International Journal of Grid and Distributed Computing
Vol. 11, No. 4 (2018), pp.71-86
http://dx.doi.org/10.14257/ijgdc.2018.11.4.07

GLSL-Driven Background Subtraction Framework Using Particle Filter to Apply Special Effects without Chroma Key

Bon-woo Gu[1], Irfan Mehmood[2] and Hyung-gi Kim[3*]

Chung-Ang University, Sejong University, South Korea
[1]bgu3797@gmail.com, [2]irfan3797@gmail.com, [3]unzi@cau.ac.kr*

Abstract

Background subtraction methods have been used in various domains such as computer vision, television, movies, media art. In media, for special effects actors are subtracted by using Chroma key in background which requires efficient target/object detection tools. In computer vision, numerous target recognition and tracking methods have been presented but most of these methods are dealing with static background. To overcome the limitation of traditional methods, particle filters-based algorithm are employed for finding target objects regardless of their dynamic background. However, computational cost depends on the number of particles. This paper presents a novel particle filter that is based on OpenGL Shading Language (GLSL) using least 10,000 particles to detect and segment out target object accurately. Number of experiments are performed to validate the effectiveness of the proposed method. It has been observed that Graphic artists can apply special effects according to their taste using the proposed method. We hope the presented work will have a significant contribution in the technological development of TV, movie, and other media industries.

Keywords: *Particle Filter, Background subtraction, GPU, OPENGL, Parallel reductions*

1. Introduction

Background subtraction was used by computer vision, TV, movie, media art. In media, for special effect actor or actress were subtracted using Chroma key in the background. It is possible to subtraction target objects and background easily through the monochromatic background. However, If the object has a color similar to the background color, it will make some error. *Hiroki AGATA et al.* have proposed check pattern background for solve monochromatic background problems[1]. Chroma key must need a special space like a studio. In computer vision, many algorithms proposed for target object tracking. Like 'Gaussian Mixture Model'(GMM)[7,9], 'Kernel Density Estimation'(KDE)[4], Eigenbackgrounds[5]. But, these algorithms are methods to distinguish between background and object in a static background environment. To respond to dynamic background changes, *Vijay Mahadevan et al.* proposed a background subtraction algorithm using dynamic texture models[6,8]. That can find target object position in the dynamic background.

In just object tracking, Particle filters are the best algorithm for finding target objects regardless of their dynamic background. However, many computation processes on the number of particles do affect badly on the efficiency and speed[10,11,12]. *Bogusław Rymut et al.* proposed an efficient object tracking with particle filter using GPU for real-time tracking[12]. However, this algorithm intended to obtain approximate position and size of an object, not to obtain a boundary. Moreover, most of the particle filter algorithms are base on GPU using

Received (December 15, 2017), Review Result (February 26, 2018), Accepted (March 9, 2018)

ISSN: 2005-4262 IJGDC

'Compute Unified Device Architecture'(CUDA) framework. So it only can be used NVIDIA GPUs[13].

In this paper, we use 'OpenGL Shading Language'(GLSL), which can be used on most GPUs, to propose a way to precisely find the boundary of a target object with at least 10,000 particles. We hope that graphic artists can easily divide target objects in an image and give the individual effects without Chroma key by using our algorithm. We hope this proposal have contributed to the technological development of TV, movie, and media art.

In Section II, we will explain about other background subtraction algorithm and CUDA, GLSL. In Section III, the framework of the proposed method is described. In Section IV, explains parallel reductions that are how to find the minimum value of a pixel in texture in GLSL. In Section V, Discusses the experiment of the proposed algorithm and the application of them for special effect. In Section VI, It is a conclusion.

2. Related Work

2.1. Kernel Density Estimation (KDE)

Elgammal et al. proposed model the background distribution by non-parametric model[4]. That is 'Kernel Density Estimation'(KDE). KDE learn background by most recent continuously background sample.

$$P(b_t) = \frac{1}{N_b} \sum_{i=1}^{N_b} f_k(b_t - b_i) \tag{1}$$

Where N_b is number of background sample. And, $b_1, b_2, ..., b_{N_b}$ is values of a pixel in background sample. f_k is kernel estimation function. It can estimate background or foreground by each pixel. If $P(b_t) < T$, that mean foreground else background. Where T is threshold by user input[2,3].

2.2. EigenBackgrounds

Eigenbackgrounds proposed by *Oliver et al.* in [5]. It is building an eigenspace background models using initial background buffer. Eigenbackgrounds applied to the whole image. So, that avoids the tiling effect of a block in the picture [3].

$$| I_{Org} - I_{pro} | > T \tag{2}$$

Where I_{Org} is original image. I_{pro} is projection image by twice projection. First, I_{Org} projected onto the eigenspace. And then back projected onto original image space. Where T is threshold by user input. If $| I_{Org} - I_{pro} | > T$, that mean foreground else background [2,3].

KDE and Eigenbackgrounds base initial background image buffer. So, those can't respond dynamic background change.

2.3. Compute Unified Device Architecture (CUDA)

Compute Unified Device Architecture(CUDA) is created by NVIDIA [13,14,15]. It can process parallel computation on GPUs without basic computer graphics

knowledge. However, CUDA is only working on NVIDIA GPUs. Other GPUs, such as AMD RADEON, will not work [13].

2.4. OpenGL Shading Language (GLSL)

OpenGL Shading Language (GLSL) is created by Khronos Group [16,17,18]. It can process parallel computation on GPUs only if the user has the necessary computer graphics knowledge. GLSL can work in any GPUs for game or visualization [16,17]. GLSL has two calculation part, which is vertex shader and fragment shader [16,18]. Vertex shader works on a calculation about each vertex position in the buffer. Fragment shader works on a prediction about each pixel colors in the object. Unfortunately, Fragment shader has no function for finding the minimum value of a pixel or maximum amount of pixel in texture. In this paper, we make parallel reductions [21,22] using GLSL to find the minimum value of pixel color on the surface. This function can detect the similar amount by each particle in image texture.

3. Proposed Framework

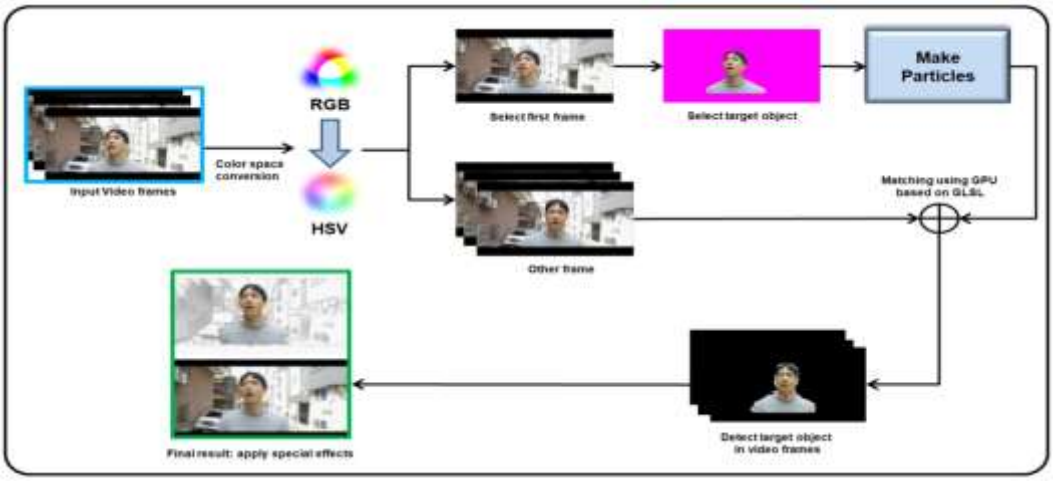

Figure 1. Overview of the Proposed Background Subtraction Framework

The process of proposed background subtraction is described in Figure 1. As HSV color space is more robust than RGB color space in terms of illumination variability [24], therefore at the first step, the image of all frames are converted from RGB to HSV. In first frame, background (without target object) is manually changed to magenta color (RGB=255, 0,255). After that, particles are made which target each of object pixels. After each frame is matched with target object particles by using GLSL, target object can be found for each frame. Finally, this result can be reproduced to various effects according to graphic artists taste.

3.1. Select Target Object

This section explains how to extract target object from first frame which will then be used for finding target object in all of other frames. User will select target object by first frame. All of the other objects except the target object will be set to magenta color.

$$F_i = Input\ video\ frames$$
$$i \in [1 \sim N] \quad (3)$$

Where F_i means input video frames. N is total number of flame in video.

$$Remove\_Pixel(\mathrm{x}) = \begin{cases} 0; & \text{if } \begin{cases} \mathrm{x_R} = 255 \\ \mathrm{x_G} = 0 \\ \mathrm{x_B} = 255 \end{cases} \\ 1; & else \end{cases} \tag{4}$$

$$Make\_Target(F) = Remove\_Pixel\left(F_{x,y}\right)$$
$$x \in [1 \sim W],\ y \in [1 \sim H] \tag{5}$$

The function $Make\_Target$ in Equation (5) is used to select the target. All pixels of the input frame F are examined by $Make\_Target$ function, and the $Remove\_Pixel$ function of Equation (4) is executed. W is the width of the input video frame and H is the height of the input video frame. The $Remove\_Pixel$ function in Equation (4) clears background for the objects whose pixel value is magenta color (RGB = 255,0,255) and will not apply on rest of the objects.

$$F^t = Make\_Target(F_1^m) \tag{6}$$

The user selects the target object manually in the first frame of the input video. In this case, target's color pixel is set to magenta color(RGB=255.0.255) In Equation (6), F_1^m is a image which includes target object and magenta color background by manually extracted from F_1. F_1^m is input to the $Make\_Target$ function to obtain F^t representing only target object. If target object obtains most of magenta color(RGB=255,0,255), user can switch to other color. $Remove\_Pixel$ function is set for distinguishing whether it is a target object or background.

3.2. Make Target Object Particles

Selected target object make particles pixel by pixel. If each particle have only one pixel value, matching results will have many errors because video frame have many similar pixels. So, each particle has covariance matrix of neighboring pixels.

$$F_{\mathrm{t}}^{HSV} = RGBtoHSV(F_1) \tag{7}$$

Equation (7) represents F_t^{HSV} by replacing pixel RGB of F_1 with HSV. HSV conversion was performed using the function provided by OPENCV[23].

$$Near\_pixels(F, x, y) = \sum_{b_x=-b_n}^{b_n} \sum_{b_y=-b_n}^{b_n} F_{x+b_x, y+b_y} \text{ , where } b_n = T_1/2 \tag{8}$$

$$NP_{end} = Near\_pixels(F^t, x, y)$$
$$x \in [1 \sim W], y \in [1 \sim H] \tag{9}$$

NP_{end} obtains the number of valid adjacent pixels through the $Near\_pixels$ function of Equation (8). In F^t, the background part is 0 and target object part is 1. So $Near\_pixels$ function counts only adjacent pixels of the target object particle. The threshold T_1 is the maximum number of adjacent pixels input by the user. If T_1 is increased, number of valid adjacent is increased which leads to more detailed appearance of each pixel by valid adjacent pixels. If T_1 will decreased, number of valid adjacent will decreased as well. If T_1 is 1, value of pixel will not change. In this case, T_1 = 3.

$$M_{cov}^{T}(F^1, F^2, x, y) = \frac{1}{NP_{end}} \sum_{b_x=-b_n}^{b_n} \sum_{b_y=-b_n}^{b_n} {F^1}_{x+b_x,y+y} \left({F^2}_{x+x,y+b_y} - \overline{C}(F^2, x, y) \right) \left({F^2}_{x+x,y+b_y} - \overline{C}(F^2, x, y) \right)^T, \text{ where } \overline{C}(F, x, y) = \frac{1}{NP_{end}} \sum_{b_x=-b_n}^{b_n} \sum_{b_y=-b_n}^{b_n} F_{x+b_x,y+b_y}$$
$$x \in [1 \sim W], y \in [1 \sim H] \quad (10)$$

In Equation (10), The M_{cov}^{T} function obtains a co- variance matrix of valid adjacent pixels. Where $\overline{C}$ is the average value of the available neighboring pixels.

$$PN_{end} = \sum_{x=1}^{W} \sum_{y=1}^{H} {F^t}_{x,y} \quad (11)$$

$$M_l^{particle} = \begin{cases} M_{cov}^{T}(F^t, F_1, x, y); & if\ {F^t}_{x,y} = 1 \\ 0; & else \end{cases}$$
$$x \in [1 \sim W], y \in [1 \sim H],\ l \in [1 \sim PN_{end}] \quad (12)$$

In Equation (12), $M_l^{particle}$ was obtained by particles of each of pixels in the first frame. If the value of F^t in the corresponding pixel value is 0, it will not be used because it represents background. If the value of F^t in the corresponding pixel value is 1, it will be used because it represents target object part. In Equation (11), PN_{end} is the number of particles. Number of particles are determined depending on target image size. If target image is large, number of particle will also increases. This allows more detailed matching using particles. However, it slower down the speed of calculation. If target image is small, number of particle will also decreased. In this case, calculation speed is faster.

3.3. Comparison Particle with Video

In this part, each particle is compared with all of the video frames. Each particle have covariance matrix. So, all video frames will have a covariance matrix pixel by pixel for matching.

$$F_i^{HSV} = RGBtoHSV(F_i)$$
$$i \in [1 \sim N] \quad (13)$$

In Equation (13), All video frames are converted to HSV using OPENCV.

$$M_{cov}(F, x, y) = \frac{1}{NP_{end}} \sum_{b_x=-b_n}^{b_n} \sum_{b_y=-b_n}^{b_n} \left(F_{x+b_x,y+b_y} - \overline{C}(F, x, y) \right) \left(F_{x+b_x,y+b_y} - \overline{C}(F, x, y) \right)^T$$
$$x \in [1 \sim W], y \in [1 \sim H] \quad (14)$$

In Equation (14), M_{cov} is similar to M_{cov}^{T} in Equation (10), However, M_{cov} uses at all pixels in the image as valid adjacent pixels, and creates a covariance matrix.

$$V_{cov}^{x,y,i} = M_{cov}(F_i^{HSV}, x, y)$$
$$x \in [1 \sim W], y \in [1 \sim H], i \in [1 \sim N] \quad (15)$$

Equation (15) makes all the frames pixels of the input video into a co-variances matrix. Total co-variances matrix $V_{cov}^{x,y,i}$ count is $W * H * N$.

$$dist(M_1^{cov}, M_2^{cov}) = \| \log(M_1^{cov}) - \log(M_2^{cov}) \|_2 \quad (16)$$

Equation (16) uses 'Log-Euclidean Metrics'[19,20] to compare the similarities of the two co-variances.

$$x, y = \arg^{x,y}_{min}\, dist(V^{x,y,i}_{cov}, M^{particle}_{l})$$
$$F^{R}_{x,y} = 1$$
$$x \in [1{\sim}W], y \in [1{\sim}H], i \in [1{\sim}N], l \in [1{\sim}PN_{end}] \quad (17)$$

Equation (17) finds the closest (most similar) position for each frame by comparing all particle co-variance $M^{particle}_{l}$ with all co-variance $V^{x,y,i}_{cov}$. F^{R} is final result image. $F^{R}_{x,y}$ set value 1 by closest (most similar) position.

3.4. Select Target Object in Other Frame

In this part, final image F^{R} is obtained. If $F^{R}_{x,y}$ value is 1, It represents the target object part. If $F^{R}_{x,y}$ value is 0, it represents no information. As shown Figure 2.(a), some particle have shown error. So, We select largest object in F^{R} and rest of the particles are considered as noise as shown in Figure 2(b). Figure 2(c) is the final cut obtained from process.

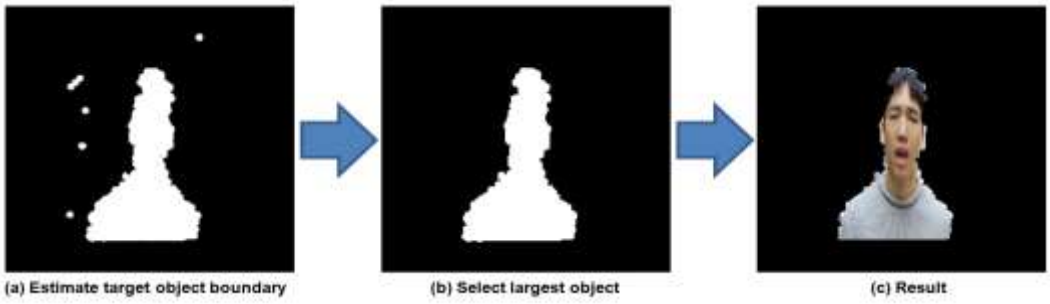

Figure 2. Select Target Object

4. Parallel Reductions in GLSL

In Equation (17), the amounts of calculations depend upon the size of the image. So, parallel calculations are performed with GLSL. GLSL, GPU processing using OpenGL [16,17,18], provides a space for storing images called 'Texture'. However, GLSL doesn't have function to find maximum or minimum value of pixel in texture. So, we have to use the method of 'Parallel Reductions' that requires manual coding but it isn't provided by GLSL. Frame Buffer objects (FBO) are used [16,18] that can make texture from screen pixels. Multiple rendering can be performed by this method. In this paper, we make parallel reductions [21,22] by using FBO.

$$V^{particle}_{x,y} = (d, x_v, y_v)^T$$
$$x \in [1{\sim}W], y \in [1{\sim}H] \quad (18)$$

In Equation (18), $V^{particle}_{x,y}$ is a feature vector for parallel computation in GLSL. $V^{particle}_{x,y}$ means one pixel in texture that is each frame of video. d stores the value calculated dist value in Equation (16). x_v, y_v stores the current position value of the pixel.

$$V^{particle}_{x,y}.d = dist(V^{x,y,i}_{cov}, M^{particle}_{l})$$
$$V^{particle}_{x,y}.x_v = x$$
$$V^{particle}_{x,y}.y_v = y$$
$$x \in [1{\sim}W], y \in [1{\sim}H], i \in [1{\sim}N], l \in [1{\sim}PN_{end}] \quad (19)$$

We made FBO texture as T_{BC}^{Min}. Now the size of each FBO texture is W x H. Correct pixel value must be extracted. So, OPENGL texture filter set 'GL_NEAREST' and wrapping set 'GL_REPEAT'. Initialize the value of $V_{x,y}^{particle}$ in first FBO texture as shown in Equation (19).

$$T_{size} = \begin{cases} W; & if\ W > H \\ H; & else \end{cases}$$
$$T_{block} = \begin{cases} W_{block}; & if\ W_{block} > H_{block} \\ H_{block}; & else \end{cases}$$

$$T_{BC}^{Min} = \sqrt[T_{block}]{T_{size}} + 1 \quad (20)$$

In Equation (20), T_{BC}^{Min} is expressed as the image size (W, H) and the block size (W_{blcok}, H_{block}). In this case, T_{BC}^{Min} is set to 15 because the image size is 200x200 and the block size is 2x2. Calculation speed of GLSL is delayed if block size is increased. Extensive calculations of GLSL must be avoided because all the computer system will also be delayed by long calculations. So, in this paper, block size 2x2 is selected for better performance.

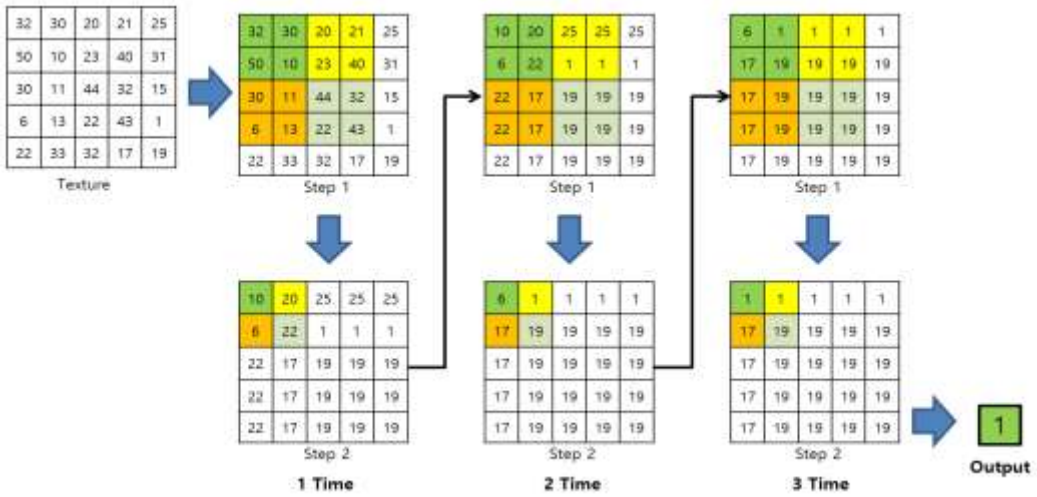

Figure 3. Find Minimum Value using Parallel Reduction in 5X5 Texture

Figure 3 shows the process of finding minimum value of pixel using parallel reductions in 5x5 texture. This resulted texture is 3 times of the parallel reductions. Image size T_{size} is 5 and block size T_{block} is 2. So $T_{BC}^{Min} = \sqrt[2]{5} + 1 = 3$.

Algorithm:

Step1: Found minimum value $V_{x,y}^{particle}.d$ in (x* T_{block})+ $(T_{block} - 1)$, (y* T_{block})+ $(T_{block} - 1)$. That is $\overline{V_{x,y}^{particle}}$.

Step2: Set value $\overline{V_{x,y}^{particle}}$ next FBO texture.

Step3: Repeat [Step 1] through [Step 2] as much as T_{BC}^{Min}.

Step4: $V_{1,1}^{particle}.x_v$, $V_{1,1}^{particle}.y_v$ of the first pixel means position of minimum value $\overline{V_{x,y}^{particle}}.d$ in T_{BC}^{Min}th FBO texture.

$x \in [1{\sim}W]$, $y \in [1{\sim}H]$, $i \in [1{\sim}N]$, $l \in [1{\sim}PN_{end}]$, $T_{black} = 2$, $T_{BC}^{Min} = 15$

After parallel reduction as much as T_{BC}^{Min}, We can get $V_{1,1}^{particle}.x_v$, $V_{1,1}^{particle}.y_v$ position of minimum value $\overline{V_{x,y}^{particle}}.d$ in T_{BC}^{Min}th FBO texture.

5. Experiments and Results

In this section, we conduct a series of experiments to validate the performance of the proposed background subtraction. We have done six experiments regarding color space, image size, block size, robustness to noise, robustness to occlusions, and result of UCSD Background subtraction dataset. And present special effect results using proposed background subtraction. All experiments were done using C / C ++, OPENGL 4.0, OPENCV 3.0, the CPU was tested on I-5 and three different GPUs(Nvidia GTX-1050 ,Nvidia GTX-750 ,Intel® HD Graphics 5500).

5.1. Color Space

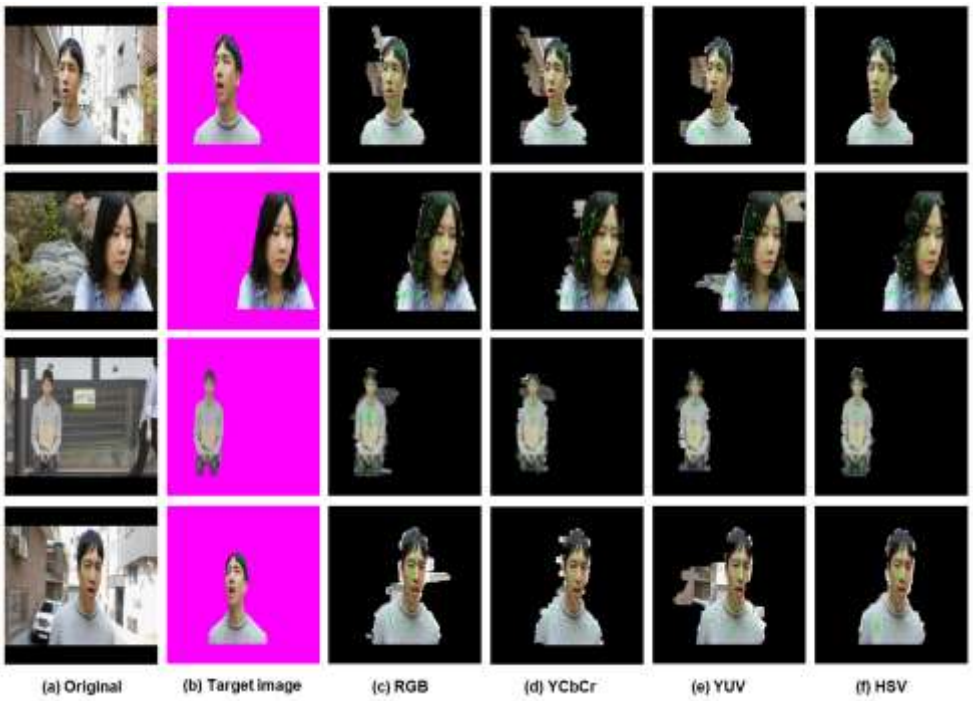

Figure 4. Color Space Experiment

In Section 5.1 comparisons of four color spaces RGB, YCbCr, YUV and HSV have been carried out. In Figure 4 the green points in image illustrates some particles position. Whereas Figure 4.(a), Figure (b), Figure (c), Figure (d), Figure (e) and Figure (f), shows the results of original image, target image(which is manually extracted by user), color space RGB, YCbCr, YUV and HSV respectively. In color space RGB, larger difference between the values of pixels is found to be giving better results that are of Figure (c). However, other images have several errors. In color space YCbCr, All images results have some errors. In color space YUV, results of most images have errors, in fact worst results were obtained. In color space HSV, most of images guarantee satisfactory results. In this paper we converted color space RGB to HSV for better performance.

5.2. Image Size

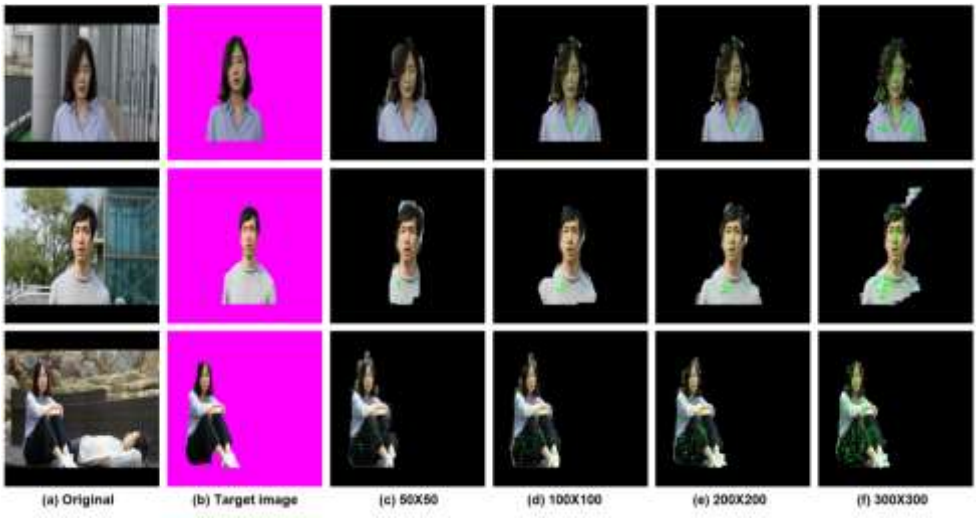

Figure 5. Image Size Experiment

The image size refers to the maximum number of particles. In Figure 5. Green points in an image shows some particles position. Whereas Figure 5.(a) is original image and Figure 5.(b) is the target image (manually extracted by user).We obtained results from the images(Figure 5.(c), Figure 5.(d), Figure 5.(e) and Figure 5.(f)) with the resolution of 50x50, 100x100, 200x200 and 300x300 respectively. Images having small size and less number of particles, as shown in Figure 5.(c), the resulting boundary of the target object will be rounded or has error. Contrary to this, images having large size and more number of particles, as shown in Figure 5.(e) , the boundary of the target object will be more detailed. However, when number of particles are more than necessary, then results will contain some errors as shown in Figure 5.(f).

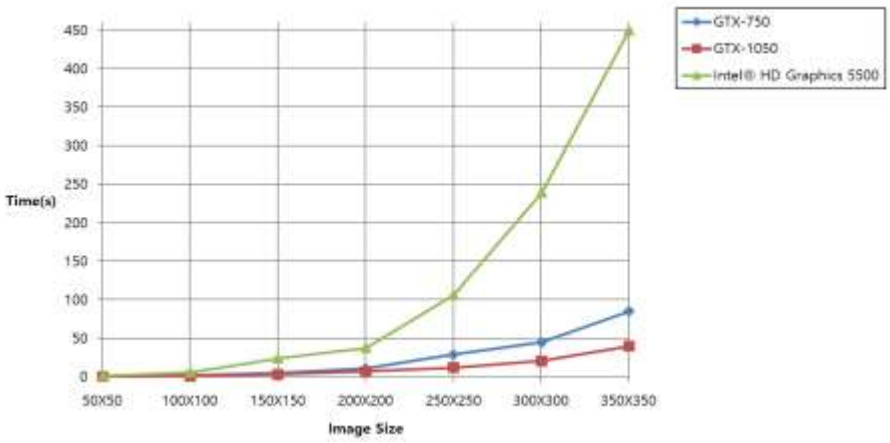

Figure 6. Rendering Speed According to Image Size

Figure 6. shows the rendering speed according to the image size. Three GPUs have same time(s) T_1 which is 3. As the size of the image increases and the maximum number of particles increases, due to this reason rendering speed also exponentially increases. Nvidia GTX-1050 and Nvidia GTX-750 maintain 100 or less second although image size is 350X350.Whereas, Intel® HD Graphics 5500

exponentially increases in over 200X200.As show in Figure 5. and Figure 6., most of videos have good results and rendering speed having image of resolution 200x200. So, in this paper we have selected image size 200x200 for better performance.

5.3. Block Size

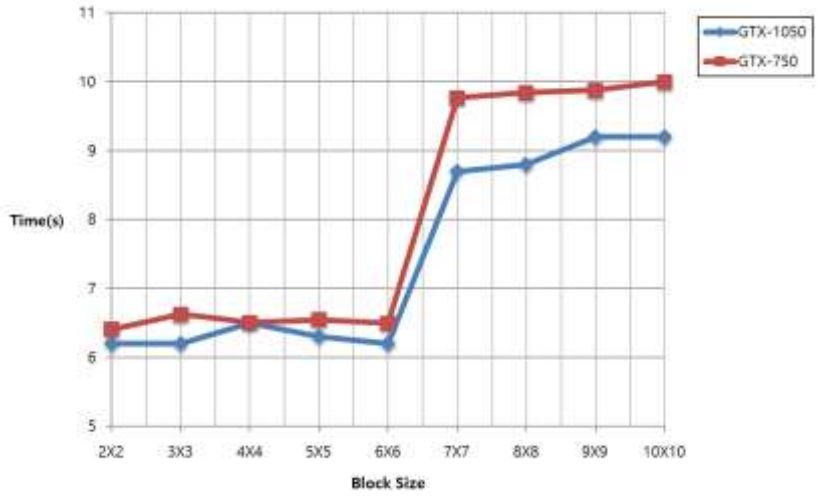

Figure 7. Rendering Speed According to Block Size

Figure 7. shows rendering speed according to Block size T_{block}. We have selected an image of size 200x200 and get rendering speed per each frame image. For increased block size, all computer system's speed was delayed. Even in Intel® HD Graphics 5500, whole system is much slowed. So, when block size is over 5x5, it can't measure the calculation speed at Intel ® HD Graphics 5500. In this paper we selected block size 2x2 to avoid the system delay problem on most of GPUs.

5.4. Robustness to Noise

Figure 8. Video Frame with Salt and Pepper Noise

We have used salt and pepper noise to produce noise in the frame image. Noise density range is 0.01~0.05. Figure 8.(a), Figure 8.(b), Figure 8.(c), Figure 8.(d), Figure 8.(e) and Figure 8.(f), are the results of noise having densities of 0.00, 0.01, 0.02, 0.03.0.04 and 0.05 respectively. In case of increased density more noise appeared on the image, vice versa. We have done experiments for rate of matching noise density.

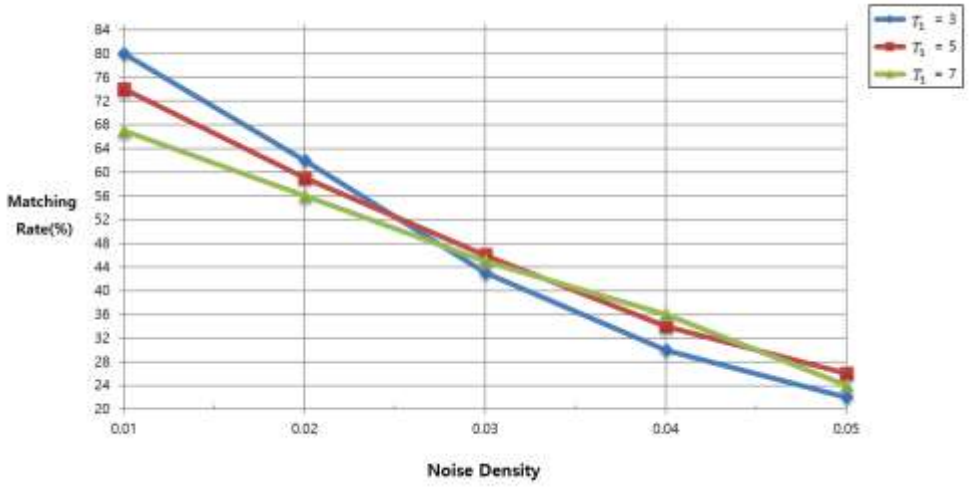

Figure 9. Matching Rate According to Salt & Pepper Noise Range

A 1000-frame video sequence is used for this sequence seven video frames are (72, 105, 342, 75, 165, 165 and 76). The matching rate was obtained by comparing the frame that has noise with the original frame. For decreased noise density, matching rate was high vice versa. In case of T_1= 3, result was best in noise density 0.01. but when density of noise was 0.05, result was worst. Average results were obtained when T_1= 5, for almost all noise densities. Whereas when T_1 =7 the results were worst then in almost all noise density.

5.5. Robustness to Occlusions

Figure 10. Video Frame with Occlusions

In this experiment we test the robustness of background subtraction to occlusion. Block size is random that ranges from [5,5] to [55,55]. Block positions were random as well that ranges from [1,1] to [200,200]. Figure 10.(a) is original image, Figure 10.(b) , Figure 10.(c) , Fig 10.(d) , Figure 10.(e) and Figure 10.(f) are the results of, block sizes[5,5], [15, 15], [35,35], [45,45]and [55,55] respectively.

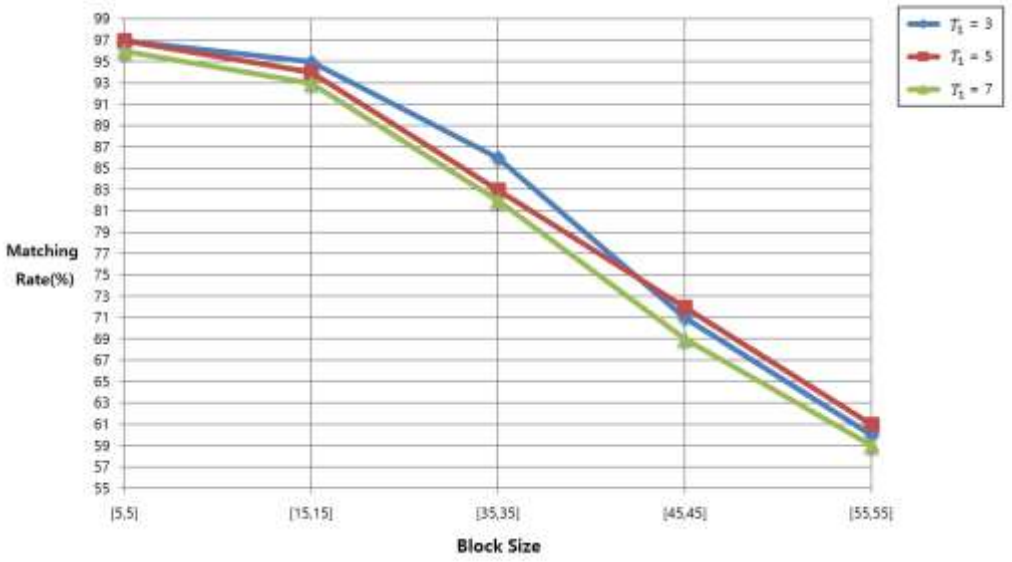

Figure 11. Matching Rate According to Block Size

A 1000-frame video sequence is used for this sequence seven video frames are (72, 105, 342, 75, 165, 165 and 76). The rate of matching was obtained by comparing the frame that have block with the original frame. For decreased block size the rate of matching was high, vice versa. In some parts of frame block occludes the result. So, Regardless of T_1 when block size increased the rate of matching was low.

5.6. Result of UCSD Background Subtraction Dataset

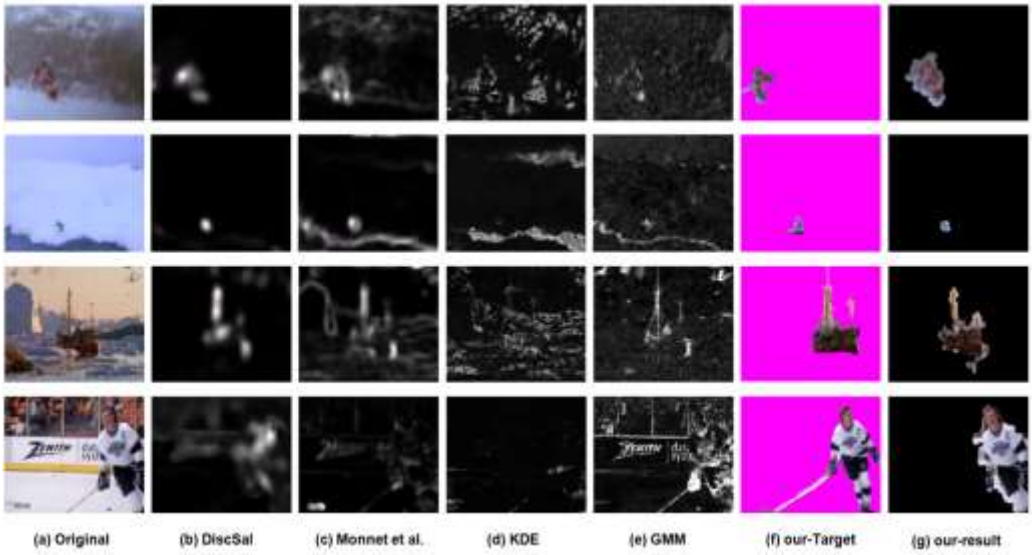

Figure 12. UCSD Dataset Result and Proposed Background Subtraction

We used UCSD Background subtraction dataset[6]. This dataset has results according to DiscSal, Monnet *et al.*, KDE and GMM in dynamic background. Figure 12.(a) illustrates original image. Figure 12.(b) shows DiscSal that is using dynamic texture models[6,8]. Figure 12.(c) is Monnet et al that is the linear dynamical model[6]. Figure 12.(d) is 'Kernel Density Estimate'(KDE) that is the nonparametric kernel density estimator[4]. Figure 12.(e) is GMM is 'Gaussian Mixture Model'[7]. Figure 12.(f) is the target image which is manually extracted by user. Figure 12.(g) is result of our proposed background subtraction. Figure 12.(c)~(e) can't extract the target object's position and boundary in dynamic background. Whereas Figure 12.(b) can extract target object's position in dynamic background. However, Boundary is not clear. Figure 12.(g) our proposed background subtraction can extract clear position and boundary than Figure 12.(b)~(e) in dynamic background.

5.7. Special Effect Result

Figure 13. Apply 4 Special Effect

We apply four special effects (Cartoon effect, Sketch effect, Water Color effect and Sketch-Background). Figure 13.(a) is the original image. Figure 13.(b) is the target image which is manually extracted by user. Figure 13.(c) is result of cartoon effect. Figure 13.(d) is result of sketch effect. Figure 13.(e) is result of water color effect. Figure 13.(f) is result of sketch-background effect. As shown Fig 13, through our proposed background subtraction, special effects can be easily applied separately to the target object and the background. Graphic artists can have various effects according to the requirements.

6. Conclusion

We have proposed background subtraction method using particle filter and developed a function for parallel reduction based on GLSL. The presented method can accurately detect target object by using at least 10,000 particles on any GPUs. We have done validation through various experiments and applied four different special effects for graphic artists. Thus, graphic artists may apply special effects very easily using our method without Chroma key. Therefore, it is believed that the current framework will contribute on the technological development of TV, movie, and media art. However, the presented method is not efficient enough to work in a real-time environment as if fails to render in real-time. This is mainly due to the

process of huge numbers of particles in video frames in fragment shader. Rendering takes at least 10 seconds per frame. Currently, fragment shader is taking major computational power. Vertex shader is faster than fragment shader in GLSL and in near future, we will try to reduce the computational complexity so that the presented method can used in realtime both for desktop and smart devices.

References

[1] H. Agata, A. Yamashita and T. Kaneko, "Chroma key using a checker pattern background", IEICE TRANSACTIONS on Information and Systems, vol. 90, no. 1, **(2007)**, pp. 242-249.
[2] N. Joubert, "Background modelling and subtraction for object detection in video", Stellenbosch University, (BEng Sci, 2009), **(2009)**.
[3] M. Piccardi, "Background subtraction techniques: a review", Systems, man and cybernetics, 2004 IEEE international conference on IEEE, vol. 4, **(2004)**.
[4] A. Elgammal, D. Harwood and L. Davis, "Non-parametric model for background subtraction", Computer Vision-ECCV 2000, **(2000)**, pp. 751-767.
[5] N. M. Oliver, B. Rosario and A. P. Pentland, "A Bayesian computer vision system for modeling human interactions", IEEE transactions on pattern analysis and machine intelligence, vol. 22, no. 8, **(2000)**, pp. 831-843.
[6] V. Mahadevan and N. Vasconcelos, "Spatiotemporal saliency in dynamic scenes", IEEE transactions on pattern analysis and machine intelligence, vol. 32, no. 1, **(2010)**, pp. 171-177.
[7] http://staff.science.uva.nl/~zivkovic/download.html, **(2009)**.
[8] V. Mahadevan and N. Vasconcelos, "Background subtraction in highly dynamic scenes", Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on IEEE, **(2008)**.
[9] C. Stauffer and W. Eric L. Grimson, "Adaptive background mixture models for real-time tracking", Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on IEEE, vol. 2, **(1999)**.
[10] A. Almeida, J. Almeida and R. Araújo, "Real-time tracking of moving objects using particle filters", Industrial Electronics, 2005, ISIE 2005. Proceedings of the IEEE International Symposium on IEEE, vol. 4, **(2005)**.
[11] C. Choi and H. I. Christensen, "RGB-D object tracking: A particle filter approach on GPU", Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on IEEE, **(2013)**.
[12] B. Rymut and B. Kwolek, "GPU-Accelerated Object Tracking Using Particle Filtering and Appearance-Adaptive Models", Image Processing and Communications Challenges, vol. 2, **(2010)**, pp. 337-344.
[13] Nvidia, C. U. D. A., "Programming guide", **(2010)**.
[14] S. In Park, "Low-cost, high-speed computer vision using NVIDIA's CUDA architecture", Applied Imagery Pattern Recognition Workshop, 2008. AIPR'08. 37th IEEE. IEEE, **(2008)**.
[15] D. Arkhipov, "Sorting with GPUs: A Survey", arXiv preprint arXiv:1709.02520, (2017).
[16] D. Shreiner and Bill The Khronos OpenGL ARB Working Group, "OpenGL programming guide: the official guide to learning OpenGL", versions 3.0 and 3.1. Pearson Education, **(2009)**.
[17] M. Woo, "OpenGL programming guide: the official guide to learning OpenGL", version 1.2. Addison-Wesley Longman Publishing Co., Inc., **(1999)**.
[18] R. J. Rost, "OpenGL shading language", Pearson Education, **(2009)**.
[19] V. Arsigny, "Log-Euclidean metrics for fast and simple calculus on diffusion tensors", Magnetic resonance in medicine, vol. 56, no. 2, **(2006)**, pp. 411-421.
[20] B. Kwolek, "Multi-object tracking using particle swarm optimization on target interactions", Advances in Heuristic Signal Processing and Applications. Springer Berlin Heidelberg, **(2013)**, pp. 63-78.
[21] M. Takahashi, "Parallel reductions in λ-calculus", Information and computation, vol. 118, no. 1, **(1995)**, pp. 120-127.
[22] M. Harris, "Optimizing cuda", SC07: High Performance Computing With CUDA, **(2007)**.
[23] G. Bradski and A. Kaehler, "Learning OpenCV: Computer vision with the OpenCV library", O'Reilly Media, Inc., **(2008)**.
[24] P. Luigi Mazzeo, "HSV and RGB color histograms comparing for objects tracking among non overlapping FOVs, using CBTF", Advanced Video and Signal-Based Surveillance (AVSS), 2011 8th IEEE International Conference on IEEE, **(2011)**.

Authors

Bon-woo Gu was born in Korea in 1992. He received his B.S. degree in digital contents from Sejong University in 2016. Currently, he is pursuing an M.S degree in game engineering at the Graduate School of Advanced Imaging Science, Multimedia & Film in Chung-Ang University. His research interests include Computer Graphics, Image Processing, HCI, Pattern Recognition, Interactive media art.

Irfan Mehmood conducts research in a number of basic and applied areas such as image and scene segmentation, motion and video analysis, perceptual grouping, shape analysis and object recognition. He has done various joint ventures with industry to develop applications for surveillance, video summarization, steganography, and deep learning based image super resolution.

Hyung-gi Kim is a professor of Art and Technology at the Graduate School of Advanced Imaging Science, Multimedia and Film, Chung-Ang University, Seoul, Korea. He received his D.N. S.A.P. in Multimedia Art from l'Ecole Nationale Supériure des Beaux-Arts de Paris in 1991. He received his D.E.A. in Multimedia and Media from Conservatoire National des Arts et Métiers in 2001 and his Ph.D. in Media Art from the Graduate School of Soongsil University in 2009. He has launched 11 solo media art exhibitions and has participated in many group exhibitions. He was chief director of the Incheon International Digital Art Festival in 2009 and director of SBS Tomorrow Festival in 2010. His research interests are projection mapping, media façades, interactive media art, and media performance.