computergraphics tübingen

Cuda Implementation of Advanced Pencil Sketch Filter

Andreas Altergott, Raphael Braun, Stefan Burnicki

August 26, 2014

Practical course:
Massive Parallel Programming SS2014
Computer Graphics

Cuda Implementation of Advanced Pencil Sketch Filter

Andreas Altergott, Raphael Braun, Stefan Burnicki

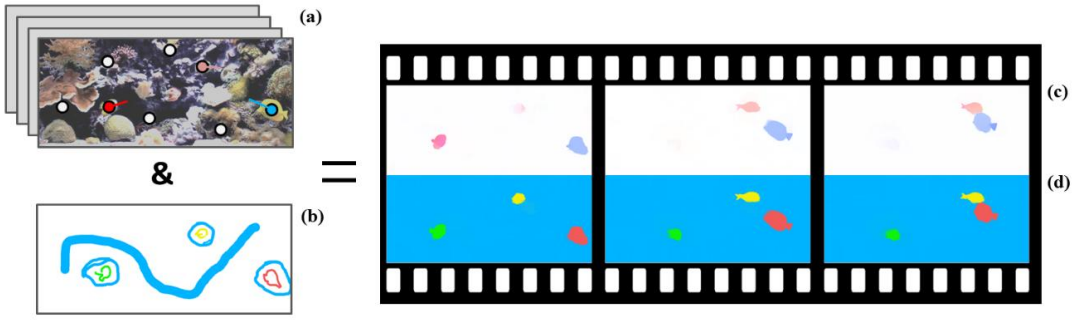

Figure 1: TODO

Abstract

This report is about blabla.

1 Introduction

Creating pencil sketches have a very long history. Throughout time artist have perfected the art of creating pencil drawings in all thinkable styles, ranging from completely abstract to stunningly photorealistic. However such skills are rare, and only well trained artists or passionated hobby artists are able to actually create pleasing looking images using nothing but the eye, a paper and a pencil. Today, in a time where cameras can be used to capture a realworld scene in a matter of seconds, with accurate proportions and colors and no skills, the need for pencil sketches seem to be gone. However, photographs do not convey the same emotions that sketches do. There is a certain timeless flair to pencil sketches that makes them just nice to look at.

Therefore designing an pencil sketch filter, which produces a pencil sketch from a regular photograph is an exciting task. The filter described in [LXJ12] is such a filter, which consists of two stages: sketching the outlines and drawing the shading. In the sketch stage the filter calculates several line convolutions and the shading stage solves a huge linear system of equations. Both tasks are computationally rather expensive and according to the authors their implementation of the filter takes 2 seconds to run on a 600×600 image. Figure 1 shows input, intermediate results and the final result of the filter.

This report presents our efforts to implement the entire filter on the GPU using Cuda, to gain realtime results. The structure of the report is as follows: The filter itself is described in Section 3, then in Section 4 we describe our implementation using Cuda. Section 5 shows our benchmarking results for various input images and Section 6 some possible optimizations and applications for our implementation.

2 Previous Work

This work is based on [LXJ12], as this paper describes the filter that is implemented. The filter requires quite some parameters. In [LXJ12] they use a parameter lerning approach to automatically select those parameters. In this work the filter is kept slightly simpler than in the origin paper, e.g. the line filter is slightly simplified and the filter parameters are left as user controllable options.

Another GPU-sketch-filter is described in [LSF10], though it uses the a regular shader-pipeline. Based on the neighbourhood informations of each pixel it calculates weather to place

a stroke at this point or not. The strokes are then rendered as stroke-textures. The strokes are made in three different detail layers based on the gradientmagnitude. Furthermore it is possible to filter animations using optical flow. The greatest difference to the filter that we implement is that they calculate each individual pencil or brush stroke based on the image and then render those strokes using stroke-textures, and our method uses traditional filters to alter the original image. Their filter is quite versatile, as it allows different painting stiles by changing the stroke-textures and weighting the detail layers.

A partially GPU based method is described in [BLC+12]. However they concentrate on rendering temporal coherent line drawing animations from a given 3D-scene. First they find active contours that are then adapted throughout the animation. The result pictures are rendered by parameterizing the and then add a certain style to the path. While this method yields amazing results for very stylistic animations, it needs 3D input data.

3 Method

The filter considers *line scetching* and *shading* separately. The following subsections shed some light on how both stages work precisely.

The input for the filter is a regular RGB-image I_{rgb}. In the first step a grayscale image I_g is calculated using the Y-channel of the YUV-transformed image I_{iuv}. I_g now serves as input for the future steps.

3.1 Line Scetching

One prominent task of the filter is to create sketchy looking outlines. The main objective here is to mimic freehand sketching, which is typically done by drawing short straight lines aligned to the outlines. The outlines are detected using a simple gradient operator. Then each pixel is assigned to a line direction and finally the pixel is drawn as part of its line.

Gradient Image The Outlines are detected and stored in the image G using gradient magnitudes:

$$G = ((\partial_x G)^2 + (\partial_y G)^2)^{\frac{1}{2}}$$

TODO Bild verweis!!

To classify each pixel to a line direction it would be possible to just use gradient directions, however those are typically noisy. Therefore a very stable convolution based direction classification is used.

Line Convolution Filter Given just the gradient magnitudes G for each line direction $i \in \{1, \cdots, N\}$ where N is the total number of lines, the convolution between G and each line segment $\mathscr{L}_i$ is calculated.

$$G_i = \mathscr{L}_i * G$$

The value $G_i(p)$ of pixel p will be very big, if that pixel lies directly on a line in G (edge) and if $\mathscr{L}_i$ is following this line, such that only big values are collected in the convolution. If the pixel doesn't lie on or close to a line it can not gather high values and therefore stay dark. Pixels which lie very close to edges can still gather some brightness if the line segment $\mathscr{L}_i$ intersects the edge. This way lines in G which follow the direction of $\mathscr{L}_i$ show up and slightly overshoot in G_i.

Now to actually draw the lines, each pixel selects its maximum value from all G_i:

$$L = \max(\{G_i\}) \quad \forall i \in \{1, \cdots, N\}$$

TODO: Bilder einfÃijgen!

3.2 Shading

Histogram Matching

Textureing Equation

4 GPU Implementation

The temporal consistency that can be achieved using the described method is usable for many applications, however as the optical flow is always needed for temporal filtering it is calculated no matter which application is implemented. Optical flow by itself can be used used for various applications like frame interpolation, motion blur or motion magnification [LTF+05].

Disparity Estimation Disparity estimation is about computing the correspondence between a pair of stereoscopic images. The result $J_{xy} = d_{xy}$ can be used as depth map for various tasks. Finding correspondences in stereoscopic images is very similar to calculating optical flow from consecutive

video frames:

$$E_{data}(d) = \sum_{(x,y)} \left|\left|I^r(x + d_{xy}, y) - I^l(x, y)\right|\right|^2$$

$$E_{smooth}(d) = \sum_{(x,y)} \left(||\nabla d_{xy}||\right)^2$$

J is sparsely initialized by feature matches between I^r and I^l and then filtered as it was done for optical flow. Figure 2 shows an example.

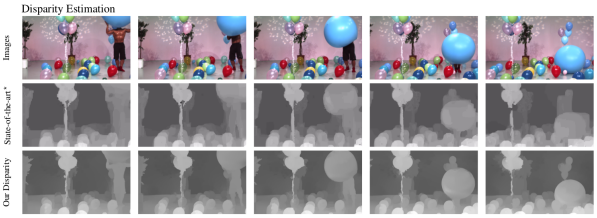

Figure 2: The results of the disparity estimation is compared of a state of the art dataset provided by the MPEG group for testing. Again the results of the proposed method is more stable in time. Less holes are randomly appearing and disappearing compared to MPEG data set.

Colorization and Scribble Propagation The approach can be used to propagate user input (pen strokes) through a given video. The scribbles can have arbitrary meaning like color input for colorization or ids for object labeling. The input scribbles s_j from frame j are used to initialize $J_j = s_j$. Then the joint temporal smoothing is applied to create intermediate frames. In average only every 20th frame needs own scribble data. **??** shows an example for this application.

Depth up-sampling Depth sensor data from devices like Microsoft Kinects often suffer from bad resolution, temporal noise and holes. The authors use their technique to filter this data (J) using the image data from the Kinects camera for the domain transform. As a result J' is getting up-sampled, holes are getting filled and noise is smoothed. Figure 3 shows a before-after comparison.

Saliency Visual saliency maps are used in many graphics applications as they highlight important regions in images. They can be used to change the aspect ratio of a video without distorting important features (persons). In this case temporal inconsistencies show up as wobbling. The authors used a

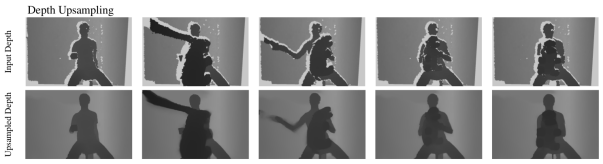

Figure 3: Low resolution and extremely noisy depth maps from a Microsoft Kinect is up-sampled and filled (white holes) using the same joint filtering operation as it is used for the optical flow calculation. The image data from the Kinect camera is used for the domain transform.

per frame saliency calculation l as initialization for $J(x, y) = l(x, y)$ and applied the temporal smoothness operation to gain consistent results. Figure 4 shows the difference between the unfiltered and the filtered saliency.

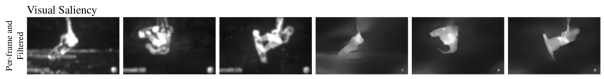

Figure 4: First three images show the per frame saliency calculation, which are then temporally filtered to create the smooth results on the right.

5 Performance

TODO

6 Future Work

TODO

7 Conclusion

A simple and efficient approach was presented that simplifies difficult global optimization problems like optical flow and disparity calculations using temporal edge aware filtering. The method creates temporal consistent results for many image based graphics problems. Sparse feature matches can be used as input for the joint filtering operation to replace costly global minimizations.

There are some limitations to the approach. It is possible that objects are not detected if they do not provide enough features to match. This issue can be addressed by adapting the parameters of the feature matcher to find as many features as possible. The effect of faulty features is minimal as their confidence is low, so they are filtered out.

As the method relies on edge preserving filtering problems occur if important object boundaries are not well defined in the input images. Unwanted bleeding artifacts can show up on such edges. Special care must be taken in the scribble propagation application to place the input scribbles close to edges, as object boundaries could otherwise be missed which causes unwanted color bleedings. A true global solution does not suffer from this problem. The authors suggested an extension that could fix this issue.

For me as writer of this report the described method in the paper was quite easy to comprehend and written in enough detail for a full reimplementation.

References

[BLC+12] Pierre Bénard, Jingwan Lu, Forrester Cole, Adam Finkelstein, and Joëlle Thollot. Active Strokes: Coherent line stylization for animated 3D models. *NPAR 2012, Proceedings of the 10th International Symposium on Non-photorealistic Animation and Rendering*, June 2012.

[GO11] Eduardo S. L. Gastal and Manuel M. Oliveira. Domain transform for edge-aware image and video processing. *ACM TOG*, 30(4):69:1–69:12, 2011. Proceedings of SIGGRAPH 2011.

[LSF10] Jingwan Lu, Pedro V. Sander, and Adam Finkelstein. Interactive painterly stylization of images, videos and 3D animations. In *Proceedings of I3D 2010*, February 2010.

[LTF+05] Ce Liu, Antonio Torralba, William T. Freeman, FrÃl'do Durand, and Edward H. Adelson. Motion magnification. *ACM Trans. Graph.*, 24(3):519–526, 2005.

[LXJ12] Cewu Lu, Li Xu, and Jiaya Jia. Combining sketch and tone for pencil drawing production. In Paul Asente and Cindy Grimm, editors, *Expressive*, pages 65–73. ACM, 2012.

[PKT09] S. Paris, P. Kornprobst, and J. Tumblin. *Bilateral Filtering: Theory and Applications*. Foundations and trends in computer graphics and vision. Now Publishers, 2009.

[RHB+11] C. Rhemann, A. Hosni, M. Bleyer, C. Rother, and M. Gelautz. Fast cost-volume filtering for visual correspondence and beyond. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 3017–3024, Washington, DC, USA, 2011. IEEE Computer Society.

[VBVZ11] Sebastian Volz, AndrÃl's Bruhn, Levi Valgaerts, and Henning Zimmer. Modeling temporal coherence for optical flow. In Dimitris N. Metaxas, Long Quan, Alberto Sanfeliu, and Luc J. Van Gool, editors, *ICCV*, pages 1116–1123. IEEE, 2011.

[Wei98] Joachim Weickert. *Anisotropic Diffusion in Image Processing*. ECMI Series. Teubner, Stuttgart, 1998.

[ZBW11] Henning Zimmer, Andrés Bruhn, and Joachim Weickert. Optic flow in harmony. *Int. J. Comput. Vision*, 93(3):368–388, July 2011.