WPI

A GPU Accelerated Virtual Scanning Confocal Microscope

James Kingsley[†], Zhilu Chen[+], Jeffrey Bibeau[‡], Luis Vidali[‡], Xinming Huang[+], Erkan Tüzel[†]

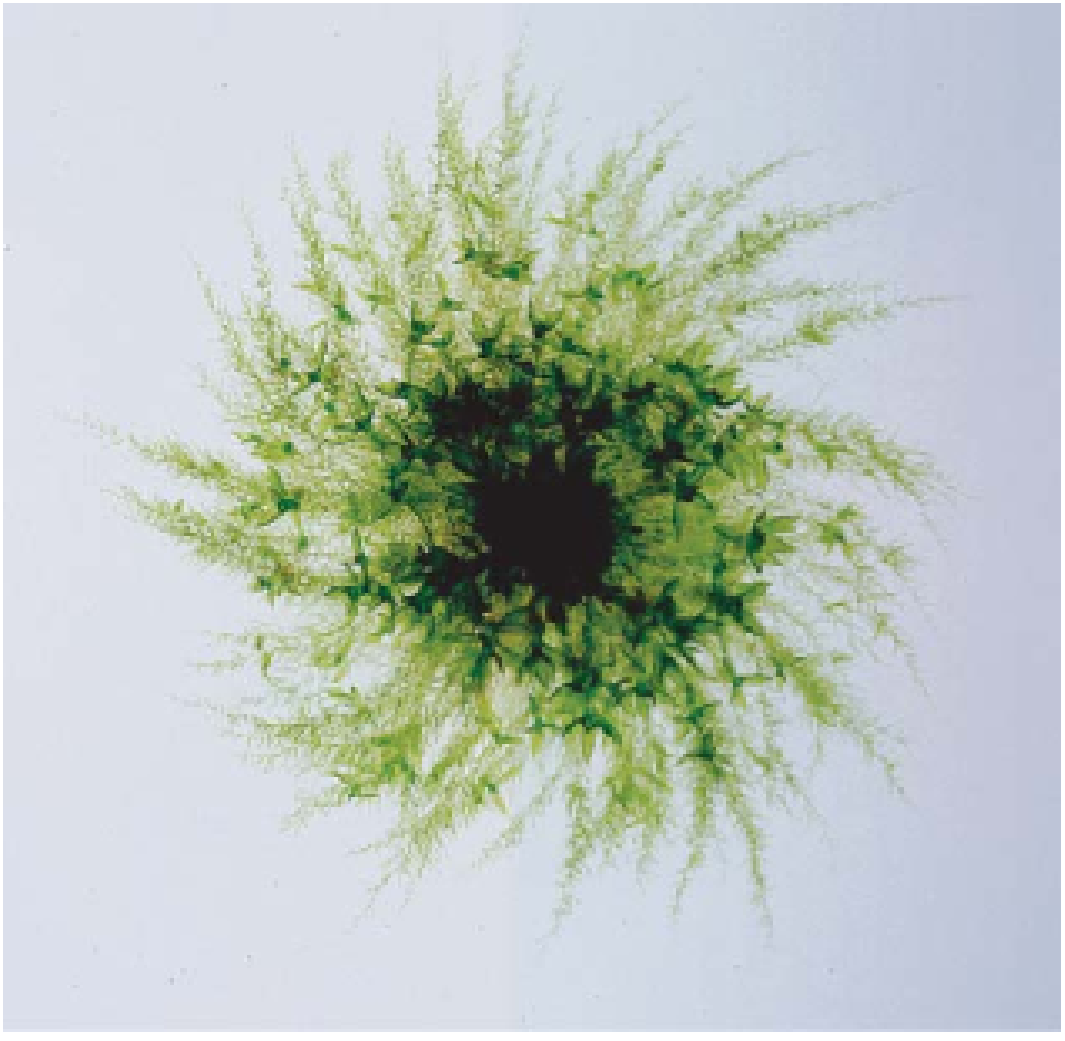
[†]Department of Physics, Worcester Polytechnic Institute

[+]Department of Electrical and Computer Engineering, Worcester Polytechnic Institute

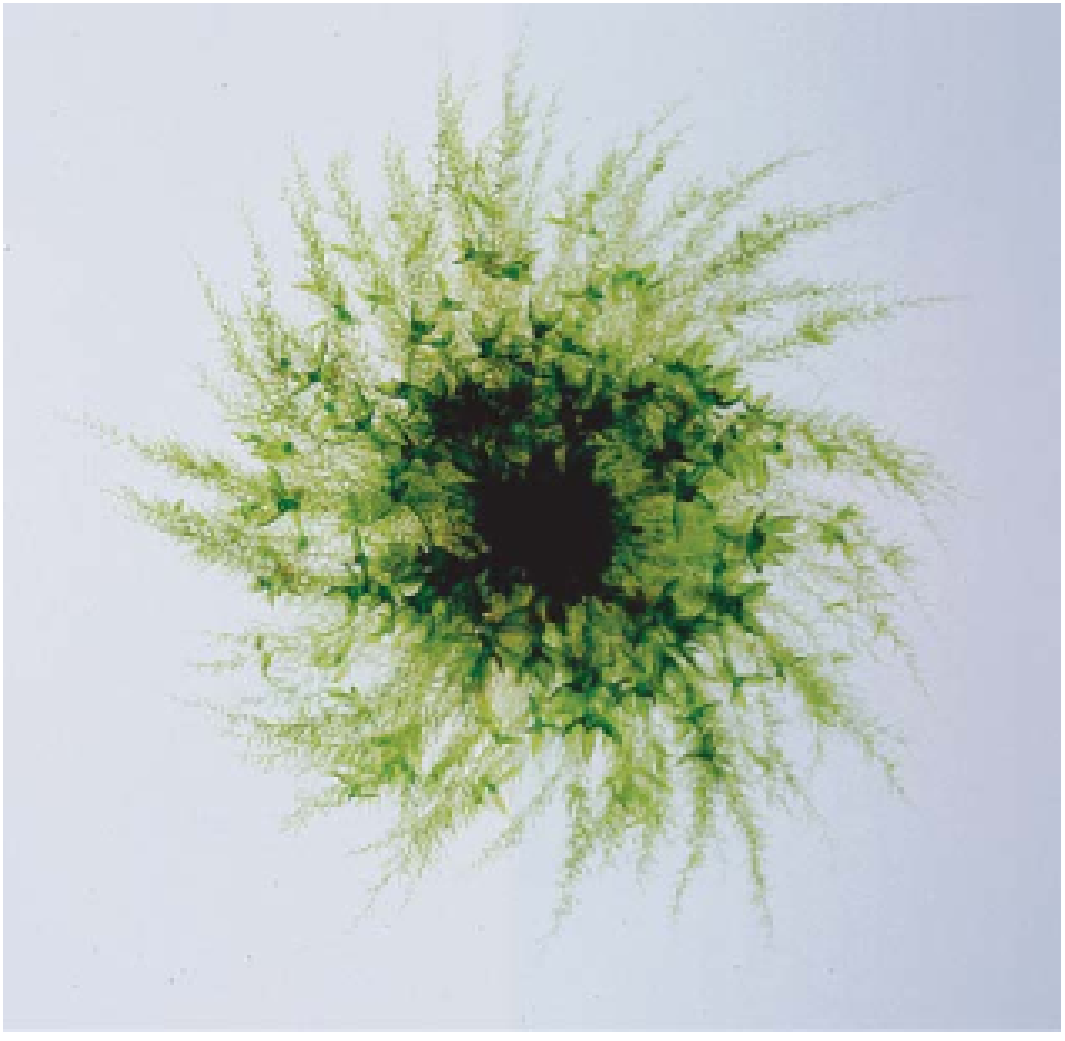
‡ Department of Biology and Biotechnology, Worcester Polytechnic Institute

September 11, 2014

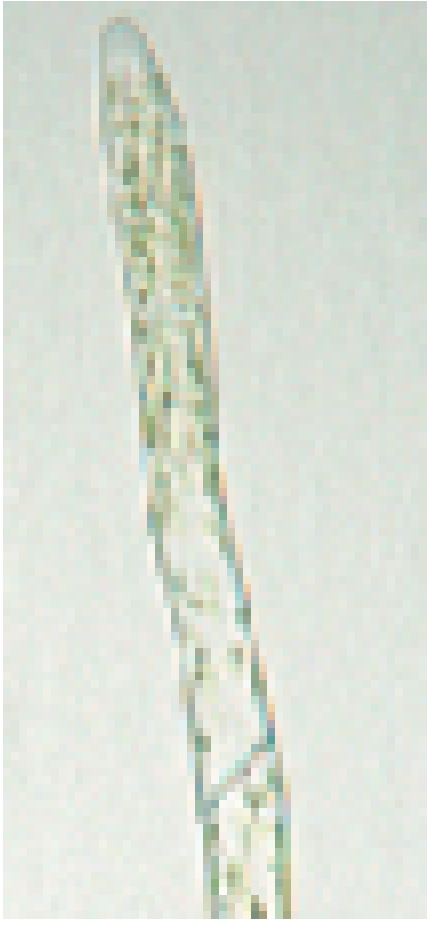
Moss: *Physcomitrella Patens*



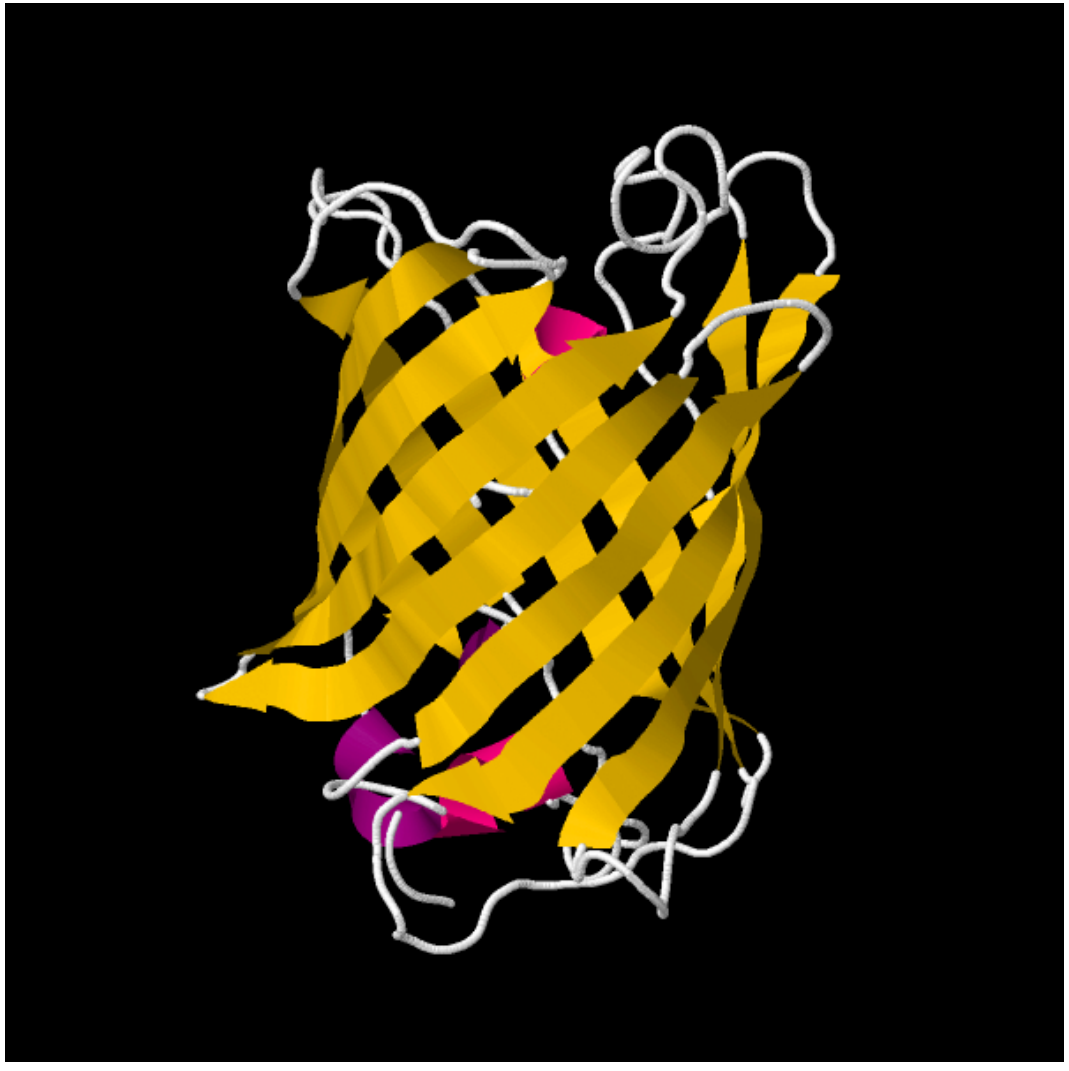
Moss: *Physcomitrella Patens*



Tip Growth

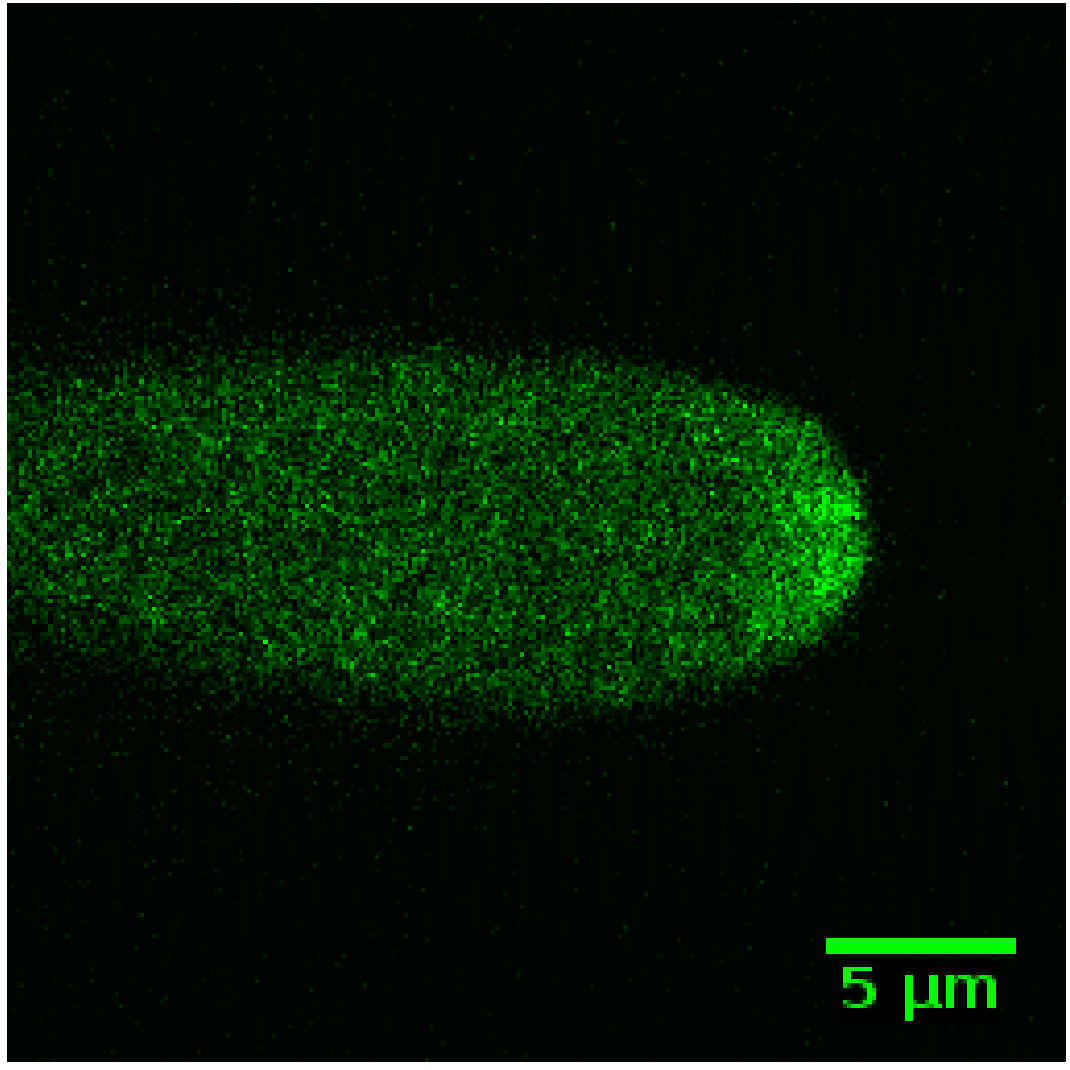


Green Fluorescent Protein



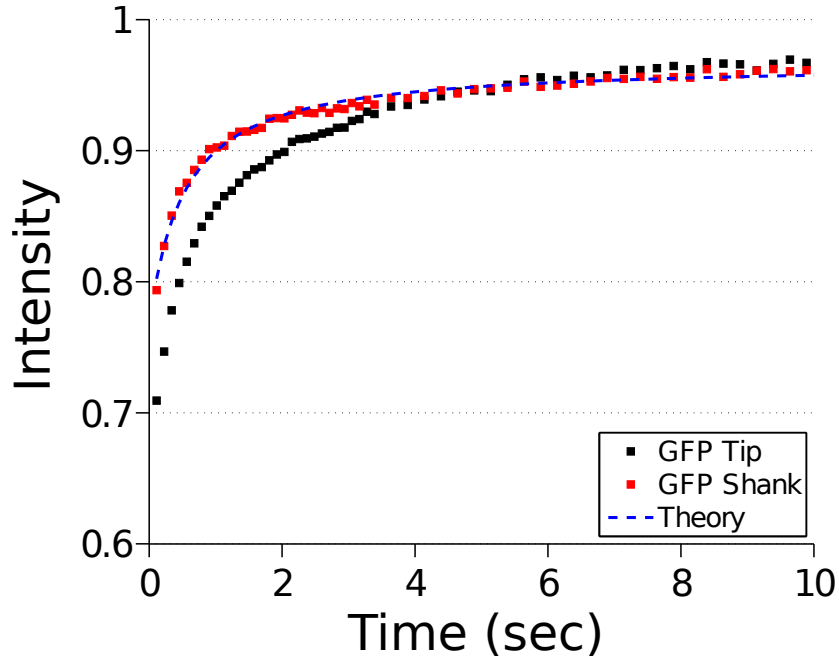
Green Fluorescent Protein:

In Moss cells



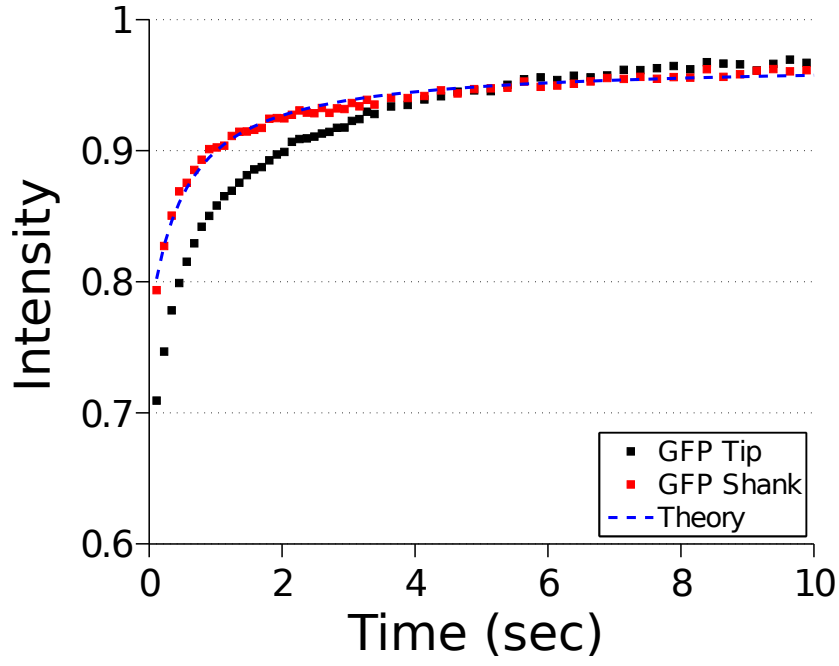
Flourescence Recovery after Photobleaching (FRAP)

- Overload and burn out an area
- Watch how quickly it recovers
- Fit that to a theoretical expression



Flourescence Recovery after Photobleaching (FRAP)

- Overload and burn out an area
- Watch how quickly it recovers
- Fit that to a theoretical expression
- Analytical solutions only exist for simple geometries



Modeling FRAP

Loop:

1. find next photobleaching or imaging event
2. run a brownian motion step out to that time
3. run the event in question
4. repeat 1

- Brownian motion model of diffusion
- Gaussian beam model of photobleaching
- Gaussian beam model of confocal imaging

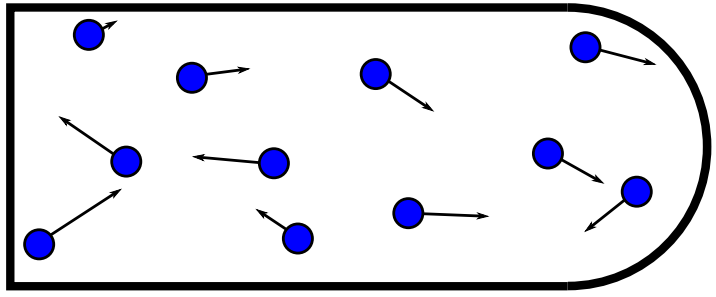
$$I(x,y,z) = I_0 w_0^2 \left(1 + \frac{z^2}{z_R^2}\right) \exp\left(-\frac{2(x^2+y^2)}{w_0^2\left(1+\frac{z^2}{z_R^2}\right)}\right)$$

- Parameters I_0, w_0, z_R.

Brownian Motion

- Movement for each particle

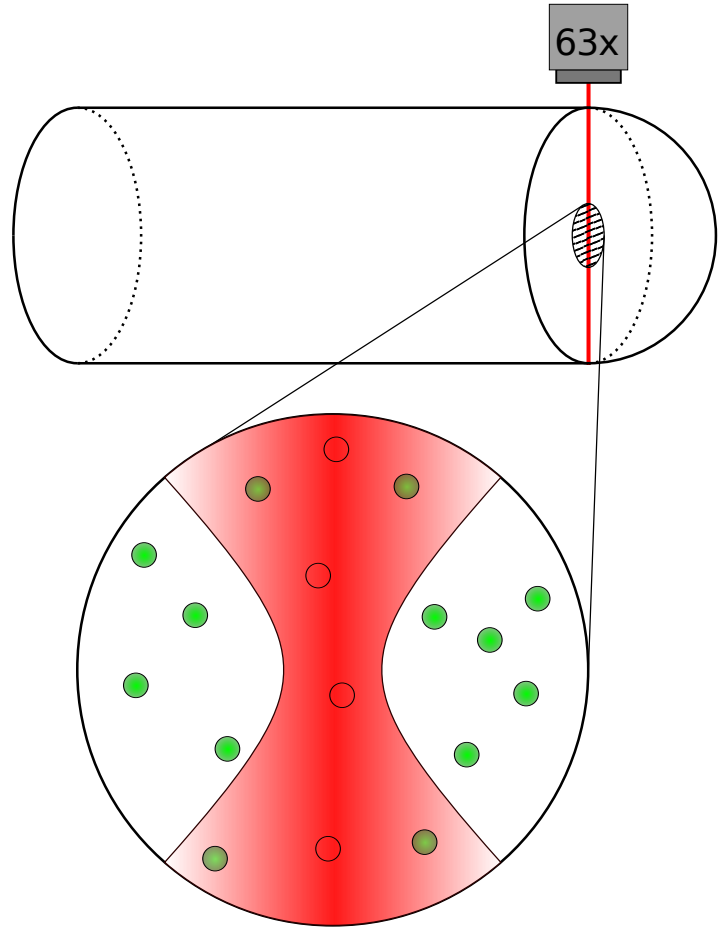
```
k_brownian=sqrt(2*diffusion_coefficient/timestep);
x+=k_brownian * randn();
y+=k_brownian * randn();
z+=k_brownian * randn();
```



- `diffusion_coefficient` fixed parameter
- `timestep` variable
- Test and handling of collisions with walls

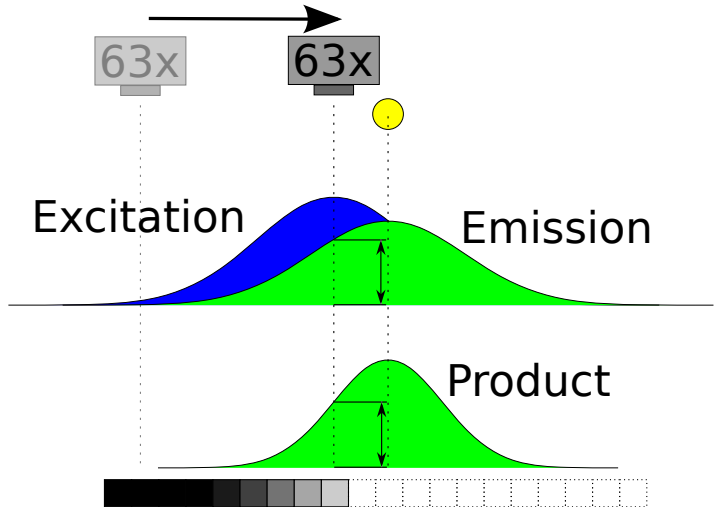
Photobleaching

- For each particle, bleach if `randu()` is less than $P_0 I(x - x0, y - y0, z - z0)$



Imaging

- In reality, scan each pixel one at a time
- In simulation, treat each line as an instantaneous action
- For each particle, start at closest pixel in the pixel line
- Increment away until our contribution ($I(x - x0, y - y0, z - z0)^2$) is less than the image quantum depth



This is embarrassingly parallel

Version	Time(s)	speedup	(cumulative)	Efficiency*	
CPU†	10728.5	-	-	2.33	9.32

* The two efficiencies are price and energy efficiency in units of particle*runs/dollar*second and particle*runs/Joule, respectively

† CPU used for comparison is a 2.4 GHz Intel Xeon E 5620, priced at $320 on Newegg and consuming 80W. Price and power consumption is divided by 8, as the CPU can run 8 parallel independent jobs at once

- A full simulation takes nearly three hours
- Particles that never interact with each other

This is embarrassingly parallel

Version	Time(s)	speedup	(cumulative)	Efficiency*	
CPU†	10728.5	-	-	2.33	9.32

* The two efficiencies are price and energy efficiency in units of particle*runs/dollar*second and particle*runs/Joule, respectively

† CPU used for comparison is a 2.4 GHz Intel Xeon E 5620, priced at $320 on Newegg and consuming 80W. Price and power consumption is divided by 8, as the CPU can run 8 parallel independent jobs at once

- A full simulation takes nearly three hours
- Particles that never interact with each other
- This is an excellent GPGPU task

CUDA

- Nvidia's Compute Unified Device Architecture
- Single instruction multiple thread (SIMT)
- Excellent for data-parallel tasks
- Requires proprietary Nvidia hardware and drivers

CUDA's compute architecture

- CPU invokes kernel functions which are queued and executed on the GPU
- kernel function execution is broken into a set of blocks
- blocks are independent; broken into warps
- each warp of threads executes the same instruction synchronously
- branches are handled by freezing threads that do not take the conditional path
- execution is asynchronous

CUDA's memory architecture

- Global memory
  - slow
  - cached
  - accessible from all threads
  - large (GB)
- Shared memory
  - shared by all threads in a block
  - fast
  - transient (does not persist longer than the block)
  - limited in size (tens of KB)
- Texture memory
- Constant memory

GPGPU version

- Naive conversion
  - Put particles in Global memory
  - Conver every “for each particle” loop into a kernel function
- All particle manipulation is of that form, so particles do not have to cross the PCIe bus
- Imaging requires atomic incrementation and a memory copy back to the CPU to output to disk

Not significantly better?

Version	Time(s)	speedup	(cumulative)	Efficiency	
CPU	10728.5	-	-	2.33	9.32
GPU† (naive)	1538.8	1.0	1.0	0.90	2.32

† GPU used for comparison is a Nvidia GTX 780 Ti, priced at $720 on Newegg and consuming 280W

Faster than CPU, but does not just justify the added price and power consumption.

Can we make it enough faster to be worth while?

Imaging is the culprit

- Out of 1,000,000 particles, an average of around 2000 participate in a given line.
- Warp divergence hurts
- One active thread in a warp is terrible.
- Prune particles to only test relevant ones?

Select particles that will be rendered?

Pro

- Minimal warp divergence

Con

- still have some divergence, as different particles may affect different numbers of pixels
- requires testing every particle anyway
- requires a perfect (generous or expensive) heuristic for determining inclusion

Sort to improve coherence?

Pro

- Less warp divergence
- Does not require expensive math operations in the sort

Con

- Requires a sorting operation: $O(n \log(n))$ doesn't scale as well as the rest
- Still requires attempting to render every particle

Partial Bucket Sort

Theory:

1. Sort items into buckets
2. ~~Use another sorting algorithm to sort the buckets~~

Practice:

1. Count how many particles will go in each bucket
2. Cumulative-sum the count to get offsets
3. Fill a pointer array with pointers to the particles, starting at the offsets

- not completely sorted
- every particle is tested, so that's fine
- $O(n)$ time

Significantly faster this time

Version	Time(s)	speedup	(cumulative)	Efficiency	
CPU	10728.5	-	-	2.33	9.32
GPU (naive)	1538.8	1.0	1.0	0.90	2.32
GPU (bucket)	271.3	5.7	5.7	5.12	13.16

We're now beating the CPU on every metric.

Significantly faster this time

Version	Time(s)	speedup	(cumulative)	Efficiency	
CPU	10728.5	-	-	2.33	9.32
GPU (naive)	1538.8	1.0	1.0	0.90	2.32
GPU (bucket)	271.3	5.7	5.7	5.12	13.16

We're now beating the CPU on every metric.

Can we do better?

Single rather than double precision

Yes.

- Imaging math is done in double precision, as a carry-over from the CPU code
- GPUs are faster at single precision
- Particle position and movement is still double precision
- Imaging can be done in single precision

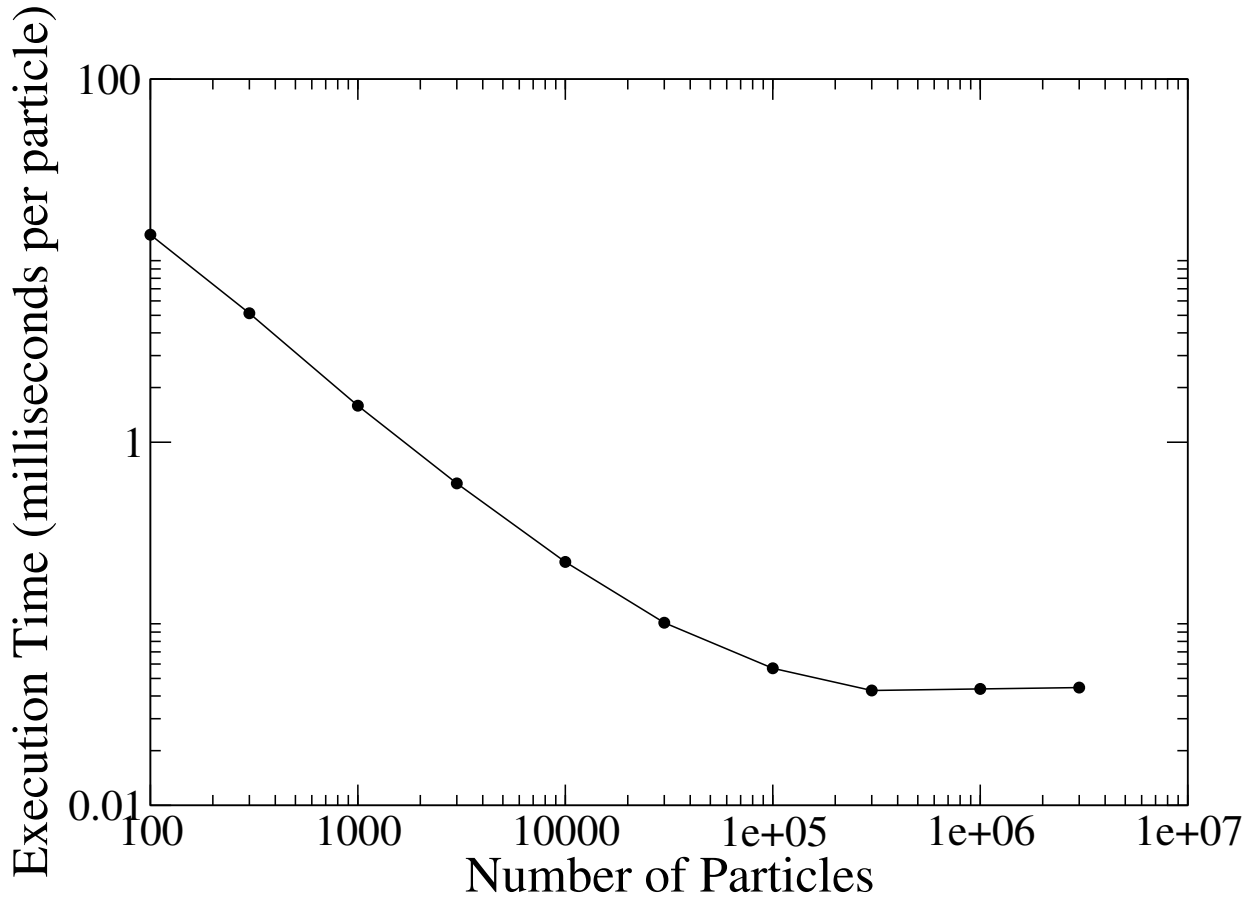
Final performance

Version	Time(s)	speedup	(cumulative)	Efficiency	
CPU	10728.5	-	-	2.33	9.32
GPU (naive)	1538.8	1.0	1.0	0.90	2.32
GPU (bucket)	271.3	5.7	5.7	5.12	13.16
GPU (float)	179.9	1.5	8.57	7.74	19.90

- 60 times faster than the CPU is great for human turn-around time: what took 3 hours takes 3 minutes
- three times more cost efficient
- two times more power efficient

Performance Scaling

- Overhead does not scale with number of particles



Conclusions

- At this point, movement and imaging each occupy about half of total run time
- Global memory bandwidth is nearly at its limit
- Memory rearrangement to improve caching appears to have no positive effect
- General Purpose Graphics Processing Unit computing is an effective technology worth using in appropriate applications

Acknowledgments

We acknowledge support from WPI startup funds, the WPI Alden Fellowship, NSF CBET 1309933, and NSF-MCB 1253444.

Acknowledgments

We acknowledge support from WPI startup funds, the WPI Alden Fellowship, NSF CBET 1309933, and NSF-MCB 1253444.

Questions?