Rendering fractal flames on the GPU

Matt Znoj Michael Semeniuk
Nicolas Mejia Steven Robertson

December 2, 2011

Flame overview

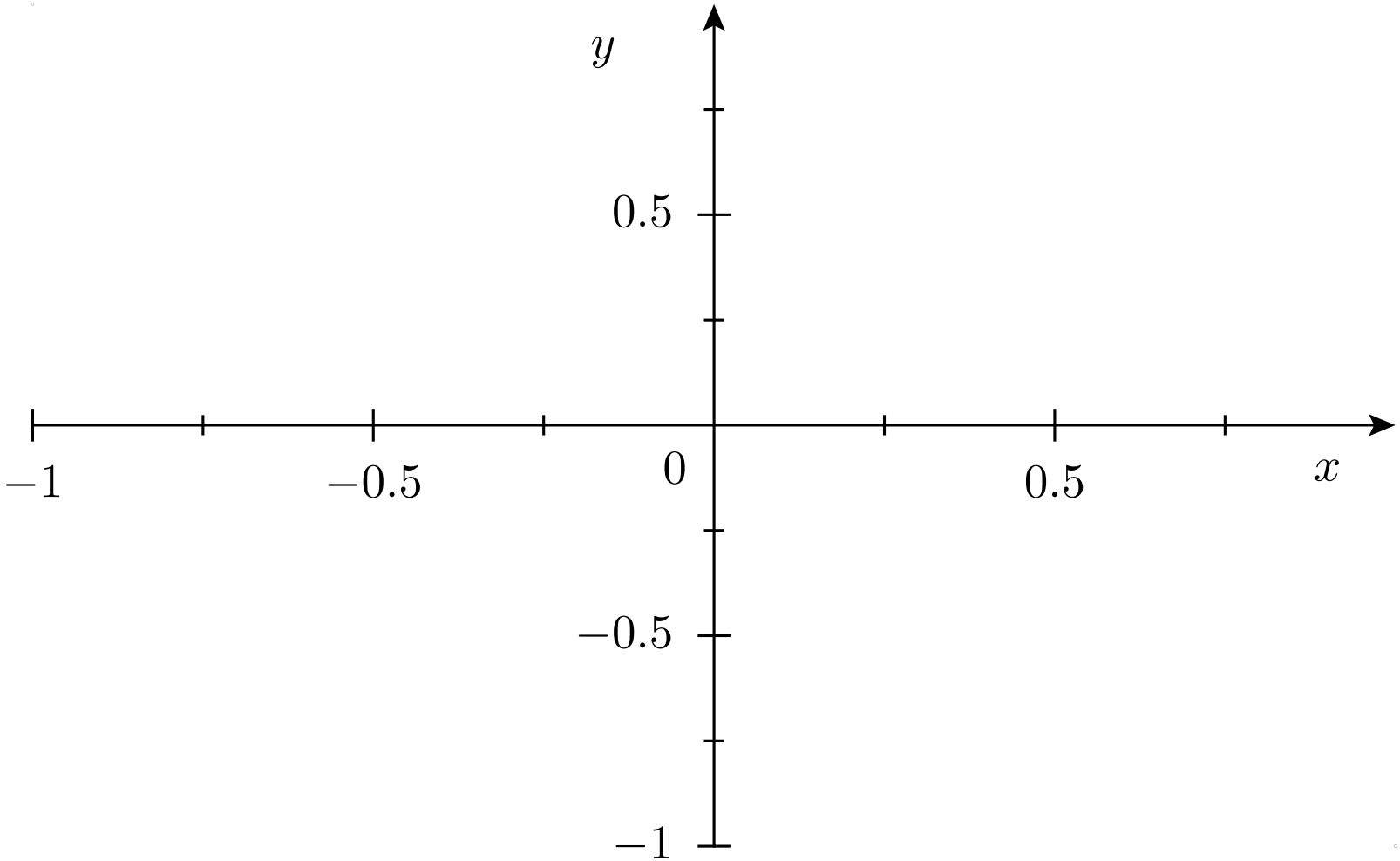
$$
\begin{aligned}
(x_{n+1}, y_{n+1}) &= (0.5x_n, && 0.5y_n) \\
(x_{n+1}, y_{n+1}) &= (0.5(x_n + 1), && 0.5y_n) \\
(x_{n+1}, y_{n+1}) &= (0.5x_n, && 0.5(y_n + 1))
\end{aligned}
$$

Flame overview

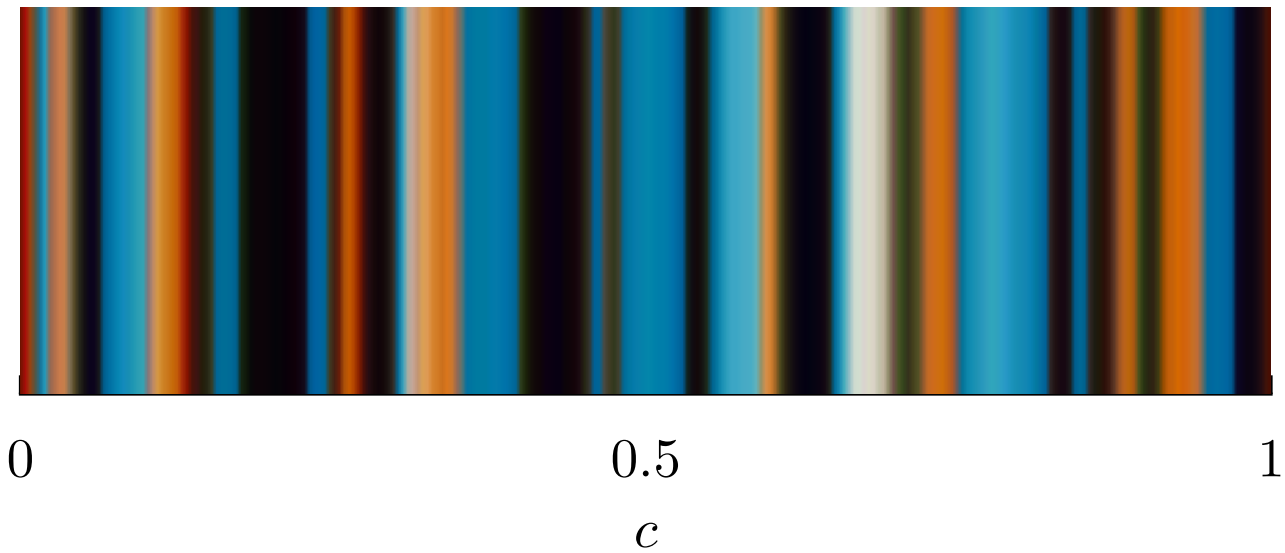
```
<flame name="electricsheep.244.04653" time="0" size="1024 1024"
center="-0.0304807 0.152945" scale="640" rotate="0"
supersample="4" filter="1" filter_shape="gaussian"
temporal_filter_type="box" temporal_filter_width="1"
quality="1000" passes="1" temporal_samples="1000" background="0 0
0" brightness="73.7913" gamma="4.28" vibrancy="1"
estimator_radius="14" estimator_minimum="0" estimator_curve="1"
gamma_threshold="0.01" palette_mode="linear"
interpolation_type="log" url="">
  <xform weight="0.122" color="0" symmetry="0" polar="0.003"
  coefs="-0.223176 1.64498 -1.64498 -0.223176 0.00173 -0.00881"/>
  <xform weight="1.829" color="1" symmetry="0" juliascope="1"
   juliascope_power="2" juliascope_dist="1"
   coefs="0.256909 0 0 0.256909 0 0" />
  <xform weight="0.458" color="0" symmetry="0" hyperbolic="2.051"
   coefs="-0.841582 -1.07778 1.07778 -0.841582 0 0" />
   <finalxform color="0" symmetry="1" perspective="1"
   perspective_angle="0.530779" perspective_dist="1.46989"
   coefs="2.01414 0 0 2.01414 0 0" />
   <!-- palette omitted -->
</flame>
```

Flame overview

Flame overview



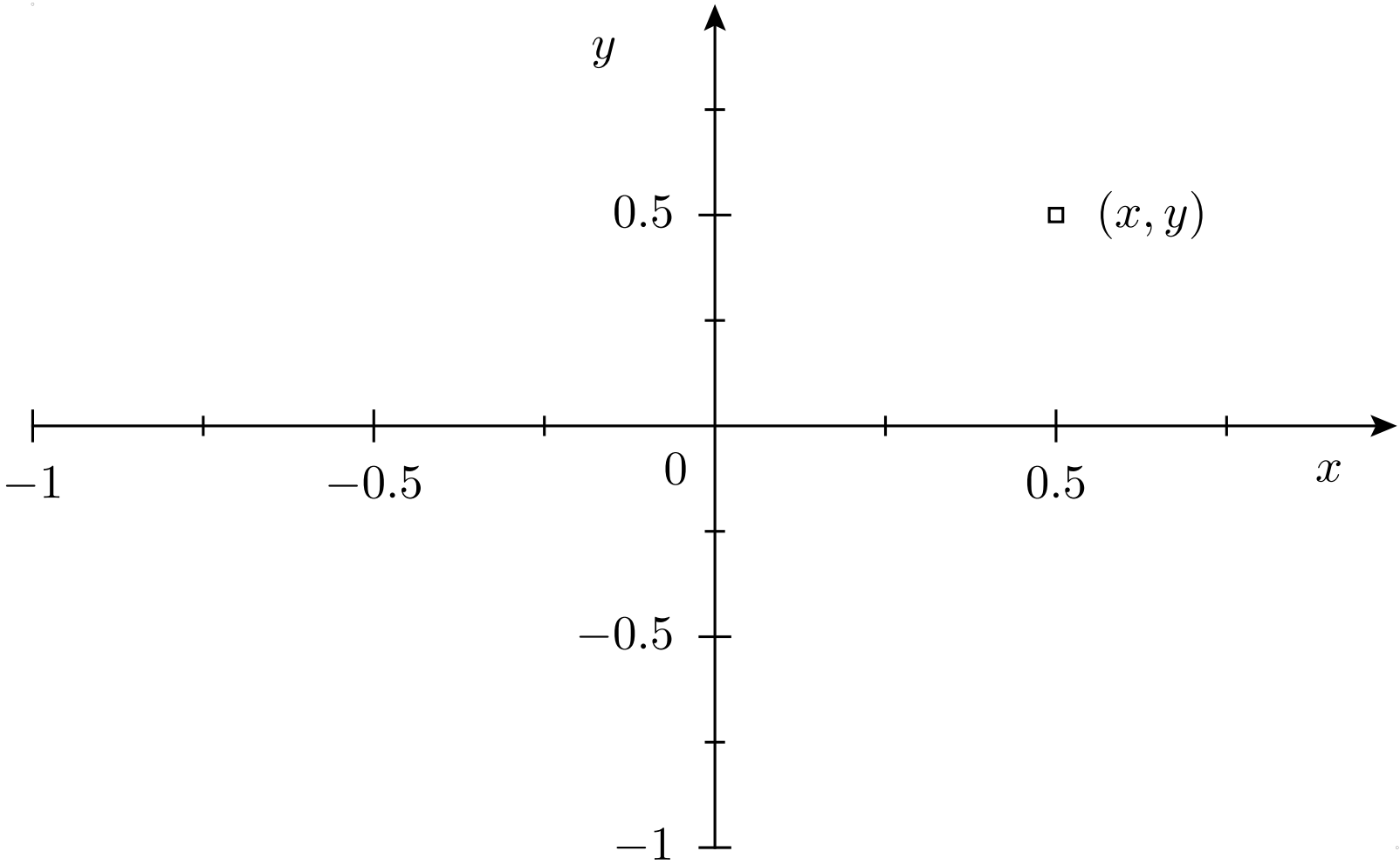
Flame overview



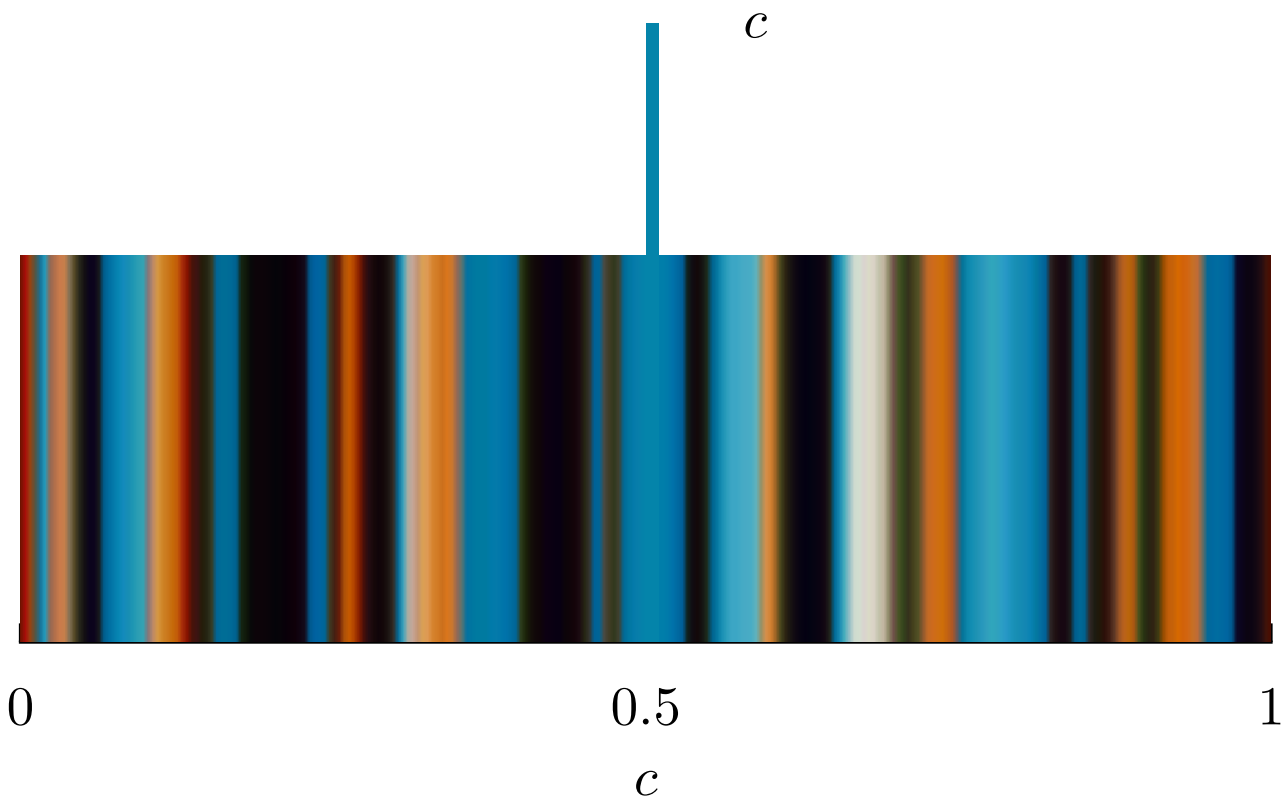
Flame overview

$$(x, y, c)$$

Flame overview



Flame overview



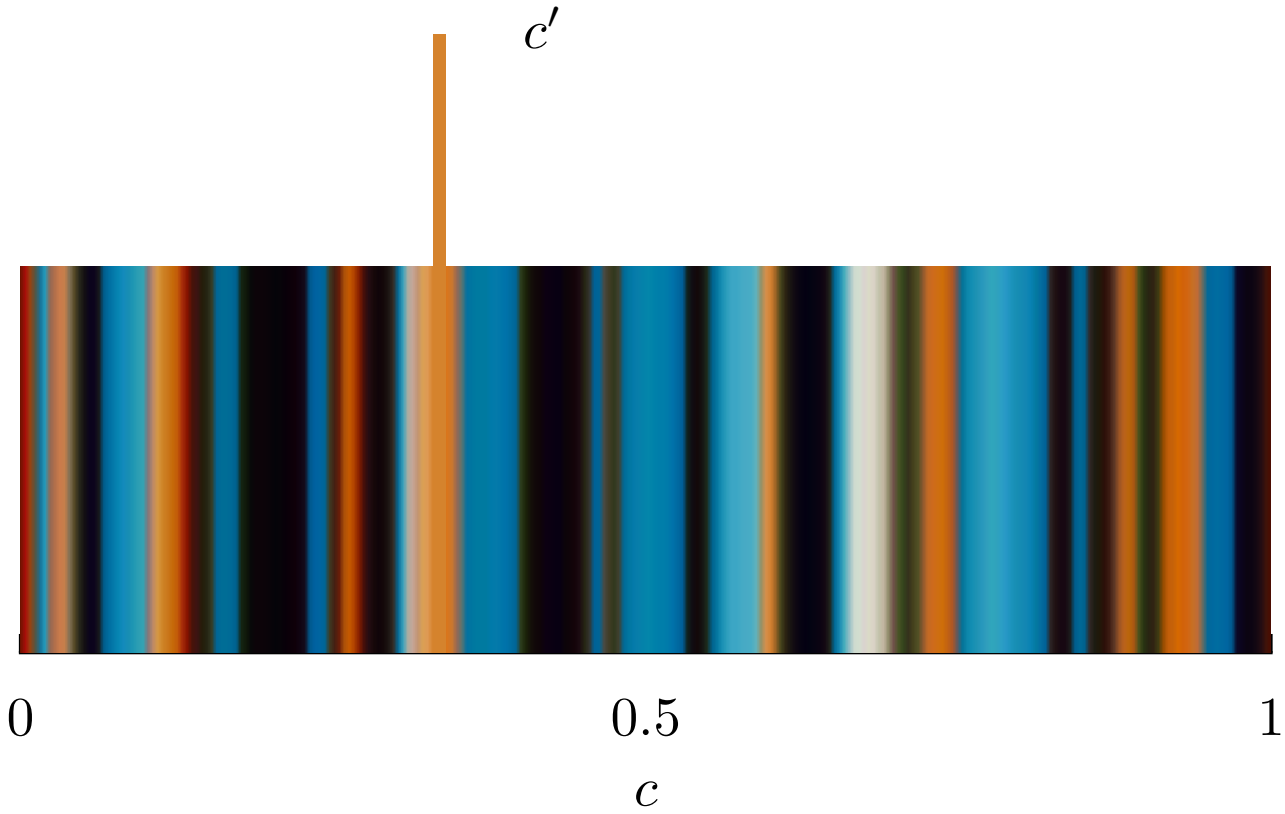
Flame overview

$$f_1(x, y, c)$$
$$f_2(x, y, c)$$
$$f_3(x, y, c)$$

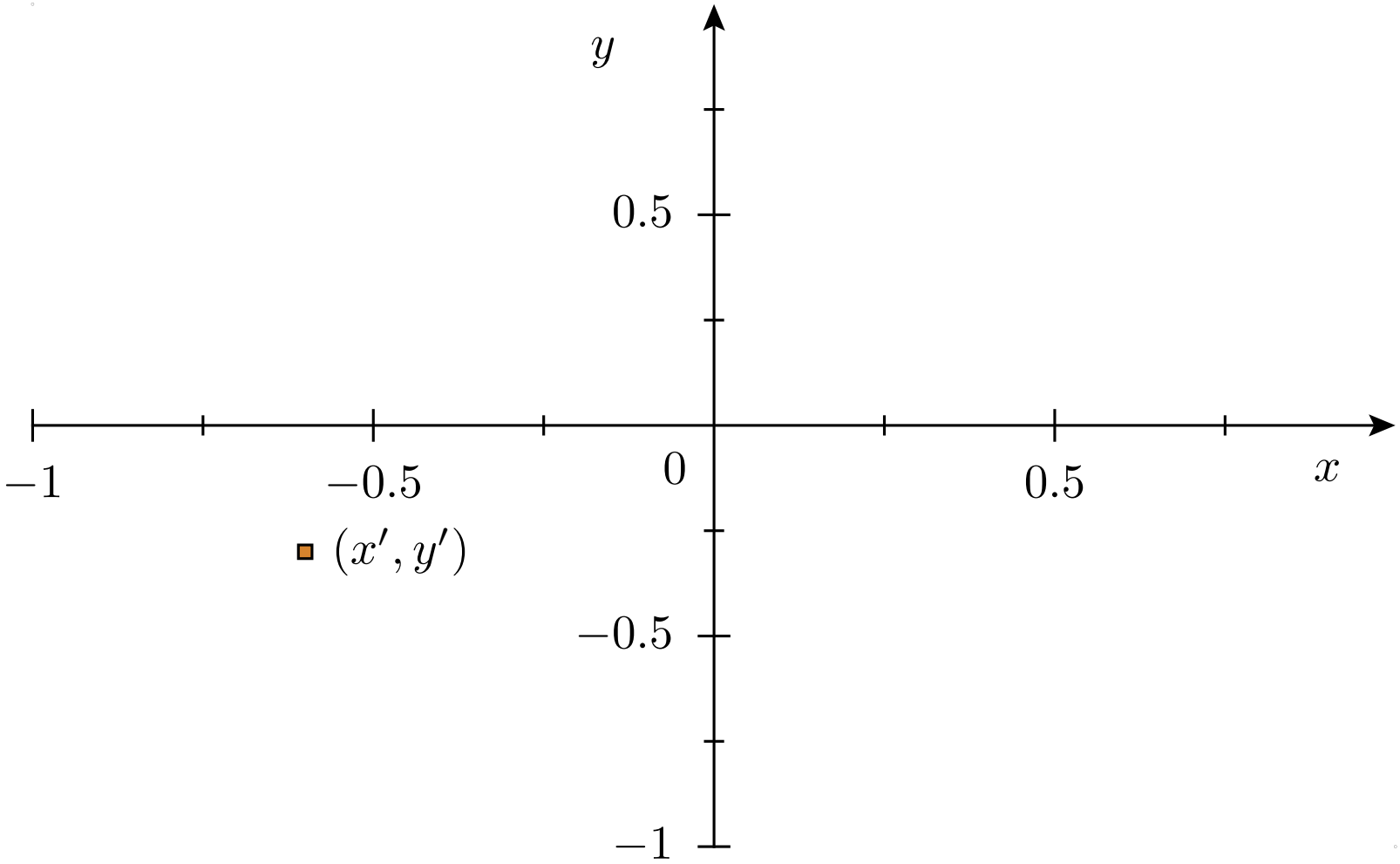
Flame overview

$$(x', y', c') = f_1(x, y, c)$$

Flame overview



Flame overview



Flame overview

40,000,000,000 iterations

Project goals

Project goals

14.2 hours wall time

- Intel Core i5 750 @ 4×2.67GHz
- 85 wall-seconds / frame
- 900 frames
- flam3 3.0, GCC 4.6

Project goals

Art requires exploration.

Project goals

Research requires large samples.

Project goals

Entertainment requires variety.

Project goals

Fully compatible.

Project goals

High quality.

Project goals

Fast.

Project goals

100 floating point operations.

Project goals

750,000,000,000 FLOPS

Project goals

1,843,200,000 samples / frame

Project goals

4.07 frames / second

Project goals

"You'd be lucky to get a twentieth of that."

Project goals

Speed target: 0.2 FPS.

Code generation

Code generation

Affine transforms

$$x' = ax + by + c$$
$$y' = dx + ey + f$$

Code generation

Variation functions

Code generation

Linear variation

$$x'' = w \cdot x'$$
$$y'' = w \cdot y'$$

Code generation

Flux variation

$$x'' = w \cdot (2 + \mathit{flux\_spread} \cdot \sqrt{\frac{\sqrt{y'^2 + (x' + w)^2}}{\sqrt{y'^2 + (x' - w)^2}}} + \cos(\arctan \frac{y'}{x' - w} - \arctan \frac{y'}{x' + w})$$

$$y'' = w \cdot (2 + \mathit{flux\_spread} \cdot \sqrt{\frac{\sqrt{y'^2 + (x' + w)^2}}{\sqrt{y'^2 + (x' - w)^2}}} + \sin(\arctan \frac{y'}{x' - w} - \arctan \frac{y'}{x' + w})$$

Code generation

Code generation

Conditional cascade

Code generation

```
if (genome.xf[1].weights[VAR_LINEAR]) {
    float w = genome.xf[1].weights[VAR_LINEAR];
    fx += w * tx;
    fy += w * ty;
}
if (genome.xf[1].weights[VAR_SINE]) {
    ...
}
...
```

Code generation

Code generation

Transistor leakage limits clockspeed

Code generation

Applications limit parallelization

Code generation

CPUs get smarter

Code generation

OpenGL shaders are parallel

Code generation

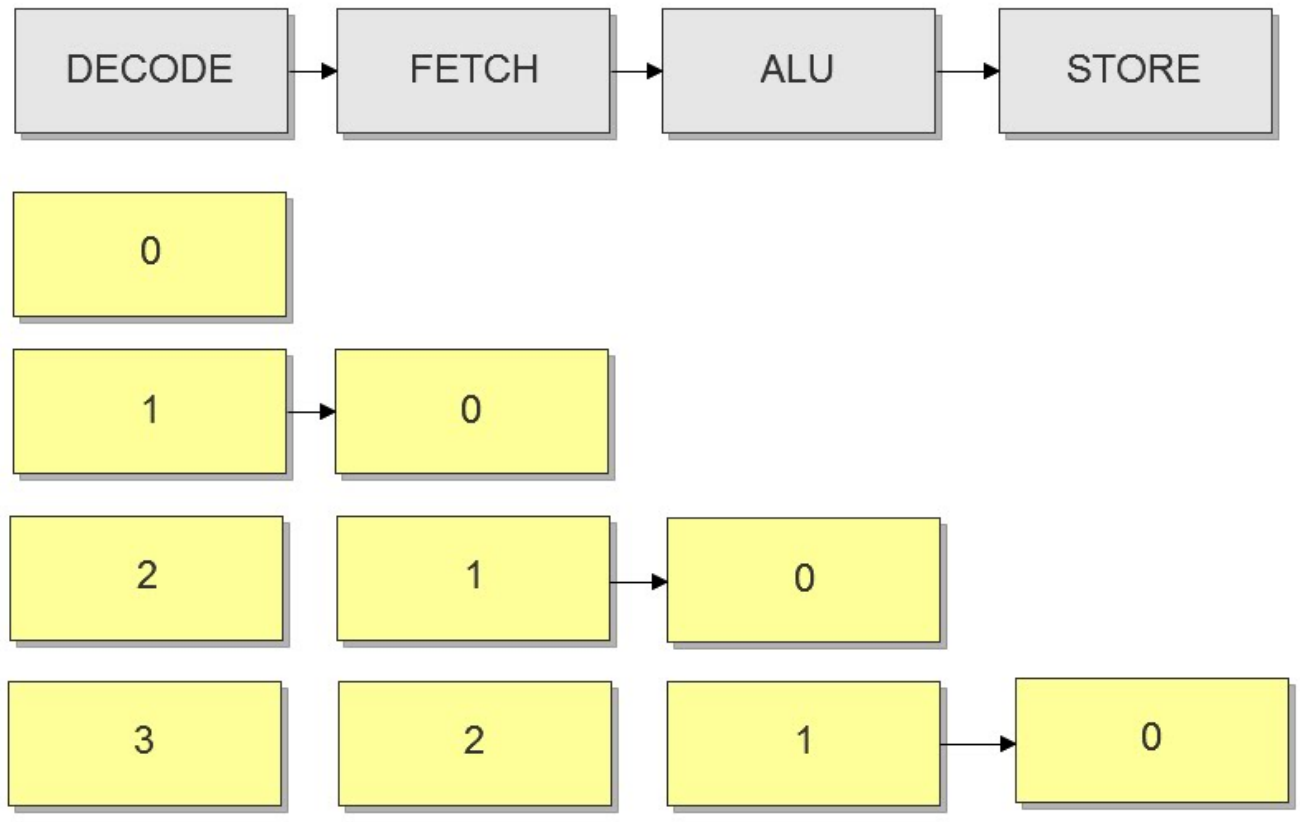
GPUs: an army of simpletons

Code generation

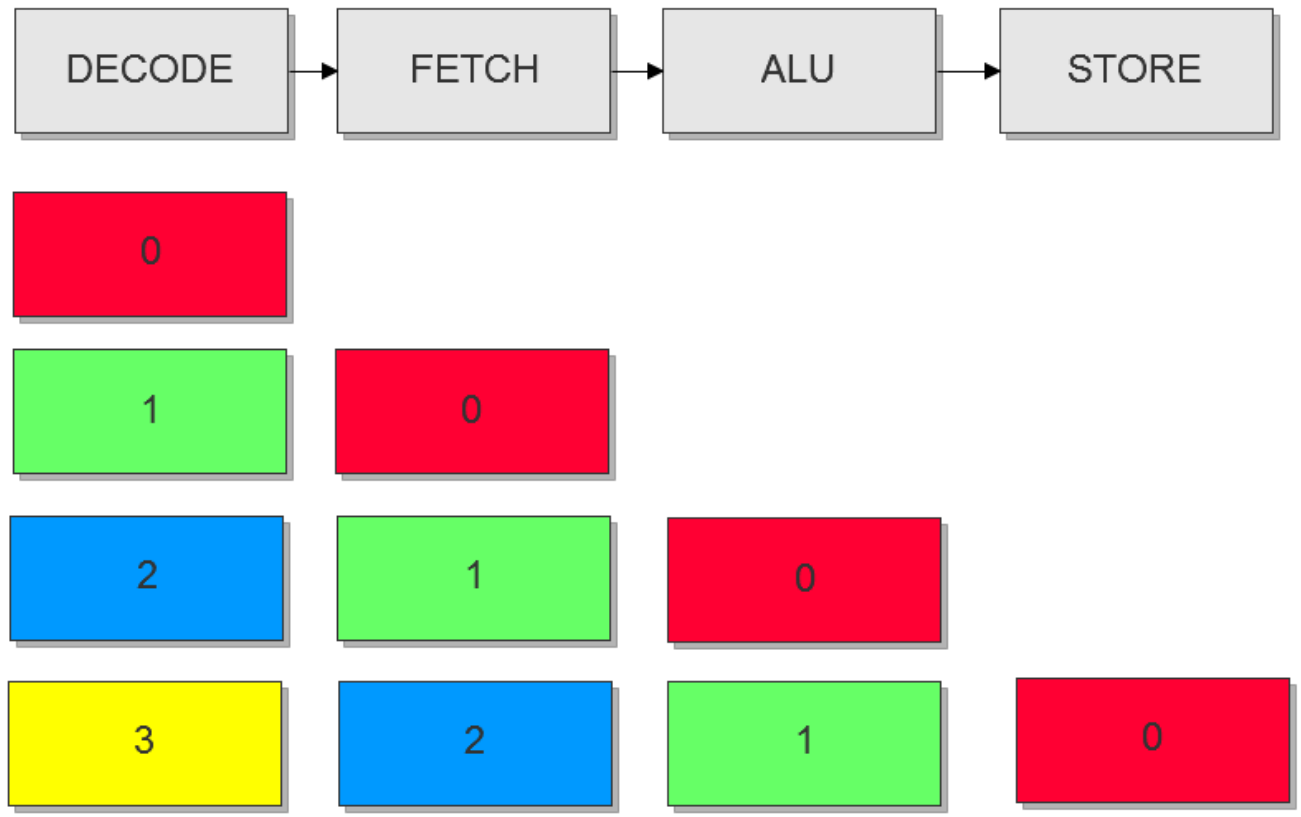
Simple hardware, complex optimizations


CUDA Core
Dispatch Port
Dispatch Port
Operand Collector
FP Unit
INT Unit
Result Queue
SM
Instruction Cache
Warp Scheduler
Warp Scheduler
Dispatch Unit
Dispatch Unit
Dispatch Unit
Dispatch Unit
Register File (32,768 x 32-bit)
Core
LD/ST
SFU
Interconnect Network
64 KB Shared Memory / L1 Cache
Uniform Cache
Tex
Texture Cache
PolyMorph Engine
Vertex Fetch
Tessellator
Viewport Transform
Attribute Setup
Stream Output

Code generation



Code generation



Code generation

Pipeline depth: 22 cycles

Code generation

Memory latency: 600 cycles or more

Code generation

Many threads, many registers

Code generation

128KB register file per shader core

Code generation

1024 threads per core

Code generation

32 registers per thread

Code generation

NVCC is... not perfect

Code generation

What's the problem?

Code generation

Conditional cascade in inner loop

Code generation

Basic blocks

Code generation

Instruction reordering

Code generation

Common sub-expression elimination


CUDA Core
Dispatch Port
Dispatch Port
Operand Collector
FP Unit
INT Unit
Result Queue
SM
Instruction Cache
Warp Scheduler
Warp Scheduler
Dispatch Unit
Dispatch Unit
Dispatch Unit
Dispatch Unit
Register File (32,768 x 32-bit)
Core
LD/ST
SFU
Interconnect Network
64 KB Shared Memory / L1 Cache
Uniform Cache
Tex
Texture Cache
PolyMorph Engine
Vertex Fetch
Tessellator
Viewport Transform
Attribute Setup
Stream Output

Code generation

Code generation

Don’t use the cascade.

Code generation

We know what paths will be taken.

Code generation

Run-time code generation

Code generation

Semantically-aware templates

Animating flames

Animating flames

Animating flames

Temporal multisampling.

Animating flames

Global parameter access is slow.

Animating flames

Shared parameter access is fast(er).

Animating flames

Shared memory is small.

Animating flames

Dynamic lookups are expensive.

Animating flames

Data structure generation

- Insert parameter references into code
- Define reference targets for compiler
- Build parameter interpolation function
- Derive device order of genome splines

Animating flames

Animating flames

Compile time on host: 5.4s

Run-time on device: 224ns (amortized)

Animating flames

batch renderer pipeline graphic?

Animating flames

```
    ld.global.u32   rv, [rt+0x100];
    setp.ge.u32      p,  rv, rn;
@p  add.u32          rt, rt, 0x100;
```

Accumulation

Accumulation

- Calculate accumulation buffer index.
- Calculate color tristimulus values.
- Add values to buffer at index.

Accumulation

Histograms on GPU are ineffecient.

Accumulation

Large transaction sizes

Accumulation

Incoherent caches

Accumulation

Small caches

Accumulation

Shared memory

Accumulation

Point log

Accumulation

Coalesced writes

Accumulation

Log replay

Accumulation

Partial address sort

Accumulation

Sorting implementations

Accumulation

MGPU sort; B40C sort

Accumulation

Cuburn sort

Accumulation

Atomics are ‘against the rules’.

Accumulation

	Cuburn	MGPU	B40C	CUDPP
7 bits	821	740	551	221
8 bits	943	813	611	251
9 bits	966	877	475	191
10 bits	862	910	528	211

Units are millions of keys per second, normalized to 32 bits.
Times taken on a GTX 560 Ti 900MHz.

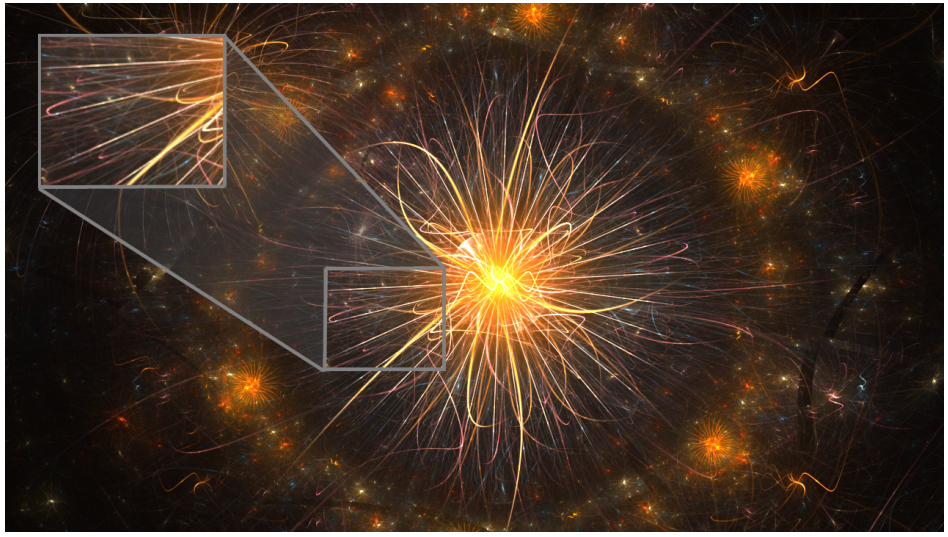
Filtering

Multisample antialiasing

Filtering

Jittered-grid antialiasing

Filtering

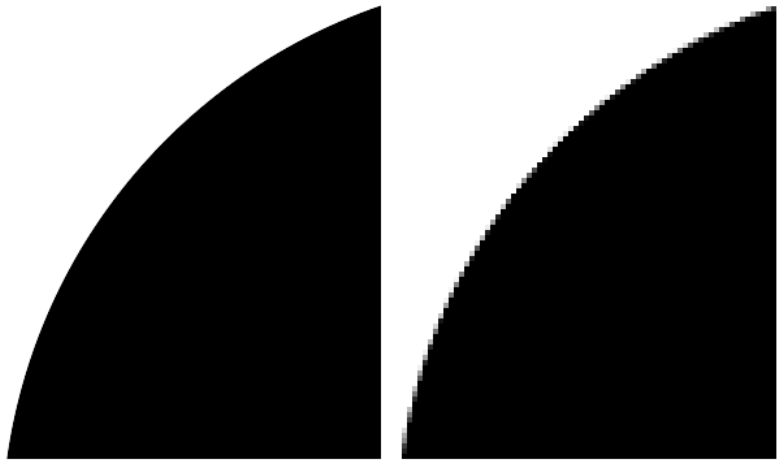


Density estimation

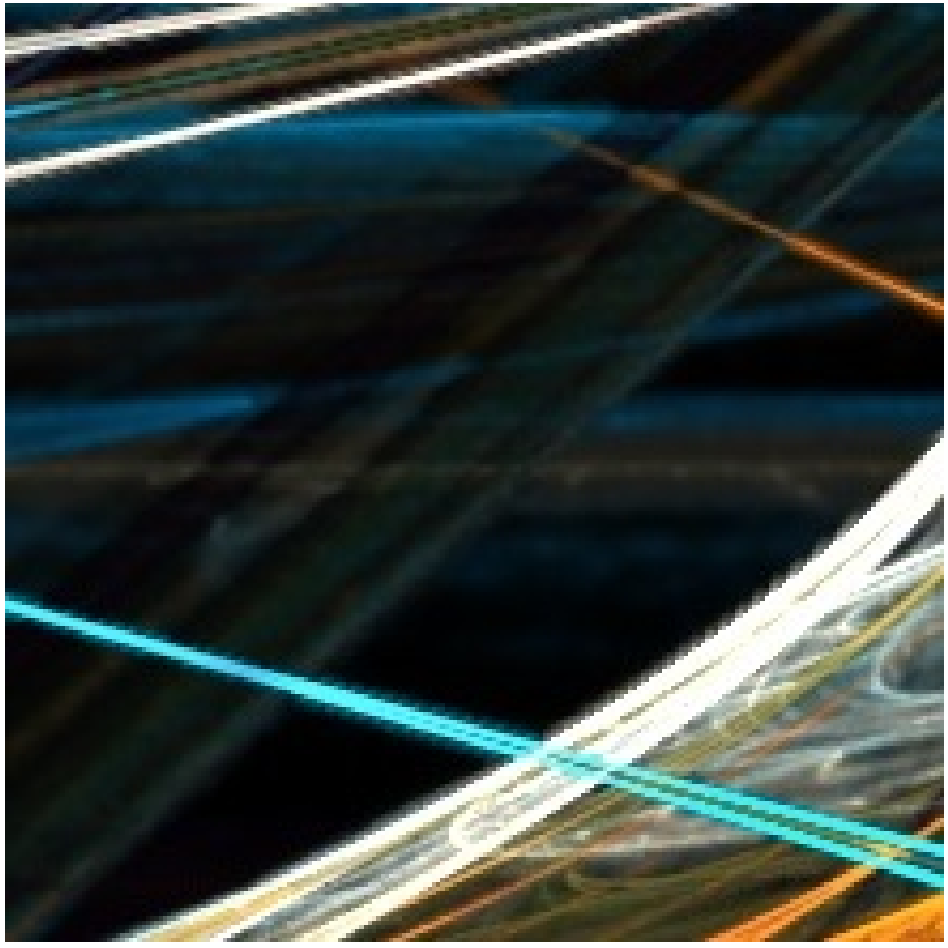
Filtering

Filtering

Filtering



Filtering



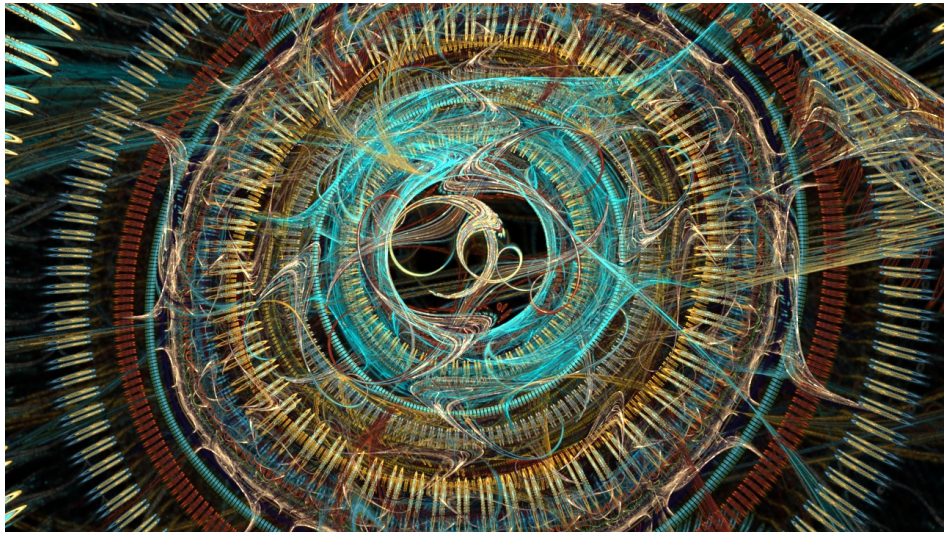
Filtering

Edge-detection convolution kernel

Filtering

$$\boldsymbol{A} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \boldsymbol{B} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

Filtering



Host-side software

Host-side software

Project status and next steps

Matt	Filtering, image enhancement, AA
Mike	Colorspace, tonemapping, writeback
Nick	RNG, `malloc()` emulation
Steve	Prototype, language tools

Project status and next steps

Is software really ever *done*?

Project status and next steps