Combining Sketch and Tone for Pencil Drawing Production

Stefan Burnicki, Raphael Braun, Andreas Altergott

November 4, 2014

Spoiler

Table of Contents

Project Objectives

- GPU implementation of the paper Combining Sketch and Tone for Pencil Drawing Production
- GPU implementation must be faster than the CPU implementation
- Understanding of a larger CUDA implementation

Implementation Steps

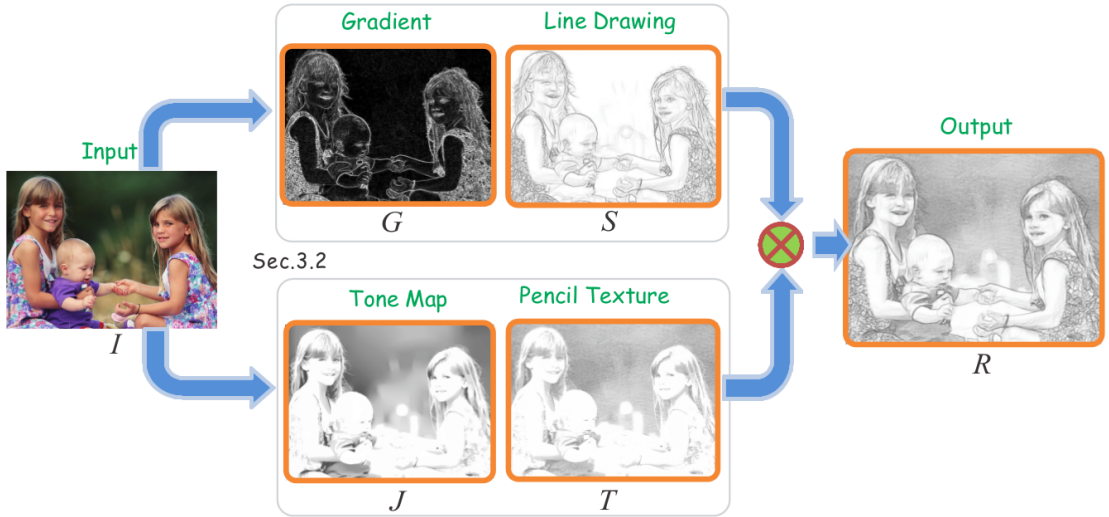
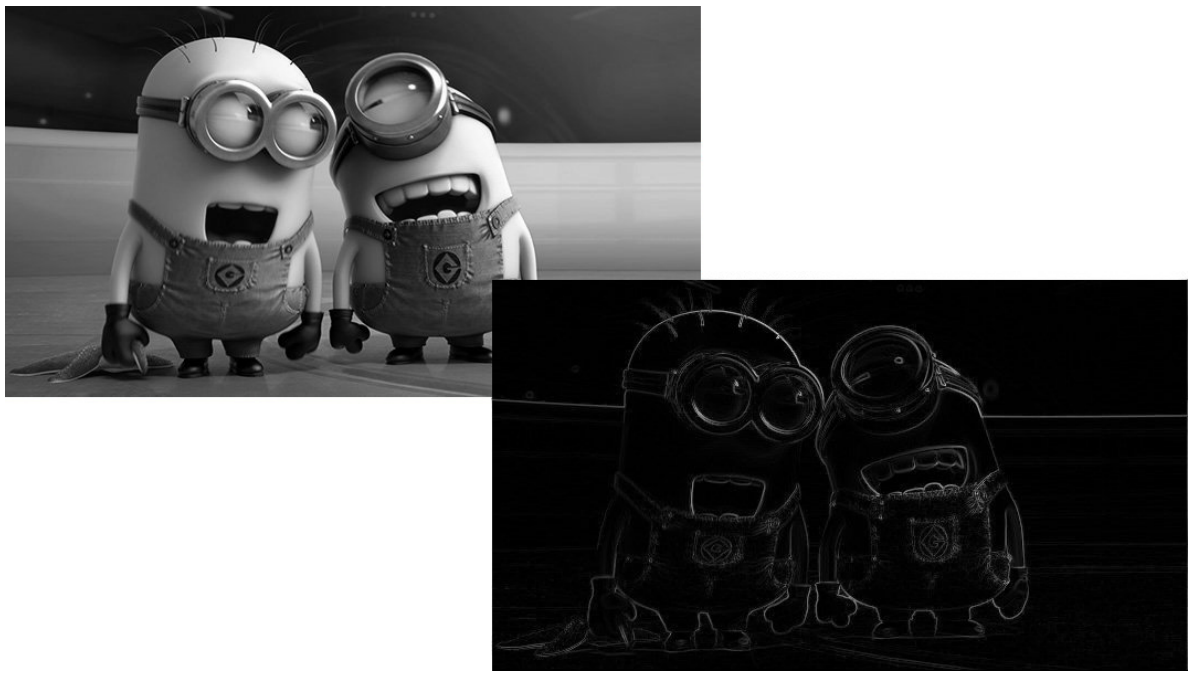


Figure : Overview of the pencil drawing framework

- Line Drawing
- Tone Drawing
- Merge of Line and Tone Drawing

Line Drawing



- Forward gradient calculation on the gray-scale version of the image with $G = \left((\partial_x I)^2 + (\partial_y I)^2 \right)^{\frac{1}{2}}$

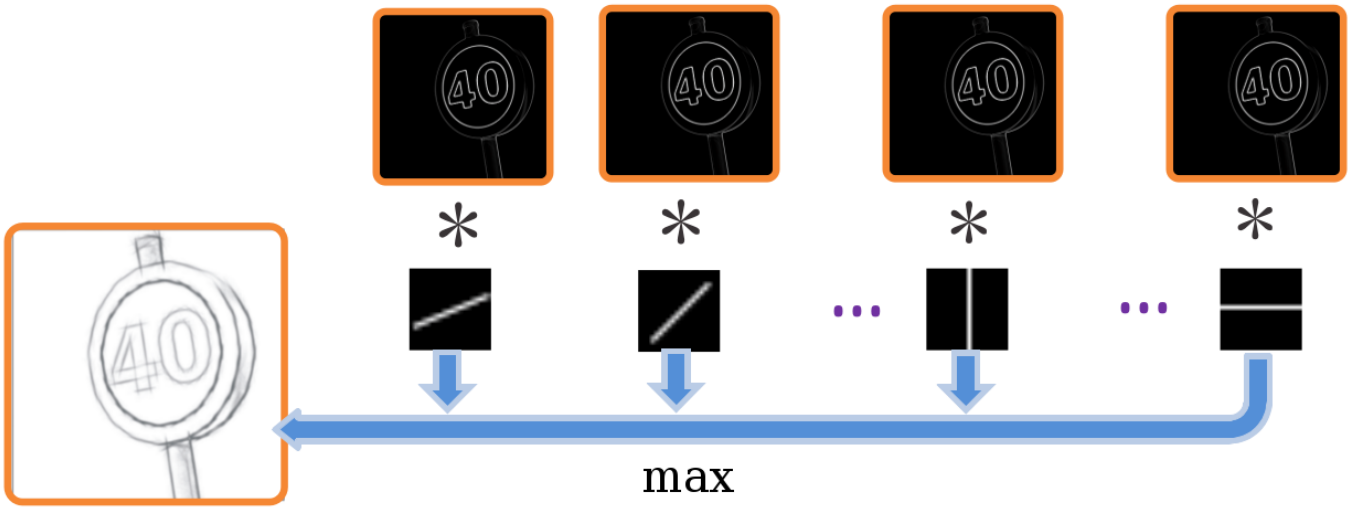
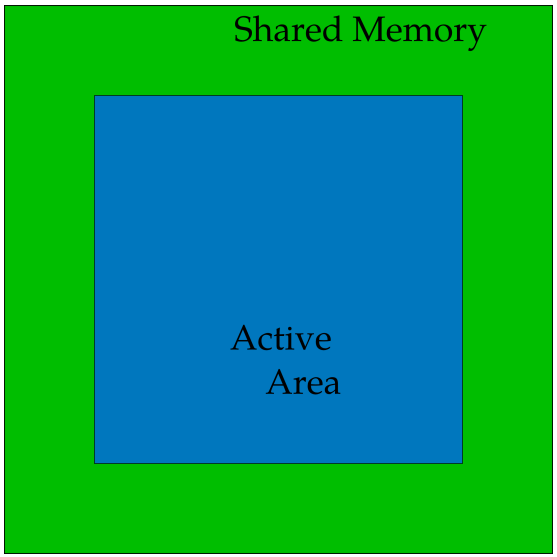
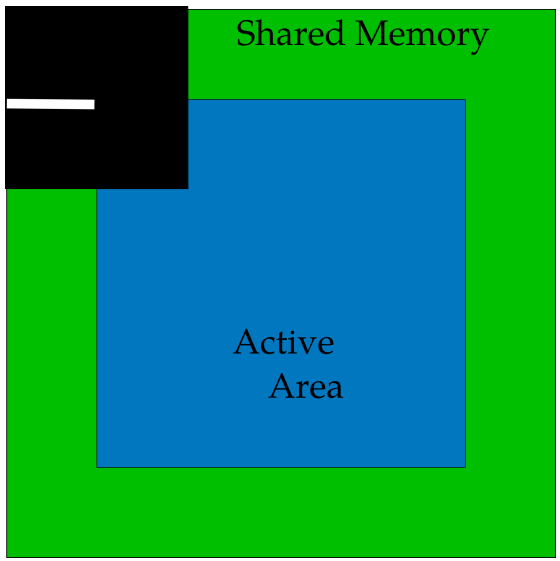


Figure : Line drawing with strokes

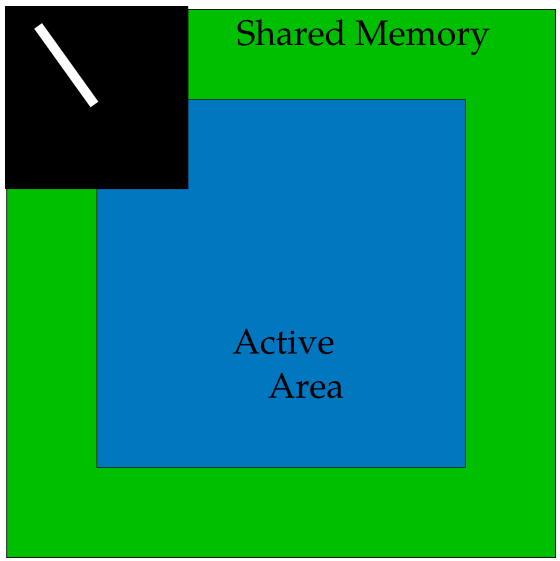
- Computation of the response map $G_i = L_i * G$, where L_i is a line segment at direction i (implemented in the convolution kernel) and $*$ the convolution operator
- Selection of the maximum response L for the pixel p: $L = \max(\{G_i\})$



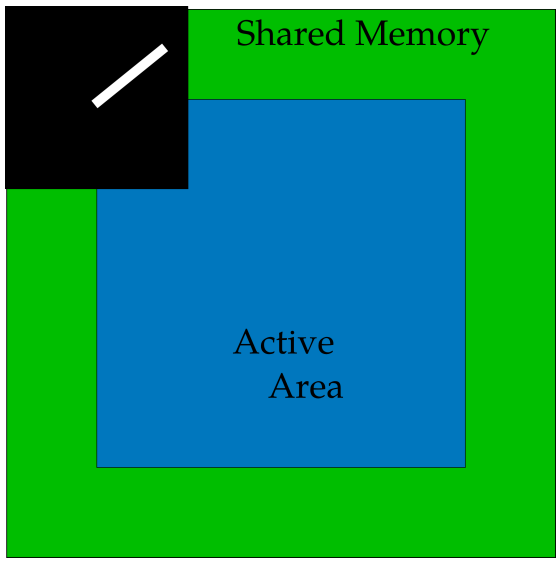
- Use Shared Memory for convolution implementation
- Final stroke map S obtained by inverting pixels of L and mapping them to $[0, 1]$



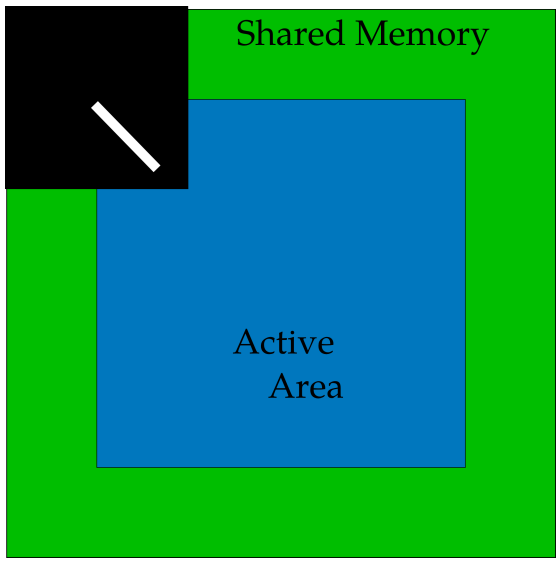
- Use Shared Memory for convolution implementation
- Final stroke map S obtained by inverting pixels of L and mapping them to $[0, 1]$



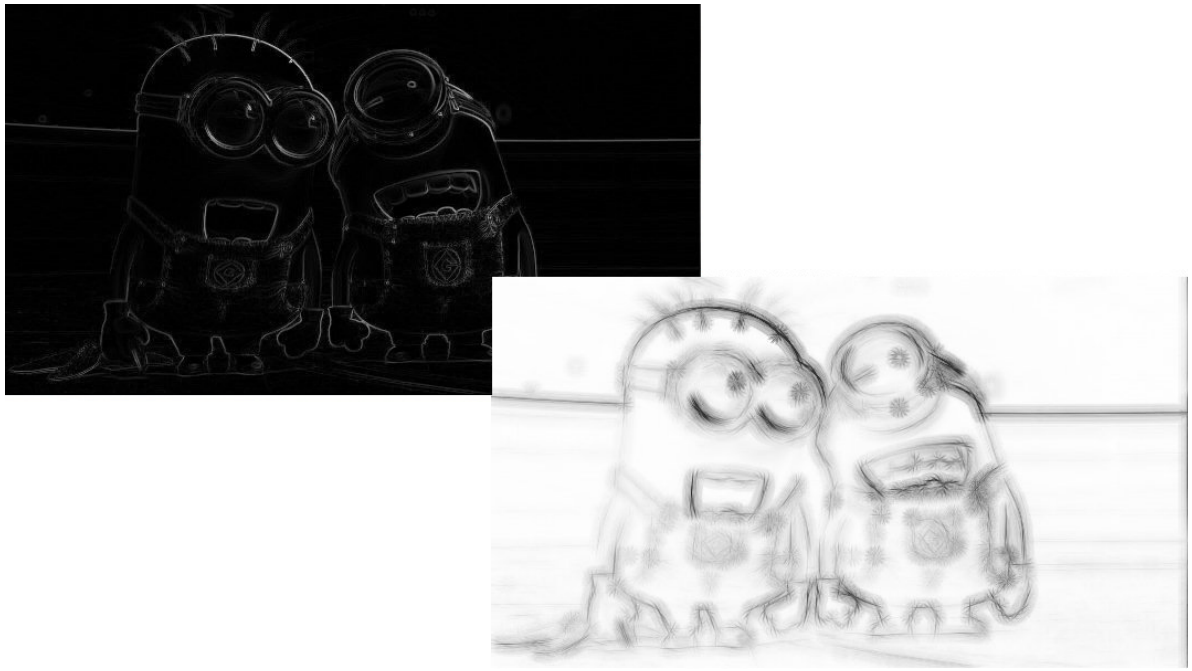
- Use Shared Memory for convolution implementation
- Final stroke map S obtained by inverting pixels of L and mapping them to $[0, 1]$



- Use Shared Memory for convolution implementation
- Final stroke map S obtained by inverting pixels of L and mapping them to $[0, 1]$



- Use Shared Memory for convolution implementation
- Final stroke map S obtained by inverting pixels of L and mapping them to $[0, 1]$



- Use Shared Memory for convolution implementation
- Final stroke map S obtained by inverting pixels of S' and mapping them to $[0, 1]$

Tone Drawing

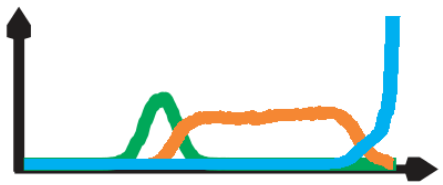


Figure : Tone distribution p_1, p_2, p_3

- Usage of constant parameters $\omega_1, \omega_2, \omega_3, \sigma_b, u_a, u_b, \mu_d, \sigma_d$
- Computation of distribution per tonal layer with $p(v) = \frac{1}{Z} \sum_{i=1}^{3} \omega_i p_i(v)$
- Computation of target tone distribution p_1, p_2, p_3
- Pencil Texture rendering

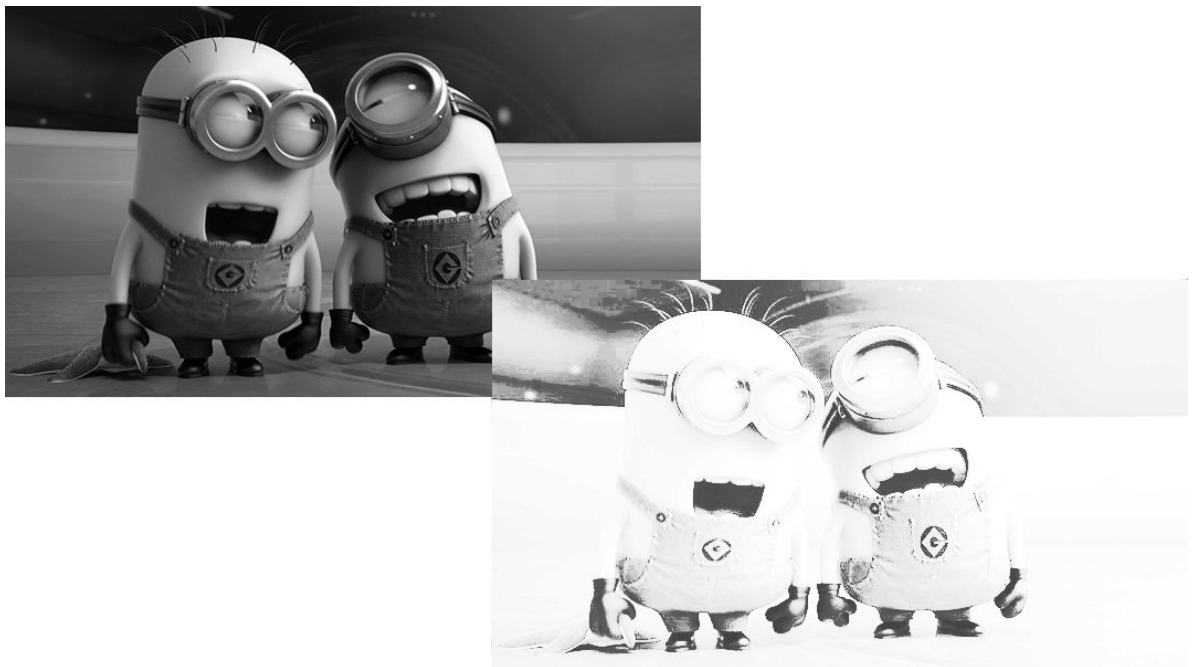
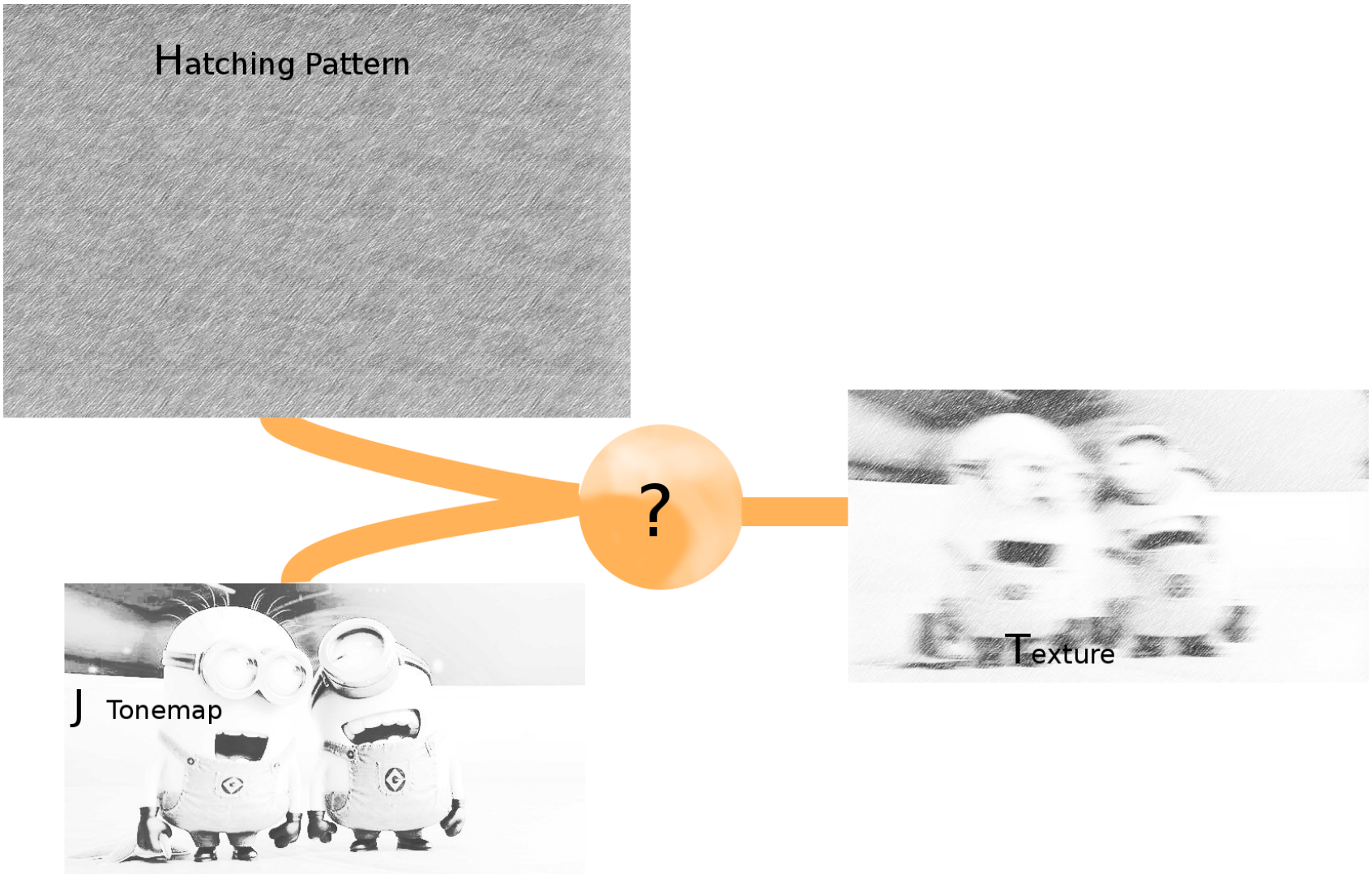
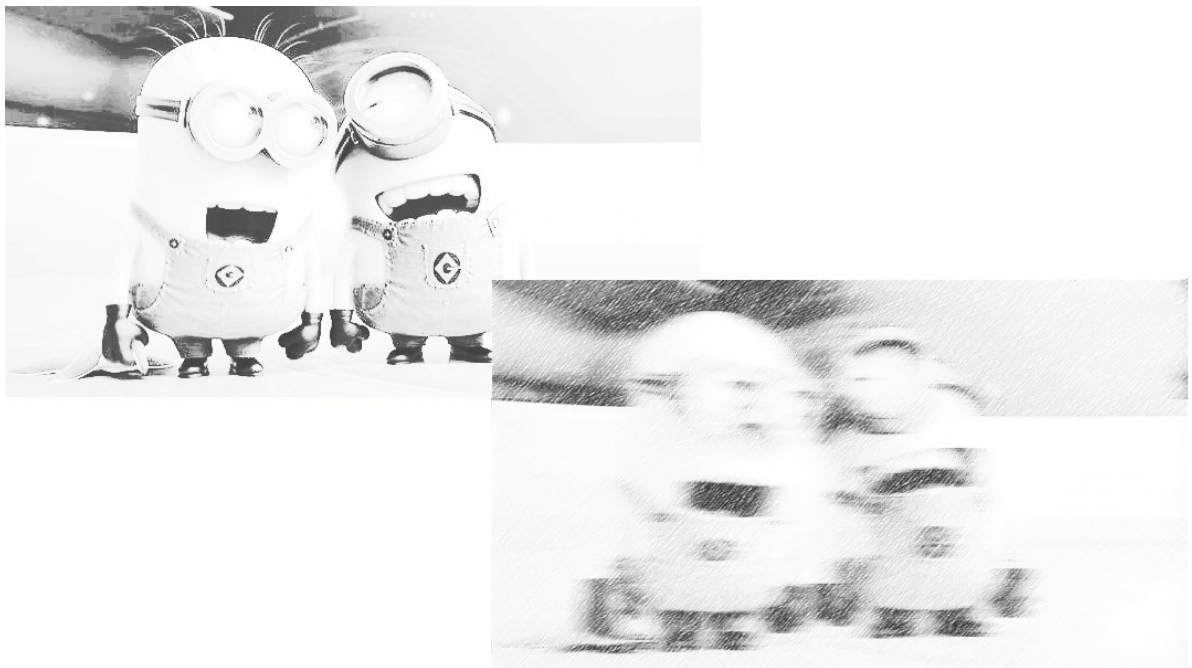


Figure : Result of tone map

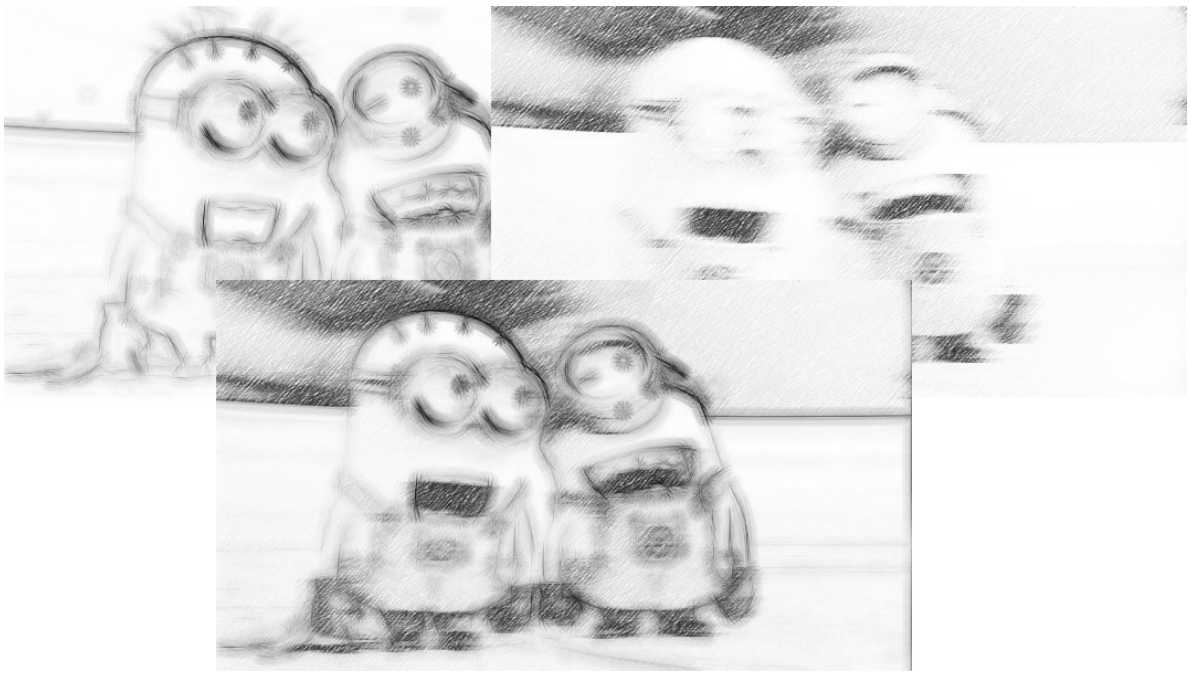

Hatching Pattern
?
Texture
J Tonemap

- Simulate preasure of pencil: H^{β}
- We need: $H(x)^{\beta(x)} \approx J(x)$ with a smooth β
- Solve in log-space: $\beta = \arg\min_{\beta} \|\beta \ln(H) - \ln(J)\|_2^2 + \lambda \|\nabla \beta\|_2^2$

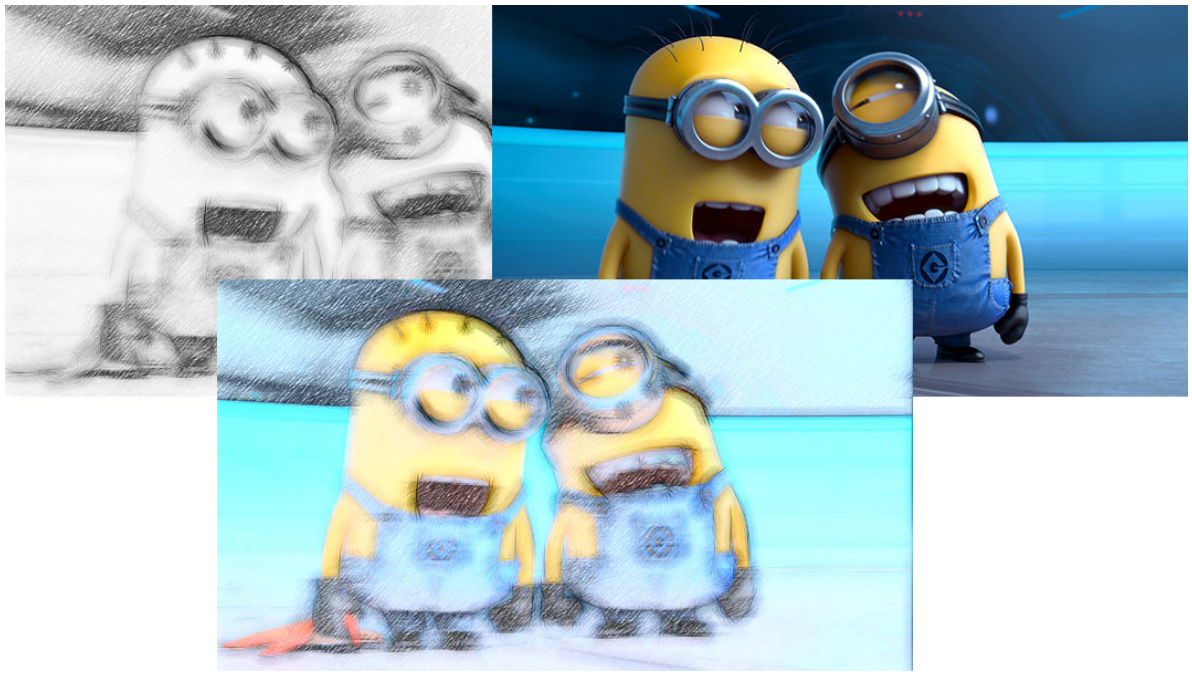


- Tikhonov regularization can be reformulated as Linear System
- Solved with conjugent gradient using cuda sparse library (cusp)

Merge of Line and Tone Drawing

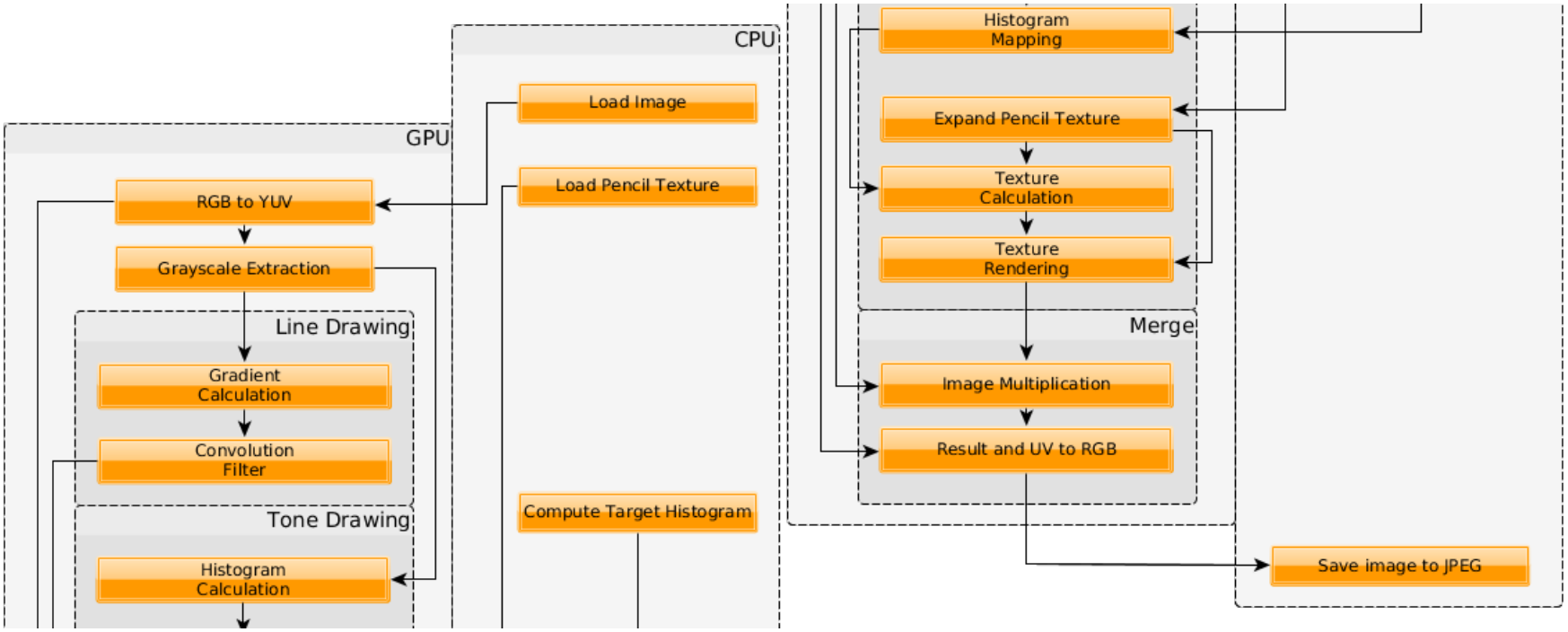


- Multiplication of S and T values per pixel in $R = S \cdot T$

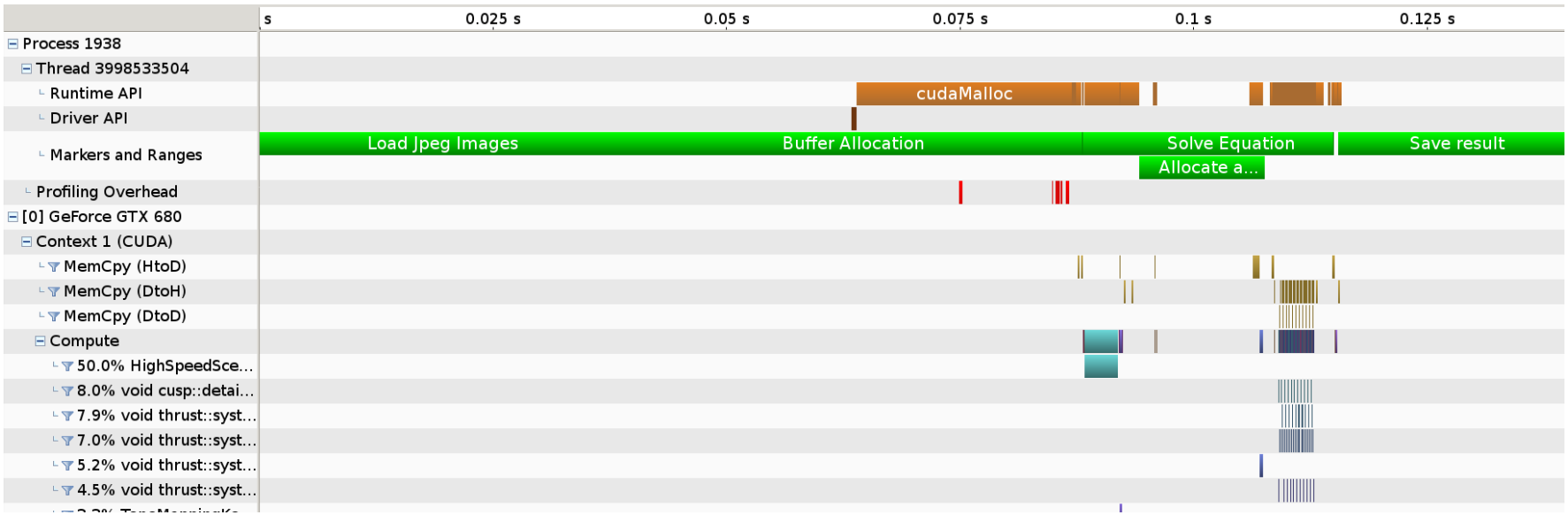


- Simply use generated output R as Y in YUV color space, keeping U and V from original image for colored output

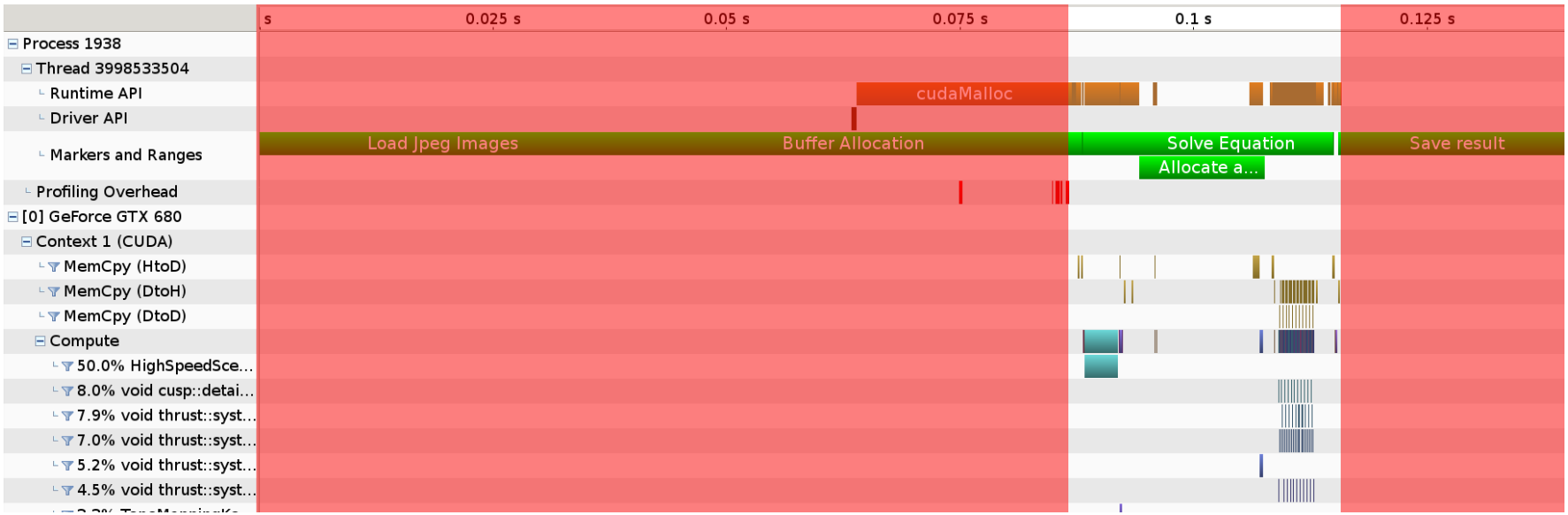
Kernel Pipeline



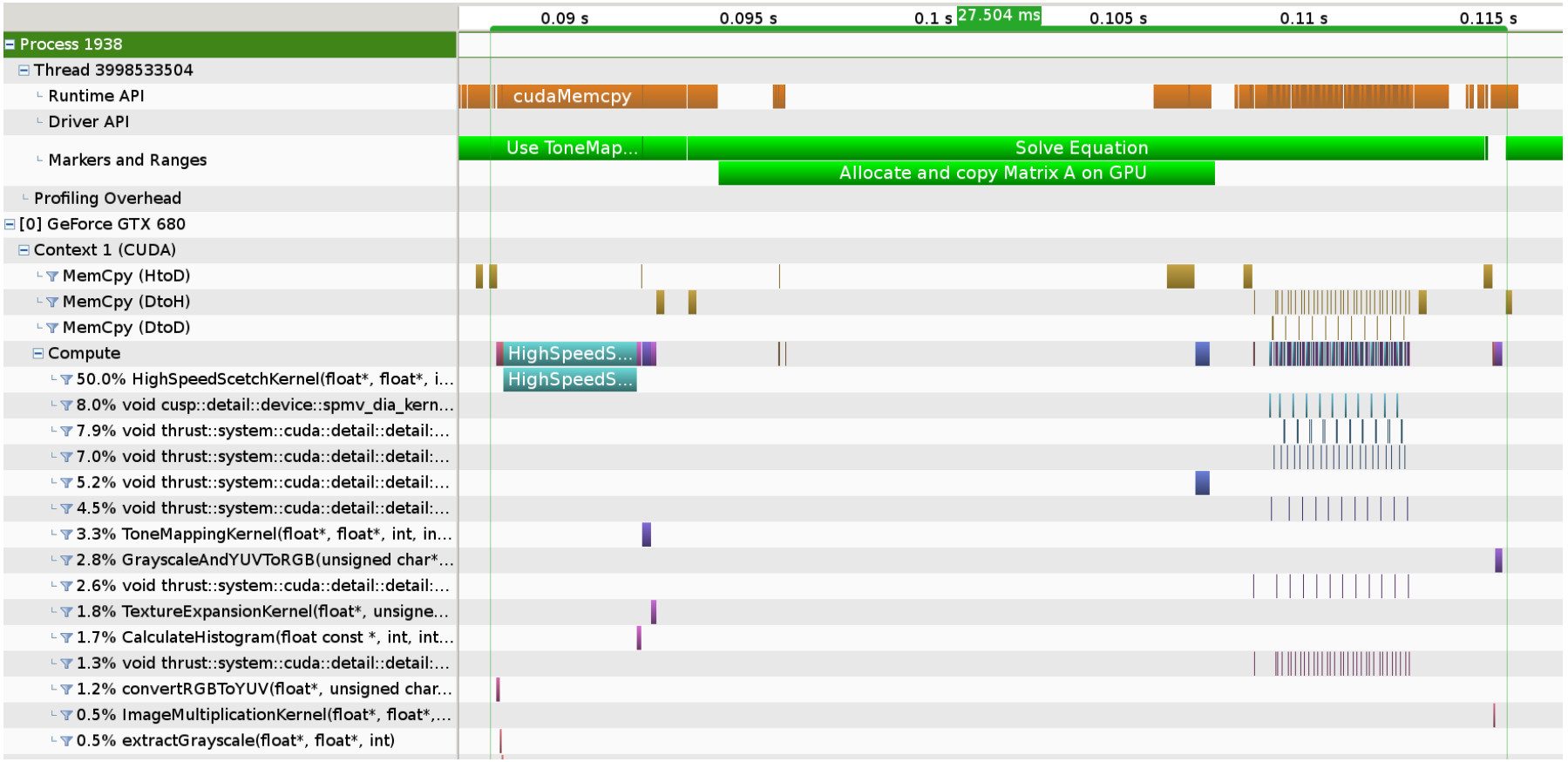
Profiling Results



One Time Overhead



Detailed View



Live Demonstration

Conclusion

- Successful GPU implementation of the sketch filter as described in the paper
- Our GPU implementation is faster than the version of the paper
- Speedup of 72.1 compared to CPU implementation of the original paper ($27.73ms$ at 600×600 pixels)
- 30 ms per frame in DVD quality
- 60 ms per frame in HD Ready
- 130 ms per frame in Full HD quality

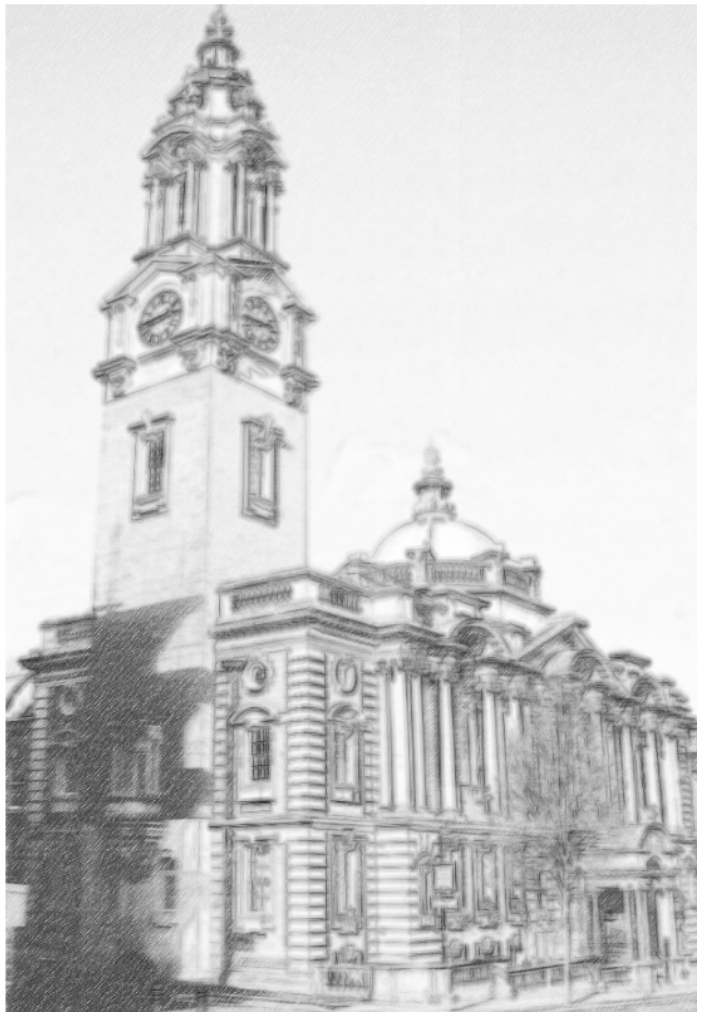


Figure : Their result on the left, our result on the right

Mastered Obstacles

- Understanding of the paper
- Newcomers to image processing
- Realization of a bigger GPU implementation

Future Improvements

- Better optimization can improve performance even more
- Pre image analyzer could automatically calculate the ideal line length for the line drawing
- Pre image analyzer could also calculate the parameters for the tone drawing

Thank you for your attention.

Questions?