

Real Time Image Segmentation

Miklos Homolya, Ravikishore Kommajosyula, Gaurav Kukreja

Technical University of Munich

April 3, 2014

Overview

- 1 Introduction
- 2 Algorithm
 - Binary Image Segmentation
 - Grayscale Image Segmentation
 - Primal-Dual Method
- 3 CUDA Implementation
- 4 Optimizations
 - Texture Memory
 - OpenGL Interoperability

Problem Definition

Binary Image Segmentation

Energy functional

$$E_1(u) := \int_{\mathbb{R}^N} |\nabla u| + \lambda \int_{\mathbb{R}^N} |u(x) - f(x)| \, dx$$

Functional derivative

$$\frac{\delta E_1}{\delta u} = -\operatorname{div} \left(\frac{\nabla u}{|\nabla u|} \right) + \lambda \frac{u - f}{|u - f|}$$

Gradient descent solver



Tony F. Chan, Selim Esedoglu and Mila Nikolova (2005)
Finding the Global Minimum for Binary Image Restoration

Sample Result

Grayscale Image Segmentation

Euler-Lagrange equation

$$\operatorname{div} \left(\frac{\nabla u}{|\nabla u|} \right) - \lambda s(x) - \alpha \nu'(u) = 0$$

where $s(x) = (c_1 - f(x))^2 - (c_2 - f(x))^2$, and $\alpha \nu'(u)$ forces u into $[0; 1]$.

Gradient descent solver



Tony F. Chan, Selim Esedoglu and Mila Nikolova (2004)
Algorithms for Finding Global Minimizers of Image
Segmentation and Denoising Models

Sample Result

Primal-Dual Method

Motivation: Gradient descent solver has slow convergence.

Primal variable u

Dual variable ξ (roughly similar to $\text{grad } u$):

CUDA Implementation

- Update kernels calls from CPU to have synchronizaton
- Update X and update U implemented as two kernels
- Image arrays swapped after each iteration
- Branching to avoid invalid memory accesses

CUDA Implementation

- Swapping images after each iteration makes things difficult
- Can not be used in gradient calculation, Can be used in divergence calculation
- Texture memory used on intermediate results X_i and X_j
- Improves the fps by 12 %

OpenGL Interoperability

What is Interoperability?

- Mapping OpenGL Resources to CUDA, to enable CUDA to read/write
- Can be used to show output from CUDA kernel, straight from GPU saving time and bandwidth

How to use OpenGL Interop?

- Set current threads OpenGL context to use for OpenGL interop with CUDA **device**.

```
cudaGLSetGLDevice(device);
```

- Create OpenGL Pixel Buffer, and register to use as CUDA buffer.

```
gl.glGenBuffers(1, &pixels);  
gl.glBindBuffer(GL_PIXEL_UNPACK_BUFFER, pixels);  
size_t size = w * h * 4 * sizeof(unsigned char);  
gl.glBufferData(GL_PIXEL_UNPACK_BUFFER, size, 0,  
               GL_DYNAMIC_DRAW);  
cudaGraphicsGLRegisterBuffer(&pixels_CUDA, pixels,  
                             cudaGraphicsMapFlagsWriteDiscard);
```

How to use OpenGL Interop?

Inside the Display Loop,

- Before starting kernel, map pixel buffer to a CUDA pointer.

```
cudaGraphicsMapResources(1, &pixels_CUDA, 0);  
cudaGraphicsResourceGetMappedPointer(&d_pixels, &size,  
    pixels_CUDA);
```

- Pass CUDA pointer as parameter for kernel. The kernel writes to the buffer in **RGBA8** format.
- After kernel execution, unmap pixel buffer.

```
cudaGraphicsUnmapResources(1, &pixels_CUDA, 0);
```

- Draw buffer

```
glDrawPixels(w, h, GL_RGBA, GL_UNSIGNED_BYTE, 0);
```

References



John Smith (2012)

Title of the publication

Journal Name 12(3), 45 – 678.

The End