움직이는 오브젝트를 위한 실시간 비디오기반 재조명 기술

비주얼 헐 오브젝트를 이용한 실시간 영상기반 재조명 기술

Real-time Video Based Relighting Technology for Moving Object

유세운, Saewoon Ryu*, 이상화, Sanghwa Lee**, 박종일, Jongil Park*
*Department of Electrical and Computer Engineering, Hanyang University,
**School of Electrical Engineering, Seoul National University

요약 본 논문은 비주얼 헐 오브젝트를 이용한 움직이는 오브젝트에 대한 실시간 영상기반 라이팅 기술을 제안한다. 본 논문에서는 특히 서로 다른 공간상의 조명 환경을 일치시키는 기술에 중점을 두고, 실시간으로 움직이는 오브젝트의 실시간 비디오 기반 재조명 기술로서 3가지 핵심 내용을 소개한다. 첫째는 비주얼 헐 데이터를 기반으로 기존에 벡터의 외적을 사용하던 방법을 개선하여 수식을 근사화시켜 연산량을 줄여서 고속으로 노말 벡터를 추출하는 방법이고, 둘째는 사용자 주변 조명 환경 정보를 효과적으로 샘플링하여 라이팅에 사용하는 점광원의 개수를 줄였으며, 세 번째는 CPU와 GPU의 연산량을 분배하여 효과적으로 병렬 고속 연산이 가능하도록 하였다. 종래의 영상기반 라이팅 기술이 정지된 환경맵 영상을 사용하거나 정지된 객체를 라이팅하였던 연구를 한 반면에 본 논문은 실시간에서 라이팅을 구현하기 위한 기술로서 고속 라이팅 연산을 위한 방법을 제시하고 있다. 본 연구의 결과를 이용하면 영상기반 라이팅 연구의 실제적이고도 폭넓은 적용이 가능할 것으로 사료되며 고화질의 콘텐츠 양산에도 기여할 것으로 사료된다.

핵심어: Image Based Lighting, Real-time, Tangible Space,

1. 서론

본 논문은 실시간 3차원 아바타 기술[1]을 이용하여 움직이는 모델에 대한 라이팅[2]기법을 개선 발전시켜 실시간 3차원 아바타의 실시간 라이팅 기술을 접목한 실감 회의 시스템 기술을 제안한다. 그림 1은 실감 원격회의 시스템의 개념도를 나타낸다.

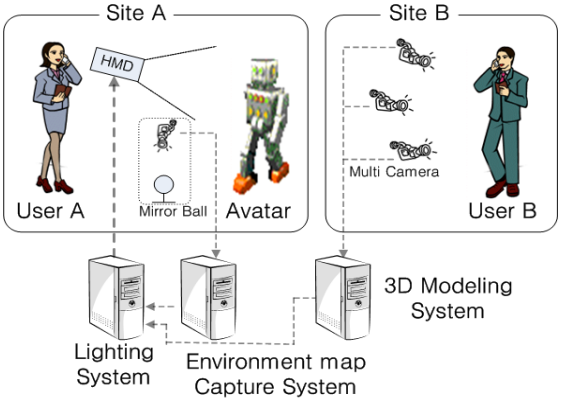


그림 1. 실감 원격회의 시스템 개념도.

일반적으로 라이팅은 객체를 3D모델로 사용하는 방식과 실제 모델에 광원을 라이팅하는 방식으로 구분된다. 객체를 3D모델로 라이팅 할 때에 뷰어 시점이 자유로운 장점이 있지만 3D 모델 표면에서 광 경로를 추적하는(ray tracking)하는 연산량이 매우 큰 단점이 있다. 실제 모델에 광원을 라이팅 하는 방식은 실제 광원이 라이팅하는 비디오영상을 촬영하므로 연산량이 적은 장점이 있지만, 특별한 조명장치가 갖춰진 스튜디오 안에서만 사용할 수 있는 단점이 있다.

본 논문은 실시간으로 서로 다른 두 공간의 조명 환경을 일치시켜 보여주는 실시간 라이팅 기술을 제안한다. 이를 위하여 제안하는 시스템은 3개의 장치로 구성한다. 첫째는 실시간 환경맵을 캡쳐하는 장치이고, 둘째는 실시간 모델정보를 취득하여 가공하는 장치이고, 셋째는 실시간으로 라이팅 영상을 생성하는 장치이다.

첫번째 환경맵을 취득하는 장치는 일반적으로 미러볼을 카메라로 촬영하여 사용하는 방식[3]이 많이 사용되고 있다. 환경맵을 취득하는 또다른 방법으로 어안렌즈를 장착한 카메라 영상 캡쳐방식[4]이 사용된다. 미러볼을 사용하는 방식은 미러볼 중간에 카메라 영상이 촬영되고 미러볼 뒷면의

환경 영상을 캡쳐할 수 없는 단점이 있지만 어안랜즈를 사용하는것에 비해 더 넓은 화각을 도출할 수 있는 장점이 있다. 반면 어안랜즈를 사용하는 방식은 미러볼보다 화각이 좁은 단점이 있지만, 순수한 환경맵을 바로 캡쳐할 수 있고, 이동하면서 환경맵을 캡쳐할 수 있는 장점이 있다. 본 논문에서 제안하는 방식은 사용자가 공간을 크게 움직이지 않는 것을 가정하고 장치 배치에 편리한 미러볼 캡쳐방식을 사용한다.

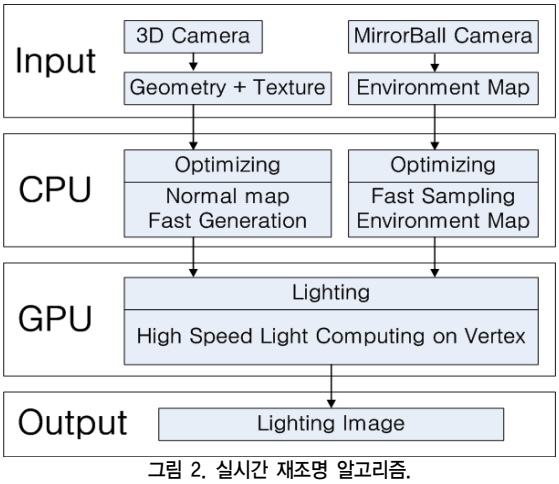


그림 2. 실시간 재조명 알고리즘.

두번째 모델정보를 취득하여 가공하는 장치는 본논문에서 실시간 3D아바타[6]의 데이터를 사용한다. 여기서 모델정보는 depth map과 texture정보이다. 이를 랜더링 하는 방식은 일반적으로 point rendering과 유사한다. 이 방법은 사실적인 표현을 위한 라이팅 연산시 point하나하나에 대한 광원의 조명 경로를 연산해야 한다.[5], 그리고 입력 데이터가 크기 때문에 GPU를 이용하여 렌더링하는 방법이 연구되고 있다.[6] 일반적으로 point 로 구성된 데이터는 모델의 해상도를 자세하게 나타낼 수 있는 장점이 있지만 그만큼 잡음에 민감하여 노이즈가 클수록 영상의 화질이 급격히 떨어지는 단점이 있다. 이에 따라 노이즈를 제거하고 방대한 데이터를 최적화해서 용량을 줄이는 방법을 사용한다. [7]

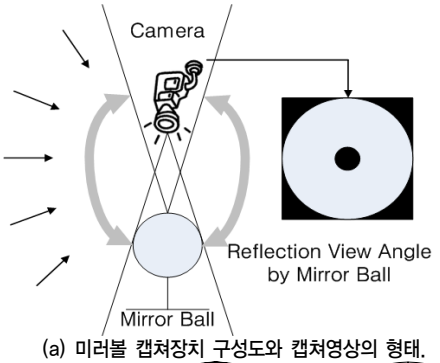
세번째 실시간으로 라이팅 영상을 생성하는 장치는 CPU와 GPU가 연동되는 연산장치이다. 컴퓨터 그래픽스의 고속 연산 방법에 관련한 연구가 활발해지면서 GPU를 활용한 다양한 응용사례들이 제시되고 있다. GPU를 이용하여 비디오 디코딩에 사용하는 방법[8]과 유한요소 해석에의 적용[9], 그리고 고속처리의 볼륨 렌더링[10]등등 다양한 분야에 사용되고 있다. 본 논문에서는 볼륨 렌더링에 사용되는 대용량의 모델 정보를 실시간으로 고해상도 환경맵을 광원으로 라이팅 연산하므로 매우 큰 컴퓨팅 능력을 필요로 한다. 효과적으로 거대한 데이터를 처리하기 위하여 CPU는 데이터 integration작업을, GPU에서는 라이팅 process를 구분하여 처리한다. GPU에서 라이팅 처리를 할 때 조명환경맵의 방향벡터 계산 및 모델정보의 최적화 및 라이팅 연산을 수행한다. 그림 2는 실CPU와 GPU를 이용하여 실시간 재조명 알고리즘을 나타낸다.

2. 실시간 재조명 시스템 구성

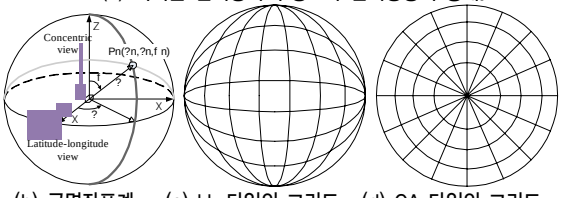
2.1 실시간 환경맵 캡처링

미러볼 이미지의 원형 부분을 추출하여 구형 프로브로 모델링한다. 그림 3은 미러볼을 캡쳐하여 샘플링 한 후 3차원 광원정보를 추출하는 방법을 나타낸다. 그림 3의 (a)는 미러볼 캡쳐장비의 구성 방법과 캡쳐되었을 때 영상의 모습을 나타낸다. 미러볼 영상의 중심부분에 촬영 카메라의 이미지가 함께 캡쳐된다.

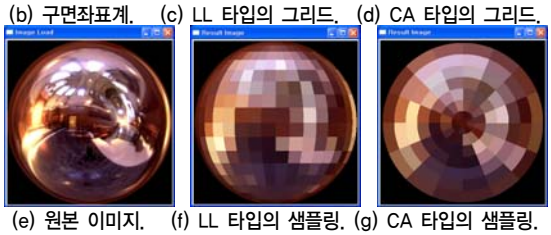
촬영된 영상은 그림 3의 (b)와같이 구형좌표계로 계산한다. 미러볼 영상을 이용하여 광원의 개수를 최소화 하면서 광원의 균일한 분포를 유지할 수 있는 효과적으로 샘플링하는 2가지 방법을 제안한다. 첫째는 그림 3의 (c)와같이 구면좌표계를 균일한 위도와 경도로 나누는 (LL: Latitude -Longitude) 방법인데, LL방식은 가운데 부분에서는 균등한 광원의 분포를 보이지만 가장자리 부분에서 다소 불균등한 점광원 분포를 갖는다. 두 번째는 그림 3의 (d)와 같이 영상의 중심을 기준으로 동심원과 균일 각도로 구간을 나누는 (CA: Concentric-Angular) 방법으로서, CA방식은 가장자리 부분의 점광원을 균등하게 분포하지만, 정면에는 셈플링 간격이 커서 광원 정보가 줄어드는 특징이 있다. 그림 3의 (e)를 이용한 각각 샘플링 결과는 (f), (g)와 같다.



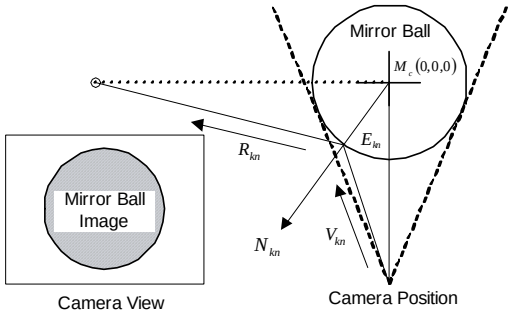
(a) 미러볼 캡쳐장치 구성도와 캡쳐영상의 형태.



(b) 구면좌표계. (c) LL 타입의 그리드. (d) CA 타입의 그리드.



(e) 원본 이미지. (f) LL 타입의 샘플링. (g) CA 타입의 샘플링.



(h) 3D 광원 입사 방향 벡터 추출.

그림 3. 환경맵 샘플링 및 3차원 광원 입상 방향벡터 추출 과정.

샘플링 영상으로부터 각 셀의 영역의 평균값을 계산하고 그 값을 광원의 칼라값으로 결정하고, 광원의 3차원 방향정보는 그림 3의 (h)와 같은 방법으로 미러볼의 반사 경로를 추적하여 계산하다. 표 1은 3차원 광원 방향 벡터 추출과정을 수식으로 표현한 것이다.

표 1. 3차원 광원 방향 벡터 추출 수식.

$$\text{Environment Map}: E_{kn}(x_e, y_e, x_e)$$
$$\text{Spherical Coordinate}: (\rho, \phi, \theta)$$
$$\text{Cartesian Coordinate}: (x, y, z)$$
$$\text{cart} \leftrightarrow \text{sph} \quad \begin{cases} x = \rho\cos\theta\sin\phi \\ y = \rho\sin\theta\sin\phi \\ z = \rho\cos\phi \end{cases}$$

$$E^s_{kn}(\rho, k\phi, n\theta) \xrightarrow{\text{cartesian}} E^c_{kn}(x_e, y_e, x_e)$$
$$E^c_{kn}(\rho\cos n\theta\sin k\phi, \rho\sin n\theta\sin k\phi, \rho\cos k\phi)$$
$$\left(k, n: \text{interval}, \phi, \theta: 1, 2, \cdots, \frac{360}{k \text{ or } n}\right)$$

이때 E^c_{kn} 는 광원의 반사 벡터를 의미한다.

$$\text{Camera Position}: C_p(x_c, y_c, x_c) = (0,0,-d)$$
$$\text{Environment Map}: E_{kn}(x_e, y_e, x_e)$$
$$\text{MirrorBallCenter}: M_c(0,0,0)$$
$$\text{Refletion Vector}: \overrightarrow{R_{kn}}(x_r, y_r, x_r)$$

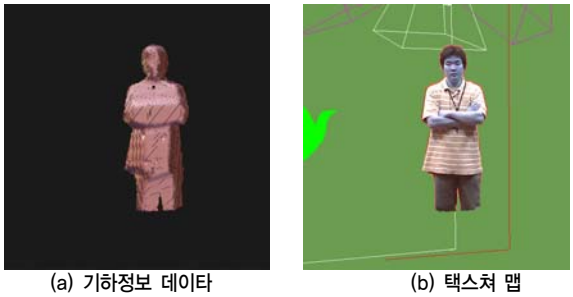
$$\overrightarrow{V_{kn}} = E_{kn} - C_p = \overrightarrow{V_{kn}}(x_e - 0, y_e - 0, x_e + d)$$
$$\overrightarrow{N_{kn}} = E_{kn} - M_c = \overrightarrow{N_{kn}}(x_e - 0, y_e - 0, x_e - 0)$$
$$\overrightarrow{R_{kn}} = \overrightarrow{V_{kn}} - 2(\overrightarrow{V_{kn}} \cdot \overrightarrow{N_{kn}})\overrightarrow{N_{kn}} = \overrightarrow{R_{kn}}(x_r, y_r, x_r)$$

$\overrightarrow{V_{kn}}$, $\overrightarrow{N_{kn}}$ 와 $\overrightarrow{R_{kn}}$ 을 정규화한 결과를 나타낸다.

$$\text{Normalize}\overrightarrow{V_{kn}}: \frac{\overrightarrow{V_{kn}}}{\|\overrightarrow{V_{kn}}\|} = \frac{1}{\sqrt{x_e^2 + y_e^2 + (x_e + d)^2}}\overrightarrow{V_{kn}}$$
$$\text{Normalize}\overrightarrow{N_{kn}}: \frac{\overrightarrow{N_{kn}}}{\|\overrightarrow{N_{kn}}\|} = \frac{1}{\sqrt{x_e^2 + y_e^2 + z_e^2}}\overrightarrow{N_{kn}}$$
$$\text{Normalize}\overrightarrow{R_{kn}}: \frac{\overrightarrow{R_{kn}}}{\|\overrightarrow{R_{kn}}\|} = \frac{1}{\sqrt{x_r^2 + y_r^2 + z_r^2}}\overrightarrow{R_{kn}}$$

2.2 실시간 사용자 모델 최적화

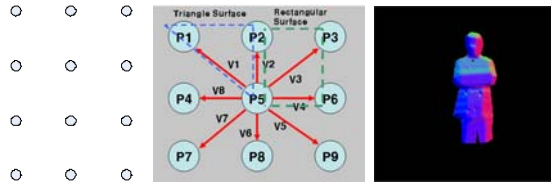
그림 4는 입력 모델을 나타낸다. 그림 4의 (a)는 기하정보를 타나내는 depth map을 나타낸다. 본 논문에서 사용하는 모델정보는 비주얼 헐로부터 도출된 depth map을 이용하는데, 노이즈 제거를 위해 low-pass 필터링을 거쳐서 사용한다.



(a) 기하정보 데이타 (b) 택스쳐 맵

그림 4. 입력하는 모델의 기하정보 데이터와 텍스쳐 맵.

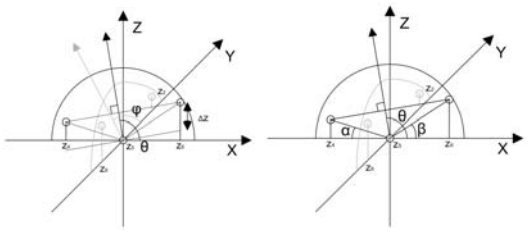
그림 5는 depth map모델 정보로부터 법선벡터를 추출하는 일반적인 방법을 나타낸다. 그림 5의 (a)는 depth map을 확대하여 모델이 픽셀들로 구성된 모습을 보여준다. 그림 5의 (b)는 가운데 픽셀을 기준으로 인접한 8개 픽셀들 사이에 가상의 선을 구성하여 법선벡터를 추출하기 위한 평면을 구성하는 과정이다. 그림 5의 (c)는 depth map의 모든 픽셀에 대하여 법선텍터를 추출하고 이것을 영상으로 표현한 결과이다. 모델을 라이팅할 때 depth map과 법선벡터 맵을 입력하여 계산을 수행한다.



(a) 픽셀 구성 (b) 단위벡터 (c) 법선벡터 맵 구성

그림 5. Depth map으로부터 법선벡터 추출 방법

실시간 라이팅을 위하여 모델을 최적화하는 법선벡터 추출과정에서 연산량을 줄일 필요성이 있다. 그림 6은 depth map을 픽셀의 2차원 좌표계에서 depth map의 고도값을 이용하여 3차원 정점(vertex)개념으로 표현하고, 인접한 정점간의 관계를 통하여 고속으로 법선벡터를 추출하는 2가지 방법을 제시한다.



(a) 고도차이를 이용한 방법 (b) 사잇각을 이용한 방법
그림 6. depth map으로부터 법선벡터 추출 방법

그림 6의 (a)는 가운데 버텍스를 기준으로 서로 마주보는 버텍스를 연결하는 선분과 가운데 버텍스가 포함되는 수평면을 이용하여 가상의 삼각형을 그릴 수 있는데, 이 삼각형의 사잇각에 직각이 되는 부분이 법선벡터가 된다. 그림 6의 (b)는 가운데 버텍스를 기준으로 인접한 버텍스간에 가상의 삼각형을 그리고 삼각형의 사잇각을 도출한다. 마주보는 삼각형의 사잇각을 이용하여 법선벡터를 도출 할 수 있다.

이때 삼각형의 사잇각을 추출하기 위해 삼각함수를 계산하는 과정이 필요한데, 이것은 미리 계산하여 결과값들을 메모리에 저장하였다가 삼각함수 계산값이 필요할 때마다 메모리에서 읽어들여서 사용하는 방식을 채택한다. 표 2는 제시된 두가지 방법을 이용하여 법선텍터를 고속 추출하는 과정을 수식으로 표현하였다. 이 방법들을 사용하면 일반적으로 법선벡터 추출에 사용되는 방식에 비하여 계산에 사용되는 연산자의 숫자가 획기적으로 줄어들게 되어 연산속도가 향상된다.

표 2. 법선 벡터 고속 추출 방법 수식.

(a): 인접한 버텍스의 고도차이를 이용하는 방법.

Number of operation: (-)연산자 6개, (+)2개, (/)2개.

$$\Delta z = \frac{z_6 - z_4}{2},\ z_6' = z_6 - \Delta z,\ \Delta z' = \frac{z_2 - z_8}{2},\ z_2' = z_2 - \Delta z'$$

$$\theta = \frac{\pi}{2} + \tan^{-1}(z_6' - z_5) \qquad \varphi = \frac{\pi}{2} + \tan^{-1}(z_2' - z_5)$$

(b): 버텍스와 중심의 사잇각을 이용하는 방법.

Number of operation: (-)연산자 8개, (/)연산자 2개.

$$\alpha = \tan^{-1}\left(\frac{z_4 - z_5}{1}\right)\ \beta = \tan^{-1}\left(\frac{z_6 - z_5}{1}\right)\ \alpha' = \tan^{-1}\left(\frac{z_2 - z_5}{1}\right)\ \beta' = \tan^{-1}\left(\frac{z_8 - z_5}{1}\right)$$

$$\theta = \frac{\pi - \alpha - \beta}{2} \qquad \varphi = \frac{\pi - \alpha' - \beta'}{2}$$

이때 법선벡터는 다음과 같이 표현된다.

$$N(x, y, z) = (\cos\theta, \cos\varphi, \sin\theta \cdot \sin\varphi)$$

2.3 GPU상에서 고속재조명 연산

모델의 영상은 모델 표면의 텍스쳐 색상이다. 사실적인 랜더링 영상을 도출하기 위하여 재조명 반사모델은 중요하다. 본 논문에서는 이색성 반사 모델(DRM: Dichromatic Reflection Model)에 기반을 둔 색상 모델을 가정하여 재조명을 연산을 수행한다. DRM은 난반사(diffuse) 물체에서의 반사 스팩트럼과 전반사 물체에서의 반사 스팩트럼이 다르다. 이것은 전반사의 스팩트럼 분포는 환경맵의 광원의 분포와 비슷하지만, 난반사(diffuse reflection)의 스팩트럽 분포는 객체 표면의 색상과 광원의 색상으로 표현된다는 것을 의미한다. 이것은 같은 광원에 의해 재조명 되더라도 각각 난반사 영상의 RGB벡터 성분은 같은 비율로 변화화지 않는 것을 의미한다[12, 13].

결국 관측되는 색상값은 표면에서 반사되는 색상(난반사 성분: diffuse component) 벡터와 광원의 색상(전반사 성분: specular component)벡터의 선형 조합으로 표현될 수 있다. 전반사 성분의 색상 벡터는 광원에 병행된다. 그리고 난반사 성분의 색상 벡터는 객체 표면의 색상에 의해 순수한 난반사 성분으로 결정된다. 반사모델에 의해 관측되는 색상값 I는 수식 (1)과 같이 표현된다.

$$I = I_d + I_s + k_a I_a \tag{1}$$

이때 I_d와 I_s는 각각 난반사(diffuse)와 전반사(specular) 성분이다. 그리고 I_a는 물체 주변에서 k_a의 상수값 비율로 반사되는 주변광(ambient)을 의미한다.

난반사 성분만을 정의하면 식 (2)와 같다.

$$I_d = k_d \cos\theta I_D \tag{2}$$

이때 k_d는 객체 표면에서 난반사되는 비율을 나타내는 상수값이고, I_D는 순수한 난반사 성분을 의미하고, 객체표면의 텍스쳐 색깔에 의해 결정된다. θ는 입사광 l_i와 표면에서의 법선벡터와의 사잇각을 의미한다. 전반사 성분은 퐁의 음영법(Phong' s shading)에 기반하여 계산한다. 전반사 성분은 $\cos^n\phi$의 비례로 되는데 ϕ는 객체 표면에서 반사되는 광선과 시점 방향사이의 각도를 의미한다. k_s는 표면에서 전반사 성분이다. 관측되는 색상은 객체 표면에서 반사되어 난반사, 전반사, 주변광 성분들의 조합으로 이루어진다. 재조명 영상의 각 픽셀의 색상값은 캡쳐된 환경맵의 광원정보로부터 객체의 3차원 표면상에서 반사모델에 의해 재조명 연산과정을 거쳐 식 (3)과 같이 도출된다.

$$I = k_a I_a + k_d \cos\theta I_D + k_s \cos^n\phi I_i \tag{3}$$

그림 7은 실시간 재조명 알고리즘의 처리과정을 나타낸다. 고속으로 라이팅 영상 합성을 위해서 CPU와 병행하여 GPU를 최적화하여 연산한다. CPU는 주로 데이터 입출력 처리에 사용하고, GPU는 조명환경 정보처리 연산과정과 모델 정보 최적화 과정과 라이팅 연산을 수행한다. 라이팅 함수는 Phong[11]의 방법을 사용하고 CPU와 GPU의 연산량 분배를 통해 고속 라이팅 연산이 가능하다.

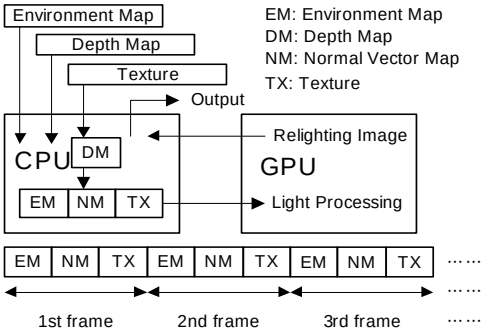
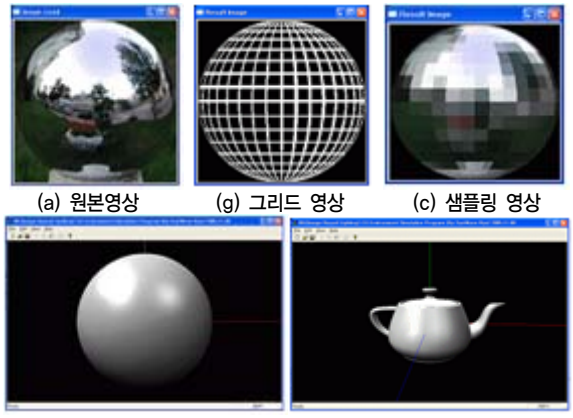


그림 7. 실시간 재조명 알고리즘의 CPU와 GPU의 연산과정.

3. 실험결과

그림 8은 환경맵을 캡쳐하여 샘플링하고 기본적인 모델에서 재조명 시뮬레이션을 수행한 결과이다. 그림 8의 (a)는 환경맵의 원본 영상이고, (b)는 샘플링 그리드 영상, (c)는 샘플링된 영상이다. (d)는 구형 모델을 재조명한 결과이며, (e)는 주전자모델을 재조명한 결과이다.

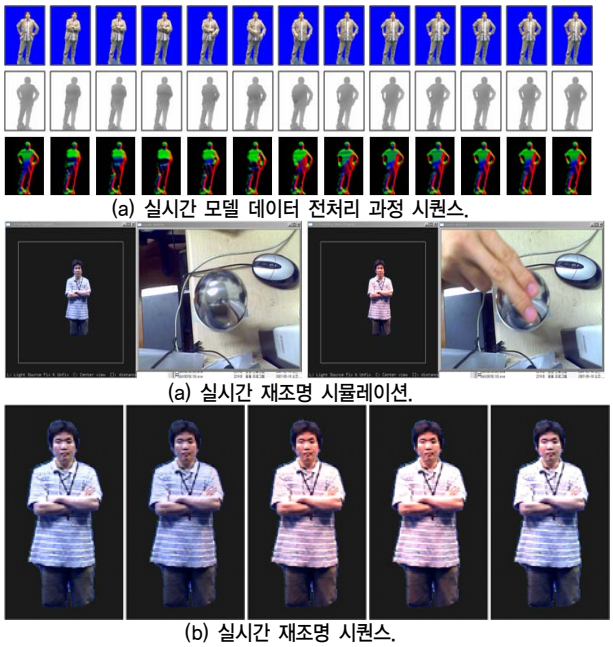


(a) 원본영상 (g) 그리드 영상 (c) 샘플링 영상

(d) 구형 모델 재조명 (e) 주전자 모델 재조명

그림 8. 환경맵의 샘플링 및 재조명 시뮬레이션 결과.

그림 9는 3차원 아바타 모델과 실시간 캡쳐되는 비디오 환경맵을 이용하여 실시간 재조명한 시뮬레이션 결과를 나타낸다. 그림 9의 (a)는 실시간 모델데이터의 depth map으로부터 법선벡터맵을 고속으로 추출하는 장면이며, 그림 9의 (b)는 미러볼 위에 물체를 움직여 환경맵을 변화시켰을 때 재조명 결과를 나타낸 것이며, 그림 9의 (c)는 변화하는 조명환에 대해 실시간 재조명 결과를 나타낸다.



(a) 실시간 모델 데이터 전처리 과정 시퀀스.

(a) 실시간 재조명 시뮬레이션.

(b) 실시간 재조명 시퀀스.

그림 9. 실시간 재조명 데모 영상.

표 3은 실시간 재조명 시스템의 CPU, GPU성능 및 입출력 데이터의 성능을 나타낸다.

표 3. 실시간 재조명 시스템 성능

Item	Performance and Spec.
CPU(Centeral Process Unit)	Intel Pentium4 3GMHz
GPU(Graphic Process Unit)	Nvidia GeForce 7900
Visaul Hull (depth map speed)	512*512, 30 (Frame/s)
Real-time Environment Map	640*480, 60 (Frame/s)
Real-time Normal Vector	30~25 (Frame/s)
Real-time Relighting	512*512, 15~20 (Frame/s)

4. 결론

종래의 영상기반 라이팅 기술이 정지된 환경맵 영상을 사용하거나 정지된 객체를 라이팅하였던 연구를 한 반면에 본 논문은 실시간에서 라이팅을 구현하기 위한 시도로서 고속 라이팅 연산을 위한 방법을 제시하고 있다. 본 연구의 결과를 이용하면 영상기반 라이팅 연구의 실제적이고도 폭넓은 적용이 가능할 것으로 사료되며 고품질의 콘텐츠 양산에도 기여할 것으로 사료된다.

참고문헌

[1] Sang-Yup Lee, Sang C. Ahn, Hyoung-Gon Kim, MyoTaeg Lim, "Real-time 3D video avatar in mixed reality: an implementation for immersive telecommunication", Source Simulation and Gaming archive Volume 37, Issue 4 (December 2006) table

of contents Symposium: virtual reality simulation, pp. 491 – 506, 2006.

[2] Charles-Felix Chabert, Per Einarsson, Andrew Jones, Bruce Lamond , Wan-Chun Ma, Sebastian Sylwan, Tim Hawkins, Paul Debevec, "Relighting human locomotion with flowed reflectance fields", International Conference on Computer Graphics and Interactive Techniques, ACM SIGGRAPH 2006 Sketches, people, puppets & pillows table of contents, Article No. 76, 2006.

[3] M. Kanbara, N. Yokoya, "Real-time estimation of light source environment for photorealistic augmented reality", Pattern Recognition, ICPR 2004. Proceedings of the 17th International Conference, pp.911 – 914 Vol.2, Aug. 2004.

[4] Li Shigang., "Full-View Spherical Image Camera" Pattern Recognition, ICPR 2006. 18th International Conference., Volume 4, pp.386 – 390, 2006.

[5] M. Botsch, M. Spernat, L. Kobbelt, "Phong splatting", Eurographics Symposium on Point Based Graphics, pp. 25~32., 2004.

[6] M. Botsch, A. Hornung, M. Zwicker, L. Kobbelt, "High-quality surface splatting on today's GPUs", Point-Based Graphics, Eurographics/IEEE VGTC Symposium Proceedings., pp. 17-24, June 2005.

[7] D. Cohen, J., G. Aliaga, D., Zhang Weiqiang, "Hybrid simplification: combining multi-resolution polygon and point rendering", Visualization, 2001. VIS '01. Proceedings, pp. 37 – 539, Oct. 2001.

[8] B. Pieters, D. Van Rijsselbergen,; W. De Neve, R. Van de Walle, "Motion Compensation and Reconstruction of H.264/AVC Video Bitstreams using the GPU", Image Analysis for Multimedia Interactive Services, WIAMIS '07. Eighth International Workshop, pp. 69 – 69, June 2007.

[9] S.E. Krakiwsky,, L.E. Turner, M.M. Okoniewski, "Graphics processor unit (GPU) acceleration of finite-difference time-domain (FDTD) algorithm", Circuits and Systems, 2004. ISCAS '04. Proceedings of the 2004 International Symposium on Vol. 5, pp.265-268, May 2004.

[10] Kyung-Seok Oh, Chang-Sung Jeong, "Acceleration technique for volume rendering using 2D texture based ray plane casting on GPU", Computational Intelligence and Security, 2006 International Conference, Vol. 2, pp.1755-1758, Nov. 2006.

[11] R. L. Cook and K. E. Torrance, "A Reflectance Model for Computer Graphics", Proceedings of SIGGRAPH 81, pp. 307~316. 1981.

[12] S. Shafer, "Using color to separate refelctance components". Color Research and Applications. pp.210.218, Aug. 1985.

[13] S. Lin, and S.W. Lee, "A representation of specular appearance". Proc. of ICCV. 1999.