# MIGRATING REAL-TIME DEPTH IMAGE-BASED RENDERING FROM TRADITIONAL TO NEXT-GEN GPGPU

*Sammy Rogmans* [*,†] , *Maarten Dumont* [†], *Gauthier Lafruit* [*], *and Philippe Bekaert* [†]

[*] Multimedia Group, IMEC
Kapeldreef 75, 3001 Leuven, Belgium
[†] Hasselt University – tUL – IBBT, Expertise centre for Digital Media
Wetenschapspark 2, 3590 Diepenbeek, Belgium

## ABSTRACT

This paper focuses on the current revolution in using the GPU for general-purpose computations (GPGPU), and how to maximally exploit its powerful resources. Recently, the advent of next-generation GPGPU replaced the traditional way of exploiting the graphics hardware. We have migrated real-time depth image-based rendering – for use in contemporary 3DTV technology – and noticed however that using both GPGPU paradigms leads to a higher performance than non-hybrid implementations. Using this paper, we want to sensitize other researchers to reconsider before migrating their implementation completely, and use our practical migration rules to achieve maximum performance with minimal effort.

***Index Terms*—** traditional, next-generation, GPGPU, migrating, real-time, image-based rendering

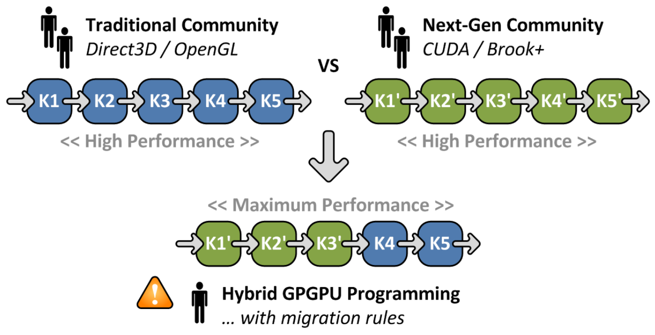

**Fig. 1**. High performance is achieved with both traditional and next-gen GPGPU, however proper hybrid programming is superior.

## 1. INTRODUCTION

For many years, people have been exploiting the powerful data parallel resources of graphics hardware for general purpose computations – a phenomenon consistently known as GPGPU. Previously, the only way to harness these computational resources, was the *traditional* approach by exploiting the graphics pipeline through the Direct3D or OpenGL API and their respective shader code. Since these APIs expose the GPU hardware in its true form – a graphics processor, it is quite cumbersome to efficiently port an application to this architecture. However, much research has been conducted in this domain, which has led to a rich and extensive amount of related knowledge and expertise [1]. More recently thanks to the GPGPU proliferation, the *next-generation* APIs CUDA and Brook+ were released to provide a means to exploit the powerful graphics processors more flexibly with a minimal learning curve requirement. These next-gen APIs should be seen as an alternative way to expose the GPU hardware, more specifically as a true generic massive data parallel coprocessor. This induces a programming style in a more familiar way to CPU programming, nonetheless, fully optimizing an application still remains expert work and low-level thinking [2]. Researchers and developers therefore either stick to traditional GPGPU to be able to reuse the existing expertise, or they are dedicated to use next-gen GPGPU for the flexibility and ease of use.

As depicted in Fig. 1, we propose to retain the valuable knowledge of traditional GPGPU, and avoid needless optimization effort in the next-gen paradigm, using the best of both worlds. To provide a case study to validate our claims, we have migrated a real-time image-based rendering (IBR) framework [3] from traditional to next-gen GPGPU, as it is the main driver behind contemporary 3DTV technology. We demonstrate how parts of the application are best kept in the traditional paradigm, because they have a more intrinsic graphics nature. As a result, the overall hybrid application performance is anticipated higher than when compared to a non-hybrid implementation. Other researchers and developers can therefore use our practical migration rules to achieve maximum end-to-end application performance with minimal effort.

This work is supported by the IWT within the Hasselt University under grant number SB071150.

## 2. DIFFERENT GPGPU PARADIGMS

The next-generation GPGPU paradigm using CUDA or Brook+, has sprouted out of necessity due to the constraints of the traditional one (i.e. using Direct3D or OpenGL). Since the traditional paradigm exposes the GPU hardware as a pipeline of four-component vector processors and raster operators with Z-testing hardware, it is purely graphics minded. The main difference with the next-gen paradigm, is that the latter abstracts the GPU as a generic coprocessor with the possibility of random writes. Moreover, it uses a user-managed cache on-chip to provide a means for explicit data transfer control between the video memory (VRAM) and the GPU chip. However, due to the great level of abstraction, the paradigm is unable to expose all available hardware in the GPU, mainly the Z-testing.

### 2.1. Traditional GPGPU

In the traditional paradigm, the four-component vector processors are exploited for GPGPU. Although primarily the fragment proces-

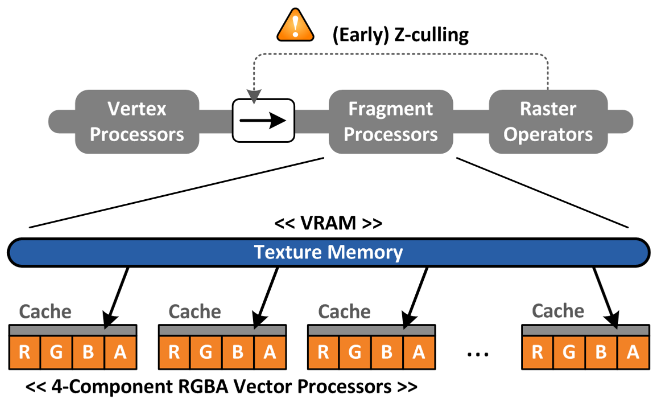

**Fig. 2**. The traditional paradigm exposes the graphics hardware as a pipeline with four-component vector processors.

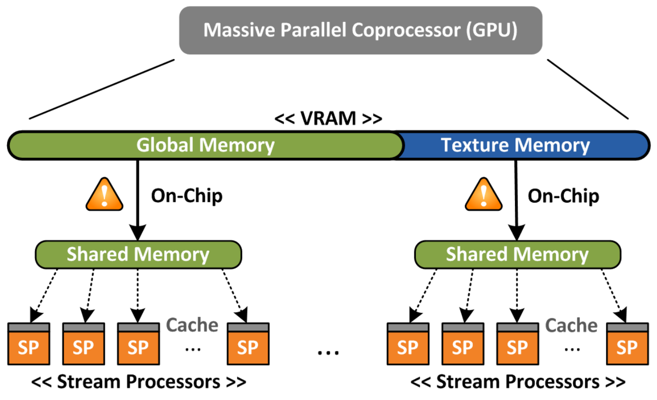

**Fig. 3**. The next-generation paradigm exposes the graphics hardware as a generic coprocessor, using a distributed-shared memory model.

sors are used (see Fig. 2), many researchers also exploit the other available resources for maximum performance [1, 4]. More specifically, *Early-Z culling* is often used to apply computational masks, resulting in executing only a sparse set of threads in an efficient way. Furthermore, threads can only read (i.e. texture lookups) from the VRAM, while the data caching is completely automatic and out of control of the programmer. The write location of the output is moreover determined in advance, and therefore offers minimal flexibility considering general-purpose computations.

### 2.2. Next-Generation GPGPU

The next-gen paradigm offers random writes and flexibility by abstracting the GPU as a generic coprocessor that exists out of multiple multiprocessors. Each multiprocessor contains an equal number of stream (scalar) processors and an on-chip shared memory, following a distributed-shared memory model (see Fig. 3). The shared memory therefore acts a user-managed cache to control the data transfers from global or texture memory inside the VRAM to the GPU.

The joint execution model uses blocks of threads inside a grid to execute individual blocks on a dedicated multiprocessor, while an (idle) multiprocessor can swap to different thread blocks. Since the next-generation paradigm exposes the graphics hardware in a more generic and familiar way, it is hereby not able to expose all available hardware contained inside the GPU.

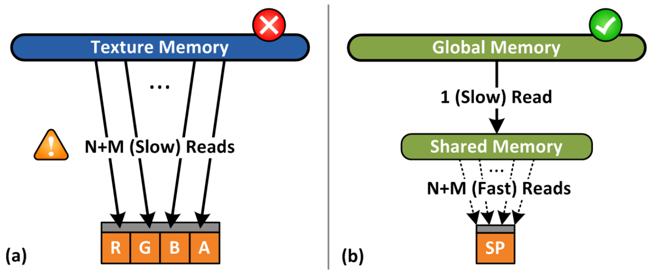

**Fig. 5**. Performing a convolution in (a) the traditional, and (b) the next-gen GPGPU paradigm.

## 3. MIGRATING IMAGE-BASED RENDERING

We have systematically migrated a real-time depth image-based rendering (DIBR) framework [3] from its traditional paradigm to next-gen GPGPU. Real-time IBR is one of the most important drivers behind contemporary 3D technologies such as autostereoscopic displays and 3DTV, hence it has a valuable research interest. Following the taxonomy of Scharstein et al. [5] and its DIBR extension of Rogmans et al. [4], the application exists out of five elementary kernels – the *cost computation*, the *cost aggregation*, the *disparity selection*, the *image warping*, and the *occlusion handling*. As depicted in Fig. 4, in the highest level of abstraction this can be seen as creating a depth map using stereo matching, and synthesizing an intermediate view using the acquired depth map.

Since the cost aggregation kernel is in its very essence a separable two dimensional convolution, it can be noticed that this kernel severely stresses the GPU hardware when used through the traditional paradigm. This is due to the large amount of data-reuse and high-speed memory transfer requirement, which are typical to any convolution. However, all other kernels map quite well to the architecture as it is exposed via traditional GPGPU.

When the kernels are one-by-one migrated to a next-generation implementation, they are noticed to map properly to the exposed architecture, thanks to its intrinsic flexibility and generality. Nevertheless, the occlusion handling kernel suddenly generates a non-trivial amount of overhead, as there is no intrinsic support for sparse computational masks within the next-gen paradigm. For this reason, it is important to evaluate all kernels individually using both GPGPU paradigms, such that initial migration rules can be composed to avoid needless effort in migrating an existing implementation.

### 3.1. Cost Computation

For a given intermediate position $s$, e.g. the center view $s = 0.5$, the cost computation takes in a stereo image pair $\mathbf{I}_0$ and $\mathbf{I}_1$, and generates a cost $C(x, y, d)$ for each disparity hypothesis $d$ out of a given search range $S$ (see Eqn. 1). The cost is consequently truncated to a maximum $\tau$, to avoid discrepancies due to noise (see Eqn. 2).

$$C(x,y,d) = |\mathbf{I}_0\,(x+sd,y) - \mathbf{I}_1\,(x-(1-s)d,y)| \tag{1}$$

$$C_\tau(x,y,d) = \min\left[C\,(x,y,d)\,,\tau\right] \tag{2}$$

In the traditional paradigm, proper care should be taken to pack four disparity hypotheses together, in order to fully utilize the four-component vector processors. If implemented like this, there will not be a large performance difference between a port to next-generation GPGPU, as both approaches can efficiently handle the presented

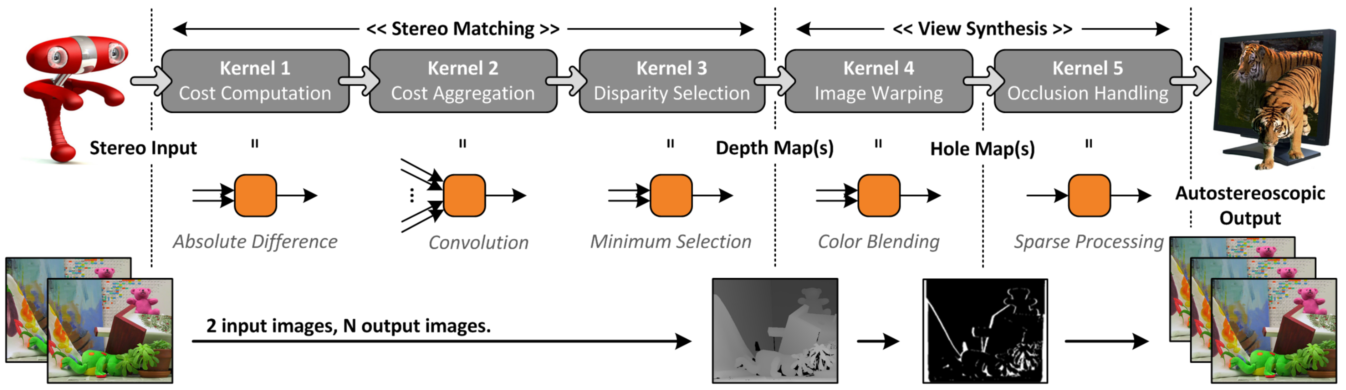

**Fig. 4**. The application chain of depth image-based rendering takes in 2 input images, and outputs a number of requested intermediate images.

computations. However, using the next-gen paradigm will not require four-component data packing, resulting in more flexibility.

## 3.2. Cost Aggregation

Consequently to the cost computation, the reliability of the cost is improved by aggregating over an $N \times M$ sized window. By applying a (separable) convolution kernel $w(u, v)$, the confident aggregated cost $A(x, y, d)$ is obtained using Eqn. 3.

$$A(x, y, d) = w(u, v) *_u *_v C_\tau (x + u, y + v, d) \tag{3}$$

Performing a convolution (most definitely a large kernel size) induces a lot of data-reuse, and its performance is therefore highly dependent of the memory transfer speed. This stresses the GPU hardware as it is exposed in traditional GPGPU, considering that memory reads (i.e. texture lookups) between the VRAM and the processors inside the GPU chip are quite slow in terms of clock cycles. In general, $N+M$ slow reads are necessary for each pixel or thread, to perform the convolution (see Fig. 5a). However, as depicted in Fig. 5b, porting this implementation to the next-generation paradigm is very beneficial for its performance. The shared memory can be used as a user-managed cache to reuse the required data straight out of this on-chip memory [6]. In other words, the convolution can be performed with a single slow read per thread, since all other data is loaded inside the shared memory by the neighbouring threads.

## 3.3. Disparity Selection

After obtaining confident costs for the different disparity hypotheses $d$ in the search range $S$, the minimum cost indicates the best possible match found. The corresponding disparity is directly correlated to the depth information, and is therefore consequently stored in a depth map $D(x, y)$, according to Eqn. 4.

$$D(x, y) = \arg\min_{d \in S} A(x, y, d) \tag{4}$$

In traditional GPGPU, the raster operators – more specifically the Z-test – is often used to automatically select and save the minimum value. This provides a means to integrate the disparity selection into the same execution kernel as the cost aggregation. This is unfortunately not the case in the next-generation paradigm, as the generality requires to manually check for the minimum, resulting in a reduced performance of the individual kernel. However, in practice, the optimal end-to-end performance should always be kept as the primary goal of the individual GPGPU paradigm selection.

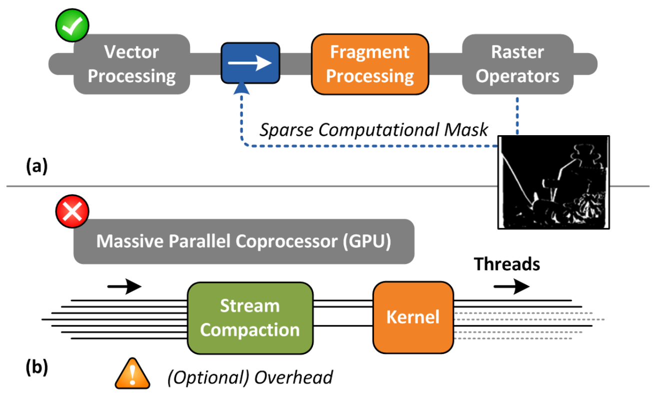

**Fig. 6**. Performing occlusion handling in (a) the traditional, and (b) the next-gen GPGPU paradigm.

## 3.4. Image Warping

After obtaining a depth map, the intermediate view $\mathbf{I}_s$ can be synthesized by blending the sampled colors of the two input images together, using the correct parallax from $D(x, y)$ (see Eqn. 5–6).

$$\mathbf{I}_s(x, y) = (1 - s)\mathbf{I}_0(x_L, y) + s\mathbf{I}_1(x_R, y) \tag{5}$$

$$x_L = x + sD(x, y), \; x_R = x - (1 - s)D(x, y) \tag{6}$$

Although a color-blend is very graphics-minded, both traditional and next-generation GPGPU will perform equally well, because the arithmetic logical units (ALUs) inside the GPU processors intrinsicly have good support for these kind of operations.

## 3.5. Occlusion Handling

During the image warping, a check is performed whether the two colors are close to identical. If not, this indicates an occluded area (i.e. the concerning pixel is only visible in one of the two input images), and the blend is therefore aborted. These (sparse) occlusions are consequently handled as a final step in the IBR process.

By exploiting the Early-Z mechanism available in the traditional paradigm, threads can be efficiently rejected, causing the occlusion handling to only run on the necessary pixels (see Fig. 6a). The Early-Z mechanism can therefore be seen as intrinsic support for sparse computational masks [7]. However, using next-gen GPGPU forces a

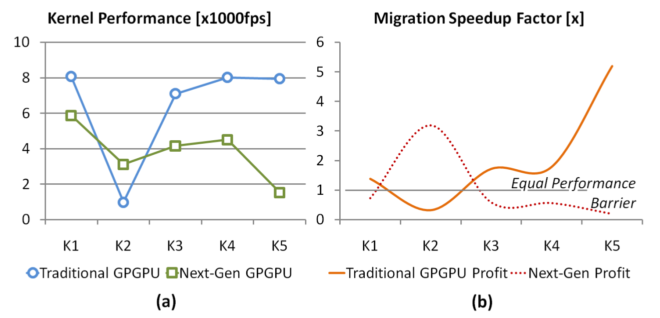

**Fig. 7**. (a) Individual kernel performance using both traditional and next-gen GPGPU, and (b) the profit by using either paradigm.

programmer to manually reject the appropriate threads. This can be implemented by thread conditionals, or by use of stream compaction to collect all threads that need execution [8], as depicted in Fig. 6b. Nonetheless, this introduces a significant overhead that is avoided by exposing the GPU hardware as the traditional pipeline.

## 4. EXPERIMENTAL RESULTS

We evaluated the cost computation (K1), the cost aggregation (K2), the disparity selection (K3), the image warping (K4), and the occlusion handling (K5) implemented in both the traditional and the next-generation paradigm. The traditional implementation uses Microsoft Direct3D, while the next-generation implementation uses the Compute Unified Device Architecture (CUDA) from NVIDIA. The performance was benchmarked on an Intel Xeon 2.8GHz, housing an NVIDIA GeForce 8800GTX with 512MB video memory.

### 4.1. Performance Evaluation

Fig. 7a depicts the benchmarks of each individual kernel using either traditional or next-gen GPGPU. The framerate of K1–3 was measured while performing four disparity hypotheses at once, to achieve a fair comparison between both paradigms. The view synthesis kernels K4–5 generated a single intermediate view $\mathbf{I}_{0.5}$.

On an individual basis, the kernel performances are quite consistent with the presented dual architecture analysis in Section 3. However, only the cost aggregation kernel K2 is significantly faster in absolute terms, when migrated to CUDA. We suspect this performance offset to manifest due to an overhead related to the GigaThread manager, which is the core mechanism to swap between running threads on the same processor (i.e. context switches). Nonetheless, considering the profit through using a proper GPGPU paradigm (see Fig. 7b), it is clear that the cost aggregation and occlusion handling gain the most speedup in the next-gen, and traditional paradigm respectively. All other kernels are close to the equal performance barrier.

### 4.2. Practical Hybrid Programming

Although the individual migration rules would propose to make a hybrid Direct3D implementation where only the cost aggregation is migrated to CUDA, we have noticed some practical constraints that affect the end-to-end performance. As the interoperability – needed for the hybrid implementation – uses a *mapping* and *unmapping* (i.e. locking and unlocking a semaphore lock in the VRAM), the memory is iteratively locked and unlocked by iterating over the disparity search range, if only K2 is implemented in CUDA. As this introduces a significant practical overhead, The lock/unlock is best performed only once, surrounding the stereo matching chain, and forcing K1 and K3 to CUDA as well. This will lead to an optimal end-to-end performance, as kernel K1 and K3 do not greatly benefit a traditional implementation.

## 5. CONCLUSIONS

We have systematically migrated a real-time image-based rendering framework from traditional to next-gen GPGPU. When considering the related kernels individually, the results indicate that cost aggregation and occlusion handling significantly increase their performance when migrated to next-generation GPGPU, and kept in the traditional paradigm respectively. The performance difference by the other kernels are far less pronounced, and could be considered almost equal. In this extent, we have found that an optimal end-to-end performance can be achieved by migrating all stereo matching kernels to next-gen GPGPU, while keeping the subsequent kernels of the view synthesis chain in the traditional paradigm. Hence, a hybrid implementation that exploits the best of both worlds is obtained.

## 6. ACKNOWLEDGEMENTS

We want to express our sincerest gratitude to Patrice Rondao Alface, who has assisted us with migrating and benchmarking our existing kernels. Sammy Rogmans would also like to thank the IWT for its PhD specialization bursary. Furthermore, all authors acknowledge the financial support from the IBBT.

## 7. REFERENCES

[1] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. E. Lefohn, and T. J. Purcell, “A survey of general-purpose computation on graphics hardware,” *EG Computer Graphics Forum*, vol. 26, no. 1, pp. 80–113, March 2007.

[2] S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, and W.-M. W. Hwu, “Optimization principles and application performance evaluation of a multithreaded GPU using CUDA,” in *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, February 2008, pp. 73–82.

[3] S. Rogmans, J. Lu, and G. Lafruit, “A scalable end-to-end optimized real-time image-based rendering framework on graphics hardware,” in *3DTV-CON, The True Vision Capture, Transmission and Display of 3D Video*, May 2008, pp. 129–132.

[4] S. Rogmans, J. Lu, P. Bekaert, and G. Lafruit, “Real-time stereo-based view synthesis algorithms: A unified framework and evaluation on commodity GPUs,” *Signal Processing: Image Communication*, vol. 24, no. 1-2, pp. 49–64, January 2009, Special issue on advances in three-dimensional television and video.

[5] D. Scharstein and R. Szeliski, “A taxonomy and evaluation of dense two-frame stereo correspondence algorithms,” *Int’l Journal of Computer Vision*, vol. 47, no. 1-3, pp. 7–42, April 2002.

[6] V. Podlozhnyuk, “Image convolution with CUDA,” in *NVIDIA CUDA SDK Documentation*, June 2007.

[7] P. V. Sander and J. L. Mitchell J. R. Isidoro, “Computation culling with explicit early-z and dynamic flow control,” in *ACM SIGGRAPH GPU Shading Course 37*, August 2005.

[8] D. Roger, U. Assarsson, and N. Holzschuch, “Efficient stream reduction on the GPU,” in *Workshop on General Purpose Processing on Graphics Processing Units*, October 2007.