KIPS Tr. Software and Data Eng.
Vol.4, No.11 pp.529~536 pISSN: 2287-5905

http://dx.doi.org/10.3745/KTSDE.2015.4.11.529

Non-Photorealistic Rendering Using CUDA-Based Image Segmentation

Hyun-Cheol Yoon† · Jong-Seung Park††

ABSTRACT

When rendering both three-dimensional objects and photo images together, the non-photorealistic rendering results are in visual discord since the two contents have their own independent color distributions. This paper proposes a non-photorealistic rendering technique which renders both three-dimensional objects and photo images such as cartoons and sketches. The proposed technique computes the color distribution property of the photo images and reduces the number of colors of both photo images and 3D objects. NPR is performed based on the reduced colormaps and edge features. To enhance the natural scene presentation, the image region segmentation process is preferred when extracting and applying colormaps. However, the image segmentation technique needs a lot of computational operations. It takes a long time for non-photorealistic rendering for large size frames. To speed up the time-consuming segmentation procedure, we use GPGPU for the parallel computing using the GPU. As a result, we significantly improve the execution speed of the algorithm.

Keywords : Non-Photorealistic Rendering, CUDA, Image, Segmentation, Colormap

CUDA 기반 영상 분할을 사용한 비사실적 렌더링

윤 현 철† · 박 종 승††

요 약

비사실적 렌더링(NPR; Non-Photorealistic Rendering)은 2차원 영상과 3차원 모델을 대상으로 하는 방법이 다르며 각각의 대상에 NPR을 적용하여 두 콘텐츠를 혼합하면 이질감이 나타나는 문제점이 있다. 본 논문에서는 3차원 객체와 영상에 있어서 각각의 대상에 카툰 및 스케치와 같은 비사실적 효과를 적용하여 조화롭게 혼합하는 기법을 제시한다. 제안 기법은 2차원 영상의 데이터를 분석하여 컬러 분포 특징을 얻고 이를 이용하여 실사 영상이나 3D 객체의 컬러 수를 줄인다. 단순화된 컬러맵과 윤곽선 에지 데이터로부터 비사실적 렌더링을 실시한다. 컬러맵 정보의 추출 및 적용 과정에서 자연스러운 장면 연출을 위해서 영상분할 과정이 필요하다. 그러나 영상분할 기법은 많은 연산을 필요로 한다. 특히 크기가 큰 입력에 대해서는 비사실적 렌더링에 많은 시간이 소요된다. 처리 시간이 많은 영상분할의 고속화를 위하여 GPU(Graphics Processing Unit)를 이용한 병렬 컴퓨팅을 할 수 있는 GPGPU(General-Purpose GPU)를 사용한다. GPGPU의 사용으로 알고리즘의 수행속도를 크게 개선하였다. 또한 영상분할 후 단순화된 컬러를 추출하여 일련의 컬러맵을 생성한 뒤 3D 객체에 NPR을 적용할 때 추출해낸 컬러맵을 적용하여 2차원 영상과 3차원 객체 간의 이질감을 줄이고 조화롭게 하였다.

키워드 : 비사실적 렌더링, CUDA, 영상, 분할, 컬러맵

1. 서 론

그동안 발전해온 사실적 렌더링과는 다르게 비사실적 렌더링은 카툰, 회화, 수묵화, 잉크 드로잉 등의 느낌을 표현하기 위해 고안된 기법이다. 비사실적 렌더링은 전달하고자 하는 점을 강조하고 중요하지 않은 부분은 단순화하거나 과감하게 생략하여 전달하는 주관적인 기법이다. 인간에게 친근한 영상의 생성을 목적으로 하며, 광고 분야나 예술 및 문화 응용 분야에서 효과적인 내용 전달 방법으로 사용되고 있다[1].

비사실적 렌더링에서 전달하고자 하는 점을 강조하기 위해 외곽선을 검출하여 추출된 외곽선을 진하게 표현하기도 하며, 중요하지 않은 부분을 단순화하거나 과감하게 생략하기 위하여 색이 비슷한 부분을 같은 색으로 표현하거나 색상을 단순화하는 기법을 이용한다[2]. 이러한 기존의 비사실적 렌더링을 더 깔끔하고 색이나 특징이 비슷한 부분을 같은 영역으로 잘 표현하기 위해서 영상 분할 기법이 필요하다[3]. 영상 분할 기법은 영상을 픽셀들의 특성을 이용하여 비슷한 특성끼리 여러 개의 픽셀들의 집합으로 나누는 과정이다[4].

※ 이 논문은 인천대학교 2014년도 자체연구비 지원에 의하여 연구되었음.
† 비 회 원 : 인천대학교 컴퓨터공학부 석사과정
†† 종신회원 : 인천대학교 컴퓨터공학부 교수
Manuscript Received : May 11, 2015
First Revision : June 25, 2015
Accepted : June 27, 2015
* Corresponding Author : Jong-Seung Park(jong@inu.ac.kr)

한편 영상 분할 기법은 많은 연산을 필요로 한다. 특히 크기가 큰 입력에 대해서는 비사실적 렌더링에 많은 시간이 소요된다. 따라서 비사실적 렌더링 기법의 개발에 있어서 수행 시간의 단축이 고려되어야 한다.

최근에는 연산속도의 향상을 위하여 GPU를 이용한 병렬 컴퓨팅 연구가 활성화되었다. GPU의 병렬 컴퓨팅을 그래픽 렌더링뿐만 아니라 일반적인 연산에서도 사용하여 연산비용이 큰 알고리즘의 수행시간을 대폭 줄여서 실시간에서도 사용되고 있다.

3차원 객체와 영상에 있어서 각각의 대상에 카툰 및 스케치와 같은 비사실적 효과를 적용한다. NPR 기법은 3D 객체 및 2차원 영상의 데이터를 분석하여 특징을 얻어낸다. 이러한 특징을 이용하여 데이터의 에지를 생성한다. 또한 기존 데이터의 컬러 분포를 단순화한다.

GPU를 이용한 병렬 컴퓨팅을 가능하게 하는 GPGPU (General-Purpose GPU)를 이용하면 연산량이 큰 알고리즘의 수행속도를 개선할 수 있다. 비사실적 렌더링은 부동 소수점 연산과 같은 간단하지만 반복적인 연산들이 대부분을 차지하므로 GPGPU에 적합하다[5].

멀티미디어 연산처리에 최적화된 NVIDIA사의 CUDA (Compute Unified Device Architecture)를 이용하면 연산 비용이 큰 부분에 해당하는 영상분할이나 필터링, 가우시안, 에지연산 등을 병렬화함으로써 수행속도를 개선하여 실시간 처리가 가능하게 한다. CUDA 프로그래밍에서는 각 연산들을 쓰레드와 블록 단위로 나누어서 계산되며 하나의 스트리밍 프로세서는 4개의 스레드(thread)를 동시에 실행할 수 있고, 하나의 블록은 하나의 스트리밍 멀티프로세서와 대응하여 동작하게 된다. 그러면 GPU는 스트리밍 멀티프로세서 단위로 블록을 처리하면서 작업을 처리하게 된다. 본 논문에서는 제안하는 비사실적 렌더링 기법을 CUDA에서 구현하여 속도 향상을 이루었다.

2. 관련 연구

2.1 비사실적 렌더링

비사실적 렌더링(NPR; Non-Photorealistic Rendering) 기법은 사실적인 표현 기법과는 다르게 손으로 그린 듯한 느낌을 주도록 표현하는 기법이다. 비사실적 렌더링은 펜 및 잉크 드로잉[6], 카툰[7] 등의 만화적인 느낌을 표현하기도 하며 회화[8], 수묵화[9], 연필 스트로크[10] 등의 손으로 그린 듯한 느낌을 표현하기 위한 렌더링 기법이다. 이 기법은 광고나 교육, 예술, 문화 등 여러 분야에서 효과적인 내용 전달 방법으로 사용되고 있다. NPR은 과감한 생략과 강조 등의 방법을 통한 주관적인 의미 전달에 중점을 두며 사실적인 표현보다는 인간에 친근한 그림 작품과 같은 영상의 생성을 추구한다.

비사실적 렌더링은 그 대상에 따라 크게 두 가지 방법으로 나뉜다. 처음 방법은 실사 영상을 대상으로 하는 비사실적 렌더링 기법이고 두 번째 방법은 3D 객체를 대상으로 하는 비사실적 렌더링 기법이다.

첫 번째 방법인 실사 영상을 대상으로 하는 방법은 크게 두 단계로 나뉜다[11]. 두 단계는 색을 단순화하는 생략하는 단계와 외곽선을 검출하여 더 진하게 나타내는 강조 단계이다. Fig. 1은 2D 영상을 대상으로 하는 비사실적 렌더링 기법의 예이다.

Fig. 1. Non-Photorealistic Rendering for 2D Images

2D 영상의 NPR과는 달리 3D 객체를 대상으로 하는 NPR은 3D 모델의 재질, 빛, 시점 등의 요소도 고려하여 렌더링해야 한다[12]. Fig. 2는 3D 모델의 비사실적 렌더링 기법을 보여준다.

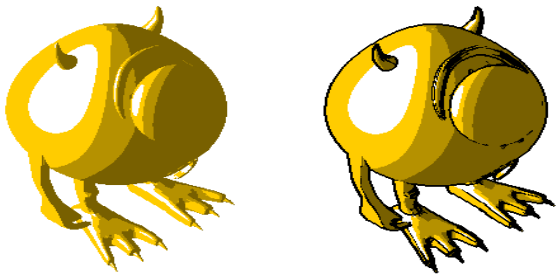

Fig. 2. Non-Photorealistic Rendering for 3D Objects

2.2 영상 분할

비사실적 렌더링에 있어서 영상 분할은 중요한 과정이다. 비사실적으로 표현하기 위해서는 색을 단순화해야 하는데 이때 영상 분할을 이용하면 비슷한 구역별로 비슷한 색으로 표현할 수 있기 때문이다.

영상 분할 기법은 영상을 여러 개의 픽셀들의 집합으로 나누는 과정이다. 영상 분할 기법은 각 픽셀에 일련의 컴포넌트 값을 부여함으로써 비슷한 특징을 가진 픽셀들끼리 같은 컴포넌트를 가지게 한다[13]. 영상 분할의 목표는 영상 분할을 통하여 영상 내에서 비슷한 색상 및 위치에 있는 픽셀들끼리 하나의 세그멘트로 묶어서 영상을 단순화하는 것이다. Fig. 3은 영상을 대상으로 인접 픽셀의 색상정보를 통해 영상을 분할하는 Felzenszwalb의 분할 기법[14]을 적용한 예이다.

Felzenszwalb의 분할 기법은 Kruskal의 에지의 최소 가중치를 바탕으로 모든 노드가 연결되도록 하는 최소 거리

Fig. 3. Photo Images and Their Image Segmentation Results

그래프의 집합 알고리즘인 MST(Minimum Spanning Tree) 알고리즘에 기초한 분할 방법이다. 이외에도 영상 분할 기법에는 그 방법에 따라 그래프를 이용한 기법[15], Quick shift를 이용한 기법[16] 등이 있다.

2.3 CUDA

많은 코어를 가진 GPU를 활용하여 병렬 컴퓨팅을 할 수 있는 기술을 GPGPU라고 한다. 이 중 NVIDIA사의 CUDA를 이용하여 많은 단순 반복 연산의 쓰레드들을 한 번에 처리하여 프로그램의 빠른 속도를 기대할 수 있다[17]. GPU에서 SM(streaming multiprocessor)은 커널함수를 호출하여 연산한다. SM은 GPU의 작은 코어중의 하나이며 구조는 Fig. 4에서 확인할 수 있다.

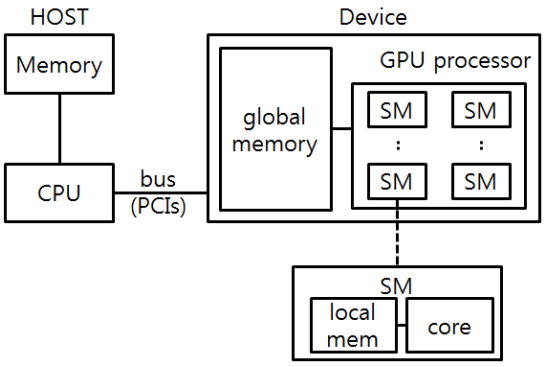

Fig. 4. Hardware Architecture of Host and Device

SM은 스레드 블록단위로 계산하며 각 스레드들은 연산이 끝나면 문맥교환을 한다. 이러한 과정은 Fig. 5에 표시되어 있다. 하나의 스레드는 하나의 커널 함수를 호출하여 계산한다. Fig. 6은 하나의 스레드가 모든 정점들을 대상으로 플래그 값을 초기화하는 커널함수를 보여준다. CUDA의 병렬 처리 방식을 사용하면 Fig. 7에서 볼 수 있듯이 프로그램을 블록과 스트리밍 멀티프로세서 단위로 나누어 데이터를 한 번에 처리하여 광범위한 응용 프로그램에서 좋은 성능을 얻을 수 있다[18].

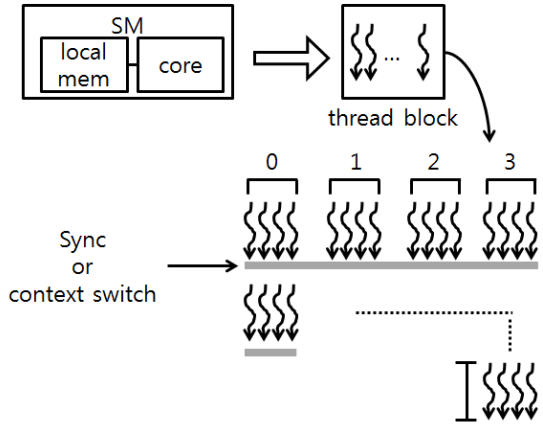

Fig. 5. A Thread Block Runs on a Single SM

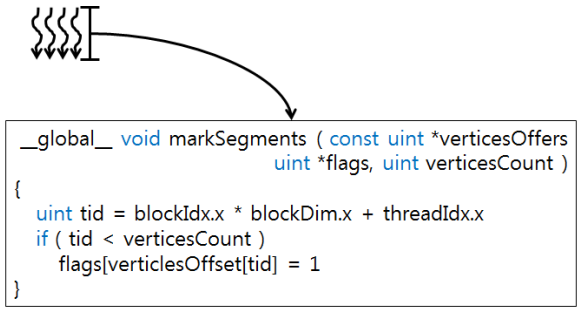

Fig. 6. A CUDA Kernel Executed by an Array of Threads

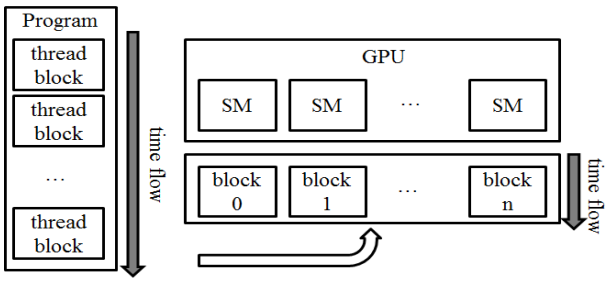

Fig. 7. GPU Parallel Processing

3. CUDA 기반 영상 분할을 사용한 알고리즘

본 논문에서는 영상과 3D 객체의 혼합된 렌더링에 있어서 적용될 수 있는 NPR(Non-Photorealistic Rendering) 기법의 연산에 GPGPU 환경인 CUDA를 이용한 알고리즘을 제시한다. 제안하는 알고리즘은 Fig. 8과 같다. 먼저 렌더링하기 위한 장치와 CUDA를 사용하기 위한 장치를 초기화하고 입력받은 영상을 바탕으로 정점과 에지, 가중치 정보를 저장한 그래프를 구축한다. 다음으로, 구축한 정보를 바탕으로 GPU에서 분할 알고리즘을 수행한다. 분할 알고리즘의 결과로 얻은 영역 영상으로부터 컬러맵을 생성한다. 마지막으로, 얻어진 컬러맵을 사용하여 3D 객체와 영상을 함께 렌더링한다.

Input : Real Photo Image I_I, 3D Object O_I
Output : NPR Image I_O
Begin
Initialize()
$I_S \leftarrow SegmentImage(\ I_I\ )$
$I_C \leftarrow DetectContour(\ I_I\ )$
$I_A \leftarrow I_S \& I_C$
$O_O \leftarrow ChangeNPRObject(\ I_S,\ O_I\ )$
$I_O \leftarrow I_A + O_O$
Render(I_O *)*
Release()
End

Fig. 8. Pseudo-Code of the Proposed Method

먼저, 분할할 실사 영상 I_I와 함께 출력할 3D 객체 O_I를 입력받는다. 다음, *Initialize()*를 통해서 CUDA를 사용하기 위한 장치와 그리기 위한 장치를 초기화한다. 다음, 영상 분할 알고리즘을 이용하여 분할된 영상 I_S를 얻는다. 영상의 윤곽선을 검출한 I_C와 앞서 분할을 적용한 영상 I_S를 합성하여 I_A 영상을 얻는다. 여기서 '&' 기호는 두 영상의 각 픽셀별 AND 연산을 의미한다. 다음, 입력받은 O_I를 비사실적으로 렌더링하고, 분할 알고리즘을 적용하여 만들어진 I_S에 사용된 색을 바탕으로 O_I의 최종 출력 컬러값을 결정하여 O_O를 만들어낸다. 최종적으로, 실사 영상을 비사실적으로 그린 I_A와 3D 객체를 비사실적으로 그린 O_I를 병합하여 최종 영상 I_O를 출력한다.

3.1 분할 알고리즘

분할 알고리즘은 Felzenszwalb의 방법에 기초하였다. Felzenszwalb의 방법과 다른 점은 그래프를 생성하는 절차 등을 고속 병렬 실행에 적합하게 개발하였다. 전체적인 분할 알고리즘은 Fig. 9와 같다.

Algorithm *SegmentImage*
Input Real Image I_I
Output Symbolic Image I_S
Begin
$G,\ w \leftarrow BuildGraph(I_I\ )$
$G=(V,E)$ with n vertices and m edges
$E \leftarrow Sort(G,w)$
For q = 1 **to** m
$O_q \leftarrow (v_i, v_j)$
If $(C_i^{q-1} \neq C_j^{q-1})$ and $(w(O_q) \leq MInt(C_i^{q-1}, C_j^{q-1}))$ **then**
$S^{q-1} \leftarrow Merge(C_i^{q-1}, C_j^{q-1})$
$S^q \leftarrow S^{q-1}$
EndFor
End

Fig. 9. Pseudo-Code of the Segmentation Algorithm

먼저 입력 실 영상으로부터 영상 픽셀 정보를 바탕으로 정점과 에지, 가중치 정보를 갖는 그래프를 구축한다. 그래프 구축 알고리즘은 Fig. 10과 같다.

Algorithm *BuildGraph*
Input : Image I_I
Output : Graph G, weight w
Begin
Reserve $G=(V,E)$, w
For y to height **do**
For x to width **do**
$V_i \leftarrow i$
$E_j \leftarrow Idx_{LO},\ w_j \leftarrow \text{distance}(P_C,\ P_{LO})$
$E_{j+1} \leftarrow Idx_{UP},\ w_{j+1} \leftarrow \text{distance}(P_C,\ P_{UP})$
$E_{j+2} \leftarrow Idx_{LE},\ w_{j+2} \leftarrow \text{distance}(P_C,\ P_{LE})$
$E_{j+3} \leftarrow Idx_{RI},\ w_{j+3} \leftarrow \text{distance}(P_C,\ P_{RI})$
EndFor
EndFor
End

Fig. 10. Pseudo-Code of the Build-Graph Algorithm

그래프 구축 알고리즘에서는 먼저 구축할 그래프의 정점과 에지, 가중치를 저장할 공간을 할당한 뒤 영상의 처음 픽셀부터 마지막 픽셀까지 전체 픽셀을 대상으로 에지와 가중치를 설정한다. 알고리즘에서 P_C, P_{LO}, P_{UP}, P_{LE}, P_{RI}은 Fig. 11에서의 배치에서의 색상값에 해당한다. 가운데 픽셀은 pixel(y*$width$+x), 아래 픽셀은 pixel((y-1)*$width$+x), 위 픽셀은 pixel((y+1)*$width$+x), 왼쪽 픽셀은 pixel(y*$width$+x-1), 오른쪽 픽셀은 pixel(y*$width$+x+1) 값으로 구할 수 있다. 이렇게 구한 픽셀의 색상값과 가운데 색상값의 가중치 w를 구하여 리스트에 넣고 이때의 픽셀의 인덱스 값을 에지 리스트에 넣는다.

	upperPixel	
leftPixel	centerPixel	rightPixel
	lowerPixel	

Fig. 11. Layout of the Adjacent Pixels

가중치 w는 에지에 연결되어있는 픽셀간의 R, G, B 값의 차이를 Euclidean 공식을 이용하여 다음과 같이 구한다. 아래 식으로 각 에지의 R, G, B의 거리를 가중치로 저장한다.

$$w = \sqrt{(P_C.r - P_{UP}.r)^2 + (P_C.g - P_{UP}.g)^2 + (P_C.b - P_{UP}.b)^2}$$

다음, 에지 가중치 w값을 기준으로 에지 E값들을 정렬한다. q번째 에지에 이어져 있는 두 정점들을 $O_q = (v_i, v_j)$ 라고

할 때, 만약 v_i와 v_j가 세그먼테이션 S^{q-1}의 다른 컴포넌트에 속해있고 그 두 컴포넌트 내부의 차이가 에지의 가중치 $w(O_q)$보다 작을 경우, 즉 $(C_i^{q-1} \neq C_j^{q-1})$ and $(w(O_q) \leq MInt(C_i^{q-1}, C_j^{q-1})$일 때 두 컴포넌트를 병합한다. 이를 에지의 숫자만큼 반복하고 최종적으로 세그먼트를 출력한다.

Algorithm *ChangeNPRObject*
Input Symbolic Image I_S, 3D Object O_I
Output O_O
Begin
  $CM \leftarrow$ CreateColormap(I_S)
  $O_A \leftarrow O_I \otimes B$
  $O_B \leftarrow O_A \otimes K_{GX} \otimes K_{GY}$
  $O_C \leftarrow O_B$*specularFactor*diffuseFactor
  $O_O \leftarrow$ ApplyColorMap(O_C, CM)
End

Fig. 12. Pseudo-Code of the NPR Algorithm

3.2 NPR 알고리즘

분할된 영상 I_S에서 사용된 일련의 컬러 값들을 저장한 컬러맵(colormap)을 생성한다. 이 컬러맵은 3D 객체를 그릴 때 영상에서 사용된 컬러값을 이용하여 렌더링하여 두 콘텐츠 간의 이질감을 줄이는 데 사용된다. 제안 NPR 알고리즘에서 3D 객체의 외양을 수정하는 알고리즘은 Fig. 12와 같다.

NPR은 크게 색 단순화와 외곽선 강조의 두 단계로 나눌 수 있다. 색 단순화 단계는 분할로 구분된 영역별 평균 색상으로 구할 수 있다. 외곽선 강조의 경우 노이즈 제거를 위해서 Equation (1)의 가우시안 필터를 컨볼루션(convolution)한다. '⊗' 기호는 컨볼루션을 의미한다.

$$B = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} \tag{1}$$

다음, 영상의 미분값을 통하여 외곽선을 검출한다. Equation (2)의 소벨 마스크를 모든 픽셀에 컨볼루션하여 외곽선을 검출해낸다. K_{GX}는 X축 외곽선을 검출하고 K_{GY}는 Y축 외곽선을 검출한다.

$$K_{GX} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix},\ K_{GY} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2}$$

검출한 외곽선과 단순화한 영상을 합성하여 중간 영상을 출력한다. 이때 단순화한 영상에 쓰인 R, G, B 컬러를 리스트로 저장하여 일련의 컬러맵을 생성한다.

3차원 모델을 렌더링할 때는 컬러의 변화가 단계적으로 뚜렷하게 바뀌도록 하고 윤곽선에 해당하는 외곽선을 검출하여 표시하게 한다. 먼저, 윤곽선을 계산하기 위한 정보를 텍스처에 렌더링한다. 다음, 확산 성분의 반사량 *diffuseFactor*와 정반사 성분의 반사량 *specularFactor*를 계산한다. 확산 성분은 노말 벡터와 빛이 향하는 벡터의 내적으로 구할 수 있고, 정반사 성분은 정점과 시야의 위치 차이와 확산 성분의 내적을 통해서 구할 수 있다. 다음, 구해진 확산 성분과 정반사 성분이 불연속적으로 바뀌도록 하여 *diffuseToon*와 *specularToon*을 계산한다. 이제 조명을 고려하여 출력 컬러를 계산한다. 각 요소는 확산 성분 F_D와 정반사 성분 F_S를 통해서 얻을 수 있다. 확산 성분 F_D는 Equation (3)으로 계산된다.

$$F_D = saturate(dot(N, L)) \tag{3}$$

여기서 L은 광원으로의 방향 벡터이고 N은 평면 노말로 다음과 같이 계산하여 얻는다.

$$L = -normalize(lightDirection)$$

$$N = normalize(input.normal)$$

한편, 정반사 성분 F_S는 Equation (4)로 계산된다.

$$F_S = pow(\max(0.0, dot(R, V)), shineness) \tag{4}$$

여기서 V은 정점에서 눈으로의 벡터이고 R은 빛이 평면에 부딪힌 후 방출되어 나가는 반사 벡터로 다음과 같이 계산하여 얻는다.

$$V = normalize(eyePosition - input.worldPos)$$

$$R = normalize(2.0 * N * dot(N, L) - L)$$

이제 실사 영상으로부터 생성된 컬러맵을 로드하여 3차원 모델의 출력 컬러와 컬러맵의 컬러를 Equation (5)의 유클리디안 거리 공식을 사용하여 거리 차이가 가장 적은 컬러를 최종 출력 컬러값으로 결정한다.

$$d = \sqrt{(r-r')^2 + (g-g')^2 + (b-b')^2} \tag{5}$$

3.3 CUDA 프로그래밍

분할 알고리즘을 CUDA를 이용하여 병렬 프로그래밍으로 구현하였다. CUDA 버전으로 수정된 분할 알고리즘은 Fig. 13과 같다.

CUDA 버전의 분할 알고리즘은 Fig. 9의 *SegmentImage* 알고리즘을 병렬화하여 고안되었다. 먼저, 영상과 그래프 크기만큼 *cudaMalloc* 함수를 이용하여 GPU 메모리에 공간을 할당한다. 다음, 정점 정보를 *cudaMemcpy* 함수를 이용하여 GPU 메모리에 복사한다. *BuildGraphCUDA()* 함수는 각 스레드당 한 개의 정점 연산을 수행한다. 이때의 영상의 좌표

Algorithm *SegmentImageCUDA*
Input Real Image I_I
Output Symbolic Image I_S
Begin
cudaMalloc()
cudaMemcpy(Host to Device)
$G,\ w \leftarrow BuildGraphCUDA(I_I)$
$G=(V,E)$ with n vertices and m edges
$E \leftarrow Sort(G,w)$
For q = 1 **to** m
$O_q \leftarrow (v_i, v_j)$
If$(C_i^{q-1} \neq C_j^{q-1})$ and $(w(O_q) \leq MInt(C_i^{q-1}, C_j^{q-1}))$ **then**
$S^{q-1} \leftarrow merging(C_i^{q-1}, C_j^{q-1})$
$S^q \leftarrow S^{q-1}$
EndFor
cudaMemcpy(Device to Host)
End

Fig. 13. Pseudo-Code of the CUDA Segmentation Algorithm

(x, y)의 정점 연산을 수행하는 스레드의 좌표는 ((*threadIdx.x* + *blockDim.x* * *blockIdx.x*), (*threadIdx.y* + *blockDim.y* * *blockIdx.y*))이다. 여기서, *threadIdx*는 블록 내에서의 스레드 위치를 나타내고 *blockIdx*는 그리드 내에서의 블록 위치를 나타내며 *blockDim*은 블록의 총 개수를 나타낸다.

CUDA를 이용하여 병렬 프로그래밍을 하려면 블록당 스레드 숫자와 블록 개수를 나누는 것이 중요하다. 즉 앞서 구축한 에지와 정점 정보 값을 GPU 메모리에서 효율적으로 병렬 처리하기 위하여 Fig. 14와 같은 알고리즘을 사용하여 블록과 블록당 스레드 숫자를 결정한다.

If $E_{TOTAL} > T_{MAX}$ **then**
$Block \leftarrow \dfrac{(E_{TOTAL} + T_{MAX} - 1)}{T_{MAX}}$
$Thread \leftarrow T_{MAX}$
Else $E_{TOTAL} \leq T_{MAX}$
$Block \leftarrow 1$
$Thread \leftarrow E_{TOTAL}$
EndIf

Fig. 14. Algorithm to Determine the Number of Blocks and the Number of Threads

스레드 단위로 나누어서 계산하려고 하는 정점이나 에지의 숫자가 스레드 최대 숫자보다 크다면 블록의 숫자는 정점이나 에지의 전체 숫자에 최대 스레드 수를 더하고 1을 뺀 뒤에 최대 스레드로 나눈 값이다. 이때의 한 블록당 가질 수 있는 스레드 숫자는 최대 스레드 숫자이다. 반면 정점이나 에지의 총 숫자가 최대 스레드보다 작을 경우에는 블록은 1개가 되며 스레드 숫자는 정점이나 에지의 총 숫자만큼 생성한다.

4. 실험 결과

4.1 실험 환경

본 연구의 실험 환경은 다음 Table 1과 같다.

Table 1. Experiment Environment

CPU	Intel i7-3770 3.50GHz
GPU	NVIDIA GeForce GTX 280
RAM	16.0GB
OS	Windows 7 Enterprise K
Library	OpenCV 2.4.8 DirectX SDK (June 2010)

본 실험에서 사용된 그래픽카드는 하나의 GPU 안에 그래픽스 프로세싱 아키텍처와 병렬 컴퓨팅 아키텍처가 모두 있는 2세대 SPA(Scalable Processor Array) 아키텍처를 기반으로 하고 있다.

4.2 NPR 결과

여러 단일 영상에 대해서 영상 분할을 실험하였다. Fig. 15는 영상 분할 결과를 보여준다. 각 영역은 영역의 평균 컬러값으로 채워서 표시하였다.

Fig. 15. Results of Image Segmentation

영상 분할이 끝나면 분할된 영상을 바탕으로 영상에 사용된 R, G, B 컬러값을 추출하여 일련의 컬러맵을 생성한다. Fig. 16은 위의 영상분할 실험에서 사용된 영상에서 추출해 낸 컬러맵을 보여준다. 영상 분할이 된 후 생성된 컬러맵에서의 컬러 수는 각각 32개, 15개, 25개, 15개이다.

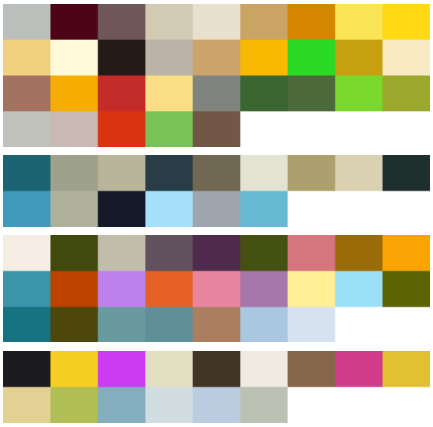

Fig. 16. Created Colormaps from Image Segmentation Results

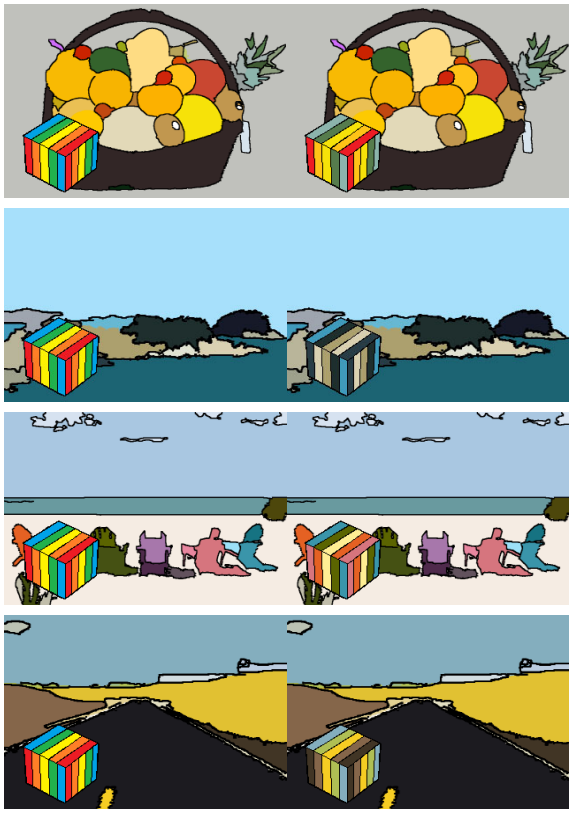

Fig. 17. Results of NPR Using the Colormaps

제안하는 NPR 알고리즘을 실제 영상과 3D 객체의 혼합 렌더링에 적용하였다. 혼합 렌더링에 적용한 결과가 Fig. 17에 있다. 띠무늬 형태의 큐브가 3D 객체에 해당한다. 왼쪽은 3D 객체에 컬러맵을 적용하기 전의 결과이고, 오른쪽은 영상으로부터 추출한 컬러맵을 3D 객체를 그릴 때 적용한 결과이다. 영상과 3D 모델을 각각 비사실적 렌더링을 적용하면 두 물체의 색상이 서로 어울리지 않는다. 반면 제안하는 방법은 컬러맵을 공유하여 렌더링하기 때문에 두 콘텐츠 간의 이질감을 줄여 조화로운 영상을 만들어낸다.

다음 Table 2는 NPR 렌더링으로 그리기 위해 줄어든 컬러의 개수를 보여준다. 제안 방법은 색상의 과감한 생략을 통해 비사실적 렌더링을 적절히 표현하였다.

Table 2. Comparison of the number of colors

	실험 1	실험 2	실험 3	실험 4
원본 컬러 수	10,649	9,565	11,103	9,102
NPR 컬러 수	32	15	25	15

4.3 실행 시간 비교

제안 방법의 실행 시간을 제안 방법과 가장 유사한 기존 방법인 Felzenszwalb 알고리즘의 경우와 비교하였다.

Table 3과 Fig. 18은 Felzenszwalb 알고리즘을 구현한 방법의 속도와 제안하는 방법을 CUDA로 구현한 방법의 속도를 영상 크기별로 측정한 표와 그래프이다.

Table 3. Comparison of the Execution Speed (Felzenszwalb vs. Proposed)

	0.12MP	0.27MP	0.48MP	0.75MP	1.1MP	1.5MP	2MP
F	1572	3752	6928	11241	16898	23661	31568
P	809	1302	1764	2469	3334	4335	5712

그래프에서의 x축은 영상 픽셀 수로 표현되는 영상 크기를 나타내고 y축은 1/1000초(ms) 단위의 실행시간을 나타낸다. Felzenszwalb 알고리즘을 사용하여 구현한 알고리즘은 영상 크기가 늘어날수록 속도가 급격하게 증가하는 반면에 CUDA를 이용하여 개발한 알고리즘은 증가폭도 적다. Felzenszwalb 알고리즘의 경우보다 모든 영상들에 대해 제안하는 알고리즘이 더 빠른 성능을 보인다.

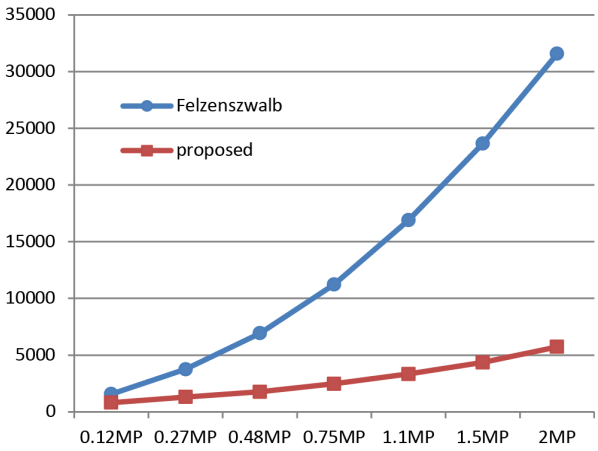

Fig. 18. Comparison of the Overall Execution Speed (Felzenszwalb vs. Proposed)

5. 결 론

실사 영상과 3D 객체의 혼합된 렌더링에 있어서 적용될 수 있는 비사실적 렌더링 기법의 연산에 GPGPU 환경인 CUDA를 이용한 알고리즘을 제안하였다. 제안 방법은 영상 분할 기법을 실사 영상에 적용하여 컬러맵을 생성하고 이를 3D 객체에 적용하는 방법으로 매우 자연스러운 NPR 결과를 생성한다.

계산량이 큰 영상 분할 과정의 고속화를 위해서 영상 분할 과정을 CUDA화하였다. 영상의 크기가 증가하는 경우에 CPU 버전은 영상의 크기에 비례하여 실행 속도가 증가하지만 CUDA 버전의 경우에는 실행 속도가 크게 증가하지 않았다. 실험 결과 전체적인 실행 속도가 크게 개선되었다.

제안 방법은 실사 영상을 배경으로 하고 그 위에 3차원 객체로 장면을 구성하는 가상현실 및 증강현실 환경에서의 렌더링 과정에서 효과적으로 사용될 수 있다.

References

[1] H. Lee and J. Choi, "Cartoon rendering techniques for real-time applications," *Korea Multimedia Society*, Vol.9, No.3-4, pp.54-64, 2005.

[2] P. Decaudin, "Cartoon-Looking Rendering of 3D-Scenes," *Research Report INRIA #2919*, Jun., 1996.

[3] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol.22, No.8, pp.888-905. 2000.

[4] Y. J. Zhang, "A survey on evaluation methods for image segmentation," *Pattern recognition*, Vol.29, No.8, pp.1335-1346. 1996.

[5] T. Son and K. Park, "Various line stylization of realtime video using the Image Difference and CUDA," *HCI'2012 Conference*, pp.382-385, 2012.

[6] S. Jang and H. Hong, "Color-based Stippling for Non-Photorealistic Rendering," *Journal of KIISE: Computer Systems and Theory*, Vol.33, No.2, pp.128-136, 2006.

[7] H. Joo, "A Study on 3D Cartoon Animation for Production in Non-Photo Realistic Rendering Technique," *Journal of The Korean Society for Computer Game*, Vol.26, No.4, pp. 179-185, 2013.

[8] H. Lee, S. Seo, and K. Yoon, "A Study on Saliency-based Stroke LOD for Painterly Rendering," *Journal of KIISE: Computer Systems and Theory*, Vol.36, No.3, pp.199-209, 2009.

[9] H. Jang, J. Jeon, and Y. Choy, "Processing Methods for Ink-and-Wash Painting in Mobile Contents," *Journal of the Korea Contents Association*, Vol.11, No.3, pp.137-146, 2011.

[10] S. Choi, "A Simple and Fast Algorithm for Real-time Pencil Strokes," *Journal of KIISE: Computer Systems and Theory*, Vol.33, No.6, pp.344-353, 2006.

[11] B. Gooch, and A. Gooch, "Non-photorealistic rendering," AK Peters, Ltd., 2001.

[12] A. Hertzmann, "Introduction to 3D non-photorealistic rendering: Silhouettes and outlines," *SIGGRAPH'99 Course*, pp.21-26, 1999.

[13] H. P. Narkhede, "Review of Image Segmentation Techniques," *International Journal of Science and Modern Engineering*, pp.2319-6386. 2013.

[14] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *International Journal of Computer Vision*, Vol.59, No.2, pp.167-181, 2004.

[15] Y. Y. Boykov and M. P. Jolly, "Interactive graph cuts for optimal boundary & region segmentation of objects in ND images," in *Proceedings of IEEE ICCV'2001*, Vol.1, pp.105-112, 2001.

[16] B. Fulkerson and S. Stefano, "Really quick shift: Image segmentation on a GPU," *Trends and Topics in Computer Vision*, Springer Berlin Heidelberg, pp.350-358, 2012.

[17] I. Buck, "GPU computing with NVIDIA CUDA," *SIGGRAPH'2007 Course*, 2007.

[18] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron, "A performance study of general-purpose applications on graphics processors using CUDA," *Journal of parallel and distributed computing*, Vol.68, No.10, pp. 1370-1380. 2008.

윤 현 철

e-mail : yhc8009@naver.com

2013년 인천대학교 컴퓨터공학부(공학사)

2013년~현 재 인천대학교 컴퓨터공학부 석사과정

관심분야 : 비사실적 렌더링, 증강현실, 카메라 추적

박 종 승

e-mail : jong@inu.ac.kr

1992년 경북대학교 전자계산학과(이학사)

1994년 POSTECH 컴퓨터공학과 (공학석사)

1999년 POSTECH 컴퓨터공학과 (공학박사)

2004년~현 재 인천대학교 컴퓨터공학부 교수

관심분야 : 게임공학, 영상처리, 증강현실