Annales UMCS Informatica AI XIII, 1 (2013) 81–87
DOI: 10.2478/v10065-012-0039-6

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI
http://www.annales.umcs.lublin.pl/

Stereoscopic video chroma key processing using NVIDIA CUDA

Jarosław Sagan[1*]

[1] *Institute of Computer Science, Maria Curie-Sklodowska University, pl. M. Curie-Sklodowskiej 5, 20-031 Lublin, Poland*

Abstract – In this paper, I use the NVIDIA CUDA technology to perform the chroma key algorithm on stereoscopic images. NVIDIA CUDA allows to process parallel algorithms on GPU. Input data are stereoscopic images with the monochromatic background and the destination background image. Output data is the combination of inputs by using the chroma key. I compare the algorithm efficiency between the GPU and CPU execution.

1 Introduction

The chroma key algorithm provides the possibility of low cost virtual video studio creation. The filmmakers use the chroma key to embed actors into virtual scene, which is often computer rendered. In this case, the filmmakers can produce many special programs for kids, weather forecast or information programs in one real studio. It reduces recording studios costs.

Filmmakers use the chroma key to record danger scenes inside safe recording studio. Many of these scenes can not be created without virtual studio because of danger and very high costs. Stereoscopic videos provide the possibility of watching real 3-Dimension videos. In the back–end the stereoscopic image is a pair of images which were made for the left and right eye separately. Because of it, to process a stereoscopic image we have to process twice more data than we use it with the 2-Dimension image.

Stereoscopic videos are created mostly by the filmmakers. Creating stereoscopic production is very popular nowadays. It is possible that 3-Dimension technology will replace the standard 2-Dimension videos in many other areas in life. The 3-Dimension screen and projectors are developed by engineers. It is possible that in the future

*jaroslaw.sagan@gmail.com

people will not be use special glasses to watch stereoscopic images. In this situation the stereoscopic technology can be used in business, education and entertainment.

2 Processing stereoscopic images

To apply any graphic algorithm on the stereoscopic images we have to process both sub images. The chroma key algorithm requires the same stereoscopic images format and images resolutions to process images as the standard 2-Dimension ones. If we have 2 incompatible input sources, we have to convert one to be compatible with the second one. Processing stereoscopic videos requires compatible frame standard and the same frame rate. If the inputs are not compatible we have to convert one to be compatible with the other one. The examplary stereoscopic chroma key processing block is shown in Fig. 1.

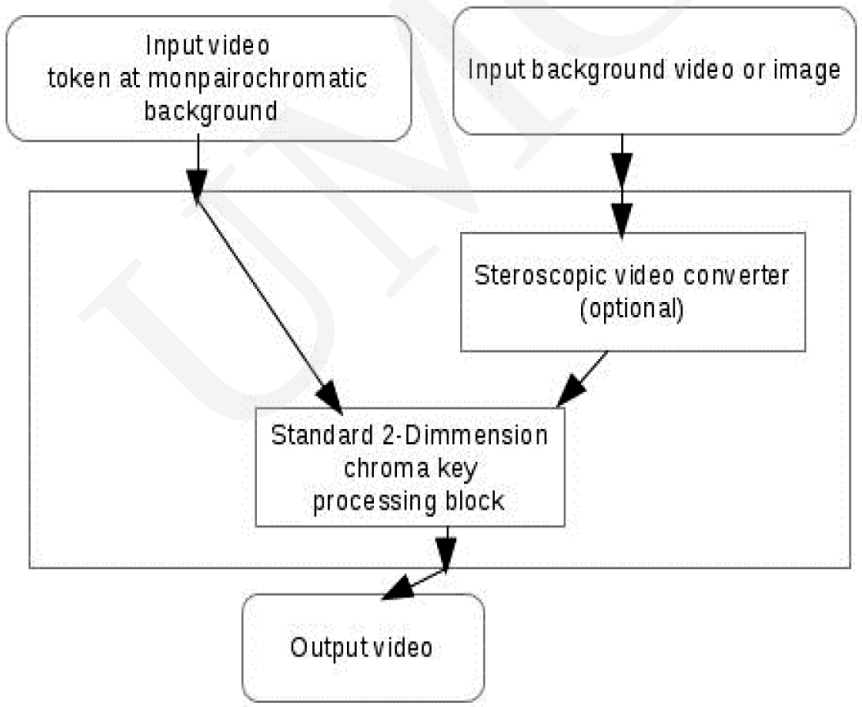

Fig. 1. Stereoscopic video processing block scheme.

3 Standard chroma-key algorithm

The standard chroma-key algorithm requires two input images: object and background and object background colour. Algorithm scans object image for pixels with

similar colour from the object background. All these pixels are replaced with the corresponding ones from the background image. The algorithm pseudo-code is presented below.

```
For(int x=0;x<objectImage.getWidth();x++)
    For(int y=0;y<objectImage.getHeight();y++)
       If(isSimilar(objectImage.getPixels()[x,y],backgroundColor)){
          objectImage.getPixels()[x,y]=backgroundImage[x,y]
}
```

The above pseudo-code writes the output image into objectImge. Because of objectImage reuses, the algorithm makes fewer assignments but we miss information abut the input objectImage.

4 3-Dimension chroma-key parallel processing

To make chroma-key processing faster we can start to process parallel. In the case of video processing, the simplest way to process the chroma-key parallel is processing frames in separated threads. In this way we have to keep buffer with the size comparable to the parallel thread count. The buffer is used to sort frames to keep the correct frame sequence. Output is delayed by defined frames count. Another way is splitting frame into parts and processing parts parallel. In this way the algorithm requires less memory than the previous option.

The chroma-key algorithm is very simple and operates on integers. The key operations are memory access, comparison, conditional jump (if). CPU is able to execute fast these operations. GPU is slower in the global memory access, which is the key operation in this algorithm. But video decoding and encoding are faster on GPU than on CPU. These operations take more time than chroma-key processing. So this is the main reason why we are going to use NVIDIA CUDA.

5 Parallel processing using NVIDIA CUDA

The processing data using NVIDIA CUDA require the multithread algorithm. The GPU device contains hundreds of cores. Sets of cores are grouped into warps. Warps execute one instruction on the multidata at the same time – SIMD architecture. Global GPU memory access is the slowest operation in the chroma-key algorithm. Nvidia Cuda technology gives programmers a technique to reduce global memory traffic – coalesced global memory access. If threads in the same warp have access to the consecutive memory bytes, the memory access will be executed in one part.

The commonly used data structure to represent images is two–dimension array of bytes. The first dimension represents lines of image. Each line contains pixel data, in this research I use four bytes per pixel (first byte – red, second byte – green, third byte – blue, fourth byte – alpha channel). To achieve the best speed for this data structure,

pixels should be processed in lines. One warp processes adjacent pixels in line. It causes that warp processes adjacent pixels and it can use coalesced global memory access. The code below presents the chroma-key kernel code.

```
__global__ void chromaKeyCuda(unsigned char* background,
                unsigned char* object, int width,
                int height, int objectWidth, int objectHeight) {

        const int y = blockIdx.y * blockDim.y + threadIdx.y;
        const int x = blockIdx.x * blockDim.x + threadIdx.x;

        if (x < objectWidth && y < objectHeight) {
                const int i = pos(x,y);

                uchar4 pixel = ((uchar4 *) object)[i];

                const int diff = pow2(pixel.x) + pow2(pixel.y) + pow2(pixel.z);

                if (diff > MAX_DIFFERENCE) {

                        int bgOffset = x + y * width;
                        ((uchar4 *) background)[bgOffset] = pixel;

                }

        }
}
```

Code 1. The CUDA chroma-key algorithm implementation.

6 Multithread chroma-key processing using CPU

CPU has different architecture from that of GPU. When we programme CPU, we have to consider other features than on GPU. CPU access to global memory is not as expensive as GPU because of cache. If program accesses a specified address in the global memory, then the processor copies data block which contains the requested value to CPU internal cache. If program process data sequentially as this data are placed in global memory it will cause efficiency benefits because data which will be processing are mostly placed in cache. Programmers should avoid multiprocessors data processing when data are placed in one global memory location because of increasing communication between cache controllers. In this case cache controllers have to communicate to keep a consistent memory snapshot. This multiprocessor processing disadvantage is reduced when we process data on a few cores of one real processor. The processor shared cache increases speed of keeping cores cache memory consistency. Another CPU

processing feature is that thread creation is expensive because of the cost of OS. If a set of operation to perform is small, the total time of processing will not be shorter than single thread processing.

According to the above features I changed the chroma-key implementation to suit the CPU features. In the chroma-key GPU implementation only one pixel is processed in one kernel execution. In the CPU implementation the thread creation cost may be higher than one pixel processing cost. In this case I have to increase instruction set size to be executed in thread. I divided the input images to parts. The parts are different from those chosen in GPU to be processed in one warp. The background image is divided by lines. One thread process: $\frac{image\,height}{threads\,count}$ subsequent lines. This implementation prevents excessive cache controllers communication. Data are processed sequentially so this implementation has an advantaget from cache. The code bellow presents the thread implementation.

```
for (int x = 0; x < objectWidth; x++)
       for (int y = yBegin; y < yEnd; y++) {
              const int i = pos(x, y);

              int diff = pow2((unsigned char) object[i * 4 ])+
                         pow2((unsigned char) object[i * 4 + 1])+
                         pow2((unsigned char) object[i * 4 + 2]);

              if (diff > MAX_DIFFERENCE) {
                    int bgOffset = x + y * width;
                    ((int *) background)[bgOffset] = ((int *) object)[i];

              }

       }
```

Code 2. The chroma-key thread main loop.

7 Research scenario

To compare algorithms implementation efficiency both on CPU and GPU I implement the algorithm for both computing platforms. Both implementations need to be optimal for their platform. Next I execute the algorithm in loop many times with timing. The research is based on walltime. Walltime timing causes an inaccuracy. The number of repetitions has to be large enough to reduce the error. The test will be executed on the research claster a couple of times to minimize an error. The resarch performed on the research cluster uses the following parameters:

Processor:	2 x Intel(R) Xeon(R) CPU X5650
Cores count:	6
Threads count:	12
GPU device:	2x Tesla S2050

I make the CPU execution test with timing. Each thread performs operations 1000 times on the same data. It simulates real implementation which processes video frames sequentially. The test implementation is faster than the potential real implementation because all data are able to load to the processor cache. If the CPU test is faster than the GPU one I have to change the research scenario. The GPU test scenario is more similar to the real implementation than the CPU test. The kernel is executed synchronously 1000 times on the same data. But in this case the data are not cached. Every time kernel accesses memory, it accesses the global GPU memory. In real implementation the kernel will access the global GPU memory for the data created by another algorithm as a video decompression algorithm. Output data are also stored in the global GPU memory so another algorithm as a video compression algorithm can access it.

The test results are presented in the char below.

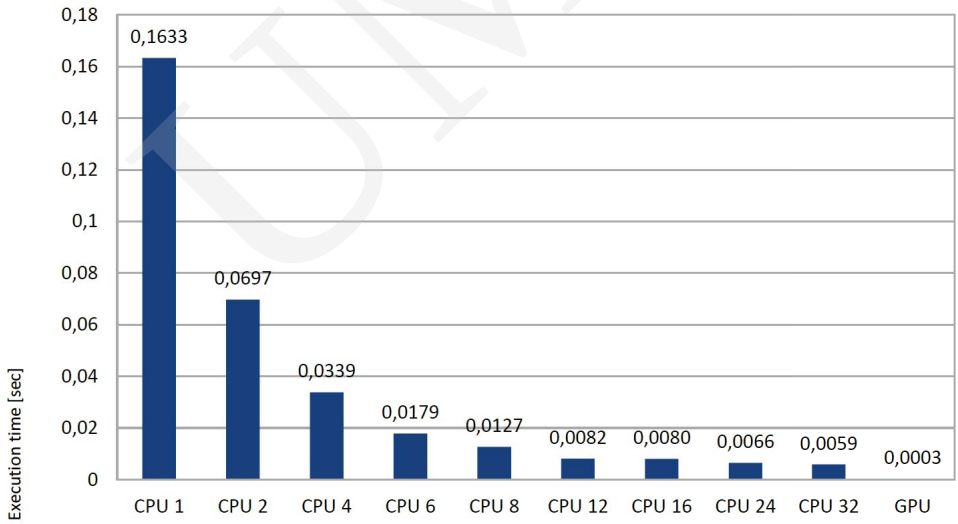

Fig. 2. Chart 1. Execution time of one chroma-key algorithm call.

This chart shows that GPU processing is faster than the fastest CPU execution. I have compared the single GPU chroma-key processing to the execution on 2 CPU. But in the case of CPU execution I take into account the time of last thread execution done. It is possible to create faster algorithm which will better schedule data parts for threads. Theoretically the time of execution of this implementation is approximated by the average thread execution time. The results of this research are presented in the

chart below. In this scenario I also take into account the time of copy data from the host to the GPU global memory card and backward.

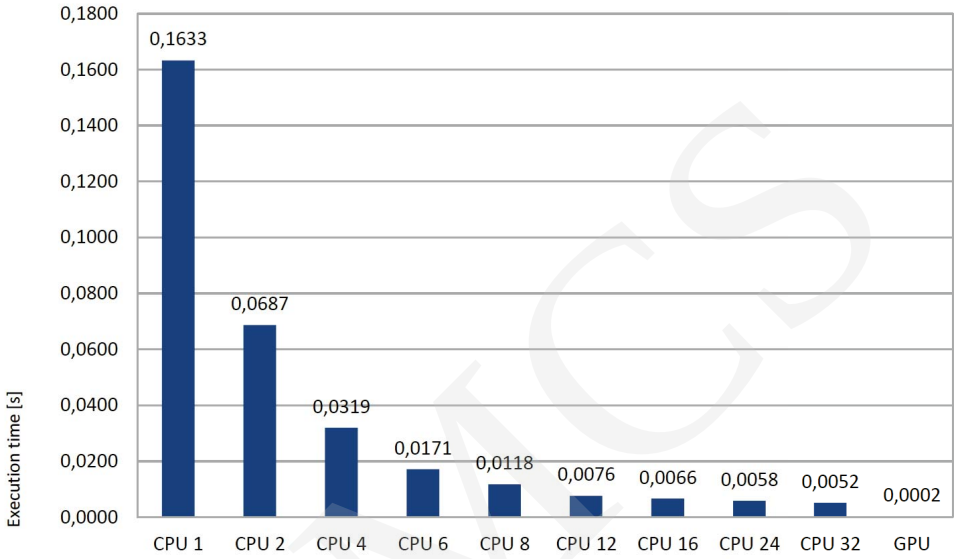

Fig. 3. Chart 2. Average thread execution time of one chroma-key algorithm execution.

The last research results are similar to the previous ones. This causes that the GPU chroma-key implementation is faster thanthat of CPU. The GPU implementation with the memory copy from the host to GPU and backward is quite slower than the execution on two CPU. The memory copy is the main cost of this way of execution. To achieve better results programmers can copy memory and execute kernel on other data than these which are currently copied. Also it is a good idea to copy the compressed video or image data to GPU and decompress it on GPU.

8 Conclusions

The chroma-key execution algorithm is commonly used in 2D and 3D videos. The research results are such that the GPU implementation executed on one GPU is about 25-times faster than theoretically the fastest CPU implementation executed on 2 CPU 6 core processors. When data are not stored on the GPU, global memory card program has to copy it from the host to GPU. The copy operation is the largest cost in this algorithm.