GPU에서 픽셀 분산을 이용한 실시간 필드심도 렌더링

이성길* · 김정현** · 최승문***

Real-Time Depth of Field Rendering using Pixel Intensity Scatter on GPU

Sungkil Lee* · Gerard Jounghyun Kim** · Seungmoon Choi***

요 약

픽셀 분산을 이용한 필드심도 렌더링 방법은 고품질의 이미지를 생성하지만, 하드웨어 가속방법의 부재로 실시간에 구현되지 못해왔다. 본 논문에서는 GPU의 포인트 스프라이트를 이용하여 픽셀 분산을 실시간에 구현한다. 분산된 픽셀의 강도는 알파 블렌딩을 통해 누적되고, 누적된 이미지를 정규화 함으로써 블러링 효과가 얻어진다. 또한, 깊이정렬이 없는 알파 블렌딩에서 발생하는 문제점인 전경 블러링의 불연속과 배경에서의 강도 침투 현상을 해결한다. 본 방법은 기존 실시간 필드심도 렌더링 방법들의 주요 결점이 없는 고품질의 필드심도 이미지를 생성한다.

Abstract

Despite high image quality of the intensity scatter method for depth of field (DOF) effects, its hardware acceleration has remained difficult due to the lack of routines/constructs amenable to simulate scattering. This paper presents a real-time DOF rendering method based on the intensity scatter using point sprites supported in the GPU as a primary blurring method. We use alpha blending for scattered intensity accumulation and resolve two accompanied problems, the foreground blurring discontinuity and the background intensity permeation. Our method does not produce any major artifacts common in previous real-time methods.

Key Words : Depth of Field, Forward-Mapping, GPU, Rendering, Post-Filtering

1. 서 론

필드심도(Depth of Field)는 렌즈 시스템에서 초점이 맞아 또렷하게 보이는 거리의 범위를 지칭한다. 실제 렌즈 시스템은 유한한 조리개를 가지고 있어 3차원의 점이 CoC(Circle of Confusion, 착란원)라고 불리는 원형의 블러링(blurring) 영역에 맺히게 된다 [1]. 필드심도 효과는 사실적인 이미지 품질 및 인간의 깊이 인식에 도움이 된다고 알려져 있다. 컴퓨터 그래픽스에서는 핀홀 카메라 모델을 쓰므로, 필드심도 효과를 생성하기 위한 몇몇 방법들이 근래에 제안되었다.

기존의 필드심도 렌더링 방법들은 크게 다중렌더링(multi-pass rendering)에 기반을 둔 정확한 방법들과 후처리(post-filtering) 기반의 효율적인 방법들로 나뉠 수 있다. 전자는 렌즈 내에서 여러 곳을 샘플링하여 렌더링한 후 누적하여 블러링 효과를 얻어 낸다[2,3]. 후처리 기반의 방법들은 부분가림(partial occlusion) 효과 없이 필드심도 효과를 근사한다.

이러한 후처리 방식 중, 실시간 필드심도 렌더링을 위한 기존방법들은 대부분 GPU (Graphics Processing Unit) 기반의 픽셀수집(pixel gather)에 기반한다 [4,5,6]. 이런 방법들은 픽셀 CoC 영역에 해당하는 주변 픽셀들의 강도를 모아서 블러링을 시뮬레이션 한다. 그러나, 이러한 접근방법은 CoC를 만드는 원래(source) 픽셀이 아니라, 맺히는(destination) 픽셀의 CoC를 기준으로 블러링을 하기 때문에 옳지 않은 방법이다. 또한, 전경에서의 불연속성과 배경에서의 강도의 누출과 같은 근본적인 결점을 내포하고 있어, 실용적으로 활용되기에 문제가 있다 [7].

수집에 기반한 방법론을 대체할 수 있는, 또 다른 후처리방법은 픽셀분산(pixel scatter)이다 [1,7]. 픽셀분산에 의한 이미지는 많은 시간을 필요로 하는 다중렌더링의 결과와 비슷한 정도의 품질을 생성할 수 있다. 픽셀분산 방법은 핀홀 모델에 의해서 렌더링된 픽셀을 CoC의 영역만큼의 스프라이트(sprite)로 확장하여, 그 주변 픽셀들에 누적함으로써 블러링 효과를 생성한다. 이렇게 함으로써, 원리적으로 올바르며, 전경 불연속성과 배경 강도 누출의 두 가지 문제를 극복할 수 있다.

본 논문에서는 픽셀분산의 이점을 살리면서, GPU로 가속하여 실시간에 처리 가능한 필드심도 렌더링 방법을 제안한다. 픽셀분산 방법은 몇 가지 문제점 때문에 GPU에서 구현되지 못하고, 주로 소프트웨어로 구현되어 왔다. 이에, 소프트웨어 렌더링은 많은 시간을 필요로 하고, 실시간에 사용되지 못해왔다 (이미지 당 몇 분 정도) [7,8]. GPU에서 구현되지 못한 주요 한계는 픽셀분산을 위한 하드웨어 구조/지원, 하드웨어 깊이 정렬, 고정확도의 이미지 누적 등의 지원 부재이다 [8]. 근래에 고정확도의 이미지 누적은 지원되게 되었지만, 여전히 앞의 두 문제는 실시간 구현을 가로막고 있다. 이에, 저자는 픽셀분산을 GPU에서 구현하기 위하여 소프트웨어 스프라이트를 하드웨어 포인트 스프라이트 (hardware ponit sprite)[9]로 대체하여 첫 번째 문제점을 해결하고, 픽셀의 깊이를 이용하여 깊이 정렬의 부재로 인한 문제점을 해결한다.

2. 일반적인 필드심도 모델

본 논문에서 제안하는 필드심도 렌더링 방법은 Potmesil [1]에 의해 제안된 필드심도 모델을 사용한다 (그림 1). 핀홀카메라 모델을 이용하여 렌더링 된 깊이 이미지(depth image)의 각 픽셀에 대해서 CoC의 지름을 구해내고 이를 필드심도 렌더링을 위한 블러링의 정도로서 이용한다.

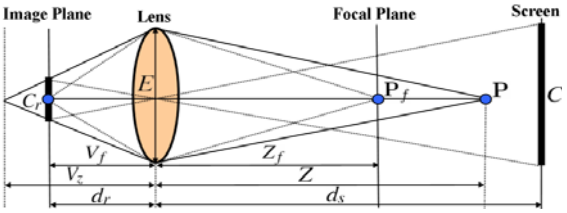

그림 1. 필드심도 모델 [1].

그림 1에서, **P**의 이미지 깊이 V_z와 초점이 맞은 $\mathbf{P}_f$의 이미지 거리 V_f는 다음과 같이 유도될 수 있다.

$$V_z = \frac{FZ}{Z-F} \quad \text{and} \quad V_f = \frac{FZ_f}{Z_f-F}$$

수식에서 F는 렌즈의 초점거리, Z는 깊이, Z_f는 초점이 맞은 깊이를 나타낸다. 위 식을 이용하여 이미지 평면의 CoC는 C_r은 렌즈의 지름 E를 이용하여 다음과 같다.

$$C_r = |V_z - V_f|\frac{E}{V_z} = \left(\frac{EF}{Z_f-F}\right)\frac{|Z-Z_f|}{Z}$$

스크린 상의 CoC C는 스크린/모니터와의 거리를 이용하여 다음과 같이 쓸 수 있다.

$$C = C_r \frac{d_s}{d_r} DPI$$

d_s와 d_r은 렌즈에서 각각 이미지 평면과 스크린까지의 거리를 나타낸다. DPI는 단위길이에서의 픽셀 수를 가리킨다. 이렇게 구해진 C는 주어진 픽셀에 대해서 블러링 되어야 하는 주변픽셀들의 영역을 찾아내는데 쓰인다.

3. 필드심도 렌더링 알고리즘

본 필드심도 렌더링의 기본 알고리즘은 Potmesil에 의해 제안된 방법의 확장이다 [1,7]. Potmesil의 방법에서 쓰인 깊이 정렬된(depth-sorted) 소프트웨어 스프라이트(또는 look-up table [1]) 대신에, 무작위로 배열된 포인트 스프라이트를 알파 블렌딩하여 블러링 효과를 얻어낸다. 주어진 3D 장면에 대해서, 컬러와 깊이 이미지는 핀홀 카메라 모델을 이용하여 매 프레임마다 하나의 RGBA 텍스쳐로 렌더링 된다. 이 텍스쳐 이미지는 프레임 버퍼(frame buffer)의 크기와 동일한 포인트들의 배열에 대응된다. 각 포인트 스프라이트의 강도(알파채널), 반지름 및 색은 2장의 식에

따라 계산된 CoC에 따라서, GPU 상의 정점(vertex) 프로그램에서 동적으로 결정된다. 이 스프라이트들의 컬러는 알파 블렌딩을 이용하여 누적되고, 최종 결과 컬러는 누적된 컬러의 알파 값으로 나누어서 구해진다.

3.1 렌더링된 픽셀의 포인트 스프라이트로의 대응

포인트 스프라이트는 GPU에서 2/3차원의 포인트를 그래픽스 하드웨어를 이용하여 스프라이트로 렌더링 하는 테크닉이다. 근래의 그래픽스 하드웨어들은 임의의 포인트들을 직사각형 영역으로 동적 확장함을 지원한다. 이렇게 확장된 영역은 CoC를 시뮬레이션 하는데 이용된다. 직사각형 영역 자체로는 원형의 CoC 모양을 나타낼 수 없지만, 최근의 텍스쳐 확장과 함께 원형이나 더 복잡한 모양까지 만들어 낼 수 있다 [9]. 본 논문의 주요 착안점은 이러한 포인트 스프라이트를 미리 렌더링된 픽셀들에 대응하고, 각 픽셀에서의 CoC에 따라 GPU 프로그램(정점 및 픽셀 프로그램)에서 확장한 후, 이를 적절히 블렌딩 하는 것이다.

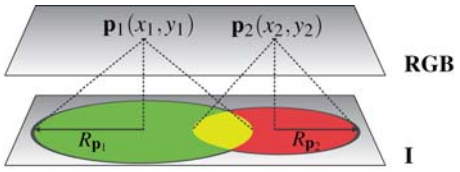

그림 2. 렌더링 된 픽셀의 포인트 스프라이트로의 대응. 두 스프라이트 사이 노란색 영역이 블러링을 위해 사용된다.

프레임버퍼의 이미지를 미리 저장한 텍스쳐의 픽셀들과 포인트 스프라이트의 대응은 다음과 같다. 실행 전의 전처리 단계에서, 이미지 해상도와 같은 크기의 포인트 배열을 생성하고, 그 렌더링 순서를 무작위로 바꾸어준다. 실행시간에 각 픽셀의 컬러 $\mathbf{RGB_p}$는 포인트의 컬러 $\mathbf{I_p}$로 다음과 같이 대응된다.

$$\mathbf{I_p} = \mathbf{RGB_p}$$

수식에서 $\mathbf{p}$는 픽셀의 위치(x,y)를 나타낸다. 여기서 각 포인트의 반지름 $R_\mathbf{p}$는 CoC 지름의 절반이다($R_\mathbf{p}=C_\mathbf{p}/2$; 그림 2). 하나의 픽셀이 여러 픽셀로 대응되므로 그 강도(intensity)는 CoC 크기에 따라 감소하고, 이는 픽셀의 알파 값에 반영된다. 스프라이트 내의 픽셀위치를 $\mathbf{q}=(x+u, y+v)$라고 할 때, 컬러의 강도는 CoC에서 균일분포(uniform distribution)를 따른다 [10]. 이에 따라, $\mathbf{q}$에서의 알파값 $A_\mathbf{q}$는 다음과 같이 정의된다.

$$A_\mathbf{q} = \begin{cases} \dfrac{K_\mathbf{q}}{\pi R_\mathbf{p}^2} & \text{if } \| \mathbf{p}-\mathbf{q} \| \le R_\mathbf{p} \\ 0 & \text{otherwise} \end{cases}$$

가중치 $K_\mathbf{q}$는 현재는 1이지만, 다음 장에서 결점들을 제거하기 위해 변형된다. 위의 과정은 실제로는 미리 정의된 균일분포의 원형 텍스쳐를 읽어오는 과정으로 대치된다. 이렇게 구해진 $A_\mathbf{q}$는 강도의 누적을 위한 알파 블렌딩에 사용된다.

3.2 픽셀누적에 의한 블러링

각 픽셀 위치에서 알파 블렌딩에 의해서 누적된 강도는 블러링 된 이미지를 나타내기 위해 사용된다(그림 2의 노란색 영역). 일반적인 8비트 픽셀형식은 부족한 정확도(precision) 때문에 픽셀의 누적으로 사용되지 못하고, 16bit 이상의 부동 소수점(floating point)형식이 사용되어야 한다.

블러링을 위해, 누적되고자 들어오는 스프라이트 내 픽셀의 알파 블렌딩은 다음과 같이 수행된다.

$$\bar{\mathbf{I}}_\mathbf{q} \leftarrow \bar{\mathbf{I}}_\mathbf{q} + A_\mathbf{q}\mathbf{I}_\mathbf{q} \quad \text{and} \quad \bar{A}_\mathbf{q} \leftarrow \bar{A}_\mathbf{q} + A_\mathbf{q}$$

수식에서 $\bar{\mathbf{I}}_\mathbf{q}$, $\bar{A}_\mathbf{q}$, $\mathbf{I}_\mathbf{q}$, $A_\mathbf{q}$ 는 누적된 컬러와 알파, 들어오는 컬러와 알파 값을 가리킨다. $\bar{\mathbf{I}}_\mathbf{q}$와 $\bar{A}_\mathbf{q}$는 매 프레임 0으로 초기화된다. 누적이 끝난 후 $\mathbf{p}$에서의 최종 픽셀 컬러는 픽셀 자신의 알파 값으로 나누는 정규화를 통하여 결정된다.

그림 3은 지금까지 설명한 방법을 이용하여 블렌딩 한 결과 예를 보여준다. 자연스러운 블러링이 생성되었지만, 초점이 맞은 평면 주위에서 두 가지 결점이 보인다. 전경에서의 불연속 블러링과 배경에서의 강도 침투 현상이다. 다음 장은 두 결점들의 해결 방법을 기술한다.

4. 전경 및 배경의 결점 제거

깊이 정렬이 없는 알파 블렌딩은 3장에서 보았듯이, 두 가지의 결점을 보인다. 전경에서 부드럽게 경계가 이어지지 못하고, 불연속적으로 끊기게 보임과 배경의 강도가 전경의 물체로 침투하여 부자연스럽게 보인다. 소프트웨어 방법에서의 깊이 정렬은 하드웨어에서 적용할 수 없으므로, 본 장에서는 깊이 정렬 없이 이 두 가지 결점을 제거하는 방법을 제시한다.

4.1 전경의 불연속 제거

전경에서 불연속이 일어나는 이유는 부분 가시도(partial visibility)가 없기 때문이다. 즉, 전경에 있는 물체의 알파 값에 따라서, 배경의 강도가 결정되어야 하나, 깊이 정렬이 없이는 정렬된 블렌딩이 불가능하므로, 전경에서의 알파 값에 비해 초점이 맞은 영역에서 알파 값이 1에 가깝게 너무 커진다.

저자는 이를 해결하기 위해 간단하면서도 효율적인 방법을 제시한다. 전경에 해당하는 물체의 강도의 증폭이 그것이다. 전경 물체의 강도를 증폭함으로써, 초점이 맞은 영역에 대해서 전경의 영향이 커져, 불연속성이 제거되고 부드러운 경계를 얻을 수 있다. 또한, 시점에 가까운 전경 픽셀이 초점이 맞은 영역에서 가까운 전경 픽셀에 비해 더 강한 강도를 가지도록 설계한다. 이러한 개념을 가상 가시도(virtual visibility)라고 정의하며, $\mathbf{q}$에서의 가상 가시도 $V_\mathbf{q}$는 다음과 같이 쓸 수 있다.

그림 3. (좌) 핀홀카메라에 의해 렌더링 된 이미지, (중) 배경에 초점이 맞은 이미지, (우) 전경에 초점이 맞은 이미지. (중)의 이미지는 전경의 물체들의 경계선이 불연속하게 보이는 결점이 있고, (우)의 이미지는 배경의 컬러강도들이 전경으로 침투하는 결점이 있다 (예: 빨간 꽃잎).

그림 4. 전경과 배경에서의 두 결점을 제거한 결과. (좌) 전경이 부드럽게 이어지는 이미지, (우) 배경의 침투현상이 제거된 이미지.

$$V_{\mathbf{q}} = (Z_f / Z_{\mathbf{p}})^{\gamma}$$

수식에서 Z_f는 $V_{\mathbf{q}}$가 초점이 맞은 영역에서 1이 되도록 하기 위해 사용되며, γ는 증폭량(amplification gain)이다. 3장에서 $K_{\mathbf{q}}$를 전경의 픽셀($Z_{\mathbf{p}}<Z_f$)에 대해서, $V_{\mathbf{q}}$로 대체함으로써, 초점이 맞은 영역을 가리는 전경 픽셀의 알파 값이 증폭되게 되고, 불연속적인 블러링이 없어지게 된다.

4.2 배경의 강도누출 제거

또 다른 결점은 배경의 강도침투이다. 소프트웨어에서의 방법 [7], 즉, 자신보다 가까운 픽셀들만 블렌딩 할 경우에는 불규칙하게 블러링 된 심각한 결과를 보여주므로, 다른 처리가 필요하다. 이에 본 장에서는 보다 세련된 방법을 적용하여 강도침투를 제거한다.

요구되는 블렌딩의 양상은 배경의 강도는 초점 영역에 영향을 미치지 않고, 초점 영역은 배경의 강도에 영향을 미치도록 하는 것이다. 이를 위해서, 배경의 침투가 제거되어야 할 느슨한 필드심도 영역(loose DOF range)를 먼저 정의한다. CoC 지름이 ε픽셀보다 작을 때($C_{\mathbf{p}}<\varepsilon$)를 필드심도 영역으로 정의하고, 이 영역에서 침투는 완전히 제거되어야 한다. 배경 픽셀($Z_{\mathbf{p}}>Z_f$)에 대해서, 요구되는 양상을 만족하는 가중치 $W_{\mathbf{q}}$는 다음과 같이 정의된다.

$$W_{\mathbf{q}} = 1 - lerp(clamp(\frac{C_{\mathbf{p}} - C_{\mathbf{q}}}{\varepsilon}), 0, clamp(\frac{C_{\mathbf{q}}}{\varepsilon} - 1))$$

lerp(*a*,*b*,*t*)와 *clamp*(*t*)는 다음과 같이 정의되는 선형보간과 클램프 함수이다. *lerp*(*a*,*b*,*t*) = *a*(1-*t*) + *bt*. *clamp*(*t*) = min(max(0,*t*),1). $C_{\mathbf{p}}$와 $C_{\mathbf{q}}$는 스프라이트와 누적되는 픽셀에서의 CoC를 나타낸다. 그림 5는 본 수식을 표현한다.

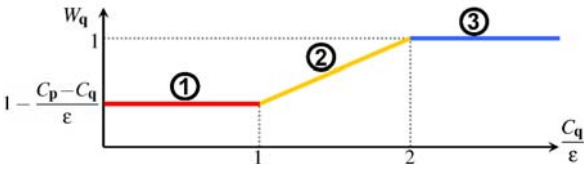

그림 5. 초점 영역으로 배경의 강도침투를 막기 위한 $W_{\mathbf{q}}$의 그래프

픽셀이 필드심도 영역에 해당한다면 (①; $C_{\mathbf{q}}/\varepsilon<1$), $\mathbf{q}$에서의 강도는 $1-(C_{\mathbf{p}}-C_{\mathbf{q}})/\varepsilon$까지 감소된다. 강도는 ②의 영역에서 점차 증가하다가, ③의 영역에서 더 이상 감소하지 않는다. $1-(C_{\mathbf{p}}-C_{\mathbf{q}})/\varepsilon$는 감소되는 강도의 양을 나타낸다. 만약, $\mathbf{p}$와 $\mathbf{q}$의 차이가 ε보다 작다면, 강도는 $1-(C_{\mathbf{p}}-C_{\mathbf{q}})/\varepsilon$까지 감소되며, 만약 그 차이가 ε보다 같거나 크다면, 강도는 완전히 감소한다. 이런 식으로 주변 픽셀들이 유사하게 블러링 되어야 하는 곳의 강도가 제대로 유지된다.

$V_{\mathbf{q}}$와 $W_{\mathbf{q}}$를 동시에 고려하여, 3장에서의 $K_{\mathbf{q}}$는 다음과 같이 바뀌게 된다.

$$K_{\mathbf{q}} = \begin{cases} V_{\mathbf{q}} & \text{if } Z_{\mathbf{p}} < Z_f \\ W_{\mathbf{q}} & \text{otherwise} \end{cases}$$

$K_{\mathbf{q}}$를 이용함으로써, 전경과 배경에서의 두 결점은 따로 다루어지고 제거된다. 그림 4는 $K_{\mathbf{q}}$를 적용하여 두 결점을 제거한 결과 이미지를 보여준다. 매우 자연스러운 필드심도 효과가 얻어졌음을 볼 수 있다.

5. 결과 및 성능

그림 6. 본 논문에서 제시된 방법을 사용하여 렌더링 된 결과 이미지.

그림 7. 전경에 초점이 맞은 렌더링결과 이미지.

그림 6은 본 논문에서 제시된 방법을 사용하여, 시점으로부터 연속적으로 길게 놓여있는 물체를 렌더링 한 결과 이미지이다. 전경, 배경 및 연속되는 지점에서 어떠한 결점도 발견되지 않는, 매우 자연스러운 이미지를 렌더링 하였다. 그림 7은 두 번째 예시로서, 전경의 Gargoyle에 초점이 맞은 것을 렌더링 한 것이다. 배경에서의 자연스러운 블러링과 배경의 강도침투 현상이 완전히 제거되어 매우 사실적인 필드심도 효과를 보여준다.

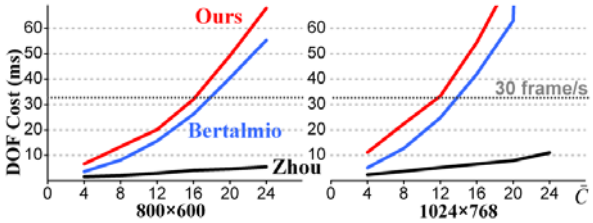

그림 8. 렌더링 성능

그림 8은 본 방법을 근래의 두 가지 방법들과 렌더링 비용으로 비교한 결과를 보여준다. 전체적으로 본 논문의 방법은 Zhou [11]의 방법보다는 느리며, Bertalmio [5]의 방법과 유사한 정도의 성능을 보여주고 있다. 훨씬 나은 품질을 보여주면서도, 실시간 렌더링 방법들과 유사한 성능을 보여주어, 본 방법이 실용적인 실시간 응용분야에 이용될 수 있음을 알 수 있다.

참고문헌

[1] POTMESIL M., CHAKRAVARTY I.: A lens and aperture camera model for synthetic image generation. ACM SIGGRAPH 15,3 (1981), 297 - 305.

[2] COOK R. L., PORTER T., CARPENTER L.: Distributed ray tracing. Computer Graphics 18, 3 (1984), 137 - 145.

[3] HAEBERLI P., AKELEY K.: The accumulation buffer: Hardware support for high-quality rendering. ACM SIGGRAPH (1990), 309-318.

[4] ROKITA P.: Generating depth-of-field effects in virtual reality applications. IEEE Computer Graphics and its Application 16, 2 (1996), 18 - 21.

[5] BERTALMÍO M., FORT P., SÁNCHEZ-CRESPO D.: Real-time, accurate depth of field using anisotropic diffusion and programmable graphics cards. In Proc. of 3DPVT'04 (2004), pp. 767 - 773.

[6] SCHEUERMANN T.: Advanced depth of field. In Game Developers Conference (2004).

[7] DEMERS J.: Depth of field: A survey of techniques. In GPU Gems: Programming Techniques, Tips & Tricks for Real-Time Graphics, Fernando R., (Ed.). Addison-Wesley Professional, 2004, ch. 23, pp. 375 - 390.

[8] KASS M., LEFOHN A., OWENS J.: Interactive Depth of Field Using Simulated Diffusion on a GPU. Tech. rep., Pixar Animation Studios, 2006.

[9] CRAIGHEAD M., KILGARD M., BROWN P.: ARB_point_sprite.http://www.opengl.org/registry/specs/ARB/point_sprite.txt.

[10] CHEN Y. C.: Lens effect on synthetic image geneartion based on light particle theory. The Visual Computer 3, 3 (1987), 125 - 136.

[11] ZHOU T., CHEN J. X., PULLEN M.: Accurate depth of field simulation in real time. Computer Graphics Forum 26, 1 (2007).

저자 소개

이성길(Sungkil Lee)
2002년 포항공과대학교 신소재공학과 학사
2002년 ~ 현재 포항공과대학교 컴퓨터공학과 박사과정
관심분야 : 가상현실, 컴퓨터그래픽스, 실시간 GPU 렌더링

김정현(Gerard Jounghyun Kim)
1987년 Carnegie Mellon University 전자컴퓨터 공학부 학사
1989년 University of Southern California 컴퓨터공학 석사
1994년 University of Southern California 컴퓨터과학 박사
1994년~1996년 NIST, Fellowship
1996년~2005년 포항공과대학교 교수
2005년~현재 고려대학교 정보통신공학부 교수
관심분야 : 가상현실, HCI

최승문(Seungmoon Choi)
1995년 서울대학교 제어계측공학과 학사
1997년 서울대학교 제어계측공학과 석사
2003년 퍼듀대학교 전기컴퓨터공학부 박사
2003년~2005년 퍼듀대학교 박사후연구원
2005년~현재 포항공과대학교 컴퓨터공학과 교수
관심분야 : 햅틱스