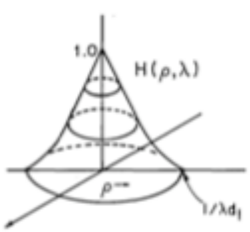
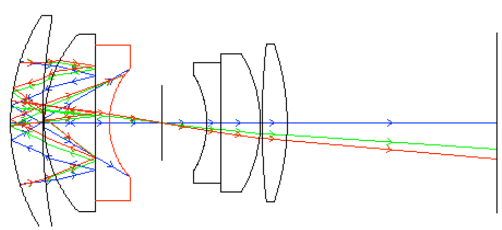
Accelerating optics simulations on modern graphics processing hardware

David Sheffield

PSYCH 221





Overview

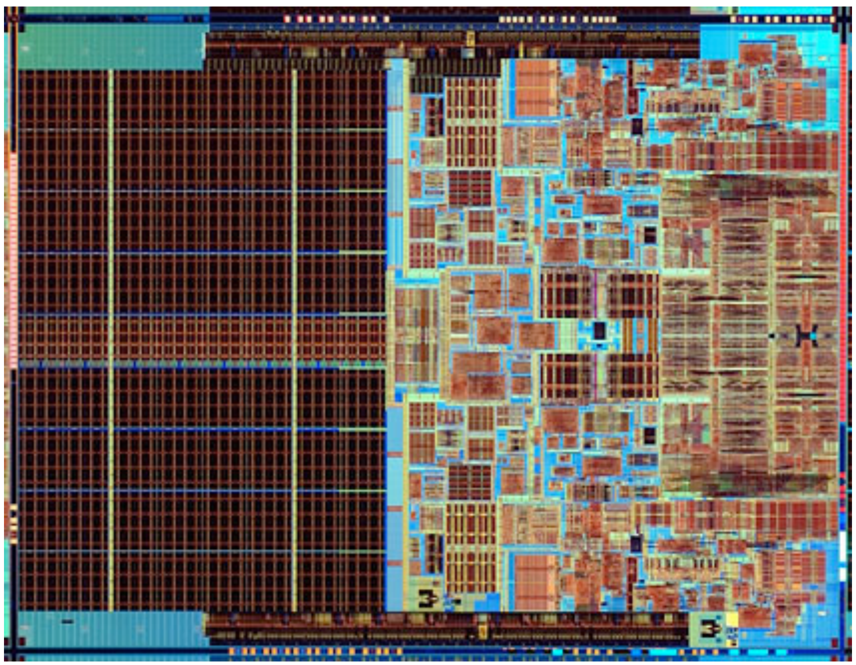
- GPU vs CPU
  - Modern computer architecture in a slide
- MATLAB Interface
- Optics Algorithms
- Implementation
- Results
- Questions
  - Interactive demo?
  - Code walkthrough?

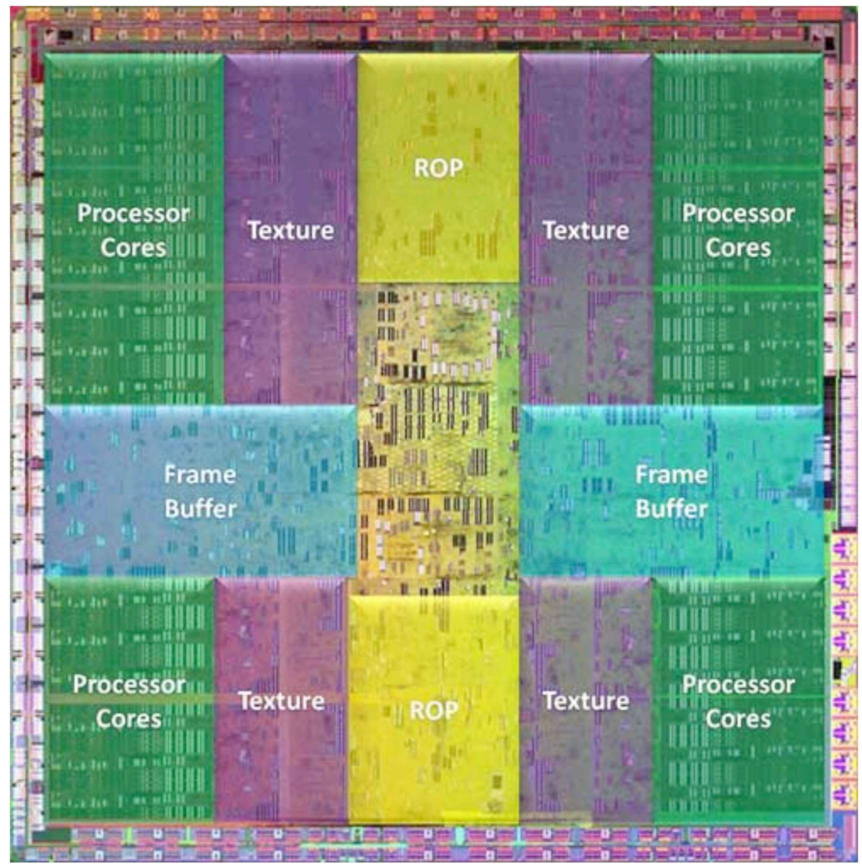
Differences: GPU and CPU

- Modern CPU
  - Lots of hardware for memory latency minimization
    - Caches
    - Branch prediction
    - Out-of-order execution
  - Not many computational resources
    - Less than ten ALUs
    - Supports double-precision floating point in hardware.
    - Fully IEEE 754 compliant
      - Specialized handlers for floating-point underflow and overflow
      - Less analysis needed to implement stable algorithms
    - Single-Instruction, Multiple-Data units operate on short vectors
    - Core 2 Duo has 143 mm^2 die in 65 nm process.
      - 291 million transistors
      - CPU alone costs about $200
  - Easy to program
    - Might not get good performance though
  - Works across a variety of applications

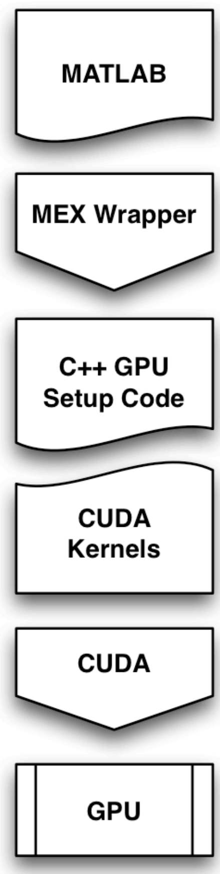
- Modern GPU
  - Lots of hardware for latency tolerance
    - High memory bandwidth
    - Out-of-order thread dispatch
    - Bulk memory transfers
  - Abundant computational resources
    - Hundreds of ALUs
    - Each scalar ALUs fed is fed by a single thread of execution
    - GTX 260 series has 576 mm^2 die in 65 nm process!
      - 1.4 billion transistors!
      - Cards available for $190
  - Harder to program than a CPU
    - Programmed with a variant of C/C++
      - Needs NVIDIA C/C++ compiler
      - Specialized runtime environment
    - Disjoint address space
    - High performance program formulation might be non-intuitive
    - Not entirely IEEE 754 compliant
      - GeForce 8xxx/9xxx series do not have double-precision support
      - Traps not handled
  - Only works well on data-parallel problems
    - Linear algebra

Die Shot




ROP
Processor Cores
Texture
Texture
Processor Cores
Frame Buffer
Frame Buffer
Processor Cores
Texture
ROP
Texture
Processor Cores

MATLAB Interface



- MATLAB accesses GPU through MEX interface
  - MEX allows MATLAB to call functions written in C/C++/FORTRAN
- NVIDIA compiler generates objects files which can be linked into a C/C++ application
  - MEX wrapper calls CUDA kernels for optics computation on the GPU.

Optics Algorithms

Algorithm 1 Shift-invariant optics algorithm

for $ww = 1$ to nWavelengths **do**
  $x \leftarrow FFT2(in\_img[ww])$
  $t \leftarrow x * otf[ww]$
  $out\_img[ww] \leftarrow IFFT2(t)$
end for

Algorithm 2 Diffraction-limited optics algorithm

for $ww = 1$ to nWavelengths **do**
  $x \leftarrow FFT2(in\_img[ww])$
  $otf = DIFF\_OTF(ww)$
  $t \leftarrow x * otf$
  $out\_img[ww] \leftarrow IFFT2(t)$
end for

Algorithm 3 Optics ray trace algorithm

for $ww = 1$ to nWavelengths **do**
  for $rr = 2$ to imgHeight **do**
    for $pp = 1$ to nPixels **do**
      $psf \leftarrow wgt[pp] * psf1 + (1 - wgt[pp]) * psf2$
      $rotpsf \leftarrow ROTATE(psf, theta[pp])$
      $ipsd \leftarrow INTERPOLATE(rotpsf, newsize)$
      $npsf \leftarrow NORMALIZE(ipsf)$
      $outimg \leftarrow outimg[xoffset, yoffset] + npsf$
    end for
  end for
end for

Implementation

Shift-invariant and diffraction-limited optics

- Modifications to opticsOTF.m
- FFT2 and IFFT2 performed on the GPU
  - Using NVIDIA's CUFFT library
- Images are padded with zeros. The image dimensions need to be both square and a power of two.
  - Increases FFT precision
  - Avoids transpose: MATLAB uses column-major indices while CUFFT uses row-major
- Diffraction-limited optics computes the OTF on the GPU
  - Fast computation on the GPU
    - Square root and arccosine are much faster on the GPU than the CPU
  - Avoids copies from the CPU memory space to the GPU memory space
  - Used formula presented by Subbarao.

Ray trace optics

- Modifications to rtPSFApply.m
- The inner loop of the ray trace algorithm is entirely performed on the GPU
- Interpolation and rotation on the GPU yield the greatest performance boost
  - The GPU has hardware for 2D-texture interpolation.
    - Used in the rasterization pipeline but accessible in general purpose computation mode
    - Very fast though little details available about accuracy
  - MATLAB's **imrotate** function is very slow
    - Poorly documented too.
  - Two different versions of rotation were written for the GPU
    - Point interpolation
      - Simple and cheap
    - Bilinear interpolation
      - Memory access is not regular and significantly more branches required
  - Neither rotation kernel exactly matches **imrotate**
    - MATLAB expert needed!

Apparatus

- MATLAB student version used
  - 32-bit binary
- ISET 3.0

	MacBook Pro	Generic Desktop 1	Generic Desktop 2
Processor	T9300	E6600	E8500
Frequency	2.5 GHz	2.4 GHz	3.16 GHz
L2 Cache	6 MB	4 MB	6 MB
Memory	2 GB	4 GB	8 GB
GPU	GeForce 8600M GT	GeForce 9800 GT	GeForce 9800 GX2
GPU Memory	512 MB	512 MB	512 MB
GPU Memory Frequency	700 MHz	900 MHz	1 GHz
GPU Bus Width	128 bit	256 bit	256 bit
GPU Threads	32	112	128
MATLAB Version	2008b	2008b	2007a
OS	OS X 10.5.6	Linux 2.6.28.19	Linux 2.6.28.19

Results: PSNR (Macbeth D50)

Shift-invariant optics

Metric	GPU Computation
Red pixel MSE	3.87
Green pixel MSE	2.49
Blue pixel MSE	1.35
Red pixel PSNR	42.26 dB
Green pixel PSNR	44.16 dB
Blue pixel PSNR	46.83 dB
Image PSNR	44.03 dB

Diffraction-limited optics

Metric	GPU Computation
Red pixel MSE	0
Green pixel MSE	0
Blue pixel MSE	0
Red pixel PSNR	84.25 dB
Green pixel PSNR	∞
Blue pixel PSNR	90.28 dB
Image PSNR	88.06 dB

Ray trace optics

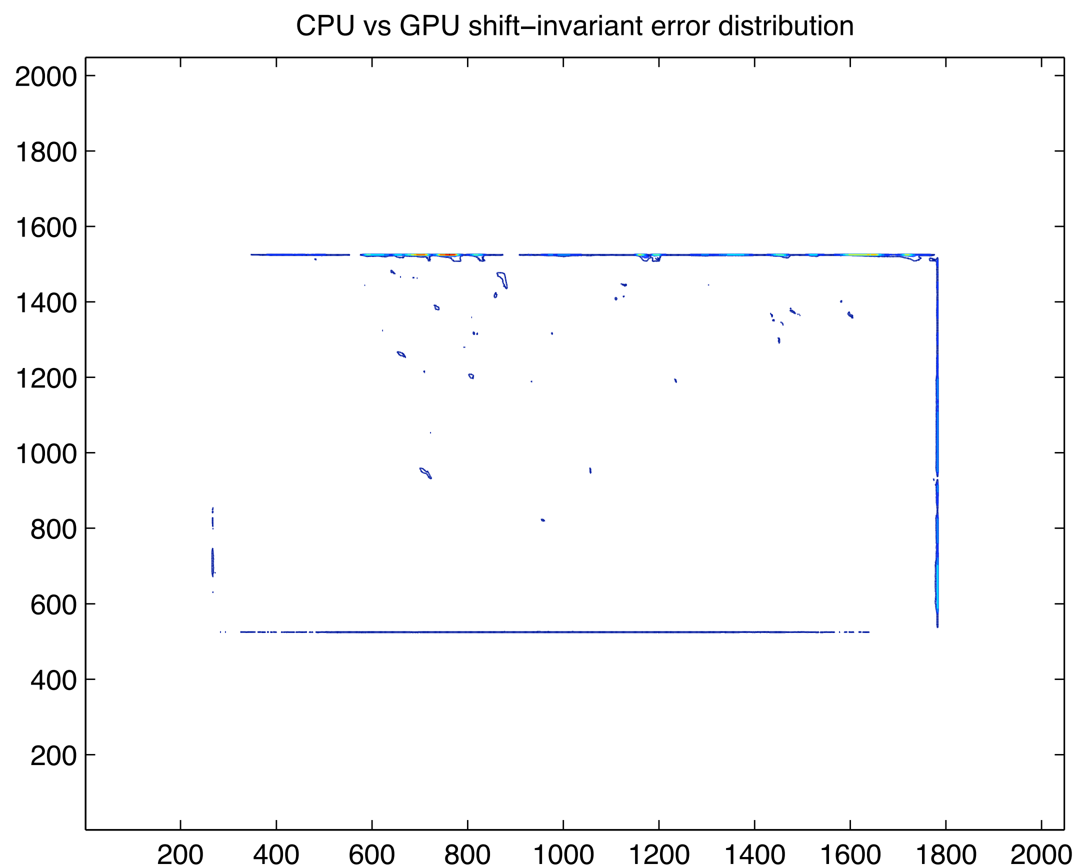
Metric	Fast CPU rotation	GPU point rotation	GPU bilinear rotation
Red pixel MSE	2.36	45.55	48.77
Green pixel MSE	1.85	35.27	38.24
Blue pixel MSE	0.80	15.29	16.32
Red pixel PSNR	44.40 dB	31.55 dB	31.25 dB
Green pixel PSNR	45.45 dB	32.66 dB	32.31 dB
Blue pixel PSNR	49.05 dB	36.29 dB	36.00 dB
Image PSNR	45.89 dB	33.07 dB	32.76 dB

Results: Shift-invariant performance (basketball team)

	MacBook Pro	Generic Desktop 1	Generic Desktop 2
CPU Run-time	255.3 sec	194.3 sec	169.6 sec
Time in **cpu_filter**	65.9 sec	45.8 sec	47.3 sec
GPU Run-time	203.4 sec	136.2 sec	102.0 sec
Time in **gpu_filter**	27.2 sec	10.3 sec	9.1 sec
Wall speed-up	1.26 x	1.43 x	1.67 x
Accelerated code speed-up	2.4 x	4.4 x	5.2 x

Metric	GPU Computation
Red pixel MSE	1.89
Green pixel MSE	2.65
Blue pixel MSE	1.47
Red pixel PSNR	45.36 dB
Green pixel PSNR	43.91 dB
Blue pixel PSNR	46.56 dB
Image PSNR	45.11 dB

Error distribution: Shift-invariant

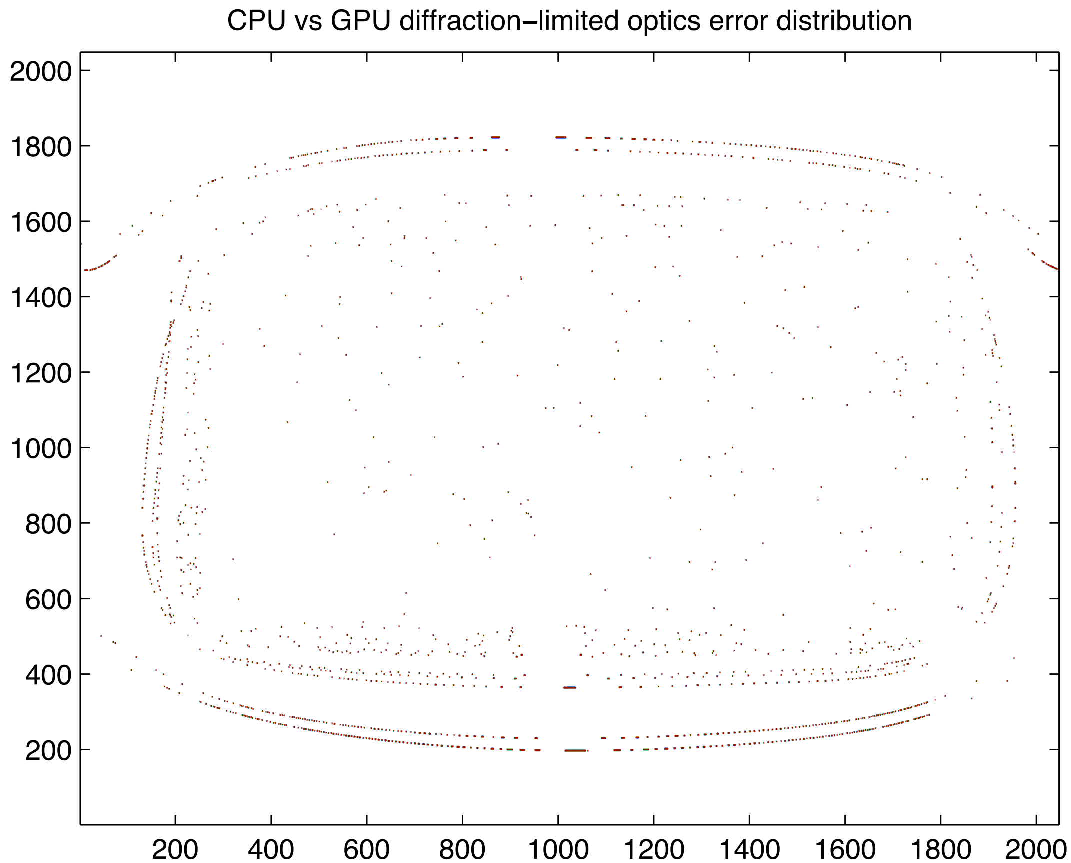


Results: Diffraction-limited performance (basketball team)

	MacBook Pro	Generic Desktop 1	Generic Desktop 2
CPU Run-time	193.9 sec	132.5 sec	119.2 sec
Time in **cpu_filter**	62.6 sec	32.8 sec	36.8 sec
Time in **oiCalculateOTF**	74.9 sec	54.1 sec	48.8 sec
GPU Run-time	90.9 sec	64.6 sec	49.7 sec
Time in **gpu_filter**	33.2 sec	11.6 sec	9.1 sec
Wall speed-up	2.13 x	2.05 x	2.40 x
Accelerated code speed-up	4.1 x	7.5 x	9.4 x

Metric	GPU Computation
Red pixel MSE	.00017
Green pixel MSE	.00024
Blue pixel MSE	.00010
Red pixel PSNR	85.75 dB
Green pixel PSNR	84.29 dB
Blue pixel PSNR	88.07 dB
Image PSNR	85.77 dB

Error distribution: Diffraction-limited



Diffraction-limited MATLAB Profile

CPU Profile

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
gui_mainfcn	1	204.968 s	0.024 s	
oiWindow	1	204.968 s	0.000 s	
oiWindow>btnSimulate_Callback	1	204.944 s	0.012 s	
oiCompute	1	186.867 s	0.012 s	
opticsDLCompute	1	186.855 s	0.048 s	
opticsOTF	1	159.534 s	0.000 s	
opticsOTF>oiApplyOTF	1	159.534 s	0.799 s	
oiCalculateOTF	31	73.021 s	0.242 s	
cpu_filter	31	58.732 s	8.288 s	
dlMTF	31	55.453 s	8.167 s	
dlCore	31	46.535 s	44.672 s	
oiGet	1479	40.594 s	9.051 s	
oiSet	119	27.200 s	0.375 s	
ieCompressData	34	26.559 s	26.498 s	
ifft2	31	24.296 s	24.296 s	
fft2	31	22.457 s	22.457 s	
ieUncompressData	98	18.464 s	18.452 s	

GPU Profile

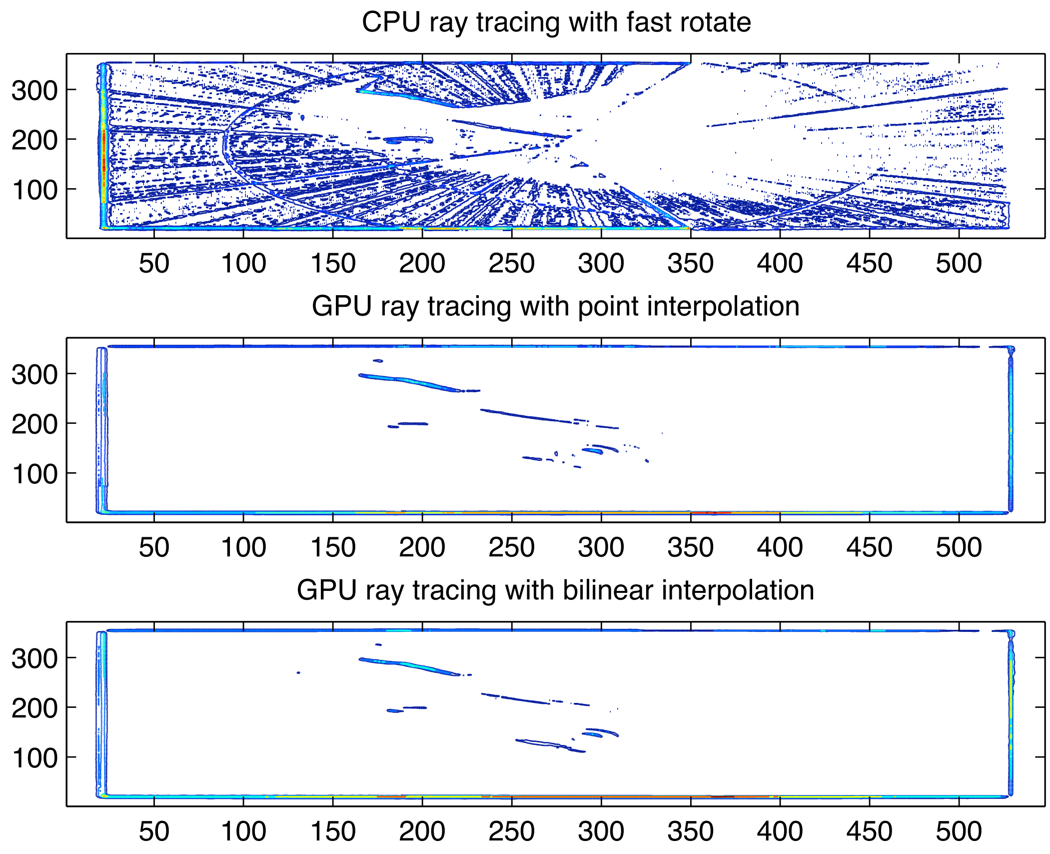
Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
gui_mainfcn	1	118.030 s	0.013 s	
oiWindow	1	118.030 s	0.000 s	
oiWindow>btnSimulate_Callback	1	118.017 s	0.000 s	
oiCompute	1	95.317 s	0.000 s	
opticsDLCompute	1	95.317 s	-0.000 s	
opticsOTF	1	64.839 s	0.013 s	
opticsOTF>oiApplyOTF	1	64.826 s	0.689 s	
gpu_filter (MEX-function)	31	34.162 s	34.162 s	
oiSet	119	30.346 s	0.384 s	
ieCompressData	34	29.643 s	29.564 s	
ieUncompressData	98	28.981 s	28.848 s	
oiGet	735	28.385 s	1.789 s	
oiSetEditsAndButtons	1	22.633 s	0.000 s	
oiWindow>oiRefresh	1	22.633 s	0.000 s	
oiPad	1	14.828 s	0.172 s	
oiCalculateIlluminance	1	13.715 s	1.842 s	
sceneShowImage	1	12.695 s	0.133 s	

Results: Ray trace performance (eagle)

	MacBook Pro	Generic Desktop 1	Generic Desktop 2
CPU Run-time	Not measured	Not measured	65000 sec
Point sampling	2759 sec	1072 sec	838 sec
Bilinear interpolation	3542 sec	1234 sec	1037 sec
Point speed-up	23.6 x	61.6 x	77.6 x
Bilinear speed-up	18.4 x	52.7 x	62.7 x

Metric	Fast CPU rotation	GPU point rotation	GPU bilinear rotation
Red pixel MSE	7.37	30.39	23.19
Green pixel MSE	15.13	76.68	60.11
Blue pixel MSE	10.32	50.14	39.60
Red pixel PSNR	39.46 dB	33.30 dB	34.48 dB
Green pixel PSNR	36.33 dB	29.28 dB	30.34 dB
Blue pixel PSNR	38.0 dB	31.13 dB	32.15 dB
Image PSNR	37.45 dB	30.94 dB	32.0 dB

Error distribution: Ray trace eagle

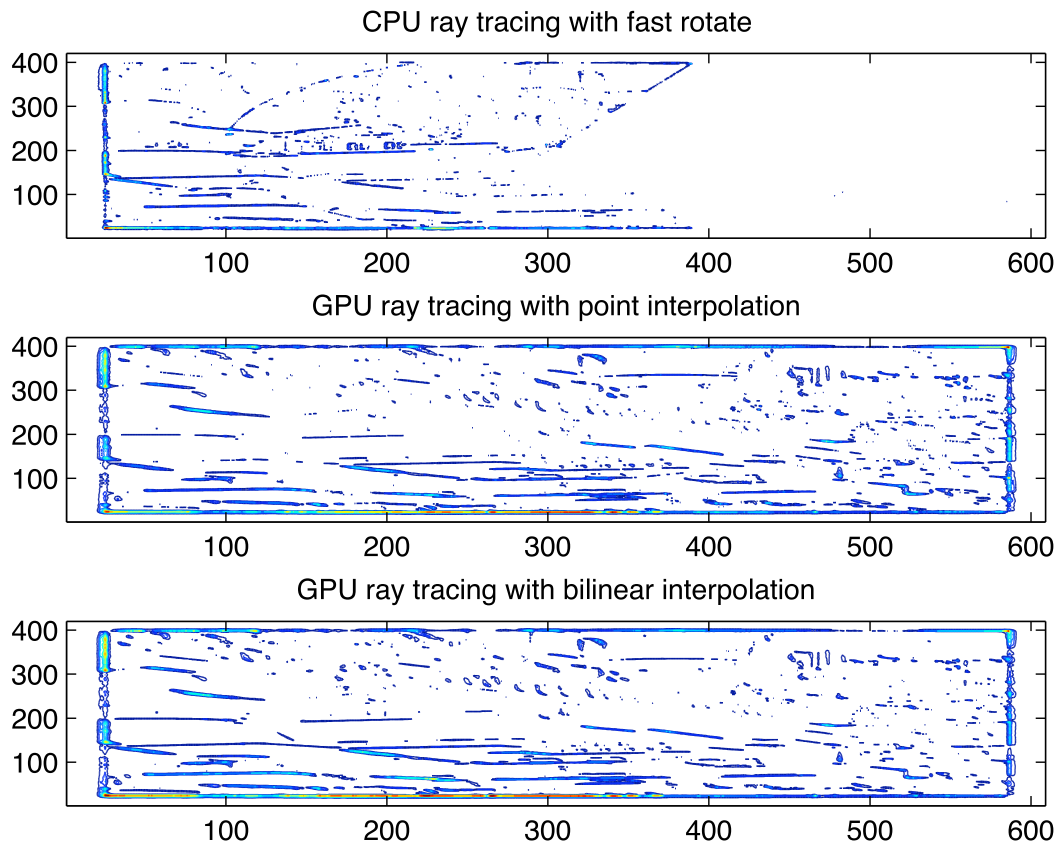


Results: Ray trace performance (Memorial Church)

	MacBook Pro	Generic Desktop 1	Generic Desktop 2
CPU Run-time	Not measured	Not measured	84732 sec
Point sampling	3475 sec	1310 sec	1048 sec
Bilinear interpolation	4409 sec	1559 sec	1297 sec
Point speed-up	24.4 x	64.7 x	81.9 x
Bilinear speed-up	19.2 x	54.4 x	64.3 x

Metric	Fast CPU rotation	GPU point rotation	GPU bilinear rotation
Red pixel MSE	1.59	37.46	41.06
Green pixel MSE	3.45	73.43	75.82
Blue pixel MSE	1.2	31.6	32.87
Red pixel PSNR	46.13 dB	32.39 dB	32.0 dB
Green pixel PSNR	42.75 dB	29.47 dB	29.33 dB
Blue pixel PSNR	47.34 dB	33.14 dB	33.0 dB
Image PSNR	45.0 dB	32.00 dB	31.15 dB

Error distribution: Ray trace Memorial Church



Conclusions

- GPU acceleration can greatly accelerate optics simulations
  - Shift-invariant: 5x
  - Diffraction-limited: 10x
  - Ray trace: 82x
- GPU ray trace optics calculation may need further tweaking for optimal accuracy
  - Better GPU rotation code to closely match MATLAB's **imrotate**
  - GPU hardware interpolation may be too inaccurate
    - Interpolation could be done in software on the GPU
      - Huge performance loss though
- Numeric analysis for computation with single-precision arithmetic would be helpful in debugging image error
  - Or just try the code on a new GPU with double-precision support