ADABOOST GPU-BASED CLASSIFIER FOR DIRECT VOLUME RENDERING

Oscar Amoros[1], Sergio Escalera[2] and Anna Puig[3]
[1]*Barcelona Supercomputing Center - CNS, K2M Building, c/ Jordi Girona, 29 08034 Barcelona, Spain*
[2]*UB-Computer Vision Center, Campus UAB, Edifici O, 08193, Bellaterra, Barcelona, Spain*
[3]*WAI-MOBIBIO Research Groups, University of Barcelona, Avda.Corts Catalanes, 585, 08007 Barcelona, Spain*

Keywords: Volume rendering, High-performance Computing and parallel rendering, Rendering hardware.

Abstract: In volume visualization, the voxel visibility and materials are carried out through an interactive editing of Transfer Function. In this paper, we present a two-level GPU-based labeling method that computes in times of rendering a set of labeled structures using the Adaboost machine learning classifier. In a pre-processing step, Adaboost trains a binary classifier from a pre-labeled dataset and, in each sample, takes into account a set of features. This binary classifier is a weighted combination of weak classifiers, which can be expressed as simple decision functions estimated on a single feature values. Then, at the testing stage, each weak classifier is independently applied on the features of a set of unlabeled samples. We propose an alternative representation of these classifiers that allow a GPU-based parallelizated testing stage embedded into the visualization pipeline. The empirical results confirm the OpenCL-based classification of biomedical datasets as a tough problem where an opportunity for further research emerges.

1 INTRODUCTION

The definition of the visibility and the optical properties at each volume sample is a tough and non intuitive user guided process. It is often performed through the user definition of *Transfer Functions* (TF). Selection of regions is defined indirectly by assigning to zero the opacity since totally transparent samples do not contribute to the final image. The use of TFs allows to store them as look-up tables (LUT), directly indexed by the intensity data values during the visualization, which significantly speeds up rendering and it is easy to implement in GPUs. In previous works, the transfer function is broken into two separated steps (Cerquides et al., 2006): the Classification Function (CF) and the optical properties assignment. The Classification Function determines at each point inside the voxel model at which specific structure the point belongs. Next, the optical properties assignment is a simple mapping that assigns to each structure a set of optical properties. In this approach, we focus on the definition and the improvement of the Classification Function and its integration into the rendering process. The main advantage of the classification approach is that, since a part of the classification can be carried on a pre-process, before rendering, it can use more accurate and computationally expensive classification methods than transfer functions mappings.

Specifically, we use a learning-based classification method that splits into two steps: learning and testing. In the *learning step*, given a set of training examples, each marked by an end-user as belonging to one of the set of the labels or categories, the Adaboost-based Machine Learning training algorithm builds a model, or *classifier*, that predicts whether a new voxel falls into one category or the other. In the *testing stage*, the classifier is used to classify a new voxel description. Thus, the learning step is done in a pre-process stage, though the testing step is integrated on-the-fly into the GPU-based rendering. In the rendering step, at each voxel value, the classifier is applied to obtain a label. We propose a GPGPU strategy to apply the classifier, interpret the voxels as the set of objects to classify, and their property values, derivatives and positions as the attributes or features to evaluate. We apply a well-known learning method to a sub-sampled set of already classified voxels and next we classify a set of voxel models in a GPU-based testing step. Our goal is three-fold:

- to define a voxel classification method based on a

Amoros O., Escalera S. and Puig A..
ADABOOST GPU-BASED CLASSIFIER FOR DIRECT VOLUME RENDERING.
DOI: 10.5220/0003369902150219
In *Proceedings of the International Conference on Computer Graphics Theory and Applications* (GRAPP-2011), pages 215-219
ISBN: 978-989-8425-45-4

powerful machine learning approach,

- to define a GPGPU-based testing stage of the proposed classification method integrated to the final rendering,
- to analyze the performance of our method comparing five different implementations with different public data sets on different hardware.

2 ADABOOST CLASSIFIER

In this paper, we focus on the Discrete version of Adaboost, which has shown robust results in real applications (Friedman et al., 1998). Given a set of N training samples $(x_1,y_1),..,(x_N,y_N)$, with x_i a vector valued feature and $y_i = -1$ or 1, we define $F(x) = \sum_1^M c_f f_m(x)$ where each $f_m(x)$ is a classifier producing values ± 1 and c_m are constants; the corresponding prediction is $\text{sign}(F(x))$. The Adaboost procedure trains the classifiers $f_m(x)$ on weighted versions of the training sample, giving higher weights to cases that are currently misclassified. This is done for a sequence of weighted samples, and then the final classifier is defined to be a linear combination of the classifiers from each stage. For a good generalization of $F(x)$, each $f_m(x)$ is required to obtain a classification prediction just better than random (Friedman et al., 1998). Thus, the most common "weak classifier" f_m is the "decision stump". Stumps are single-split trees with only two terminal nodes. If the decision of the stump obtains a performance inferior to 0.5 over 1, we just need to change the polarity of the stump, assuring a performance greater (or equal) to 0.5. Then, for each $f_m(x)$ we just need to compute a threshold value and a polarity to take a binary decision, selecting that one that minimizes the error based on the assigned weights.

In Algorithm 1, we show the *testing* of the final decision function $F(x) = \sum_1^M c_f f_m(x)$ using the Discrete Adaboost algorithm with Decision Stump "weak classifier". Each Decision Stump f_m fits a threshold T_m and a polarity P_m over the selected m-th feature. In testing time, x^m corresponds to the value of the feature selected by $f_m(x)$ on a test sample x. Note that c_m value is subtracted from $F(x)$ if the hypothesis $f_m(x)$ is not satisfied on the test sample. Otherwise, positive values of c_m are accumulated. Finally decision on x is obtained by $\text{sign}(F(x))$.

We propose to define a new and equivalent representation of c_m and $|x|$ that facilitate the parallelization of the testing. We define the matrix $V_{f_m(x)}$ of size $3 \times (|x| \cdot M)$, where $|x|$ corresponds to the dimensionality of the feature space. First row of $V_{f_m(x)}$ codifies the values c_m for the corresponding features that have been considered during training. In this sense, each position i of the first row of $V_{f_m(x)}$ contains the value c_m for the feature $mod(i,|x|)$ if $mod(i,|x|) \neq 0$ or $|x|$, otherwise. The next value of c_m for that feature is found in position $i + |x|$. The positions corresponding to features not considered during training are set to zero. The second and third rows of $V_{f_m(x)}$ for column i contains the values of P_m and T_m for the corresponding Decision Stump. Thus, each "weak classifier" is codified in a channel of a 1D-Texture, respectively.

As our main goal is to have real time testing, we deal with two main possibilities: a GLSL-programmed method on the fragment shader and an OpenCL/CUDA implementation using the OpenCL/CUDA-GL integration. We choose OpenCL for portability reasons. Using GLSL, the gradient calculation habitually is computed on CPU because it's faster. Then the results are send to the GPU as shown in Figure 1. The testing and visualization stages can be computed into the fragment shader to obtain good speedups. Through OpenCL, in contrast to GLSL, we can control almost all the hardware so we can solve the gradient problem faster in the GPU using an adaptation of the Micikevicius algorithm (Micikevicius, 2009). Thus, the gradient calculation and the classification steps can be computed into the GPU reducing the PCIe transfers and computing each step faster. In the OpenGL side we use a 3D Texture Map to visualize the models. The integration of OpenCL and OpenGL allows to avoid sending the OpenCL labeled voxel model back to the Host and visualize it directly. The OpenGL layer loads the data to the graphics card and then OpenCL obtains the ownership of the data from the global memory, processes it and returns ownership to OpenGL when finished.

3 GPGPU IMPLEMENTATION: INTRODUCING WORK GROUP SHARING

As shown in Figure 1, we propose two OpenCL kernels: the gradient and the classification or testing kernel.

Algorithm 1: Discrete Adaboost testing algorithm.

```
1: Given a test sample $x$
2: $F(x) = 0$
3: Repeat for $m = 1,2,..,M$:
   (a) $F(x) = F(x) + c_m(P_m \cdot x^m < P_m \cdot T_m)$;
4: Output $\text{sign}(F(x))$
```

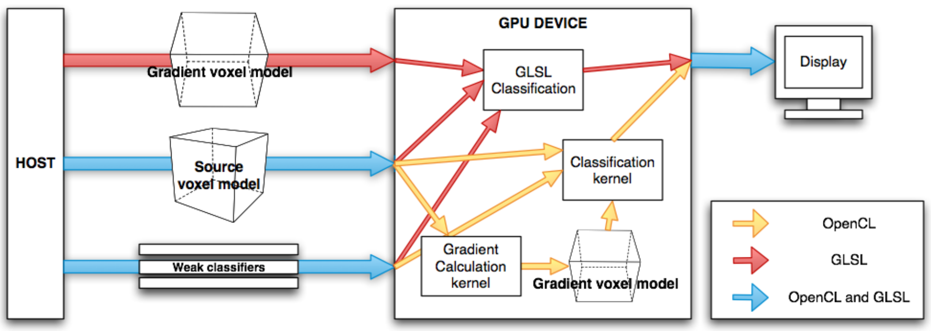


Figure 1: GPGPU implementation overview: GLSL and OpenCL approaches.

Next, we overview our proposed OpenCL classification kernel algorithm. The eight features considered for each sample by our binary classifier are: the spatial location (x, y,z), the sampled value (v), and its associated gradient value and magnitude (gx, gy, gz, $|g|$). Our binary classifier has a total of N possible c_m values, with $N = 3 \cdot M$. We create a matrix of Work-Groups (WG) that covers the x and y dimensions of the dataset, whereas the component z is computed in a loop. Each WG classifies one voxel. Inside each WG, we define $N \cdot 8$ threads or WorkItems (WI) where N is a multiple of two. Each WI computes a single step with the three weights weak classifiers and produces a value. These $N \cdot 8$ values will be reduced at the end of the execution. This process parallelizes the step 3 of the Discrete Adaboost testing algorithm defined in Algorithm 1. Finally, the sign of this computed value ($sign(F(x))$) is used to obtain the label of the processed voxels.

The way we are using threads and Global Memory transfers follows what we call Work Group Sharing (WGS), a short form of Work Group global memory transfer sharing. Our WGS method is characterized by:

- Counter intuitive global memory use. A work group reads minimum global memory data and produces the result for a single voxel. Classifying different voxels allows the work group to read at maximum global memory bandwidth. It is as to say that several work groups share a single global memory transaction, but in fact we are using only one WG.
- To process n voxels we can use 240 threads serializing n steps instead of using n threads serializing 240 steps each one. That gives a greater number of threads and so forth better performance (latency hiding) and scalability.
- Local memory gets alleviated. We store n half voxels instead of 240 for each workgroup.

In summary, finer grain parallelization, more local memory and more registers available allow to extra tune the code for faster execution.

4 SIMULATIONS AND RESULTS

In order to present the results, first, we define the data, methods, hardware platform, and validation protocol.

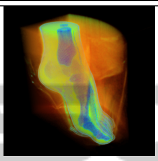
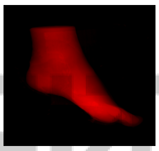
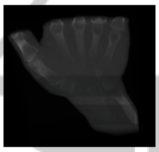
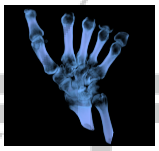
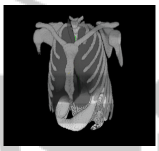
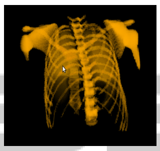
- **Data.** We used three datasets: the *Thorax* data set represents a phantom human body; *Foot* and *Hand* are CT scan of a human foot and a human hand, respectively.
- **Methods.** We use a Discrete Adaboost classifier with 30 Decision Stumps and codified the testing classifier in Matlab, C++, OpenMP, GSGL, and OpenCL codes.
- **Hardware Platform.** We used a Pentium Dual Core 3.2 GHz with 3GB of RAM and equipped with a NVIDIA Geforce 8800 GTX with 1 GB of memory running a 64-bit Ubuntu Linux distribution, a PC with a quad core Phenom2 x4 955 processor with 4GB of DDR3 memory equipped with an NVDIA Geforce GTX470 with 1,28 GB of memory. The viewport size is 700×650.
- **Validation Protocol.** We compute the mean execution time from 500 code runs. For accuracy analysis, we performed stratified ten-fold cross-validation.

The **classification performance** of the Adaboost-GPU classifier on each individual dataset is analyzed in Table 1. We defined different binary classification problems of different complexity for the three medical volume datasets. Last column of the table shows the number of *weak classifiers* required by the classifier in order to achieve the corresponding performance. For the different binary problems we achieve performances between 80% and 100% of accuracy. These performances depend on the feature space and

Table 1: Testing step times in seconds of the different datasets. The different labellings to learn increases the number of weak classifiers needed to test them. Testing times has been obtained running our OpenCL implementation on a GTX470 graphic card.

Dataset	Size	Features	Weak classifiers	Accuracy	Learning	Testing (GPU)
Foot	128x128x128	Bones and Soft tissue	1	99.95%	2.3s	0.0461s
Foot	128x128x128	Finger's bone	8	99.89%	11.45s	0.1567s
Foot	128x128x128	Ankle's muscle	7	99.21%	10.01s	0.1611s
Thorax	400x400x400	Vertebra and Column	3	99.01	3.2s	0.7157s
Thorax	400x400x400	Bone and lungs	30	84.15%	33.14s	1.9253s
Thorax	400x400x400	Bone and liver	30	78.28%	32.8s	1.9154s
Hand	244x124x257	Bone	1	100%	2.8s	0.1653s

Table 2: Results and times in seconds of the integrated OpenCL GPU-based renderings in the GTX470 graphic card.

Foot		Hand		Thorax	
					
0.1256s		0.1653s		1.9253s	

its inter-class variability. Binary problems which contain classes with a higher variability of appearance require more weak classifiers in order to achieve good performance. This increment of weak classifiers also implies an additional learning time. However, the testing time of the GSGL approach basically depends on the size of the data set and on the number of weak classifiers learned in the training stage. We can conclude that there also exists a constant time in the loading of data into GPU, and that the variability in the testing times is non-significant.

In Table 2, we analyze the **testing performance** for the different CPU-GPU implementations and hardware. First of all, we have compared the time performance of our GPU parallelized testing step in relation to the CPU-based implementations and the GLSL approach. We show the averaged times of the five implementations with the different sized datasets. Our proposed OpenCL-based optimization has a speed up of 89.91x over a C++ CPU-based algorithm and a speed up of 8.01x over the GLSL GPU-based algorithm. Finally, Table 3 shows the visualization of the three datasets and the corresponding timings of their visualizations, with the integrated in the rendering pipeline.

5 CONCLUSIONS

In this paper, we presented an alternative approach in medical classification that allows a new representation of the Adaboost binary classifier. We also defined a new GPU-based parallelized Adaboost testing stage using a OpenCL implementation integrated to the rendering pipeline. We used state-of-the-art features for training and testing different datasets. The numerical experiments based on large available data sets and the performed comparisons with CPU-implementations show promising results.

Table 3: Testing step times in seconds of the different datasets with the five implementations. GLSL and OpenCL times has been obtained using the GTX470 graphic card.

Dataset	Size	Matlab	CPU	OMP	GLSL	OpenCL
Foot	128x128x128	18.32s	9.63s	8s	1.32s	0.12s
Hand	244x124x257	67.29s	26s	20s.	2.86s	0.16s
Thorax	400x400x400	114.28s	33.76s	25s	4.41s	1.92s

ACKNOWLEDGEMENTS

This work has been partially funded by the projects TIN2008-02903, TIN2009-14404-C02, CONSOLIDER INGENIO CSD 2007-00018, by the research centers CREB of the UPC and the IBEC and under the grant SGR-2009-362 of the Generalitat de Catalunya, and the CASE and Computer Science departments of Barcelona Supercomputing Center.

REFERENCES

Cerquides, J., Lpez-Snchez, M., Ontan, S., Puertas, E., Puig, A., Pujol, O., and Tost, D. (2006). Classification algorithms for biomedical volume datasets. *LNAI 4177 Springer*, pages 143–152.

Friedman, J., Hastie, T., and Tibshirani, R. (1998). Additive logistic regression: a statistical view of boosting. In *The annals of statistics*, volume 38, pages 337–374.

Micikevicius, P. (2009). 3d finite difference computation on gpus using cuda. In *General Purpose Processing on Graphics Processing Units*, GPGPU-2, pages 79–84, New York, NY, USA. ACM.