FLOG : graphics hardware accelerated image processing library

Rosario De Chiara Ugo Erra Vittorio Scarano
ISISlab
Dipartimento di Informatica ed Applicazioni “R.M. Capocelli”
Università di Salerno
84081, Baronissi (Salerno), Italy
{dechiara,ugoerr,vitsca}@dia.unisa.it

***Abstract*— The search for realism at high frame rate leads to the development of programmable hardware graphics, with a powerful flow processor and large memory. In this paper we present FLOG (Filter Library on GPU) which is an example of general purpose computation boosted up by GPU. FLOG is an API designed to easily implement image filters on graphics hardware: from a combination of operations that manipulates images as operands FLOG generates an optimized Cg source code that will run on graphics hardware, executing desired operations.**

I. INTRODUCTION

Graphics hardware is a fascinating new field of research that gloves new lights on traditional number crunching computation The search for realism at high frame rates leads to the development of programmable hardware graphics, with a powerful flow processor and large memory. One of the milestone in its development is the capability of running programs written in a particular assembly code on its processor [1]. The importance of this evolution has been emphasized by the name GPU (Graphics Processing Unit) given to this processor, in fact similarly to the CPU, it has general purpose instructions plus a set of specialized instructions designed to accomplish, in a unique step, quite complex operations widely used in graphics (e.g. interpolations, complex texture mapping etc…). Another meaningful advantage of using the GPU is the availability of parallel computations for free, indeed the GPU executes the programs on a multi-pipeline architecture implementing an efficient SIMD (Single Instructions Multiple Data) paradigm of parallel computation. Due to its parallel nature, GPU is probably doomed to outperform CPU by an increasing margin [2], [3].

Of course all this power available for free in consumer graphics board must hide some traps that make challenging its use in general purpose computations instead of 3d graphics rendering for which it has been designed for. One of the main trap is the absence of quite useful programming paradigms like variable length cycles (that is, cycles must be unrolled). Actually this limitation appear to be only temporary, in fact there are rumors from the manufacturers about a stack implemented on GPU. Another limitation intrinsic in the use of the GPU are the slow accesses for reading from its memory, that is the passage [CPU memory] $\mapsto$ [GPU memory] is fast while the passage [GPU memory] $\mapsto$ [CPU memory] is slow. This can be explained considering the “one way” architecture of the bus (AGP) connecting the graphics board to the CPU, in fact the reading from the GPU is accomplished reading directly from the frame buffer that is the last stage of the rendering pipeline, this is the reason because the passage [GPU memory] $\mapsto$ [CPU memory] is usually avoided [4].

Writing general purpose algorithms on highly specialized hardware like GPU is not an easy task, it is fundamental to understand that no matter the similarity between the GPU assembly and a generic CPU assembly, the design of the programs is not intuitive at all; for example, it is necessary to map computational concepts like an input or an output to concept like texture and frame buffer respectively. And not always this mapping is obvious and often it has to face strict hardware limits. Some attempts have been tried to make this development easier: for example BrookGPU is a general purpose stream language designed to run arithmetic intensive, parallel algorithms [5].

II. FLOG : FILTER LIBRARY ON GPU

There are several ways of programming GPU running through raw assembly code to some ANSI C inspired language. In FLOG the Cg (C for graphics) language is used. Cg has been designed by *n*VIDIA and support different profile to best adhere to the various hardware available from different manufacturer [6]. FLOG is framework designed to enable user to easily design complex image filters able to be executed on GPU without considering the tricky creation of Cg source code: an efficient Cg source code is automatically generated by FLOG that also manages the binding of parameters. In figure 1 is showed the architecture.

The core of FLOG is the concept of operand, an operand can be of three types: an image buffer called `Texture`, a `Constant` (both matrices and floats) and a `Function` that is a snippet of Cg code. In general operands can be mixed up with meaningful algebraic operation implementing the logic of the filter, all operations are wrapped up in a suitable data type: `Filter`. The `Filter` data types has two important methods: `begin()` and `end()`, every operation that involves Operand after a `begin()` statement is tracked down, and on the `end()` statement the Cg source code is generated. At this point the source code can be executed with the method

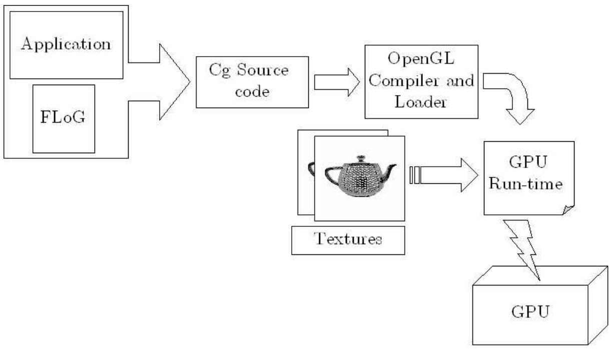

Fig. 1. FLoG architecture.

`run()` that load the Cg source code, compile it using the Cg compiler and runs it on the GPU. The binding of parameters for the filter is made easy by the availability for every texture operand of a `set(void *)` method that fills up the image buffer of the texture.

In FLOG the semantic of filter design is directly borrowed by the common mathematical expressions semantic. As an example we provide the Sobel's filter [7]:

```
::Filter f;
::Texture A, B, I;
::Function horDer, verDer;

f.declareOperand(I);

f.declareOpearand(horDer);
f.declareOpearand(verDer);

f.begin();
 A = horDer.applyOn(1, I.returnSampler());
 B = verDer.applyOn(1, I.returnSampler());
 f.out = Texture::sqrt(A * A + B * B);
f.end();
```

In this example `A` and `B` are simple temporary values so they are not declared as filter operand, this will allow to generate simpler code.

```
void main(
 in float2 texCoord : TEXCOORD0,
 out float4 color : COLOR,
 uniform samplerRECT texture0) {

 // Hard work
 texSample1 = applyhorDer((texCoord) , (texture0) );
 texSample2 = applyverDer((texCoord) , (texture0) );

 // Color output at the end
 color =
  sqrt(texSample1 * texSample1 + texSample2 * texSample2);
}
```

A common solution for multi-pass computation using GPU is the use of pBuffer that is a specialized buffer in which the GPU render the image in the last stage of pipeline. The use of pBuffer technique is avoided in FLOG using temporary variables allowing to keep the calculation *inside the GPU*.

The use of pBuffer is also avoided in the common situation of compositing two or more kernel based (convolution) filters: `Sobel(Gauss(A))`. In case of using the pBuffer the partial result of `Gauss(A)` is kept in pBuffer and is used as input for `Sobel()`. Using FLOG the combination of the two convolution filters is accomplished in a unique step recursively calculating the filters. This process is automatized. Early results say that this technique is limited by the hardware because the recursion implies the use of a lot of registries. This limitation is probably doomed to be overcame by the introduction of a stack [8].

A. Implementation details

The target hardware on which the development has been carried out is *n*VIDIA FX 5800 with NV30 processor. This is a good choice for FLOG because it provides a larger number of instructions for fragment programs, registers and texture accesses respect on other graphics hardware manufacturers. Another important feature of Cg shading language (version 1.2) is that it is more reliable than the standard GLSlang available for OpenGL that is not yet available.

III. Conclusions

We summarize here the advantages that FLOG provides:

- Ease of designing filters using common mathematical operators like arithmetics operator (`+`, `-`, `*`,...), functions (`sqrt`, `log`, ...). Their behavior is kept intuitive.
- Let the user to focus on the logic of the filter hiding the tricky part of parameters binding, context switching and all the paraphernalia needed to use Cg in a way it has not designed for. All this work has the goal of making FLOG of general use for those interested in using the GPU for image filtering without referring to some single purpose solutions ([9], [10]).
- Some heuristics have been implemented in FLOG : for example, an heuristic to efficiently implement the compositing of two or more stages of a filter is used to avoid the use of pBuffer. This solution is bounded by the current number of registers available. Particular attention is paid in minimizing the accesses to texture and the number of instructions.

FLOG is a step toward the real time image processing exploiting consumer graphics hardware. For its nature it is suitable to be used in field where keeping the CPU free for other tasks is paramount (e.g. sonification, pre/post-processing of digital video).

References

[1] M. Macedonia, "The GPU enters computings mainstream," *Computer*, vol. 36, no. 10, pp. 106–108, 2003.

[2] Semiconductor Industry Association, "International technology roadmap for semiconductors," Dec. 2002.

[3] B. Khailany, W. J. Dally, S. Rixner, U. J. Kapasi, P. Mattson, J. Namkoong, J. D. Owens, B. Towles, and A. Chang, "Imagine: Media processing with streams," *IEEE Micro*, pp. 35–46, 2001. [Online]. Available: http://cva.stanford.edu/publications/imagine-ieeemicro/

[4] T. J. Purcell, C. Donner, M. Cammarano, H. W. Jensen, and P. Hanrahan, "Photon mapping on programmable graphics hardware," in *Graphics Hardware 2003*, 2003.

[5] "BrookGPU." [Online]. Available: http://graphics.stanford.edu/projects/brookgpu

[6] R. Fernando and M. J. Kilgard, *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[7] K. R. Castleman, *Digital image processing*. Prentice Hall Press, 1996.

[8] S. Green, "NVIDIA OpenGL extensions new," in *Game Developers Conference*, 2004.

[9] K. Moreland and E. Angel, "The FFT on a GPU," in *Graphics hardware*. Eurographics Association, 2003, pp. 112–119.

[10] F.Colantoni, N.Boukala, and J. Rugna, "Fast and accurate color image processing using 3d graphics cards," in *8th International Fall Workshop: Vision Modeling and Visualization*, Nov. 2003.