A Real-time Video Illustration using CUDA

JiHyung Lee[1], Yoon-Seok Choi[1], Bon-Ki Koo[1], and Chi Jung Hwang[2]

[1] Electronics and Telecommunications Research Institute,
138 Gajeong-ro, Yuseong-gu, Daejeon, 305-700, Republic of Korea
{ ijihyung, ys-choi, bkkoo}@etri.re.kr
[2] Chungnam National University, Department of Computer Science,
79 Daehangno, Yuseong-gu, Daejeon, 305-764, Republic of Korea
cjhwang@cnu.ac.kr

Abstract. According to advancements in video technology, there are lots of needs for various special effects of videos. The conventional image-transform effects could be applied to video streams, but non-photorealistic rendering effects are not easy to apply. For example, cartoon or illustration effects have expensive costs in video transformation which makes it difficult to execute in real-time. In this paper, we suggest a video transformation system with illustration effects. It is designed to apply the illustration effects to the video stream directly and is implemented to achieve real time performances using the GPU hardware with NVIDIA's CUDA.

Keywords: non-photorealistic rendering, video, illustration, real-time, CUDA

1 Introduction

Recently, videos have become quite common in a human's life. That is, videos are a very familiar and popular media. So, there are demands to make videos with unique style.

The easiest way to make a unique video is putting in special effects in it. To achieve this goal, many special effects for videos were designed in the past. The early results are derived from image transformations. The simple color conversions such as black &white and sepia toning can be easily applied to videos on a real-time through conventional applications.

However, video effects from more complex image transformations like non-photorealistic rendering effects can't be applied easily. Non-photorealistic rendering effects are a variety of effects as if a person is directly painting a picture on the simple image. For example, illustration and watercolor effects in an image takes a long time to be generated and it is similar in videos. Several studies have been made on video effects in the fields of non-photorealistic rendering, but those were simply used as small images.

This paper aims to apply non-photorealistic rendering effects to various image sizes in videos in real-time. Among non-photorealistic rendering effects, illustration effects are emphasized. For real-time performances, it is designed and implemented using the GPU hardware with NVIDIA's CUDA.

2 Previous work

Gooch et al. [1] suggests a facial illustration method. In his research, the illustration technique and the caricature system, which are put in features of an individual, are introduced. Even though he is mainly dealing with the caricature system, his illustration technique for the face description also shows positive results.
Holger [2] introduces the technique which converted images in videos into images of cartoon style on real-time. First, an image is abstracted by applying the bilateral filter repeatedly. Second, the luminance quantization and DoG edge detection technique are applied to the abstracted image. Results of the processes are combined and then the final cartoon-style image is produced. And Klein [3] suggests the video mosaic method which uses not only simple image but also video stream.

3 Video illustration

In this chapter, our real-time video illustration is described. At first, we mention basic illustration algorithm and explain modified algorithm for real-time processing. Then, we deal with the NVIDIA's CUDA code in order to implement our algorithm.

3.1 Illustration

Illustration effects are based on brightness perception of black and white images. The brightness of an object depends on the light reflected by itself and the object's background. Even if the brightness is a little different, relatively, it can be seen the same. As shown in Fig. 1, below four circles have equal intensities. While the circles in (a) seems to be the same, those in (b) seems to be different due to their backgrounds.

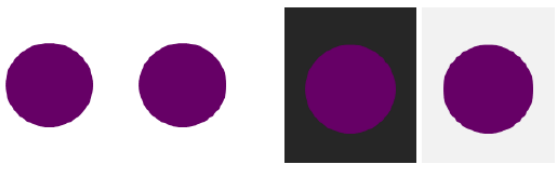


Fig. 1. Example of brightness perception

The illustration is composed of three basic operations: differentiation, integration, and threshold. First, two blurred images can be generated by applying two different sizes of Gaussian blur filters to an original image. Second, the difference between the corresponding pixels of two images is calculated and the value is integrated. The size of the Gaussian blur filter increased than the former step with 1.6-fold, and the same task is repeated several times. Finally, the last image based on an original image value, an accumulated difference value, and a pre-defined threshold value is produced. Refer to the below algorithm in Fig. 2.

```
Program Illustration(SrcImage)
{
  GrayImage = ConvertToGray(SrcImage);
  v1, v2, b : Image ;
  for s=1 to S {
    fAlpha = pow (1.6, s);
    nKernelSize = DecideKernelSize(fAlpha);
    GaussianFilterWeight= MakeGaussianFilter(nKernelSize);
    v2 = GaussianFilter(v1, GaussianFilterWeight);
    b +=  (v1 - v2) / (coeff + v1);
    v1 = Copy(v2);
  }
  FinalImage = Threshold(b, GreyImage, fThresValue);
  return FinalImage;
}
```

Fig. 2. Image illustration algorithm

3.2 Separable Gaussian Blur Filter

We divided the Gaussian Blur process into two passes to reduce computation costs. In the first pass, a one-dimensional kernel is used to blur an image in only horizontal or vertical direction. In the second pass, another one-dimensional kernel is used to blur in remaining direction. The results of those passes are the same as convolving with a two-dimensional kernel in a single pass. Equation 1 shows the basic Gaussian Blur filter and equation 2 represents the separable Gaussian Blur filter for our purpose. The separable Gaussian Blur filtering requires less computation costs. Fig. 3 shows the step images which are created in each pass.

$$F(x,y) = c^2 \sum_i \sum_j exp(-\frac{(x_i - x_a^2)}{2\sigma^2}) exp(-\frac{(y_i - y_a^2)}{2\sigma^2}) \cdot t(x_i, y_j) \quad \textbf{(1)}$$

$$I(x,y) = c \sum_i exp(-\frac{(x_i - x_a^2)}{2\sigma^2}) \cdot t(x_i, y_j)$$
$$F(x,y) = c \sum_j exp(-\frac{(y_i - y_a^2)}{2\sigma^2}) \cdot I(x_i, y_j) \quad \textbf{(2)}$$

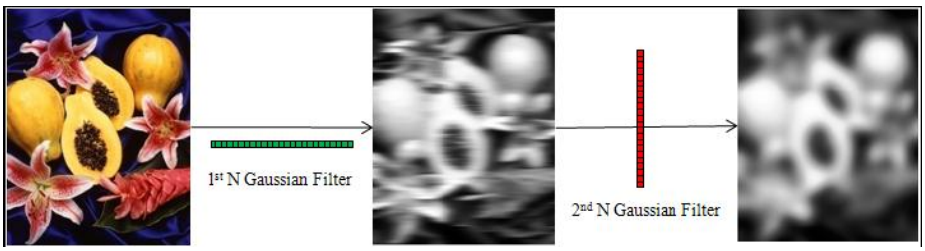


Fig. 3. Images generated by separable Gaussian Blur filter in each pass

3.3 Implementation using CUDA

We convert the image illustration algorithm by single thread using CPU to multiple threads using current GPU which is capable of parallel computation. The texture array provided by NVIDIA's CUDA is used in order to assign data at each GPU processor and to avoid data bottle neck generated in the SIMD command execution of CUDA. An image is inputted into the texture array and an intermediate value is stored in the illustration process. Also, our implementation of CUDA employs the separable Gaussian Blur filter to reduce computation costs. Below CUDA pseudo codes in Fig. 4 represents these explanations in detail.

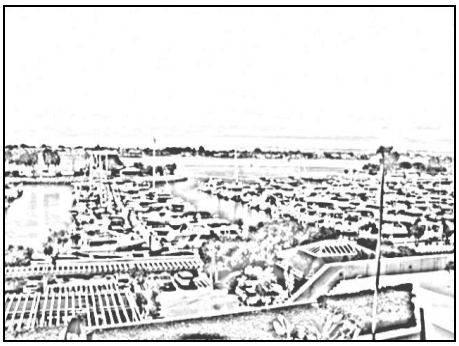
```
Program cudaComputeIllust (*pSrcData, *pDestData, nWidth, nHeight,
fAlphaScale, fThresLumi)
{
   float     *pV1, *pV2, *pV3, *pB;
   cudaArray *arrTex1, *arrTex2, *arrGuassWeight;
   dim3      threadBlock(blockSize, blockSize, 1);
   dim3      blockGrid(iDivUp(nWidth, threadBlock.x), iDivUp(nHeight,
threadBlock.y), 1);
    // Copy the data in main memory to GPU memory
    cudaMemcpy (pV1, pSrcData, nBytes, cudaMemcpyDeviceToDevice) );
    for (s = 1 ; s<= N ; s++) {
        MakeGussianKernel(s);
        // Horizontal Gaussian Blur filter
        cudaThreadGaussRow2D <<<blockGrid, threadBlock, 0>>> (pV3,
nWidth, nWidth, nHeight, nAnchorX);
        // Vertical  Gaussian Blur filter
        cudaThreadGaussColumn2D <<<blockGrid, threadBlock, 0>>> (pV2,
nWidth, nWidth, nHeight, nAnchorX);
        // Get the differences between orignal & blurred image
        cudaThreadComputeB <<<nHeight, 128 >>> (pB, nWidth, nWidth,
nHeight, fCoeff);
        // Swap original image and blurred image
        swap (pV1, pV2);
    }
   // Get the Final Image
  cudaThreadComputeFinal <<<nHeight, 128 >>> ((Pixel4 *)pDestData,
nWidth, nWidth, nHeight, fThresLumi);
}
```

Fig. 4. CUDA pseudo code for our illustration algorithm

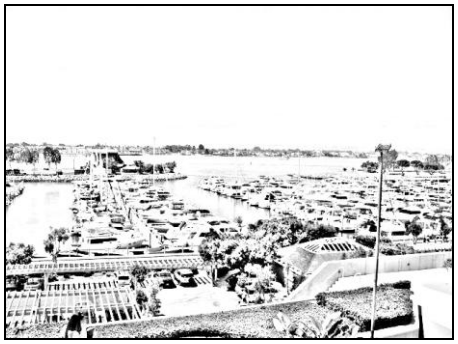
4 Experiments and Results

To find advantages of our real-time video illustration, we conduct two kinds of experiments. The first experiment is about the quality of video illustration. To check the illustration quality, comparison between result images using illustration module in Adobe Photoshop and our results. In Fig. 5, (a) is a source image, (b) is the result image of illustration by Adobe Photoshop, and (c) is the result of our illustration. Comparing the quality of (b) and (c), we can't find much difference. Therefore, when we apply illustration effects to video streams, we can obtain high image quality.

(a) Source Image



(b) Illustration by Adobe Photoshop



(c) Our Illustration Image

Fig. 5. Images generated by separable Gaussian Blur filter in each pass

The second experiment is about real-time performances. In this experiment, we implemented a variety of cases: CPU-based method using OpenCV library [5] and GPU-based method using NVIDIA CUDA. The former is divided into 2 methods according to the form of the Gaussian Blur filter. In short, there are CPU-based method using OpenCV (CPU1), and CPU-based method with the separable Gaussian Blur filter (CPU2), and NVIDIA CUDA method (CUDA). Therefore, we can compare the results of above three methods.

Because performance of an image or a video transformation relies on its image size (resolution), we can test the performances of three methods, using various image sizes. Table 1 represents the number of illustration frames in a video to be generated per second. The image sizes of videos used in our experiments are 320x180, 640x360, 960x540, and 1280x720. All methods were tested on an Intel Q9550 CPU PC with Microsoft Windows XP and a NVIDIA GeForce GTX260.

Table 1**.** The performance evaluation of 3 methods under various video resolutions.

Resolution	CPU1	CPU2	CUDA
320 x180	32.26	66.67	148
640x360	8.54	16.13	62.5
960x540	4	7.09	32.36
1280x720	2.56	3.77	21.28

As shown in Fig. **6**, the GPU-based method using NVIDIA CUDA shows more impressive results when comparing with the other methods. Particularly, if the image size of a video is enlarged, the performance difference grows because computation costs of the Gaussian Blur filter increase.

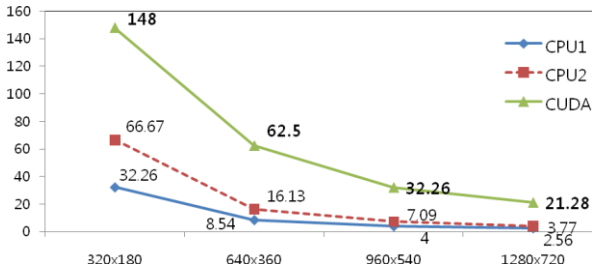


Fig. 6. Comparison of 3 methods (FPS)

5 Conclusion

In this paper, we suggest a video transformation system which is designed to apply illustration effects to video streams directly. It is also implemented by using the GPU hardware with NVIDIA's CUDA for real-time performances. Our system has more effective performances than systems simply with CPU. Excellent results come out in videos of the HD resolution (1280x720) as well. Illustration effects are not influenced in color distributions or in contents of videos because calculation is performed pixel by pixel. Therefore, performances according to the resolutions of videos can be expected for the real application.
In the future, we hope to create real-time video transformation systems with other non-photorealistic rendering effects like cartoon, photo-mosaics, and watercolor.

Acknowledgments. This work was supported by the IT R&D program of MCST/MKE/IITA. [*2008-F-030-02*, Development of Full 3D Reconstruction Technology for Broadcasting Communication Fusion]

References

1. Bruce Gooch, Erik Erinhard, Amy Gooch: Human Facial Illustrations: Creation and Psychophysical Evaluation. ACM Transactions on Graphics, Vol. 23, No. 1, pp. 27--44 (2004)
2. Holger Winnemöller, Sven C. Olsen, Bruce Gooch: Real-Time Video Abstraction. Proceedings of ACM SIGGRAPH 2006, pp. 1221--1226 (2006)
3. A. W. Klein, T. Grant, A. Finkelstein, M. F. Cohen: Video mosaics. Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering, pp. 21--28 (2002)
4. CUDA: Compute Unified Device Architecture. http://www.nvidia.com/object/cuda-home.html
5. OpenCV: Open Computer Vision Library, http://sourceforge.net/projects/opencvlibrary/