GPU Histogram Computation

Oliver Fluck* Siemens Corp. Research
Shmuel Aharon† Siemens Corp. Research
Daniel Cremers‡ Siemens Corp. Research
Mikael Rousson§ Siemens Corp. Research

1 Introduction

Due to the immense computational power of today's graphics processors (GPU), general purpose computation on GPUs has become a vivid research area. The performance of algorithms running on GPUs highly depends on how well they can be arranged to fit and exploit the processors single instruction multiple data (SIMD) architecture. Many tasks that are considered simple on a CPU such as grouping and counting of values of a domain for statistical purposes appear rather challenging to be implemented on a GPU.
In this poster, we present a method to compute histograms in shader programs. On the example of image segmentation, we show that our method enables iterative and histogram guided algorithms to run efficiently on graphics hardware without costly CPU intervention. Using an image segmentation example, we demonstrate how the algorithm can be optimized for smaller regions of interest inside larger domains.

2 Exposition

We store input data in a 2D texture with power of two dimensions and subdivide the domain into evenly sized and independent regions (tiles). The tile size depends on the histogram granularity. By using a RGBA texture, a histogram of n bins leads to a tile size of log(n) x log(n) texture elements (texels). The number of tiles equals the input texture size divided by the size of the tiles. Using this scheme, many local histograms will be computed in parallel. Each texel of a tile counts the occurrence of values inside its tile for a particular histogram interval (bin). The according bin is determined using texture coordinates. Thus, after a single pass of log(n) x log(n) texture fetches per texel, every tile represents a local distribution of values in its region. To obtain a global distribution of values inside a domain, we combine all tiles to a single global histogram. Therefore, we sum up tiles by applying a texture reduce operation in a fashion similar as presented in [Krueger and Westermann 2003]. During each rendering pass, we halve texture size in each dimension. The ith texel of a local histogram is summed up with the ith texel of the three - in positive texture coordinate direction - adjacent tiles. This way, all n x n tiles of a domain are combined to a final histogram after log(n) passes.
As a demonstration, we implement our histogram algorithm in the data term of a level set formulation [Kim et al. 2002]. GPU implementations of level sets have been introduced to provide real time user interaction by outperforming CPU implementations by a factor of 15 [Lefohn et al. 2003]. However, previous GPU implementations don't provide methods regarding more sophisticated data terms for higher reliability.
With our method, histograms for inside and outside the segmented regions can be updated quickly while the level set front propagates. We measure timings on a 2.8GHz Pentium IV CPU and a nVIDIA GeForce 7900 GT, PCI-Express graphics card. The segmentation runs inside an image of 256x256 pixels with two 64-bin histograms representing intensity distributions for inside and outside the segmented regions. In order to obtain a relieable representation, we need to update the distributions frequently while the level set front propagates. The GPU calculation for both histograms takes 1.6 ms in total and allows us to keep the entire level set computation on the GPU. In contrast to that, transfering the level set surface to CPU memory and writing computet histograms back to the GPU, would lead to a total time of 5.8 ms until the segmentation process can proceed with the next iteration.

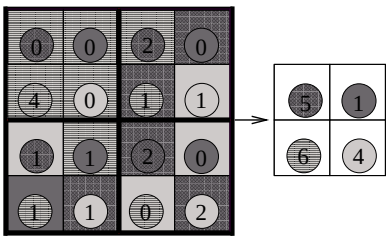

Figure 1: Schematic illustration. The occurrence of a value inside a tile is counted (circled number) in one particular texel of each tile. Tiles are summed up to a global histogram by applying a texture reduce operation.

In cases where counting is desired only inside small regions of interest (such as in our segmentation example), we can further decrease the algorithm's execution time:
A domain subdivided into m x m tiles will be represented by a histogram tile map of m x m texels. Each texel contains the maximum value of the level set function over all values inside a tile. Each positive value inside the map activates the tile it represents. For disabled histogram tiles, the counting fragment program will not be executed. This technique decreases the number of expensive texture fetches during the actual calculation significantly. The tile map itself is created by applying a *max* operator to the level set function during log(m) pixel wise texture reduce passes. Thus, the tile map is created with a few inexpensive passes.

3 Conclusion

We have introduced an algorithm to compute histograms on graphics cards. We showed, using an image segmentation example, that our method allows iterative and data dependent processes to run efficiently on GPUs without data transfer between CPU and GPU memory.

References

KIM, J., FISHER, J., YEZZI, A., CETIN, M., AND WILLSKY, A. 2002. Nonparametric methods for image segmentation using information theory and curve evolution. *IEEE Internation Conference on Image Processing, Rochester*.

KRUEGER, J., AND WESTERMANN, R. 2003. Linear algebra operators for gpu implementation of numerical algorithms. *ACM Transactions on Graphics (TOG) 22*, 3, 908–916.

LEFOHN, A. E., KNISS, J. M., HANSEN, C. D., AND WHITAKER, R. T. 2003. Interactive deformation and visualization of level set surfaces using graphics hardware. *IEEE Trans. on Visualization and Computer Graphics*, 75–82.

*e-mail: oliver.fluck@siemens.com
†e-mail: shmuel.aharon@siemens.com
‡e-mail: dcremers@cs.uni-bonn.de
§e-mail: mikael.rousson@siemens.com