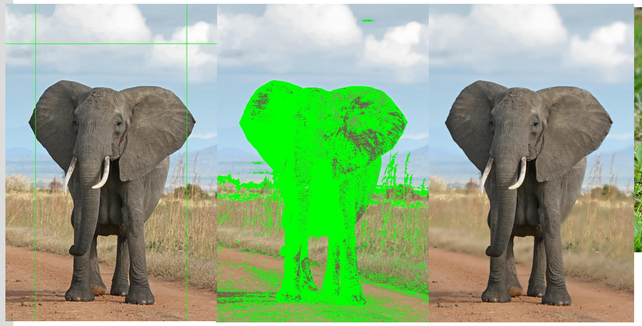
Parallel Portrait Mode

Ariel Davis (azdavis) Jerry Yu (jerryy)

https://github.com/azdavis/parallel-portrait-mode



Background Region | Mask | Output Image

Introduction

Portrait mode has traditionally been done by high end DSLR camera hardware, which the foreground of the subject is in focus and the background is blurred. We have chosen to use a software-only image processing approach to this problem, segmenting the image into the foreground and blurring the background.

Algorithm

1. Designate the borders of the image as the background. Top, left, and right 1/8 of image.
2. Get a color distribution of the background region. For each background pixel, increment the appropriate color count bucket. (Color Counts)
3. Create a foreground mask by finding every pixel with a color not in the background color distribution. Filter background colors that make up less than 1 percent of the background area. (Build Mask)
4. For every unmasked pixel, add it to the mask if pixel with a color not in the background color distribution. (Refine Mask)
5. Blur all the pixels that are not within the foreground mask. We used a convolution with a circular filter to emulate the bokeh effect. (Blur)
6. For each pixel in the mask, add the original pixel of the image into the blurred version of the image.

Approach

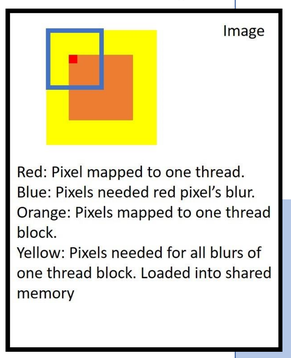
GPU:
- Parallelize over pixels, one thread per pixel for all steps
- Atomic add into color distribution
- Load all pixels needed for blurring block into shared memory

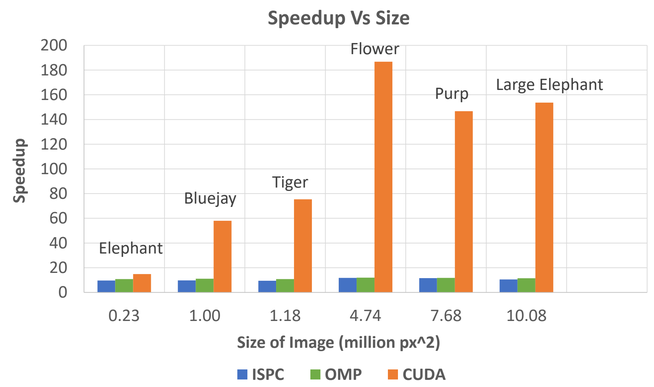
CPU OMP:
- Parallelize over rows, one thread per row for all steps
- Foreground pixels do not require blur, uneven work per row
- Dynamic scheduling for blur, small granularity to even work

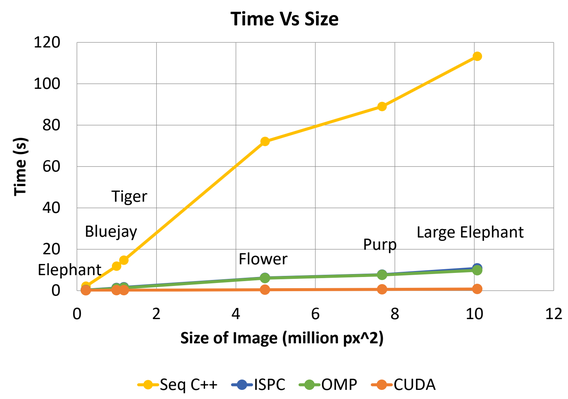
ISPC:
- Dynamic scheduling of tasks, one row per task
- SIMD over each column

Other Challenges:
- Synchronizing after every step
- Managing data cross CPU/GPU and ISPC formats

GPU Thread Mapping:







Speedup

	Size	OMP	ISPC	CUDA
Elephant	389 × 584	10.8240x	9.6238x	14.8264x
Bluejay	1200 × 832	11.1078x	9.7846x	58.0582x
Tiger	1400 × 845	10.8540x	9.5010x	75.3517x
Flower	2242 × 2112	**11.9523x**	**11.7707x**	**186.7298x**
Purp	2944 × 2609	11.7518x	11.5943x	146.6925x
Large Elephant	2592 × 3888	10.7315x	10.5507x	153.6602x

Time

	C++	OMP	ISPC	CUDA
Elephant	2.0887s	0.1930s	0.2170s	0.1409s
Bluejay	11.8096s	1.0632s	1.2070s	0.2034s
Tiger	14.6917s	1.3536s	1.5463s	0.1950s
Flower	72.0560s	6.0286s	6.1216s	0.3859s
Purp	88.9941s	7.5728s	7.6757s	0.6067s
Large Elephant	113.224s	**9.8219s**	**10.7315s**	**0.7369s**

Time Breakdown For Large Elephant (s)

	C++	OMP	Speedup	CUDA	Speedup
Init	0.0118	0.0087	-	0.1803	0.0655
Color Counts	0.0283	0.0181	-	0.0000	726.2308
Build Mask	0.0640	0.0152	4.2228	0.0026	24.8173
Refine Mask	0.0337	0.0162	2.0788	0.0000	1984.235
Blur	**112.73**	**9.4200**	11.9673	0.1695	665.0706
Clean Up	0.3547	0.3436	-	**0.3844**	-
Total	113.22	9.8219	11.5278	0.7369	153.6602

Conclusions

- GPU is better for image processing than CPU
  - GPU: 20 SM * 1024 threads each > CPU: 16 hyperthreads
  - GPU: Parallelize over pixels, CPU: Parallelize over rows
- CUDA shared memory gave 2x speedup
- CUDA bottleneck was File I/O data transfer, saving image
- CUDA has relatively low overhead on simple operations, OMP has higher overhead with less arithmetic intensity
- OMP/ISPC bottleneck was computing blur
- OMP threads faster than ISPC tasks and SIMD due to scatter/gather and type conversions
- OMP/ISPC dynamic scheduling with low overhead with low granularity

References

Foreground segmentation algorithm was a simplified version of grab cut algorithm: https://dl.acm.org/citation.cfm?id=1015720