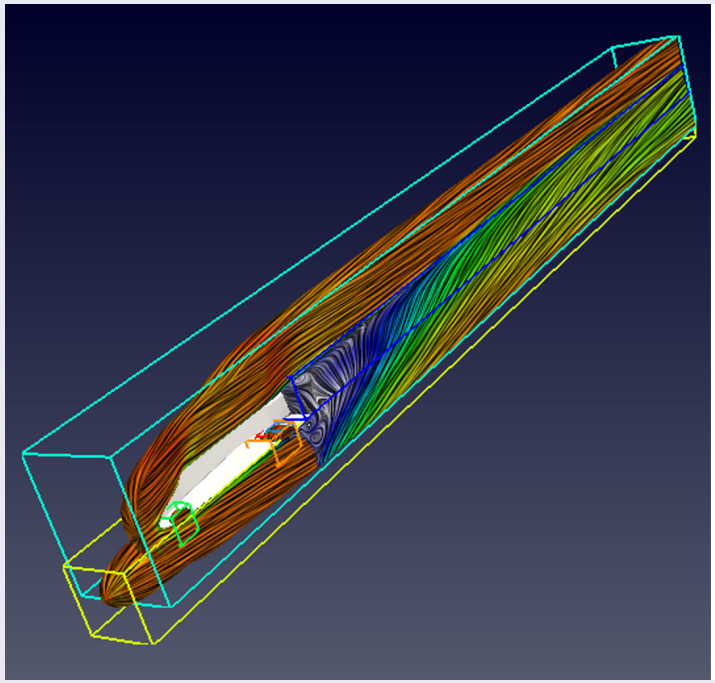


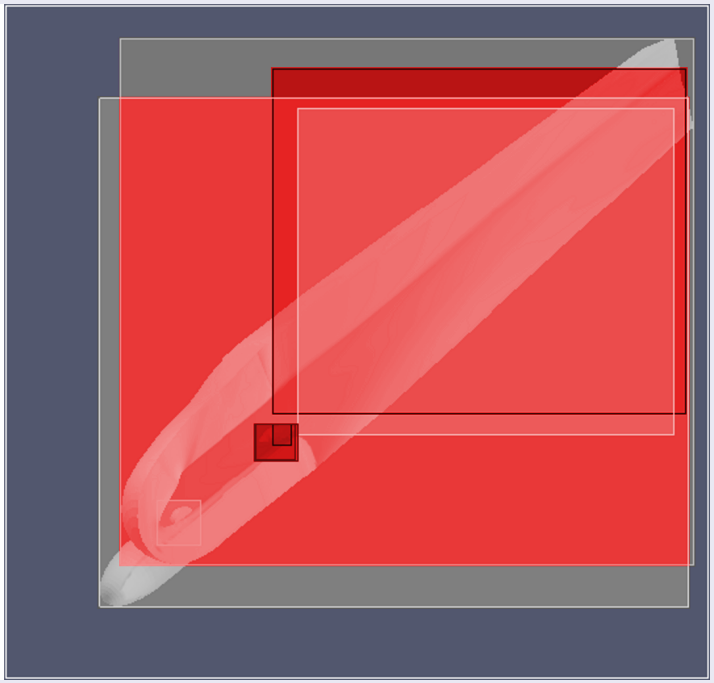
In-place Composite Lazy. Process on the existing screen space extents.

- ✓ Simple, works with IceT
- ✓ It's the best if there's no overlap (ie slice)

Parallelization

Working in screen space





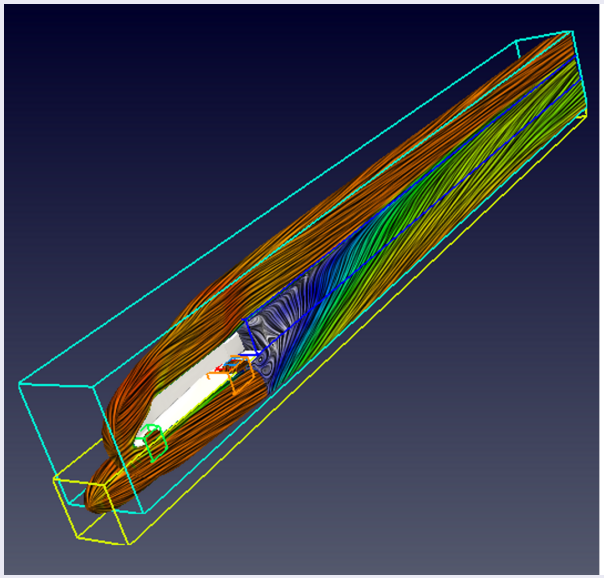
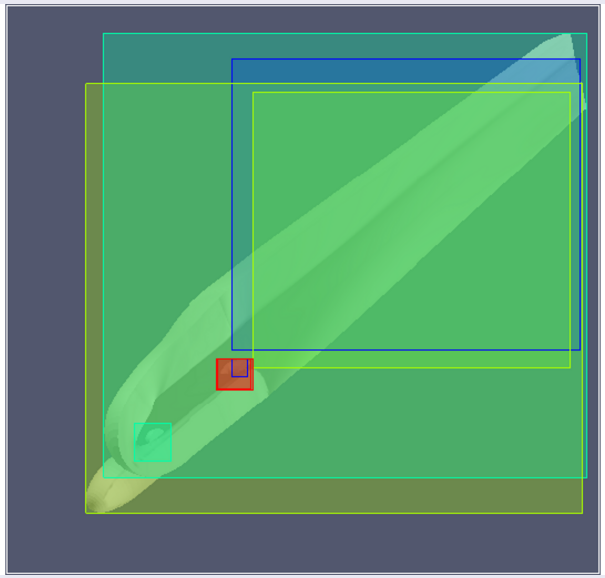
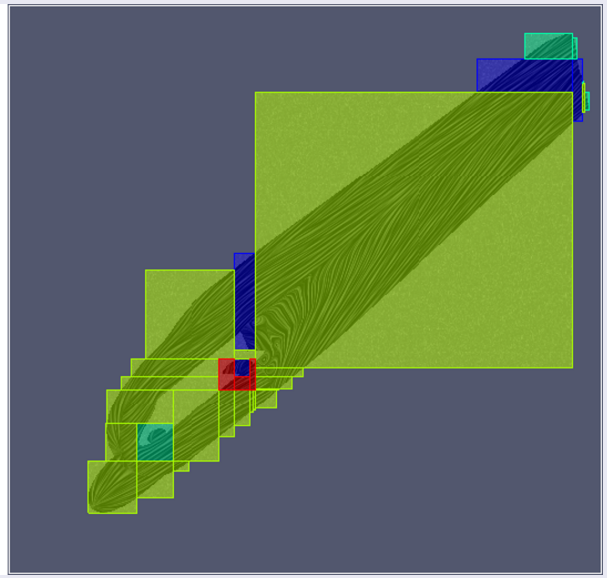
In-place Composite Lazy. Process on the existing screen space extents.

- ✓ Simple, works with IceT
- ✓ It's the best if there's no overlap (ie slice)
- ✗ view dependent/not load balanced
- ✗ high communication costs. all process pairs with overlapping regions exchange data
- ✗ redundant computation can limit performance

screen-space overlap is costly!!

Parallelization

Working in screen space

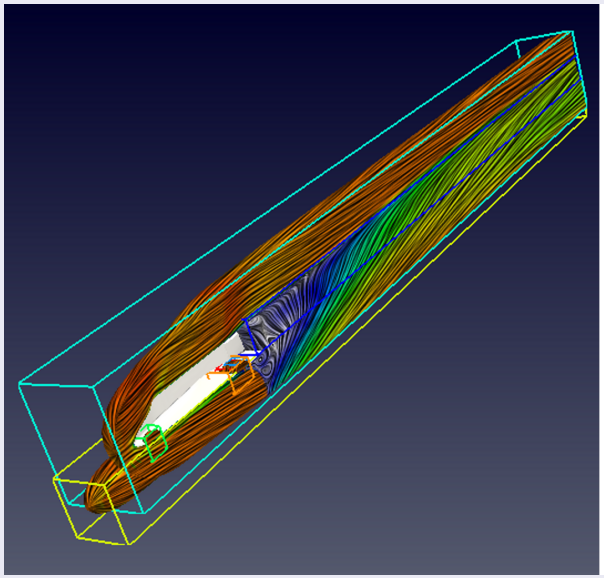
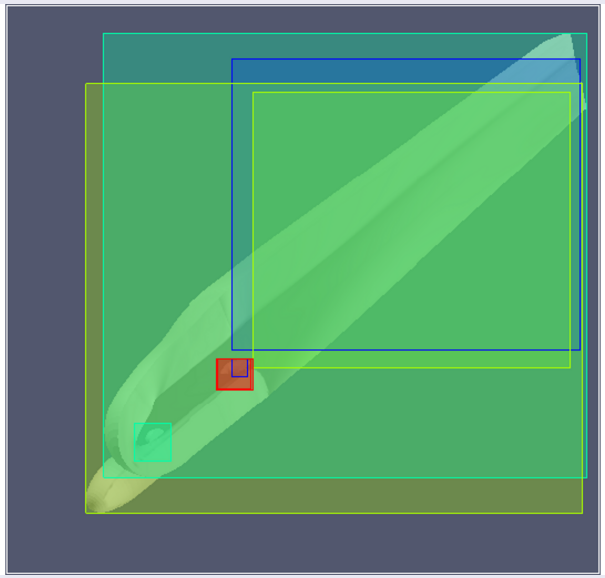
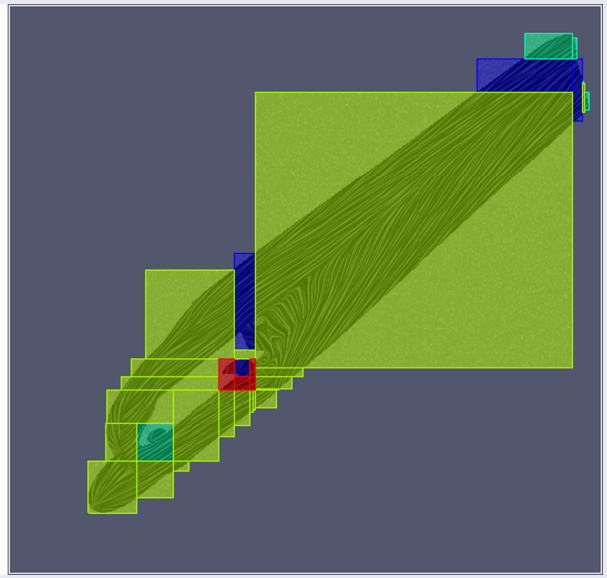




In-place Disjoint Composite *Less* lazy. Process on *unique parts* of existing screen space ext.

- ✓ reduced comminication costs by $\frac{1}{2}$ for each overlapping processor pair
- ✓ computing once per pixel is a big win, esp. w/o GPUs
- ✓ smaller regions further reduce comm and compute costs

Parallelization

Working in screen space

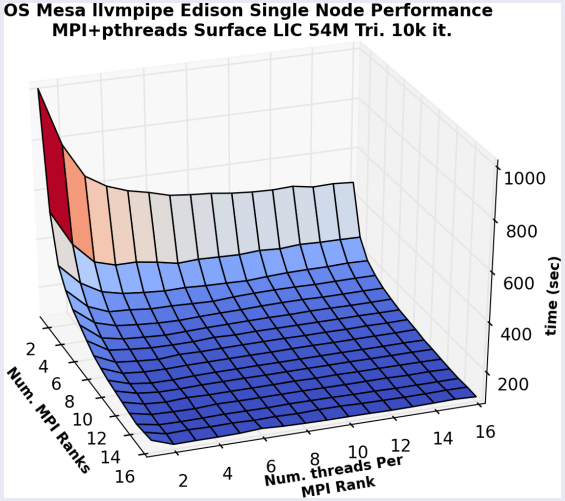




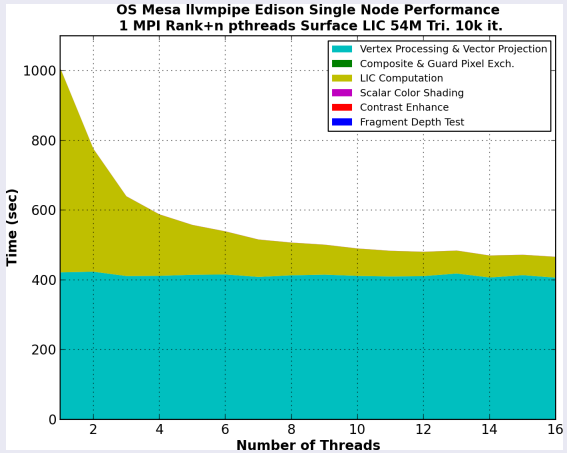
In-place Disjoint Composite *Less* lazy. Process on *unique parts* of existing screen space ext.

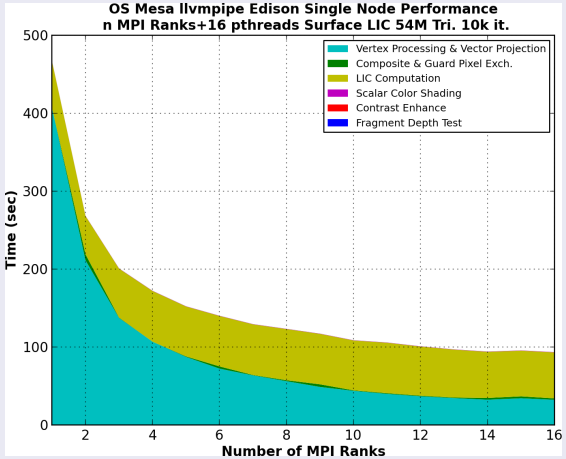
- ✓ reduced comminication costs by $\frac{1}{2}$ for each overlapping processor pair
- ✓ computing once per pixel is a big win, esp. w/o GPUs
- ✓ smaller regions further reduce comm and compute costs
- ✗ disjointification adds some overhead
- ✗ more patches means more guardpixels means more communication
- ✗ doesn't solve load balancing issues

Parallelization

Single node performance



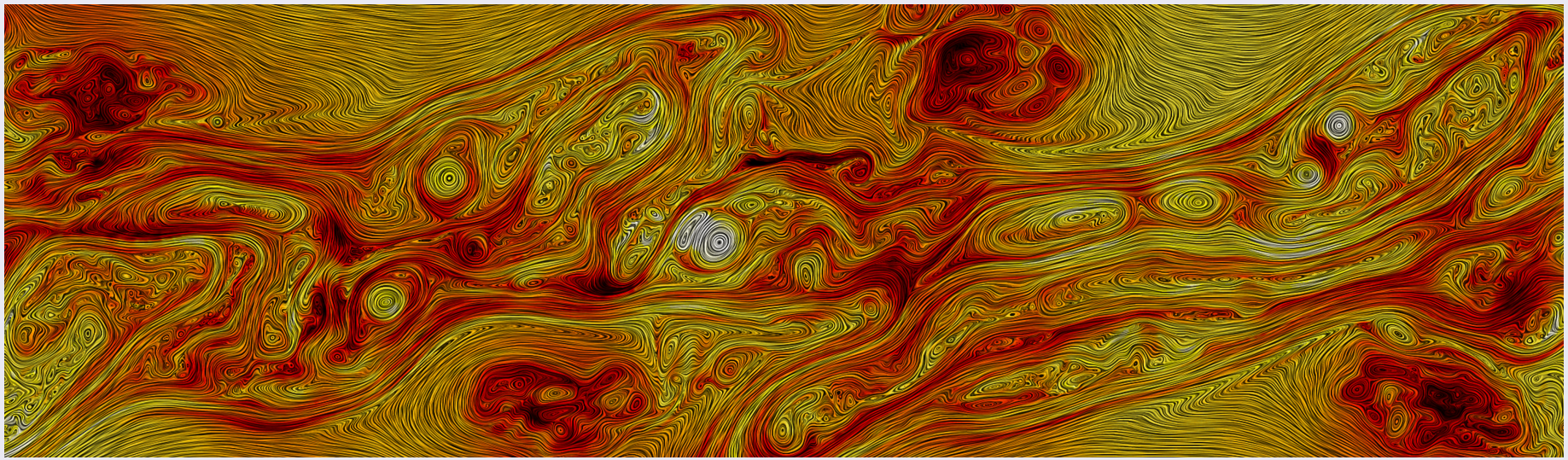


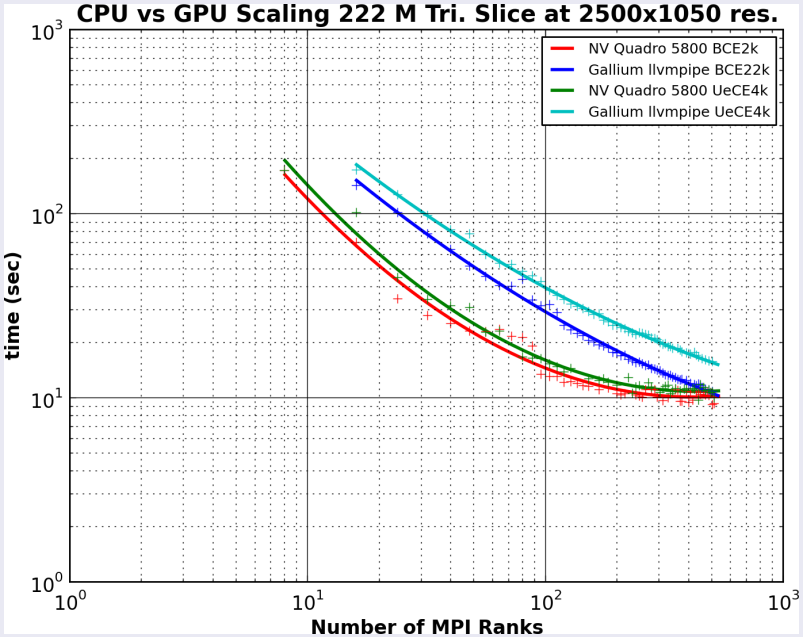


- w/o GPU
  - Best results when number of threads on the node is equal to the number of hyperthreads.
  - For benchmarks above and below run 1/2 packed, 4 processes per socket, each with 4 threads.
- w GPU
  - Dual GPU nodes
  - Run fully packed 4 processes per GPU

Parallelization

Performance and Scaling w/wo GPU

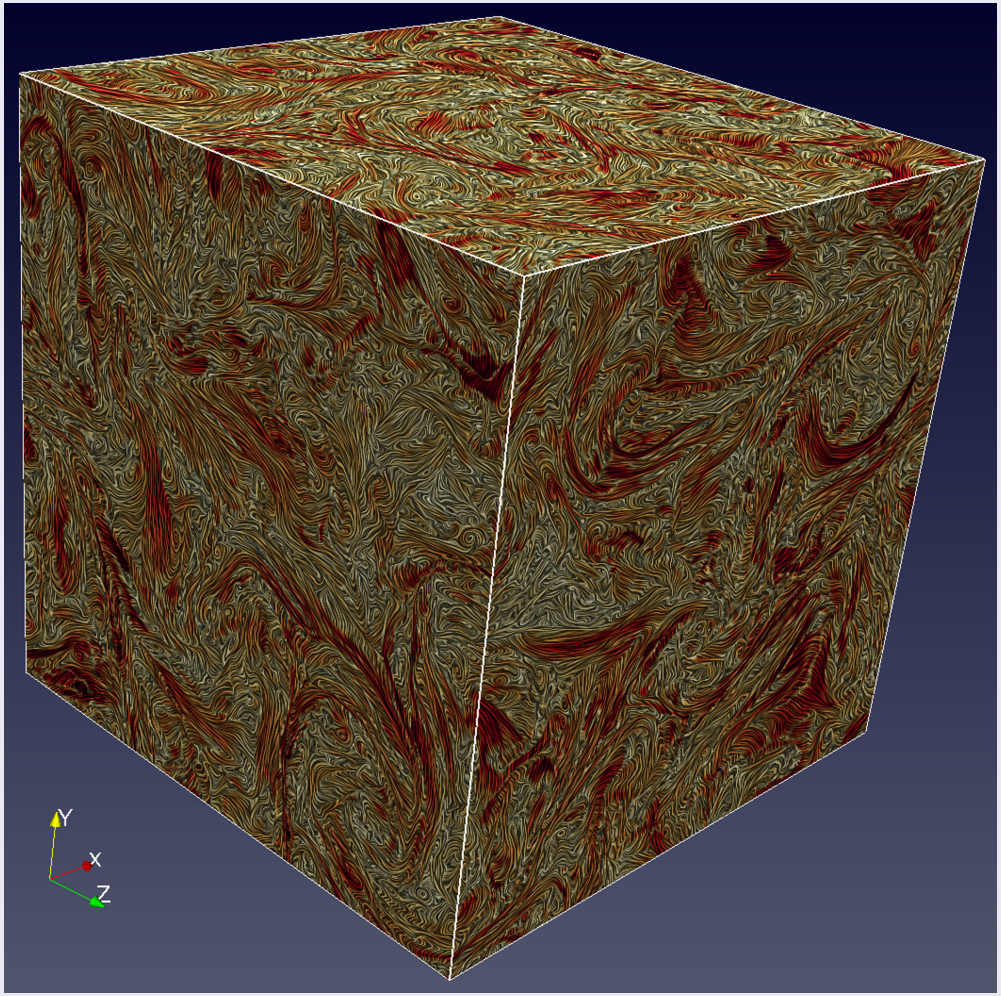


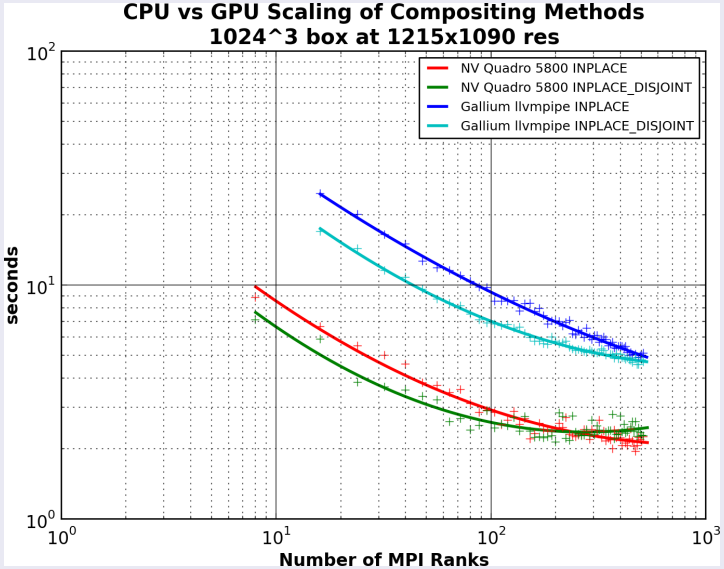


- 2D 16384x1890 cell PIC plasma turbulence simulation. 7TB of data generated 40000 cores in 72 hours on Jaguar.
- 222 Million triangles. 2550x1050 rendering
- GPU faster but CPU scales better
  - CPU is really slow!
  - GPU needs work to amortize the data transfers over PCI
  - more processes $\longrightarrow$ more guard pixels $\longrightarrow$ more comm
  - Edison has faster lower latency network hardware

Parallelization

Performance and Scaling of Compositing Alg w/wo GPU

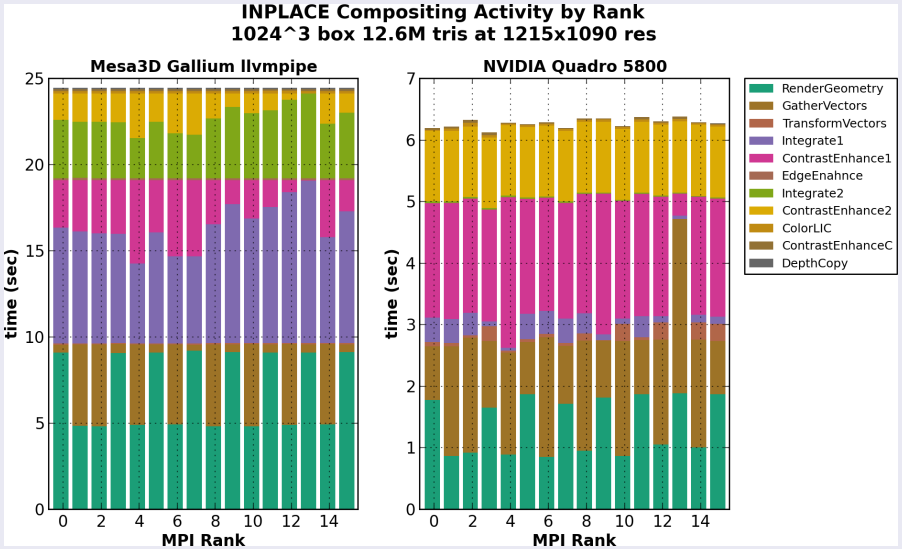


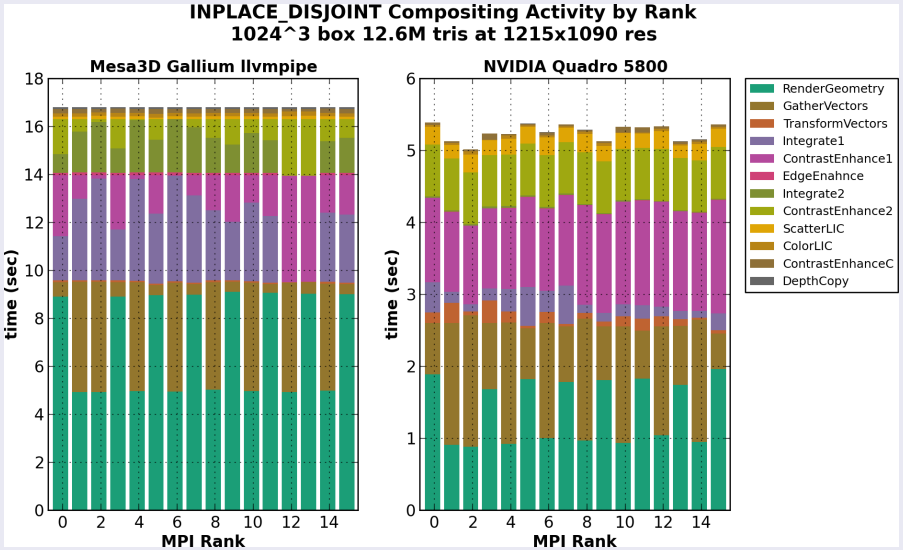


- 3D 1024x1024 MHD plasma turbulence simulation
- 54 Million triangles. 1215x1090 rendering
- GPU faster but CPU scales better
  - Same as before
  - Disjointification may be resulting more guard pixels and fragmentation

Parallelization

Performance and Scaling of Compositing Alg w/wo GPU



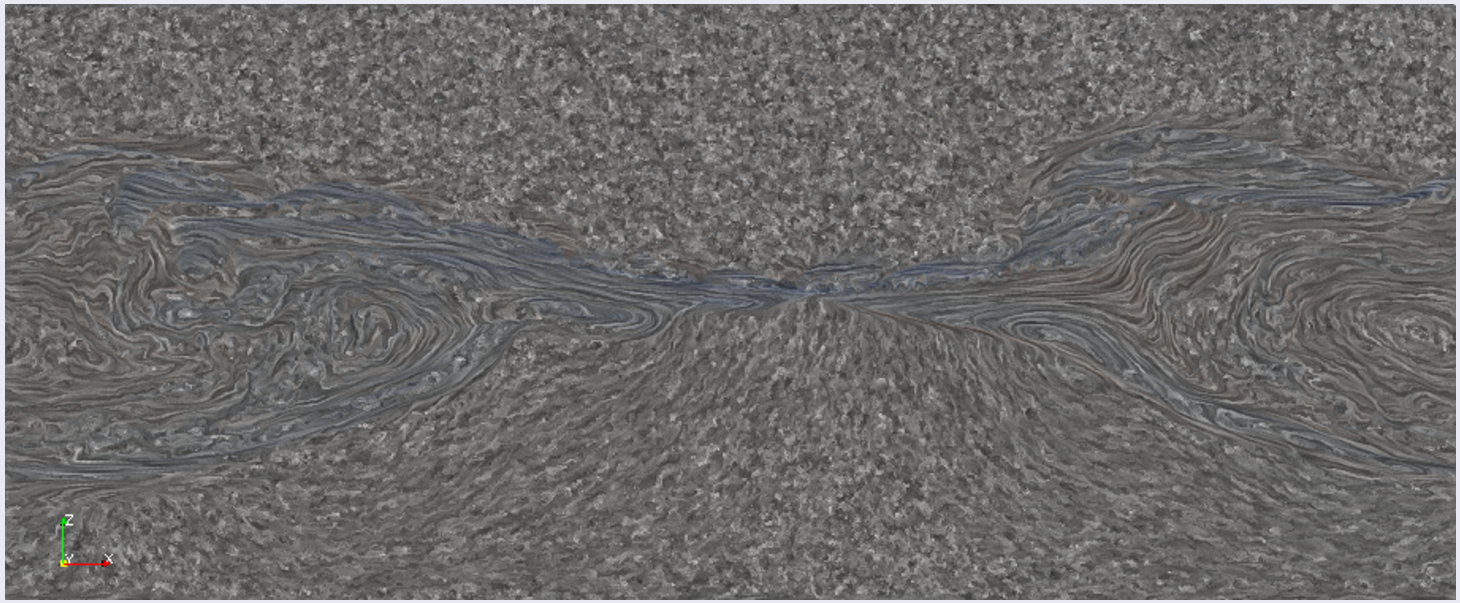


- 16 rank run, slowest run, differences are easier to spot
- first column : w/o GPU , disjointification shaves off time in integration (purple) and communication(brown), scatter stage(yellow second row) adding very little overhead
- second column : w GPU , communication is the bottleneck, gather (brown), contrast enhance (pink, yellow).

Addressing visual and perceptual issues

Issues addressed by our work

Motivation:



Issues:

- Results are highly dependent on inputs, many inputs, noise, vector, surface, lighting, view, camera, etc
- Convolution of noise naturally reduces contrast and narrows dynamic range
- Combination of LIC with pseudocolored lit geometry often loses subtle variations in pseducoloring, lighting.

What properties in LIC/shaders enable best combination with lit pseudocolored geometry?

Addressing visual and perceptual issues

First: How is LIC combined with lit pseudocolored geometry?

Blend:

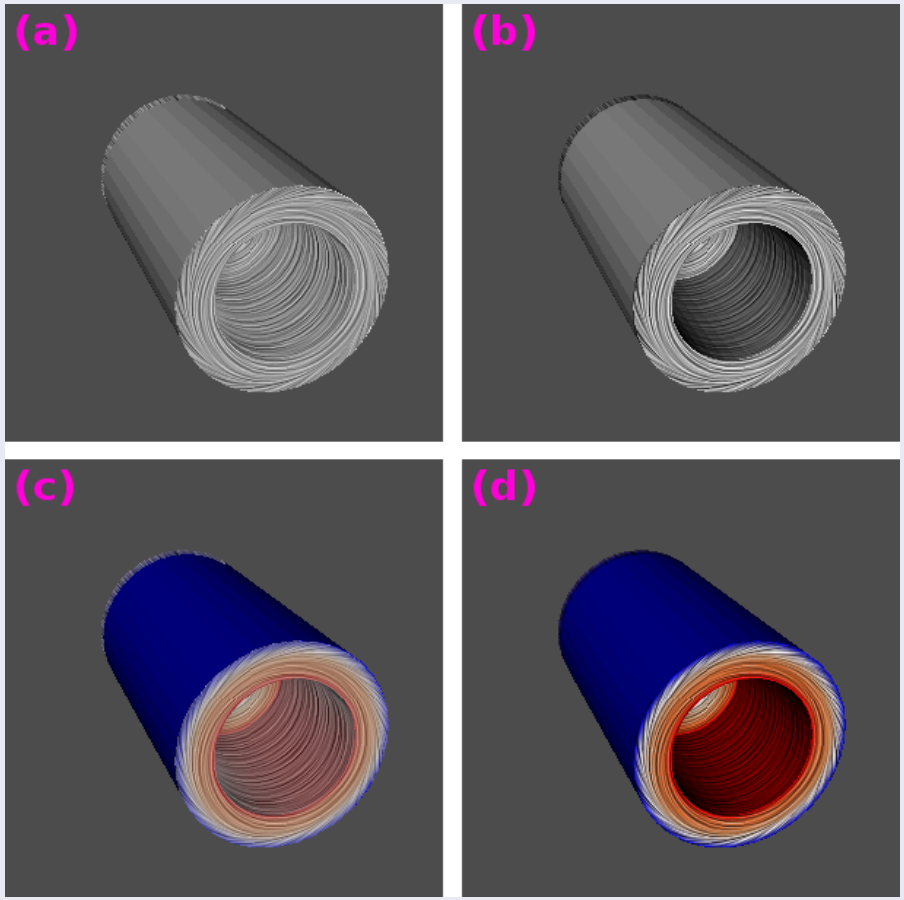
$$c_{ij} = L_{ij} * I + S_{ij} * (1 - I) \quad (1)$$

where the indices i, j identify a specific fragment, c is final RGB color, L is LIC gray scale value, S is the scalar RGB color, and I is a constant ranging from 0 to 1.

Multiply:

$$c_{ij} = (L_{ij} + f) * S_{ij} \quad (2)$$

where the indices i, j identify a specific fragment, c is final RGB color, L is LIC gray scale intensity, S is the scalar RGB color, and f is a biasing parameter, typically 0, that may be used for fine tuning.

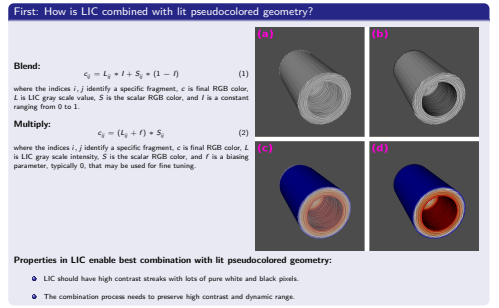


Properties in LIC enable best combination with lit pseudocolored geometry:

- LIC should have high contrast streaks with lots of pure white and black pixels.
- The combination process needs to preserve high contrast and dynamic range.

A screen space GPGPU surface LIC algorithm for parallel interactive exploration of large composite datasets

Addressing visual and perceptual issues



blend

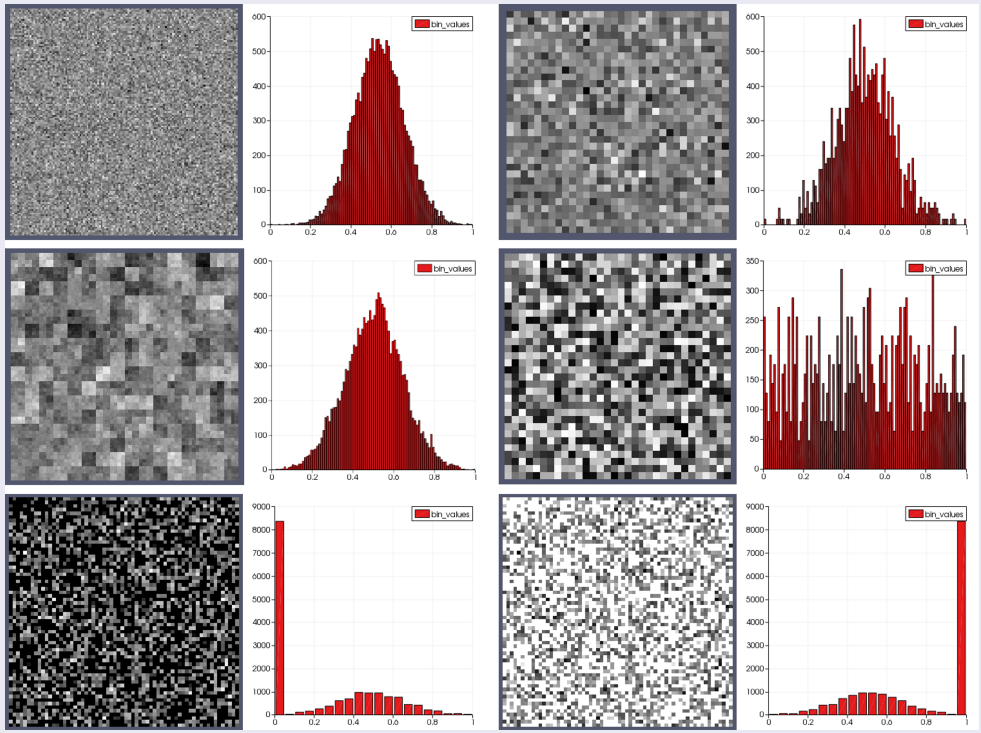
* Decreasing *I* to obtain brighter colors diminishes the intensity of the LIC, and vise versa.
* When colors are bright the LIC is difficult to see.
* Gives a better result curved surfaces than on planar ones.

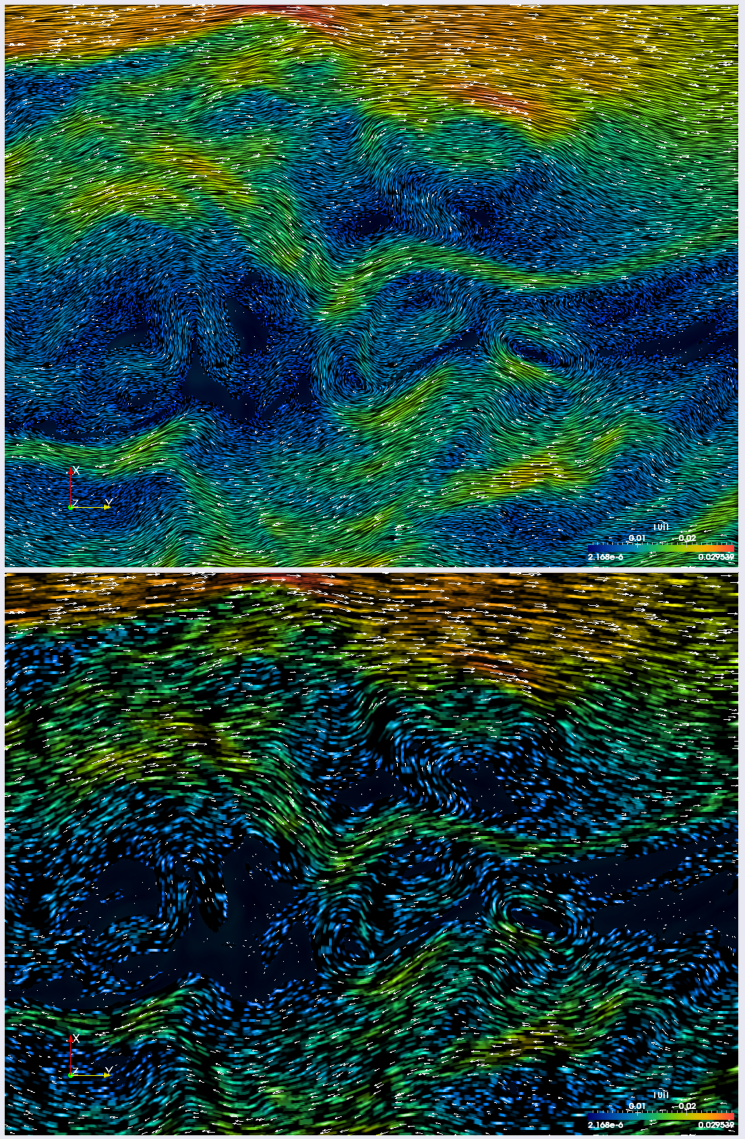
multiply

* Results in crisp LIC and bright colors.
* potentially oversaturated colors

Addressing visual and perceptual issues

Noise distribution and properties

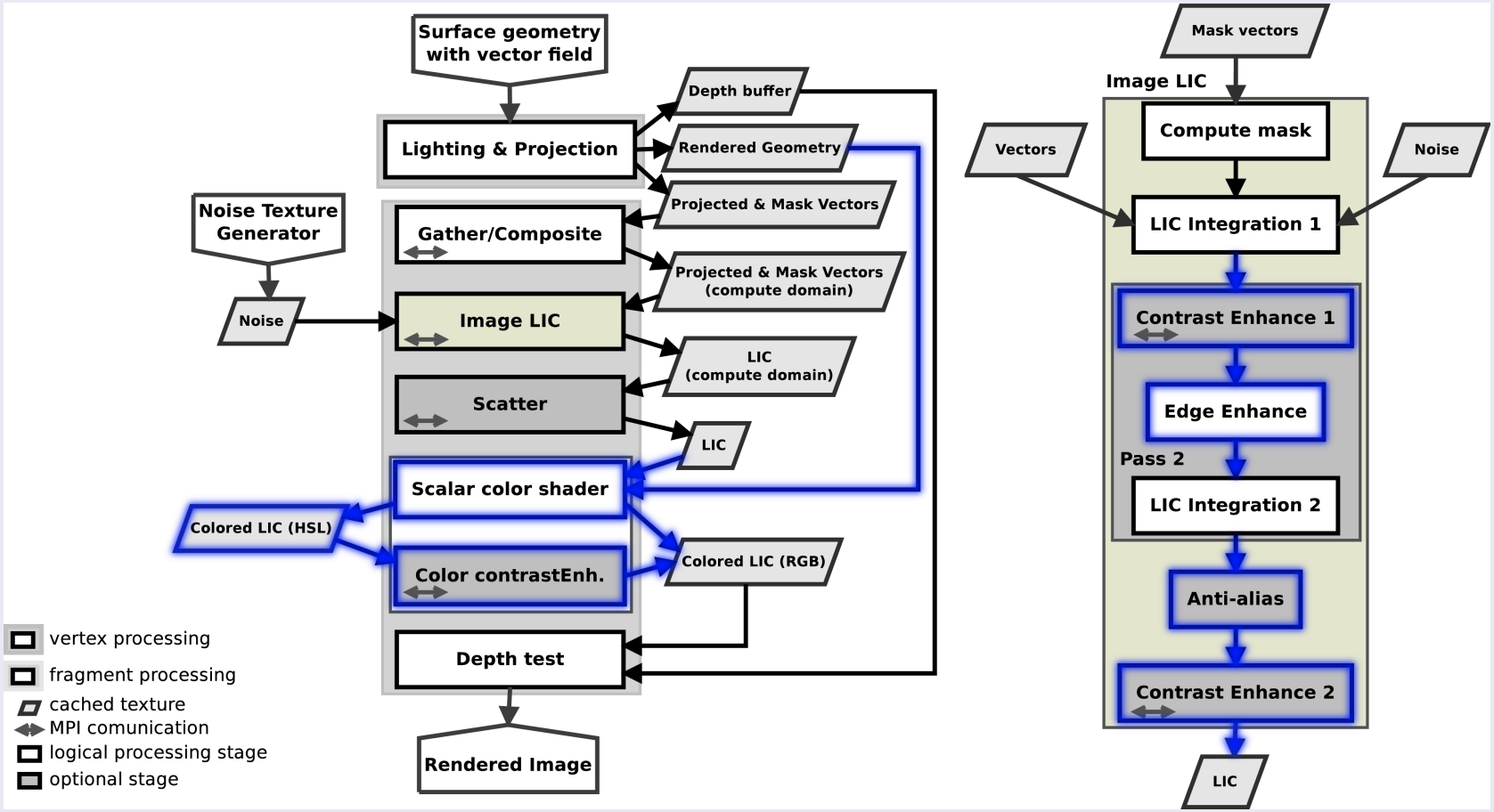




- 9 DOF
  - size, distribution, num levels, grain size, min/max value, impulse probability, background color, seed
- better control over streaking. size, width, intensity

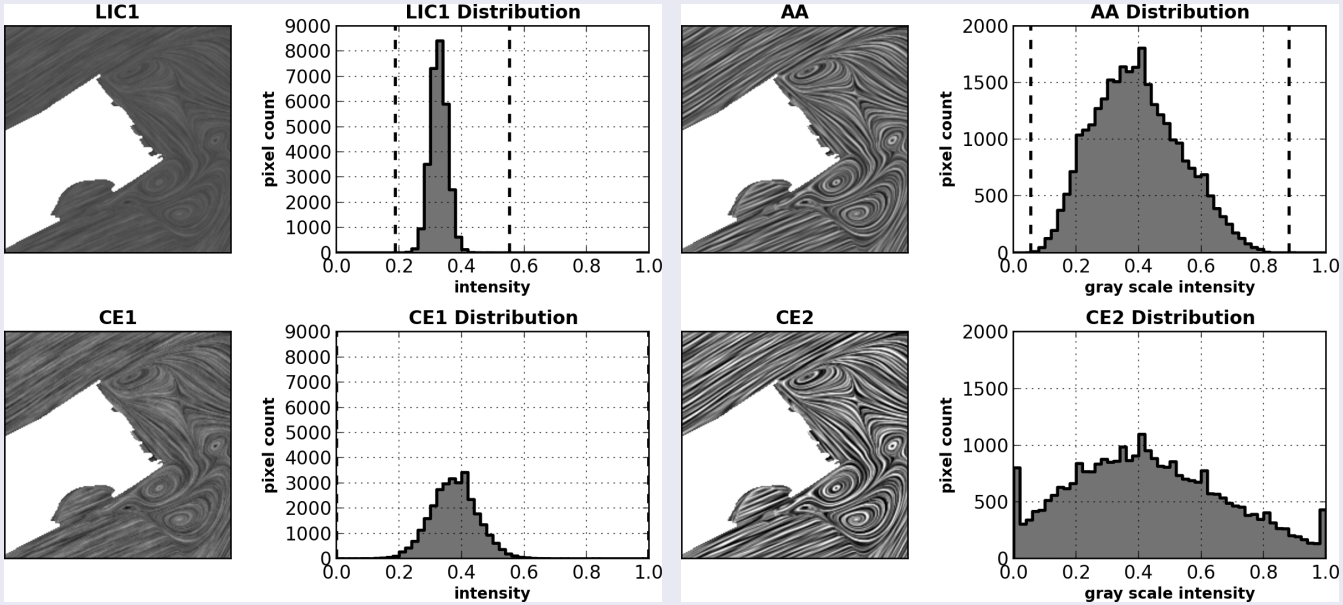
Introduction

Screen space + OpenGL ⇒ use image processing algorithms!



Visual and perceptual issues

Histogram Stretching:
enhances streak contrast and can increase numbers of pure white and black pixels



Linear re-mapping of intensity values

$$I_{ij} = \frac{I_{ij} - m}{M - m} \tag{3}$$

where, the indices i, j identify a specific fragment, I is the fragment's gray scale intensity, m is the intensity to map to 0, M is intensity to map to 1.

Automating the procedure

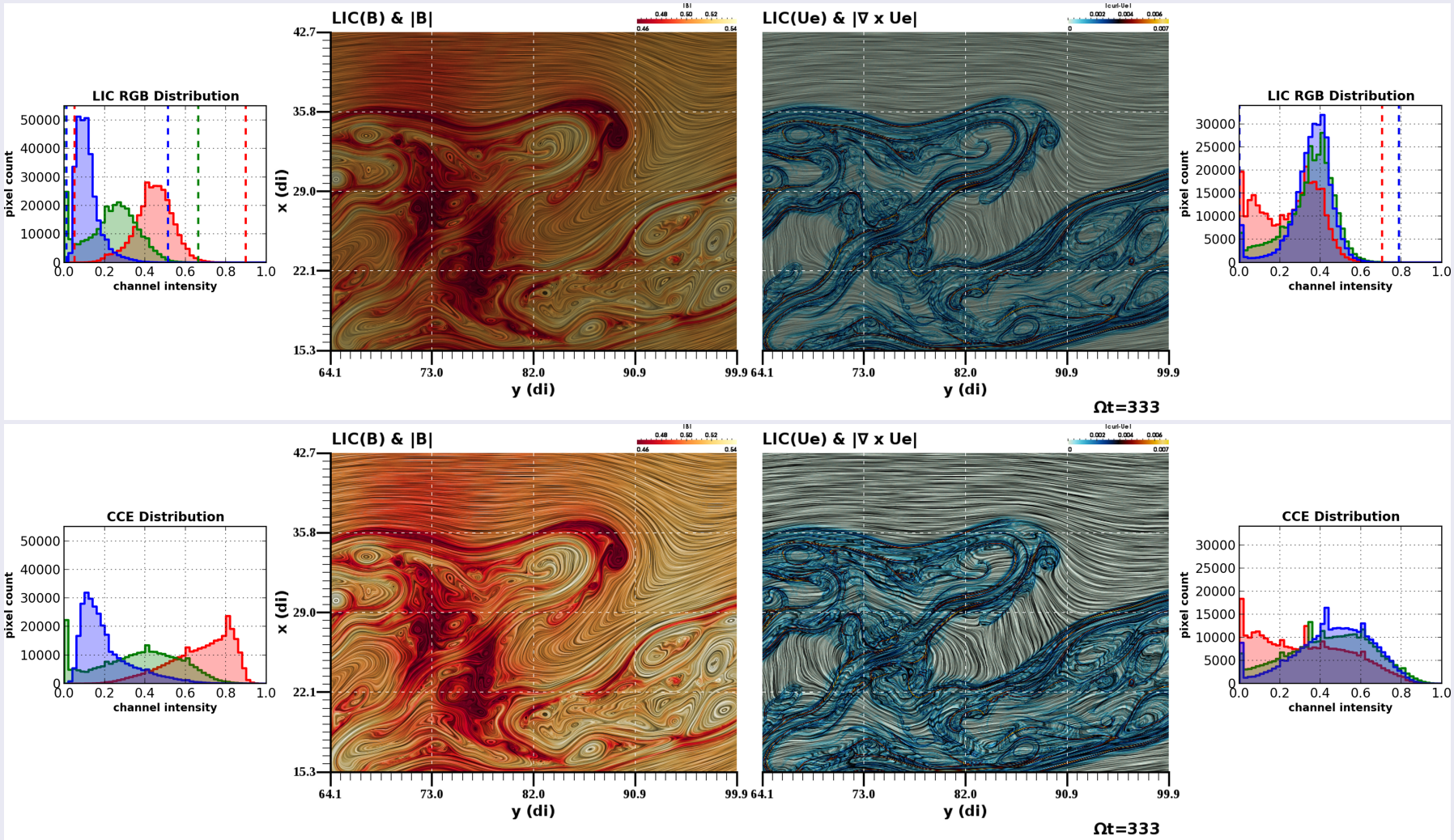
$$m = min(I) + F_m * (max(I) - min(I)) \tag{4}$$

$$M = max(I) - F_M * (max(I) - min(I)) \tag{5}$$

where, min/max are computed on all fragments, F_m and F_M are optional adjustment factors.

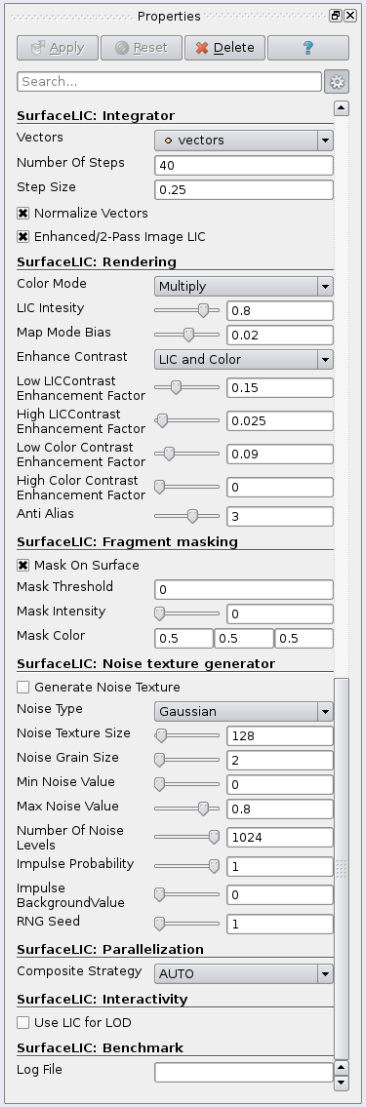
Visual and perceptual issues

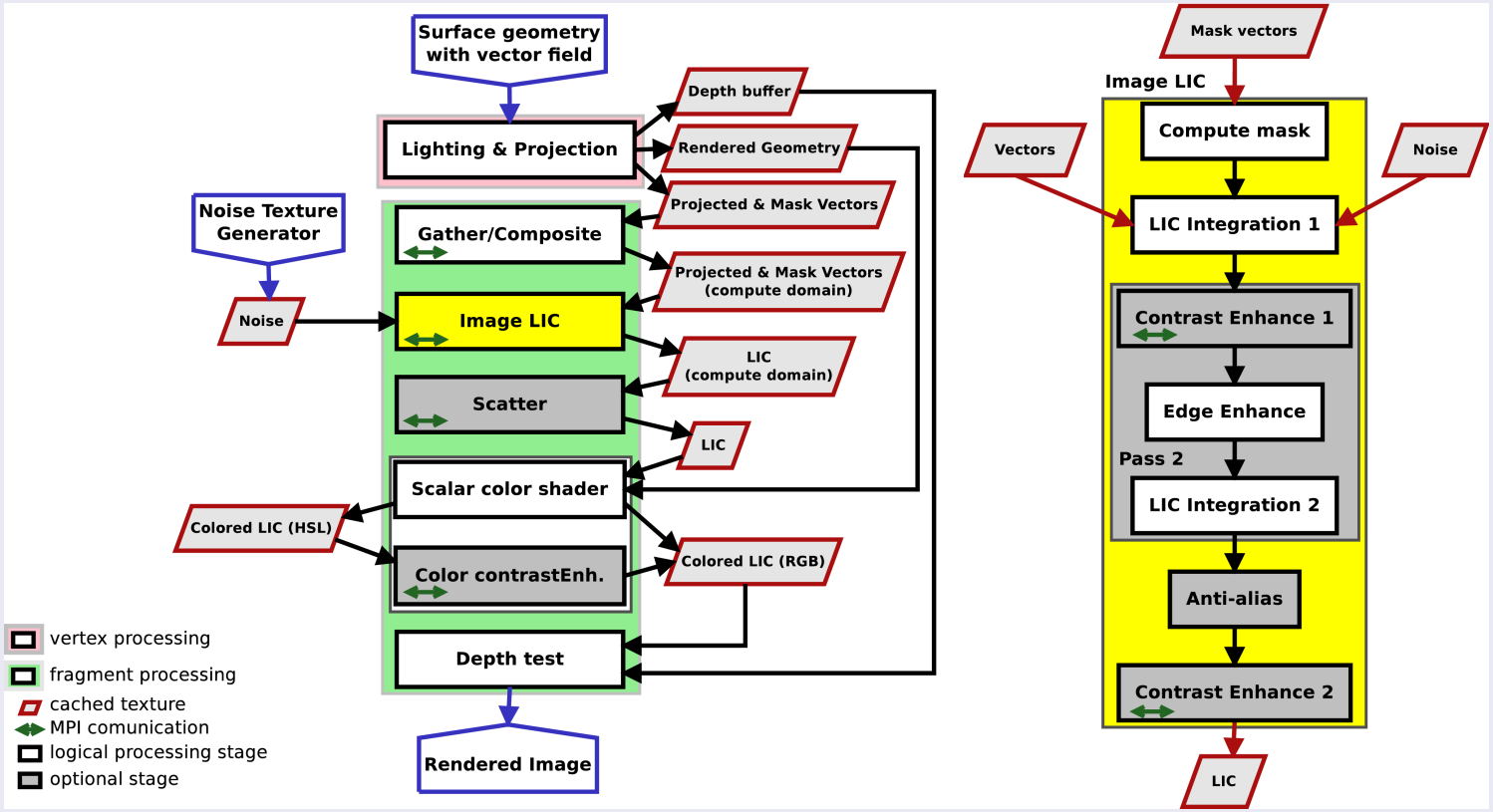
Color Space Histogram Stretching: Applied on lightness channel after HSL conversion



Interactivity

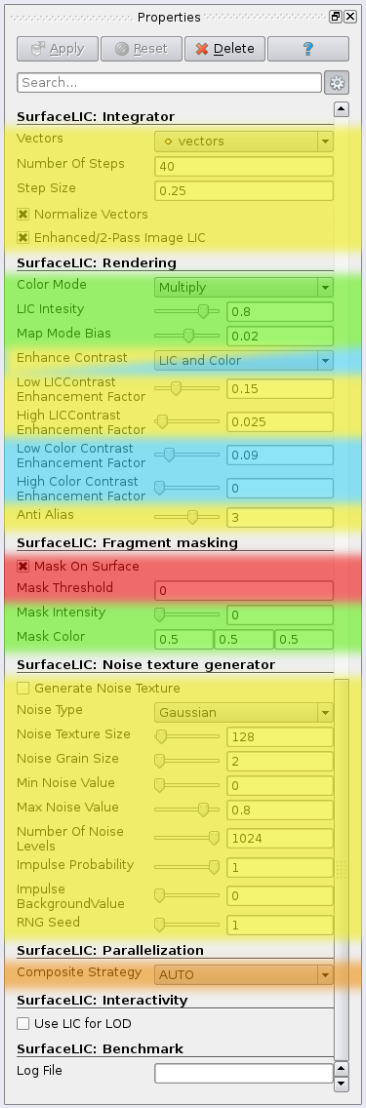
efficienciently respond to user tweaks

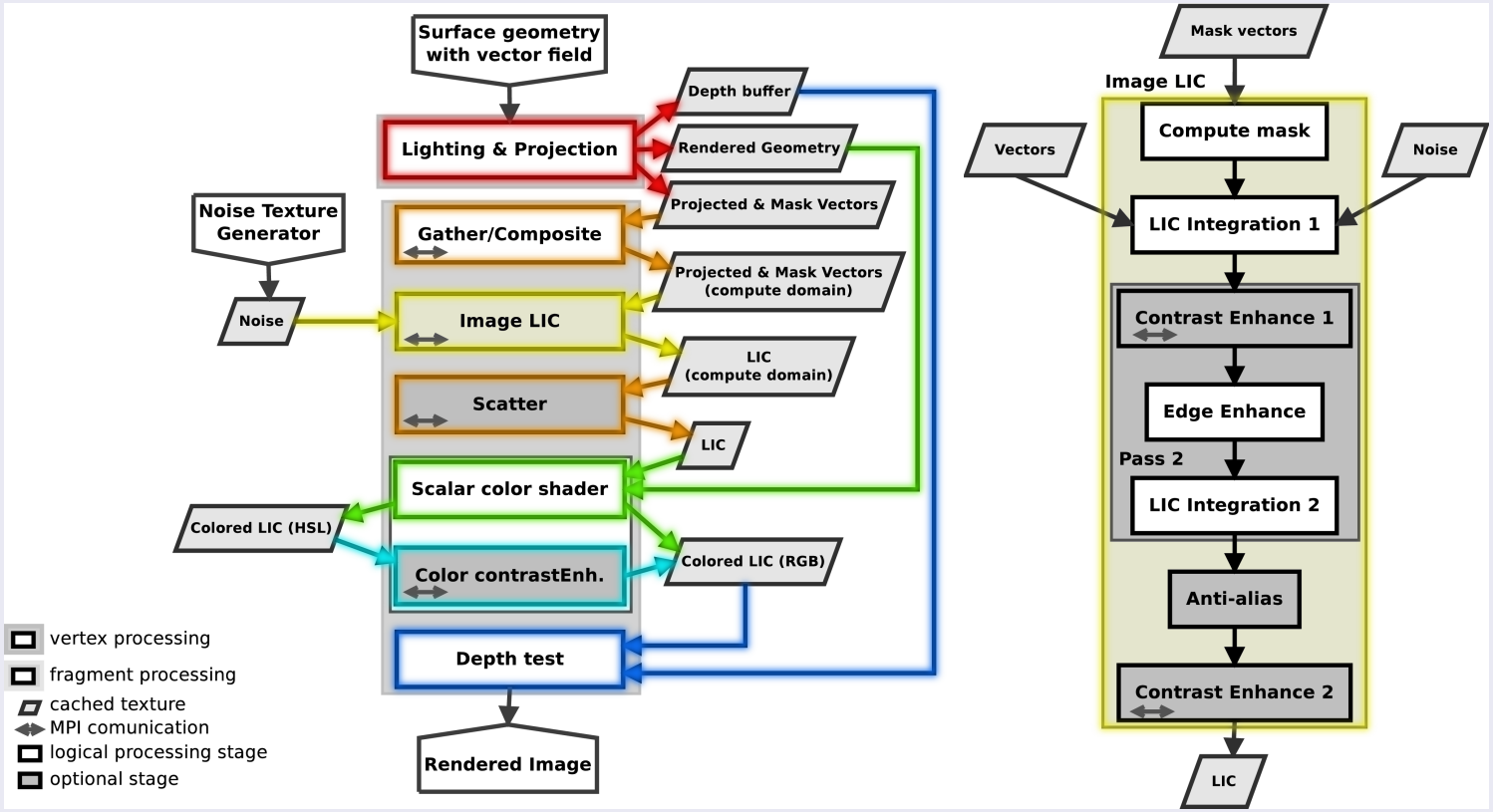




Interactivity

efficienciently respond to user tweaks

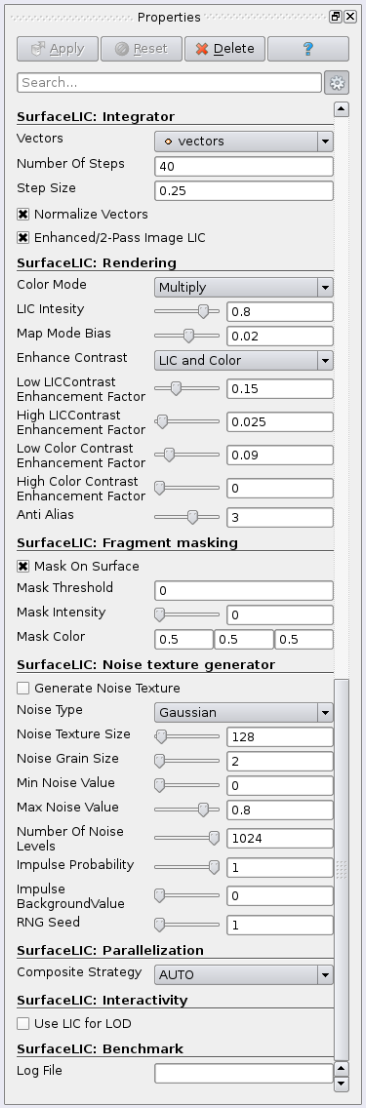


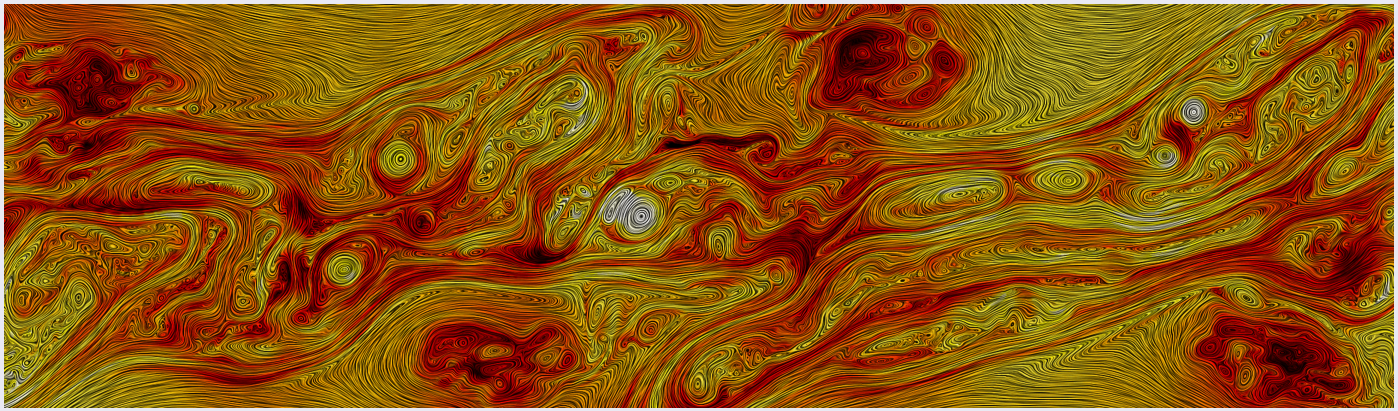


- cache each stage of the pipeline in a texture
- group parameters by stage, only run modified and downstream stages

Results

What did we accomplish?





1. Apply image processing techniques to produce high contrast high dynamic range pseduocolored results $\Leftrightarrow$ capture fine details and accurately represent the vector field
2. Expose many knobs $\Leftrightarrow$ Choices are left to runtime in order to effectively deal with data, view, and surface variablity
3. Optimzations for interaction $\Leftrightarrow$ make interaction practical
4. Parallelizaiton $\Leftrightarrow$ process very large datasets
5. Deployment in ParaView