특집논문-06-11-4-12

다시점 3차원 방송을 위한 OpenGL을 이용하는 중간영상 생성

이 현 정[a)‡], 허 남 호[b)], 서 용 덕[a)]

View Synthesis Using OpenGL for Multi-viewpoint 3D TV

Hyunjung Lee[a)‡], Namho Hur[b)], and Yongduek Seo[a)]

요 약

본 논문은 다중 카메라에서 얻은 영상과 그 영상으로부터 얻은 깊이영상(depth map)을 사용하여 중간시점에서의 영상을 생성하기 위하여 OpenGL 함수를 적용하는 방법을 소개한다. 기본적으로 영상기반 렌더링 방법은 카메라로 직접 얻은 영상을 그래픽 엔진에 적용하여 결과 영상을 생성하는 것을 의미하지만 주어진 영상 및 깊이정보를 어떻게 그래픽 엔진에 적용시키고 영상을 렌더링 하는지에 관한 방법은 잘 알려져 있지 않다. 본 논문에서는 카메라 정보로 물체가 생성될 공간을 구축하고, 구축된 공간 안에 색상영상과 깊이영상 데이터로 3차원 물체를 형성하여 이를 기반으로 실시간으로 중간영상 결과를 생성하는 방법을 제안하고자 한다. 또한, 중간영상에서의 깊이영상도 추출하는 방법을 소개하도록 하겠다.

Abstract

In this paper, we propose an application of OpenGL functions for novel view synthesis from multi-view images and depth maps. While image based rendering has been meant to generate synthetic images by processing the camera view with a graphic engine, little has been known about how to apply the given images and depth information to the graphic engine and render the scene. This paper presents an efficient way of constructing a 3D space with camera parameters, reconstructing the 3D scene with color and depth images, and synthesizing virtual views in real-time as well as their depth images.

Keyword : OpenGL, multi-viewpoint 3D TV, depth map rendering, view synthesis

Ⅰ. 서 론

차세대 방송 서비스로 현재 개발 중인 3DTV의 구성 요소로는 3차원 영상획득, 데이터 압축 및 전송, 저작 및 분배, 디스플레이 기술 등이 있다. 전체적인 시스템의 구성과 MPEG 국제표준화 작업 등 최근의 3DTV 및 방송에 대한 연구 동향은 참고문헌[1]에 잘 소개되어 있다.

3차원 영상을 획득하기 위하여 현재 사용하고 있거나 개발 중인 방법으로는 다중 색상 카메라의 영상에 다중 스테레오 정합 기법을 적용한 수동형 깊이영상 획득 방법과 적외선, 레이저, 색상패턴 등을 사용하는 능동형 깊이영상획득 방법이 있다. 3차원 영상출력의 입장에서 보면 다양한 획득 장치를 사용하여 얻어진 3D 데이터를 이용하여 사용자가 원하는 시점에서의 영상으로 렌더링하는 방법을 개발해야 할 필요가 있다. 렌더링과 관련하여 가장 먼저 생각할 수 있는 방법으로는, 하나의 고정된 좌표계에 대하여 면과

a) 서강대학교 미디어공학과
Sogang University Dept. of Media Technology
b) 한국전자통신 연구원
Electronics and Telecommunications Research Institute
‡ 교신저자 : 이현정(whitetobi@sogang.ac.kr)
※본 연구는 ETRI <SmarTV 기술개발(3차원 AV 기반기술 개발)>의 위탁연구 및 2006년도 정부(과학기술부)의 재원으로 한국과학재단의 지원을 받아 수행된 연구임(No. R01-2006-000-11374-0)

색상으로 이루어지는 단일 3차원 그래픽 모델을 제작하는 것이다. 이렇게 모델을 제작할 수만 있으면 현재 컴퓨터 그래픽스 분야에서 이미 개발해 둔 OpenGL과 같은 실시간 렌더링 엔진을 그대로 사용하는 것이 가능하므로 하드웨어 기반의 고속 영상 렌더링을 손쉽게 구현할 수 있을 뿐만 아니라 3차원 입체 디스플레이를 위한 렌더링 엔진을 따로 개발해야 하는 수고를 덜게 되는 장점이 있다. 잘 알려진 바와 같이 여러 개의 3D 센서에서 얻은 결과를 하나의 단일 모델로 구성하는 기술은 현재도 국제적으로 관심을 받고 있는 분야이다[2][3]. 단일 입체 모델이 가지는 다양한 장점에도 불구하고 이를 직접적으로 적용하지 못하는 이유는 여러 3차원 획득 장치들의 출력으로부터 하나의 단일 모델을 구성하는 것이 쉽지 않기 때문이다. 따라서 이들 획득 장치들의 출력을 직접적으로 사용하기 위한 방법을 고안해야 할 필요가 있으며 이를 위해서는 GPU 프로그래밍과 같은 매우 전문화된 방법이 적용되어야 할 필요가 있다. 그러나 현재의 3DTV 개발 단계에서는 3차원 획득 장치 뿐만 아니라 출력장치에 대한 종속성 때문에 GPU 프로그래밍을 개발하는 것이 용이하지만은 않은 것이 현실이다.

본 논문에서는 여러 대의 카메라에서 얻은 색상 영상으로부터 스테레오 정합을 통하여 얻은 깊이영상을 사용하여 OpenGL로 렌더링하기 위한 방법을 소개한다.

두 카메라 사이의 임의시점에서 본 가상 영상을 생성하는 것은 스테레오를 위한 보정 카메라 정보를 사용하면 쉽게 수행된다[4]. 그러나 잘 알려진 바와 같이 이 방법은 후방향 워핑(backward warping) 방정식이 존재하지 않기 때문에 전방향 워핑(forward warping)에 의해서만 렌더링을 수행해야 한다. 따라서 렌더링의 결과로 얻어지는 영상에서 많은 픽셀들이 색상을 할당받지 못하는 현상을 초래하므로 영상 필터링이나 보정을 통하여 비어있는 픽셀들에 대한 처리를 해주어야만 한다. 그리고 여러 개의 소스 픽셀들이 중간영상에서 하나의 픽셀로 매핑되는 경우 깊이 값에 의한 깊이버퍼링(depth-buffering)을 수행해 주어야한다. 반면에 OpenGL과 같은 그래픽 엔진에서 렌더링을 하게 되면 동일한 입력 데이터를 사용하면서도 텍스쳐 매핑과 같은 기법을 사용할 수 있어서 중간영상의 해상도를 더 좋게 유지 할 수 있으며 깊이버퍼링(depth-buffering) 등의 처리를 이미 개발된 그래픽 엔진에 맡길 수 있는 장점이 있다.

또한 컴퓨터 그래픽 물체와 실사 물체의 합성에 의한 증강현실 또는 혼합현실의 효과를 얻기 위해서는 OpenGL 렌더링을 반드시 구현해야만 한다. 기존의 비디오 기반 증강현실이 2차원의 실사 영상에 2차원의 그래픽 영상을 합성하는 것을 주요 목표로 삼았지만, 앞으로의 3DTV에서는 3차원 실사를 3차원 그래픽스와 합성하는 기능을 기본적으로 갖추어야 할 것이다. 이를 위해서는 하나의 그래픽 엔진에서 실사 부분과 그래픽스 부분을 동시에 처리해야만 하며 이를 위한 테스트 기반으로서 OpenGL은 좋은 도구가 된다.

2장에서는 추출된 깊이 정보와 색상 영상을 그래픽 엔진에 적용하기 위해 알아야 하는 바늘구멍 카메라 모델을 소개하고, 컴퓨터 비젼과 컴퓨터 그래픽에서 분야에서 바늘구멍 모델이 어떤 차이점을 가지고 있는지 알아본다. 또한, 공통으로 사용하는 카메라 보정으로부터 얻어지는 보정변수에 대하여 설명하고, OpenGL 인터페이스를 사용하여 깊이 정보와 색상 영상으로 임의시점에서의 중간영상을 생성하는 방법 및 중간시점의 색상영상과 깊이 정보를 동시에 획득하는 방법을 렌더링 결과와 함께 소개하고, 다른 방법으로의 렌더링 방법과 비교 분석하여 3장에서 결론을 내리고자 한다. 본 논문에서 사용한 색상 영상과 깊이 정보는 참고문헌[5]의 웹페이지에서 제공하는 데이터이다.

Ⅱ. OpenGL을 사용하는 렌더링의 방법

OpenGL을 사용한 렌더링 방법의 전체적인 순서는 그림 1과 같다. 이와 같은 과정을 거쳐 최종 중간영상을 생성하기 위해서는 우선 바늘구멍 카메라 모델과 보정 변수에 대해 알아야 한다.

1. 바늘구멍 카메라 모델과 보정 변수

바늘구멍 카메라를 기본 모델로 삼는 투영방정식은 예전부터 컴퓨터 비젼과 컴퓨터 그래픽스 분야에서 사용되어

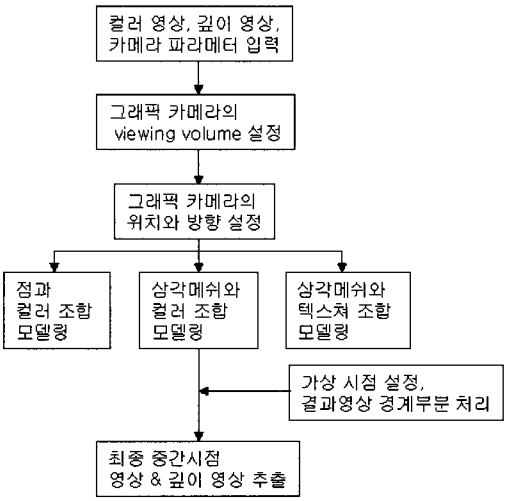


그림 1. 제안 알고리즘에 관한 구조도
Fig. 1. flowchart of our algorithm

왔지만 두 분야에서의 가장 큰 차이점으로는 카메라의 시선방향을 선택할 때 사용하는 좌표계의 설정이 서로 다르게 되어 있다는 것이다. 컴퓨터 비젼 분야에서 사용하는 바늘구멍 카메라 모델에서는 카메라 좌표계를 선택할 때 카메라의 시선방향을 Z-축의 양의 방향으로 정하는 것이 일반적이다[6]. 공간의 한 점 $[X,Y,Z,1]^T$이 영상의 한 점 $x=[u,v,1]^T$ 으로 투영되는 관계식은 (1)과 같이 정의된다.

$$x = K[R|T]\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{1}$$

식 (1)에서 K는 카메라의 CCD 센서와 같은 실제 영상면에 투영된 영상을 픽셀단위의 디지털 영상으로 변환하는 기능을 하며, 카메라 내부 변수 행렬 (internal parameter matrix)라고 부른다. K는 focal length, aspect ratio, skew, principal point 등을 포함하여 다양한 카메라 내부 변수를 나타낼 수 있다. 본 논문에서는 skew=0 인 경우만 고려하도록 하겠다. 내부 변수를 구하는 방법은 이미 컴퓨터 비젼 분야에서 오랫동안 연구되어 왔는데, 실질적으로는 Z. Zhang이 제안한 방법[7]을 기반으로 제작한 Camera Calibration Toolbox를 가장 많이 사용하는 것으로 알려져 있다[8].

식 (1)의 변수 중 R과 T 는 공간의 물체에 대하여 정의된 좌표계를 카메라에 대하여 정의된 좌표계로 변환하는 관계를 나타내는 것이며, 각각은 회전 행렬과 직선이동 벡터를 의미한다. 이 방정식에서 등호는 영상좌표의 마지막 요소를 1로 만들어주는 과정을 포함하고 있으며 일반적인 등호와는 의미가 다른데, 이는 영상 좌표값 (u,v)로의 투영 방정식은 실제로는 식(2)와 같이 비선형 방정식으로 주어지기 때문이다.

$$\begin{aligned} u &= u_0 + f_x \frac{r_{11}X + r_{12}Y + r_{13}Z + T_X}{r_{31}X + r_{32}Y + r_{33}Z + T_Z} \\ v &= v_0 + f_y \frac{r_{21}X + r_{22}Y + r_{23}Z + T_Y}{r_{31}X + r_{32}Y + r_{33}Z + T_Z} \end{aligned} \tag{2}$$

식 (1)의 선형 투영 방정식에서 내부 변수 행렬 K는 식 (3)과 같이 주어진다.

$$K = \begin{pmatrix} f_x & 0 & u_0 \\ & f_y & v_0 \\ & & 1 \end{pmatrix} \tag{3}$$

이 행렬에서 f_x는 f_y에 aspect ratio 값이 곱해져 있는 것으로 최근의 카메라들은 가로 세로 비율이 1:1인 경우가 많다. 카메라 보정 단계에서 이 변수들을 어떻게 정의하느냐에 따라 주어지는 값이 달라지므로 본 논문에서는 가로 축과 세로축에 대하여 각기 다른 스케일링 값(f_x, f_y)을 사용하는 것으로 하였다.

지금까지는 컴퓨터 비젼 분야에서 사용되는 투영방식에 대해 살펴 보았는데, 컴퓨터 그래픽 분야에서 사용하는 투영 카메라는 컴퓨터 비젼 분야에서 사용하는 방법과 다르게 시선방향을 Z-축의 음의 방향으로 설정하도록 되어 있다[9][10]. 그래서 컴퓨터 비젼의 방법으로 구한 입체 형상 데이터를 컴퓨터 그래픽스로 보여주는 경우에는 몇 가지 사항을 고려해 주어야 하며 다음 절에서 상세히 설명하도록 하겠다.

2. OpenGL 인터페이스에 의한 영상기반 렌더링

OpenGL을 이용하여 중간영상을 생성하기 위해서는 전

체적으로 그래픽 카메라에 대한 설정 작업이 필요하고, 실제로 영상을 획득하는 카메라와 동일한 카메라 특성을 지니는 그래픽 카메라를 정의하기 위해서는 여러 변수를 잘 설정해 줄 필요가 있다.

2.1 그래픽 카메라의 viewing volume 설정

그래픽 카메라의 viewing volume을 설정하기 위해서는 먼저 최종적으로 생성되는 영상의 크기를 고려하여 렌더링된 영상의 픽셀 단위 크기를 설정해 주어야 한다. 이는 실제 카메라의 출력 영상과 동일한 크기로 그래픽 영상이 출력되도록 설정하면 되고, 이를 위하여 glViewport() 함수를 사용하여 OpenGL에서 생성되는 영상의 윈도우 크기를 설정하면 된다.

$$\textit{void } \textbf{\textit{glViewport}} \textit{ (int x, int y, int width, int height);} \quad (4)$$

이 함수의 처음 두 인자 (x,y)는 윈도우의 위치를 나타내는 것이며, 나머지 두 인자 (width, height)가 최종 영상의 픽셀단위 크기를 나타내므로 만약 실제카메라에서 얻은 영상의 크기가 1024x768 이면 이 값을 동일하게 사용하면 된다.

glViewport() 함수에서 정한 크기로 중간시점 영상을 생성하기 위해서는 카메라의 위치로부터 깊이 영상과 색상 영상으로 생성될 3차원 물체가 위치할 영역을 설정해 줄 필요가 있다. OpenGL에서는 그래픽 카메라의 viewing volume을 설정하는 glFrustum() 함수의 매개변수를 이용함으로써 이와 동일한 기능을 구현하는 것이 가능하다.

$$\textit{void } \textbf{\textit{glFrustum}} \textit{ (double left, double right, double bottom, double top, double near, double far);} \quad (5)$$

함수의 매개변수 값들이 표시하는 바를 그림 2에 도시하였고, 그림 2에서 주목해야하는 부분은 가까운 쪽에 있는 평면이 실제 카메라의 영상면(image plane)과 대응하도록 설정해야 한다는 것이다. 이 영상면은 Viewing volume의 앞쪽 면을 컴퓨터 비젼에서 사용하는 $z=1$인 위치의 영상면과 대응시킴으로써 실제 카메라와 가상 카메라를 동일한 형태로 만들 수 있다. 카메라 보정에 의해 얻는 내부 변수들을 이용하면 left, right, bottom, top 에 해당하는 값들을 구하는 것이 가능하다. 내부 변수를 이용하여 viewing volume을 정의하기 전에 컴퓨터 비젼과 컴퓨터 그래픽스에서 정의하는 변수들의 대응 관계를 살펴보면 표 1과 같다.

표 1을 살펴보면 카메라 내부 변수 중 skew 정보가 없는 것을 볼 수 있다. 이는 skew값이 포함되게 되면 영상평면이 기울어지는 현상이 발생 하기 때문에 skew가 없는 바늘구멍 카메라 모델을 사용하여 그 카메라의 내부 변수를 그래픽 카메라의 viewing volume을 정의하는 변수로 변환하는 것이 가능하게 되고, 그래픽 카메라와 실제 카메라의 투영

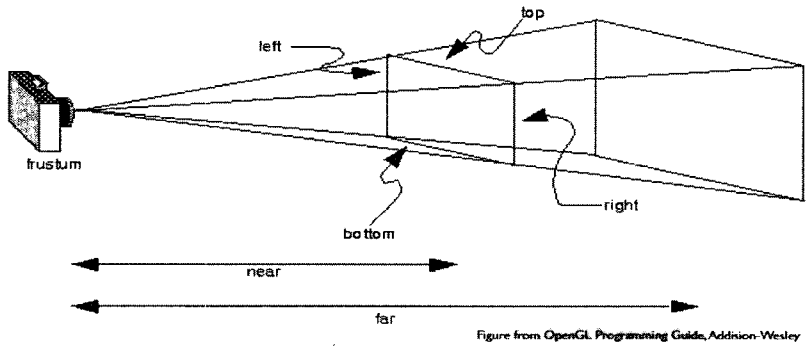


그림 2. glFrustum() 함수에 의한 viewing volume의 정의
Fig. 2. Determination of viewing volume according to glFrustum() function

표 1. 컴퓨터 비전과 컴퓨터 그래픽스에서 정의하는 변수의 대응 관계
Table 1. Relationship between parameters of computer vision and graphics

컴퓨터 비전에서 사용하는 변수		컴퓨터 그래픽스에서 사용하는 변수	
내부변수	초점거리 (focal length) principal point	left, right, top, bottom	glFrustum ()
외부변수	rotation translation	rotation translation	Model-View Transformation

특성이 동일하게 설정되는 결과를 얻을 수 있다. 그러면, 카메라 보정으로부터 얻어진 카메라 내부 변수 행렬 K가 다음과 같다고 하자.

표 1을 살펴보면 카메라 내부 변수 중 skew 정보가 없는 것을 볼 수 있다. 이는 skew값이 포함되게 되면 영상평면이 기울어지는 현상이 발생 하기 때문에 skew가 없는 바늘구멍 카메라 모델을 사용하여 그 카메라의 내부 변수를 그래픽 카메라의 viewing volume을 정의하는 변수로 변환하는 것이 가능하게 되고, 그래픽 카메라와 실제 카메라의 투영 특성이 동일하게 설정되는 결과를 얻을 수 있다. 그러면, 카메라 보정으로부터 얻어진 카메라 내부 변수 행렬 K가 다음과 같다고 하자.

$$K=\begin{pmatrix} f_x & 0 & u_0 \\ & f_y & v_0 \\ & & 1 \end{pmatrix} \tag{6}$$

실제 카메라의 보정으로부터 얻어지는 f_x는 $Z=1$인 영상면을 픽셀단위의 디지털 영상으로 변환할 때 영상면을 X-축 방향으로 확대시키는 스케일링 역할을 하게 된다. 따라서 디지털 영상의 경계 좌표값을 내부 변수 행렬을 사용하여 역으로 환산함으로써 그래픽 카메라의 glFrustum() 함수에 필요한 변수를 아래와 같이 구할 수 있다.

$$\begin{aligned} &\text{left:} && l=\frac{-u_0}{f_x} \\ &\text{right:} && r=\frac{W-1-u_0}{f_x} \\ &\text{bottom:} && b=\frac{-v_0}{f_y} \\ &\text{top:} && t=\frac{H-1-v_0}{f_y} \\ &\text{near:} && n=1 \end{aligned} \tag{7}$$

예를 들어, 디지털 영상의 왼쪽경계좌표는 $y=0$으로 주어진다. 실제 영상면에서 이 값을 l 이라 하면, 카메라 내부 변수들에 의해 $f_x \times l+u_0=0$의 관계를 만족하게 된다. 식 (7) 중에서 첫 번째 식은 이러한 관계를 나타내는 것이고, 나머지 식들도 동일한 아이디어를 적용하여 얻을 수 있게 된다. 마지막 식은 viewing volume의 가까운 쪽 면 (near plane)이 그래픽 카메라에서는 영상 렌더링을 위한 하나의 절단면을 의미하지만 실제 카메라와 대응시켜서 생각해보면 이 면을 실제카메라의 CCD 평면 즉 $Z=1$인 영상면으로 인식하는 것이 가능하다. 따라서, 처음 네 개의 변수를 위와 같이 정의하고 가까운 쪽 면의 위치를 $n=1$로 설정하면 실제 카메라의 투영방식과 동일한 투영 특성을 가지게 만드는 것이 가능해진다.

2.2 그래픽 카메라의 위치와 방향 설정

2.1 절에서 언급한 바와 같이 카메라의 viewing volume을 설정하고 렌더링 결과 영상의 픽셀 크기를 실제 영상의 크기와 동일하도록 설정하였으므로 그래픽 카메라의 위치와 방향을 실제 카메라의 위치와 방향에 맞도록 설정하기만 하면 결과적으로 얻어지는 렌더링 영상이 실제카메라의 입력 영상과 동일하게 될 것이다. 그래픽 카메라의 위치나 방향을 바꾸더라도 카메라의 투영특성이 동일하므로 결과적으로 얻어지는 렌더링 결과에서는 실제 카메라를 운용하여 얻는 효과를 구하는 것이 가능하게 된다. 임의시점에 대한 변수를 설정하는 것은 어렵지 않으나, 실제 카메라와 동일한 위치 및 방향으로 그래픽 카메라를 설정해야할 필요가 있을 때에는 실제 카메라로부터 주어지는 카메라 외부 변수(rotation, translation)를 활용해야 할 필요가 있다.

실제 카메라에 대한 투영방정식(projection equation)과 깊이 영상(depth map)에 대한 관계는 식(1)과 같다. 이 식

에서 $x=[u,v,1]^T$는 영상의 좌표를 나타내고, $[X,Y,Z,1]$는 깊이 정보로부터 주어지는 물체의 공간 좌표이다. OpenGL에서 그래픽 카메라의 위치 및 방향 설정과 관련하여 한 가지 주의해야할 점은 OpenGL에서는 그래픽 카메라가 보는 방향이 $-Z$ 방향이라는 것이다. 따라서, 영상에 보이는 모든 물체의 Z 좌표값은 음수라야만 한다. 통상적으로 컴퓨터 비젼에서 정의하는 Z 값들은 모두 양수이므로, 주어진 깊이 영상으로부터 구한 Z 값이 양수인지 검사하고, 만약 양수 이면 식(8)과 같이 부호를 바꾸어 줌으로써 이 문제를 해결하는 것이 가능하다.

$$x = K[R|-T]\begin{bmatrix} -X \\ -Y \\ -Z \\ 1 \end{bmatrix} \tag{8}$$

이는 투영방정식의 고유특성 때문에 투영으로 얻어지는 좌표값이 일정하므로 가능하며, 식(9)와 같이 투영방정식의 결과가 동일한 것을 확인하면 좀 더 명확하게 이해될 수 있다.

$$\frac{r_{11}X+r_{12}Y+r_{13}Z+T_X}{r_{31}X+r_{32}Y+r_{33}Z+T_Z} = \frac{-r_{11}X-r_{12}Y-r_{13}Z-T_X}{-r_{31}X-r_{32}Y-r_{33}Z-T_Z} \tag{9}$$

Z 값에 대한 부호 문제는 위와 같이 해결한 것으로 가정하고, 식(1)에서 원래 주어진 투영방정식을 공간 좌표에 대한 변환의 개념을 사용하여 표시하면 식(10)과 같이 된다.

$$x = K[I|0]\begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{10}$$

이 식은 다음과 같이 풀어 설명할 수 있다. 공간 좌표 $[X,Y,Z]$에 주어진 색상을 이용하여 점이나 면을 설정하고, 이들 전체를 회전(R) 및 직선이동(T) 변환을 한 후 그래픽 카메라로 자연스럽게 투영하면 원하는 결과가 얻어지게 된다. 결론적으로, 이제까지 언급한 OpenGL 렌더링 순서를 OpenGL 프로그래밍의 순서에 따라 요약하면 다음과 같다.

- glViewport()를 사용하여 픽셀단위 영상의 크기를 실제 카메라에서 얻는 크기와 동일하게 설정한다.
- glFrustum()을 사용하여 그래픽 카메라의 투영특성을 설정한다.
- Model-View transformation을 적용한다. 초기 위치를 실제 카메라의 위치 및 방향으로 설정하기 위하여 깊이 영상과 함께 주어진 카메라의 회전 및 이동 행렬 값을 사용한다.
- 깊이 영상을 사용하여 점 또는 면의 모델을 해당 공간 좌표에 정해준다.

3. depth map으로부터 OpenGL의 점/면 모델링

실제 카메라에서 얻어진 색상 영상과 깊이 영상이 3차원 데이터로서 주어지면, 이 데이터를 컴퓨터 그래픽에서 사용하는 데이터 형식으로 변환하여야 한다. OpenGL에서 지원하는 3차원 형상의 모델 중에서 사용가능한 방법으로 크게 세 가지를 들 수 있는데 복잡도에 따라 다음과 같이 나열할 수 있다.

- 점(vertex)과 색상의 조합
- 삼각메쉬(triangular mesh)와 삼각형 각 점에서의 색상의 조합
- 삼각메쉬와 텍스쳐(texture) 영상의 조합

입력 데이터로서 깊이 영상은 각 픽셀에서의 색상과 함께 주어진다. 따라서 깊이 영상에서 주어진 픽셀 위치 (u,v)의 깊이 값을 읽고 이를 3차원 공간에서의 좌표로 변환하는 과정이 필요하다. 디지털 영상의 픽셀 위치 (u,v)로부터 실제 영상면(x,y)로 변환하는 과정은 카메라의 내부변수 행렬 K의 역을 곱하는 것으로 주어진다.

$$\begin{aligned} x &= (u-u_0)/f_x \\ y &= (v-v_0)/f_y \end{aligned} \tag{11}$$

그러면 깊이 영상에서 얻어진 값 Z를 이용하여 이 점의 나머지 3차원 좌표 X와 Y를 식 (12)를 이용하여 구할 수 있다.

$$x = \frac{r_{11}X + r_{12}Y + r_{13}Z + T_X}{r_{31}X + r_{32}Y + r_{33}Z + T_Z}$$
$$y = \frac{r_{21}X + r_{22}Y + r_{23}Z + T_Y}{r_{31}X + r_{32}Y + r_{33}Z + T_Z} \quad (12)$$

이 두 방정식에서 다른 변수값 들은 이미 투영 방정식으로 주어져 있으므로 두 개의 계산해야할 변수 X와 Y에 대하여 두 개의 선형 방정식을 얻게 된다. 따라서, 간단히 나머지 3차원 좌표를 구하면 픽셀 위치 (u,v)에 대한 3차원 공간 좌표 (X, Y, Z)를 얻게 된다. 또 이 점에서의 색상 (R,G,B)는 입력 색상 영상에서 얻는다.

3.1 점과 색상의 조합 모델링

입력 색상 영상의 각 픽셀에 대한 3차원 공간 좌표를 계산하였으면 OpenGL에서의 점(Vertex) 정의에 의해 3차원 공간 좌표에 한 점을 두는 것이 가능하다. 이는 아래 표 2에 보인 바와 같은 C 함수 호출에 의해 이루어진다. 이 방법은 렌더링 시간이 매우 빠른 장점이 있지만 중간영상 합성을 위하여 그래픽 카메라를 실제 카메라와 다른 위치로 이동하게 되면 점과 점사이의 공간이 렌더링 되지 않고 나타나게 된다. 그림 3의 두 영상은 이 방법으로 렌더링한 결과를 보인 것이다. 왼쪽 영상은 그래픽 카메라를 OpenGL 프로그램을 사용하여 두 대의 실제 카메라 사이에 배치하여 획득한 색상 영상이다. 오른쪽 영상은 왼쪽 영상에 대한 깊이영상(depth map)에 해당한다. 실제 깊이영상은 실수 형태의 데이터 형식을 가지기 때문에 영상으로 표시하기

표 2. 점과 색상의 조합에 의한 3차원 공간 좌표의 OpenGL 표현 방법
Table 2. OpenGL code for displaying 3D coordinates based on point and color combination

```
glBegin (GL_POINTS);
for (int y=0; y<Height; y++)
for (int x=0; x<Width; x++)
{
        glColor3ub (image[y][x].R, image[y][x].G, image[y][x].B);
        glVertex3d (X[y][x], Y[y][x], Z[y][x]);
}
glEnd ();
```

그림 3. 점과 색상의 조합에 의한 중간영상 렌더링 결과
Fig. 3. Rendering result from point and color combination

위하여 256레벨로 조절하여 가까운 부분일수록 밝은 효과가 나도록 하였다. 이 깊이영상은 중간영상을 획득하기 위한 카메라의 시점에 대하여 물체 중심의 좌표계로 표시한 것이다. 이 정보는 아래 4절에서 좀 더 자세히 설명하겠지만, 최근에 생산되는 3차원 입체 디스플레이 중에서 깊이영상과 색상영상을 입력으로 받는 기종에서 사용하기 위한 것이다. 일례로, 네델란드 Philips 사의 3D-Wow 계열 제품들은 깊이영상과 색상 영상을 입력으로 받아 3D 영상을 출력한다.

3.2 삼각메쉬와 색상의 조합 모델링

입력 영상뿐만 아니라 깊이영상 조차도 영상 그리드 위에서 정의되어 있으므로 자연스럽게 영상의 그리드를 기반으로 삼각메쉬를 형성하는 것이 가능하다. 표 3에서는 삼각메쉬를 형성하고 삼각메쉬의 각 점(vertex)에 해당 색상을 할당하는 방법을 OpenGL을 사용하는 C 코드로 보였다.

삼각메쉬를 형성할 때 주의해야할 점으로는 그림 4과 같이 삼각형의 꼭지점의 나열 순서를 잘 고려하여 형성된 삼각면의 수직방향 즉 면의 법선벡터의 방향이 항상 일정하도록 해야 한다는 것이다. 이 방법을 사용하면 중간영상을 합성하더라도 점과 점사이가 하나의 면으로 연결되기 때문에 자연스러운 형상을 렌더링 하는 것이 가능하다. 아울러, 제한적이긴 하지만 그래픽에 의한 가상의 조명과 같이 OpenGL에서 제공하는 부가적 기능을 사용하는 것이 가능해진다.

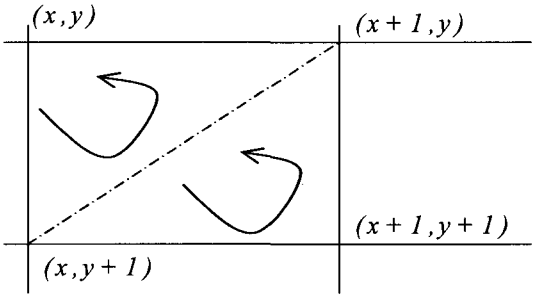


그림 4. 삼각 메쉬 정의 시 삼각형 면의 방향
Fig. 4. Ordering direction of triangular mesh

표 3. 삼각메쉬와 색상의 조합에 의한 3차원 물체의 OpenGL 표현 방법
Table 3. OpenGL code for 3D objects based on triangular mesh and color combination

```
glBegin (GL_TRANGLES);
for (int y=0; y<Height-1; y++)
for (int x=0; x<Width-1; x++)
{
      glColor3ub (image[y][x].R, image[y][x].G, image[y][x].B);
      glVertex3d (X[y][x], Y[y][x], Z[y][x]);
      glColor3ub (image[y+1][x].R, image[y+1][x].G, image[y+1][x].B);
      glVertex3d (X[y+1][x], Y[y+1][x], Z[y+1][x]);
      glColor3ub (image[y][x+1].R, image[y][x+1].G, image[y][x+1].B);
      glVertex3d (X[y][x+1], Y[y][x+1], Z[y][x+1]);

      glColor3ub (image[y+1][x].R, image[y+1][x].G, image[y+1][x].B);
      glVertex3d (X[y+1][x], Y[y+1][x], Z[y+1][x]);
      glColor3ub (image[y+1][x+1].R, image[y+1][x+1].G, image[y+1][x+1].B);
      glVertex3d (X[y+1][x+1], Y[y+1][x+1], Z[y+1][x+1]);
      glColor3ub (image[y][x+1].R, image[y][x+1].G, image[y][x+1].B);
      glVertex3d (X[y][x+1], Y[y][x+1], Z[y][x+1]);
}
glEnd ();
```

그림 5. 삼각메쉬와 색상의 조합에 의한 렌더링 결과
Fig. 5. Rendering result from triangular mesh and color combination

그림 5는 삼각메쉬와 색상 조합에 의한 렌더링 결과를 예시하였다. 오른쪽 흑백 영상은 왼쪽 색상 영상에 해당하는 깊이영상이다. 깊이 값을 변환하여 가까울수록 밝은 색을 가지도록 256 레벨로 나타내었다. 모든 픽셀 그리드에 대하여 삼각메쉬를 형성하였기 때문에 가운데 사람의 왼쪽 어깨 부분처럼 깊이 값의 차이가 큰 부분에서는 선형보간에 의한 렌더링이 이루어졌다.

3.3 삼각메쉬와 텍스쳐의 조합 모델링

그래픽 카메라를 실제 카메라 주변에 두었을 때 렌더링된 영상의 세밀함 또는 사실성을 고려하면 삼각메쉬의 각 점에 색상값을 사용하는 것 보다 주어진 색상영상을 사용하여 텍스쳐 맵핑을 하는 것이 더 효율적이다. OpenGL을 이용한 텍스쳐 매핑 방법은 표 4를 보면 알 수 있다.

텍스쳐 매핑을 위해서는 입력 색상 영상을 텍스쳐 버퍼에

표 4. 삼각메쉬와 텍스쳐 매칭에 의한 3차원 물체의 OpenGL 표현 방법
Table 4. OpenGL code for 3D objects based on triangular mesh and texture mapping

```
glEnable (GL_TEXTURE_2D);
glBegin (GL_TRANGLES);
for (int y=0; y<Height-1; y++)
for (int x=0; x<Width-1; x++)
{
      glTexCoord2f (x/texSize, y/texSize);
      glVertex3d (X[y][x], Y[y][x], Z[y][x]);
      glTexCoord2f (x/texSize, (y+1)/texSize);
      glVertex3d (X[y+1][x], Y[y+1][x], Z[y+1][x]);
      glTexCoord2f ((x+1)/texSize, y/texSize);
      glVertex3d (X[y][x+1], Y[y][x+1], Z[y][x+1]);

      glTexCoord2f (x/texSize, (y+1)/texSize);
      glVertex3d (X[y+1][x], Y[y+1][x], Z[y+1][x]);
      glTexCoord2f ((x+1)/texSize, (y+1)/texSize);
      glVertex3d (X[y+1][x+1], Y[y+1][x+1], Z[y+1][x+1]);
      glTexCoord2f ((x+1)/texSize, y/texSize);
      glVertex3d (X[y][x+1], Y[y][x+1], Z[y][x+1]);
}
glEnd ();
```

복사해야 하는데 이때 사용하는 텍스쳐 버퍼의 가로 또는 세로의 크기는 표 4에서 texSize로 표시하였다. OpenGL에서는 텍스쳐 버퍼의 크기를 2의 지수승으로 지정하도록 하였는데 입력 영상이 1024x768인 경우 1024x1024 크기의 텍스쳐 버퍼가 필요하며 텍스쳐 맵핑에서 사용하는 좌표는 0과 1사이로 스케일링 하여 입력한다. 따라서, 이미지에서 (u,v)의 좌표는 텍스쳐 맵에서는 $(u/1024, v/1024)$의 좌표를 가지게 된다.

그림 6에서는 삼각메쉬와 텍스쳐 맵핑에 의한 렌더링 결과를 예시하였다. 왼쪽 색상영상에서 검은색 부분은 깊이영상의 차이가 큰 부분으로, 5절에 서술한 바와 같이 임계값을 설정하여 차이가 큰 경우에는 삼각메쉬가 형성되지 않도록 하였기 때문에 생긴 것이다. 이 부분은 다른 시점에서 획득한 입력 데이터를 사용함으로써 처리할 수 있다. 이 결과는 그림 7에서 모두 여덟 개의 (깊이영상, 색상영상) 짝을 사용하여 렌더링한 결과를 보였고, 그림 6의 영상에서 데이터가 없었던 부분이 잘 렌더링 되어 있음을 관찰할 수 있다.

4. 중간시점 깊이 영상(depth map)의 생성

일반적인 2차원 디스플레이를 사용하여 결과를 보여주는 경우나, 3차원 다시점 디스플레이 중에서 Stereographic 사의 SynthaGram 같은 종류의 디스플레이로 3차원 출력을 보여주는 경우에는 평행으로 설치된 9대의 가상카메라로 얻은 영상을 입력으로 받는다. 따라서, OpenGL의 2차원 영상 렌더링 기능을 사용하여, 그래픽 카메라를 9개의 위치에 설정 각각의 중간영상을 합성하고 그 결과를 SynthaGram 입력으로 사용하는 것이 가능하다. 그렇지만 Philips 사의 3D-Wow 디스플레이 같은 경우는 입력으로 3D 깊이 영상과 색상 영상(텍스쳐)을 사용한다. 따라서 중간시점에서 새로운 영상을 합성한 후 OpenGL 프레임버퍼에서 색상영상을 획득하는 것뿐만 아니라 그 색상 영상에 해당하는 깊이영상을 생성해야 할 필요가 있다. 이를 위해서는 OpenGL의 깊이 버퍼(depth buffer)의 값을 읽고 이를 물체중심 좌표계에 대한 깊이정보로 변환하는 함수로 OpenGL Utility Library (GLU)에서 제공하는 함수들 중에 ***gluUnProject ()*** 가 있다.

```
int gluUnProject (double winX, double winY,
        double winZ, const double *modelview,
        const double *projection,
        const int *viewport, double *objX,
        double *objY, double *objZ);
```

함수의 매개변수를 살펴보면, (winX, winY, winZ)는 픽셀단위의 디지털 영상의 특정위치 (winX, winY) 위치의 깊이(depth) 값 winZ를 나타내는데 표 5의 함수를 호출함

그림 6. 삼각메쉬와 텍스쳐 맵핑에 의한 렌더링
Fig. 6. Rendering result from triangular mesh and texture mapping

표 5. 특정 위치의 깊이 값을 계산해주는 함수
Table 5. A function computing the depth of a 3D point

```
double winZ[Width*Height];
glReadPixels (0,0, Width, Height, GL_DEPTH_COMPONENT, GL_DOUBLE, winZ);
```

표 6. modelview, projection, viewport 행렬 획득 함수들
Table 6. Functions for computing modelview, projection, viewport matrices

```
GLint viewport[4];
GLdouble modelview[16];
GLdouble projection[16];

glGetDoublev( GL_MODELVIEW_MATRIX, modelview );
glGetDoublev( GL_PROJECTION_MATRIX, projection );
glGetIntegerv( GL_VIEWPORT, viewport );
```

표 7. 깊이영상 생성 순서
Table 7. Order of depth map generation

```
1  for (int x=0; x<Width; x++)
2     for (int y=0; y<Height; y++)
3        {
4        winX = (double)x;
5        winY = (double)viewport[3] - (double)y;
6        int k = x + Width*(Height-1-y);
7        gluUnProject( winX, winY, winZ[k],
8                      modelview, projection, viewport,
9                      &objX, &objY, &objZ);
10       depthmap[y][x] = (double) objZ;
11       }
```

으로써 영상전체에 대한 winZ값을 깊이 버퍼에서 얻을 수 있다.

그리고, modelview, projection, viewport 는 각각 현재 카메라 시점에서의 model-view transformation 행렬, 투영 행렬(projection matrix), viewport 행렬에 해당한다. 이 값들은 표 6에서의 OpenGL 함수 호출에 의해 획득가능하다.

끝으로 물체중심의 좌표계에 대한 깊이영상은 표 7과 같은 순서로 구해진다.

이 프로그램 코드에서 여섯 번째 줄은 디지털 영상의 y-축 방향이 OpenGL에서는 왼쪽 아래가 원점이지만, 디지털 영상의 픽셀단위에서는 왼쪽 위가 원점이 되기 때문에 필요한 부분이다. 이 부분에 대한 처리를 하지 않으면 영상의 위와 아래가 뒤집힌 형태로 나타나게 된다.

마지막으로, 만약 현재 카메라를 중심으로 설정된 좌표계에 대한 깊이영상을 얻고 싶으면 앞에서 구한 modelview 행렬을 위에서 구한 (objX, objY, objZ)에 곱하면 된다. modelview 행렬은 카메라 중심좌표와 물체 중심좌표계 사이의 회전변환과 직선이동변환을 담고 있다.

5. 색상 영상 및 깊이 영상의 경계부분 처리

실제 카메라로부터 주어지는 깊이영상에서 경계부분은 상대적으로 거리의 변화가 큰 부분을 의미한다. 일반 밝기 영상에 대하여 밝기의 차이를 이용한 경계선을 구할 수 있

는 것과 마찬가지로 깊이영상에 대해서도 동일한 연산을 적용할 수 있다. 삼각메쉬를 형성할 때 불연속 부분에 대한 경계를 그대로 유지하게 되면, 그래픽 카메라의 위치가 실제 카메라의 위치와 많이 다른 경우 불연속 경계 부분에서 심하게 왜곡된 보간 결과를 얻게 된다. 미리 정해둔 임계치(threshold)를 기준으로 이웃한 두 픽셀사이의 깊이 차이가 큰 경우에는 두 픽셀 사이의 연결을 끊어 줌으로써 삼각메쉬를 형성할 때 불연속이 되는 부분들을 허용할 수 있다. 그림 6의 영상에서 보이는 렌더링 되지 않은 부분들이 다른 입력 데이터의 깊이영상에 의해 렌더링 되어 사라졌음을 볼 수 있다. 오른쪽 깊이영상 결과는 4절의 중간영상에 대한 깊이영상을 구하는 방법을 적용하였는데 새롭게 랜더링 된 부분도 원본 영상들의 깊이영상들 처럼 부드럽게 이어져 있음을 볼 수 있다.

6. 결과 비교

지금까지 OpenGL을 사용하여 중간시점을 렌더링하는 방법에 대해서 살펴보았다. 하지만, 중간 시점을 합성하는 방법으로 OpenGL을 이용한 방법 이외에도 이미지 기반 렌더링(IBR)을 통해서도 결과를 생성할 수 있다. 이 두 방법은 서로 장단점을 가지고 있지만, 이미지 기반 렌더링 방법에 비해 OpenGL을 사용한 렌더링 방법이 장점이 많은 것을 알 수 있다. 그 내용을 살펴보면 다음과 같다.

그림 7. 8대의 입력 카메라로부터 주어진 정보를 사용하여 렌더링 한 중간영상과 깊이영상
Fig. 7. Novel view and depth map synthesis using 8-view input data

그림 8. 다른 영상에 대한 가상 시점 렌더링 결과 이미지
Fig. 8. Other novel view rendering results

(a) (b)

그림 9. (a) 이미지 기반 렌더링 결과 (b) 제안 알고리즘으로 렌더링 한 결과

Fig. 9. (a) IBR rendering result (b) Our algorithm rendering result

본 논문에서 제안하는 알고리즘은 3차원 공간내에 영상 모델링이 끝난 후에는 사용자가 임의의 시점을 지정해 주면 실시간으로 결과 영상이 렌더링 될 수 있는 장점을 가지고 있다. 반면에, 영상 기반 렌더링 방법으로는 한 장의 결과를 얻기 위해 픽셀 단위로 렌더링 작업을 해주어야 하기 때문에 시간이 많이 걸린다는 단점이 있다. 뿐만 아니라, 그림 9에서 보는 바와 같이 (a)의 경우 가장 왼쪽에 있는 사람의 얼굴 부분을 보면 얼굴 부분이 완전하게 렌더링 되지 않는 결과를 볼 수 있다. 이 문제는 깊이정보에 대한 버퍼링 작업을 해주면 해결할 수 있지만, 이전 보다 작업 시간이 좀 더 걸리게 된다. 이러한 문제점 외에도 (a) 렌더링 방법은 픽셀 단위의 렌더링 방법이므로 렌더링을 하게 되면 색상값이 결정되지 않는 픽셀이 생기게 된다. 이것은 렌더링 된 주변의 픽셀 값으로 채워주게 되는데 이로 인해 원하지 않는 결과를 얻을 수 있게 된다. 또한, 이미지 기반 렌더링 방법은 중간시점에 대한 깊이 영상을 얻기 힘들어 증강현실 혹은 혼합현실을 응용하기 어렵다는 단점도 가지고 있다. 이와는 반대로 제안하는 알고리즘은 픽셀 단위의 렌더링 방법이 아니므로 물체 내부에서는 홀이 생기지 않고, 영상을 3차원으로 재구성하였기 때문에 증강현실을 표현함에 있어 손쉽게 구현이 가능하다는 장점을 가지고 있다.

Ⅲ. 결 론

본 논문에서는 다중 카메라에서 주어진 색상 영상과 깊이영상을 입력으로 OpenGL을 사용하여 임의시점에서의 중간영상을 합성하는 방법과 중간영상에서의 깊이영상 추출 방법을 제안하였다. 영상기반 렌더링의 실시간 처리를 위해서는 GPU를 직접 제어하는 프로그램의 개발이 필수적이지만, 본 논문에서는 OpenGL에서 제공하는 그래픽 렌더링 엔진을 사용하여 물체가 생성될 공간을 형성한 후 그 공간 내에 3차원 물체를 위치시키고 실시간으로 중간영상을 합성하는 것이 가능한 것을 알 수 있었다. 특히 3차원 디스플레이의 표준이 정해지지 않은 현재 시점에서는 다양한 3차원 디스플레이를 테스트해야 할 필요가 있으며 OpenGL을 사용하면 기본적인 컴퓨터 그래픽스의 기능을 사용한 중간시점 렌더링이 가능하며 아울러 중간시점에서의 깊이영상도 쉽게 구할 수 있다는 장점을 가지고 있다. 아울러, 3DTV의 콘텐츠 구현에 있어 기본적 기능으로 간주되는 실사와 컴퓨터 그래픽스를 혼합하는 증강현실 또는 혼합현실을 구현하고자 할 때에도 OpenGL 렌더링 방법을 사용한다면 쉽고 빠르게 결과를 얻을 수 있을 것이다.

참 고 문 헌

[1] O. Schreer, P. Kauff, T. Sikora, "3D Videocommunication : Algorithms, concepts and real-time systems in human centred communication," John Wiley & Sons Inc, 2005.

[2] J.K. Seo, G. Sharp, S. Lee, "Range data registration using photometric features,"IEEE International Conference on Computer Vision and Pattern Recognition, 2005.

[3] C. Je, S. Lee, R.-H. Park, "High-contrast color-stripe pattern for rapid structured-light range imaging,"European Conference on Compuer Vision (ECCV), 2004.

[4] D. Scharstein, "View Synthesis Using Stereo Vision,"Ph.D. Thesis, Cornell University, January 1997.

[5] C. Zitnick, Sing Bing Kang, M. Uyttendaele, S. Winder, R. Szeliski, "High-quality video view interpolation using a layered representation,"ACM SIGGRAPH, 2004

[6] R.I. Hartley and A. Zisserman, "Multiple View Geometry," Cambridge University Press, 2nd Edition, 2002.

[7] Z. Zhang, "Flexible camera calibration by viewing a plane from unknown orientation,"IEEE International Conference on Computer Vision, 2000.

[8] J.Y. Bouguet, "Camera calibration toolbox for matlab," http://www.vision.caltech.edu/bouguetj/calib_doc

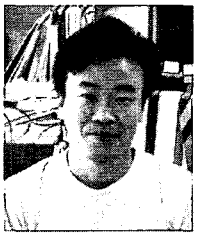
[9] D. Shreiner, M. Woo, J. Nelder, T. Davis, "OpenGL Programming Guide: The Official Guide to Learning OpenGL, 5th Edition," Addison-Wesley Professional, 2005.

[10] D. Shreiner, Editor, OpenGL Reference Manual, "The Official Reference to OpenGL, 4th Edition," Addison-Wesley Professional, 2004.

저 자 소 개

이 현 정

· 2003년 2월 : 성신여자대학교 컴퓨터공학과 학사
· 2005년 8월 : 서강대학교 영상대학원 미디어공학과 석사
· 2005년 9월 ~ 현재 : 서강대학교 영상대학원 미디어공학과 박사과정
· 주관심분야 : 3DTV, novel view synthesis



허 남 호

· 1992년 2월 : 포항공과대학교 전기전자공학과 학사
· 1994년 2월 : 포항공과대학교 대학원 전기전자공학과 석사
· 2000년 2월 : 포항공과대학교 대학원 전기전자공학과 박사
· 2000년 4월 ~ 현재 : 한국전자통신연구원 전파방송연구단 방송시스템연구그룹 3DTV시스템연구팀
· 주관심분야 : 3DTV, Free-viewpoint TV

서 용 덕

· 1992년 2월 : 경북대학교 전기전자공학과 학사
· 1994년 2월 : 포항공과대학교 대학원 전기전자공학과 석사
· 2000년 2월 : 포항공과대학교 대학원 전기전자공학과 박사
· 2005년 ~ 현재 : 서강대학교 영상대학원 미디어공학과 교수
· 주관심분야 : 3DTV, Non-metric augmented reality, Flexible Camera Self-Calibration