특집논문-08-13-6-08

GPGPU를 이용한 고속 영상 합성 기법

신 홍 창[a], 박 한 훈[b], 박 종 일[a]‡

Fast View Synthesis Using GPGPU

Hong-Chang Shin[a], Hanhoon Park[b], and Jong-Il Park[a]‡

요 약

본 논문은 3차원 디스플레이 시스템에서 카메라의 기하 정보 및 참조 영상들의 깊이 맵 정보가 주어졌을 때, 다수의 중간 시점 영상을 실시간으로 생성하는 고속 영상 합성 기법을 제안한다. 기본적으로 본 논문에서는 영상 합성 기법의 모든 과정을 GPU에서 병렬 처리함으로써 고속화 할 수 있었다. 병렬처리를 이용한 고속화 효율을 높이기 위해 최근 NVIDIA사에서 발표한 CUDA™를 이용하였다. 영상 합성을 위한 모든 중간 과정을 CUDA로 처리하기 위해 병렬구조로 변환하고, GPU 상의 고속메모리의 사용을 극대화하고, 알고리즘 구현을 최적화함으로써 고속화 효율을 높일 수 있었다. 결과적으로 본 논문에서는 양안 영상과 깊이 지도를 이용하여 가로 720, 세로 480 크기의 9개의 시점 영상을 0.128초 이내에 생성할 수 있었다.

Abstract

In this paper, we develop a fast view synthesis method that generates multiple intermediate views in real-time for the 3D display system when the camera geometry and depth map of reference views are given in advance. The proposed method achieves faster view synthesis than previous approaches in GPU by processing in parallel the entire computations required for the view synthesis. Specifically, we use CUDA™ (by NVIDIA) to control GPU device. For increasing the processing speed, we adapted all the processes for the view synthesis to single instruction multiple data (SIMD) structure that is a main feature of CUDA, maximized the use of the high-speed memories on GPU device, and optimized the implementation. As a result, we could synthesize 9 intermediate view images with the size of 720 by 480 pixels within 0.128 second.

Keyword : 3D display, View synthesis, GPGPU, CUDA

I. 서 론

3차원 디스플레이는 기존의 2차원 디스플레이 영상에 깊이 정보를 부가하여 사용자로 하여금 입체감을 느끼게 하여 실감형 멀티미디어 서비스를 즐길 수 있도록 해준다. 최근 3차원 입체 영상 디스플레이에 관한 연구개발이 활발하게 진행되고 있으며, 실제 상용화된 제품들도 출시되고 있다[1].

본질적으로 3차원 멀티미디어 서비스를 제공하려면, 여러 시점에서 찍은 실제 카메라 영상을 디스플레이 기기에 제공해줄 수 있어야 한다. 하지만 시점의 개수와 동일한 수의 많은 카메라가 필요하며, 이들 카메라 간의 동기화 및 방대한 데이터 처리와 전송 등의 현실적인 문제가 해결되

a) 한양대학교 전자통신컴퓨터공학과
Department of Electronics and Computer Engineering, Hanyang University.
b) 일본 NHK 방송기술연구소
NHK Science & Technical Research Laboratories.
‡ 교신저자 : 박종일(jipark@hanyang.ac.kr)

어야 한다. 이러한 현실적인 문제로 인해서 한정된 시점 영상을 이용하여 여러 중간 시점 영상을 생성하는 영상 합성 방법이 3차원 디스플레이 분야에서 중요한 이슈가 되고 있다. 이러한 방법의 예로 여러 대의 카메라를 사용하는 다안 스테레오그램 방식을 이용하여 특수 안경이 없어도 입체화가 가능한 범위를 넓히거나 측면 부분도 보이도록 중간 시점을 보간 생성하는 연구가 있었다[2].

양안 영상을 이용하여 중간 시점 영상을 생성하는 방법에 대한 연구도 있었다. Park 등은 두 카메라 시점의 영상으로 깊이 지도를 추정하는 방법과 이를 이용하여 중간 시점을 생성하여 3차원 디스플레이에 적용하는 방법은 제안하였다[3]. 한정된 수의 시점 영상과 깊이 지도를 이용하여 그 사이의 중간 시점들을 생성할 때에는 정보가 없거나 부정확한 물체 경계 부분이 흐릿해지거나 유령현상이 생기지 않도록 정확하게 보관하는 것이 중요하다. 이러한 시각 효과를 줄이기 위해 MAP(maximum a posteriori probability) 기준을 기반으로 하여 “winner-takes-all" 기법을 쓰는 방법도 있었다[4]. 하지만 이런 기법들은 최적화 해를 구하는 데에 많은 시간이 소요된다는 큰 단점이 있다.

3차원 디스플레이로의 실용화를 위해서 여러 중간 시점 영상을 동시 생성할 수 있는 3차원 영상 제작 기술 개발을 하려면 좀 더 구체적인 방법들에 대해 논의가 이루어져야 한다. 특히 실용화를 위해서는 효과적인 중간 시점 생성 방법도 중요하지만 이를 실시간으로 생성하는 문제도 중요하기 때문에 영상 합성 기법을 실시간으로 생성할 수 있는 고속화 기법에 대한 논의도 이루어져야 한다.

최근 컴퓨터 비전 분야에서도 속도를 개선하기 위해 GPU기반의 알고리즘에 대한 연구가 활발히 진행되고 있다. 크기, 회전 불변의 특징점을 검출하는 방법과 캐니 에지 검출방법을 그래픽 하드웨어를 이용하여 향상되는 속도를 비교한 연구가 있었으며[5,6], 화상 복원, 스테레오 비전, 영상 분할 등의 분야에서 최적화 해를 찾는 데에 쓰이는 강력한 방법인 그래프-컷(Graph Cuts)을 그래픽 하드웨어를 이용하여 구현한 사례도 있었다[7]. 이러한 연구들은 그래픽 하드웨어를 범용 연산 목적으로 활용할 수 있도록 고안된 NVIDIA사의 CUDA™를 이용하여 그래픽 하드웨어 구조에 최적화되도록 알고리즘을 병렬로 구현함으로써 향상된 속도를 얻을 수 있었다.

본 논문에서도 기존 영상 합성 기법의 속도 측면에서의 한계를 해결하고자 GPU기반의 고속 영상 합성 기법을 제안한다. 영상 합성을 이용하여 여러 시점을 동시에 생성하는 데에 소요되는 시간을 효과적으로 줄이고자, 영상 합성 기법을 병렬화하고 NVIDIA사의 CUDA™를 이용하여 구현하였다. 일반적으로 모든 영상처리 알고리즘이 그렇듯 현존하는 영상 합성 기법에도 품질과 소요 시간 사이에 상충 관계가 있다. 이러한 측면에서 본 논문에서는 영상 합성 기법의 고속화에 중점을 두었고 제안한 기법의 3차원 디스플레이로의 실용 가능성을 알아본다.

본 논문은 다음과 같이 구성된다. 2장에서는 3차원 디스플레이 시스템에서의 영상 합성의 원리와 방법에 대해 설명하고, 3장에서는 병렬처리를 이용한 과정별 고속화 방안을 제시하고, 4장에서는 실험 결과를 통하여 과정별 소요시간을 분석하였으며, 5장에서는 본 논문의 결론과 앞으로의 연구방향을 언급한다.

II. 3차원 디스플레이 시스템

일반적으로, 3차원 디스플레이 시스템은 다수의 시점에서의 3차원 영상을 동시에 제공해 주어야 한다. 이를 위해서는 다수의 카메라를 이용하여 다수의 시점에서 얻어진

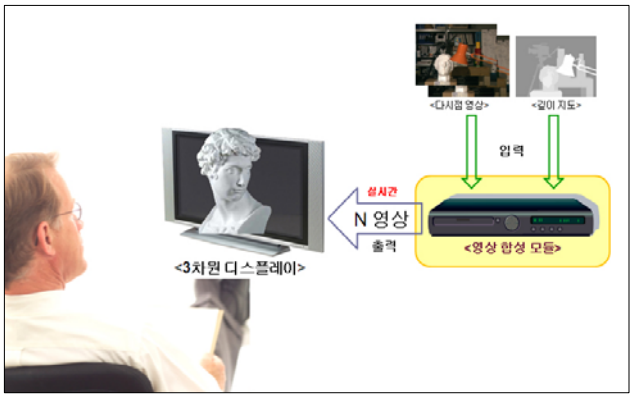


그림 1 . 3차원 디스플레이 시스템
Fig 1. 3D display system

영상을 전송해 주어야 하는데, 시점의 수가 많아질수록 데이터량이 급격히 증가하여, 3차원 영상의 실시간 제작이 불가능해질 수 있다. 그러나 영상 합성 기술을 이용하면 적은 수의 카메라로부터 획득한 영상을 전송하여, 다수의 시점을 생성해낼 수 있다. 그림 1은 본 논문에서 제안하는 영상 합성을 이용한 3차원 디스플레이 시스템 개략도이다.

개략도를 보면 스테레오 카메라로 촬영한 좌우 영상과 깊이 지도가 전송되면 디스플레이 장치 내의 내부 모듈에서 영상 합성 작업을 하여 N개 시점에서의 영상을 만들고, 이를 3차원 디스플레이 장치를 이용해서 동시에 출력해준다. 이와 같이, 영상 합성을 이용할 경우, 모든 시점을 전송하는 것에 비해 데이터 전송량은 크게 줄일 수 있으나, 디스플레이 시스템에서 많은 수의 시점 영상을 합성해 내야 한다. 시점의 개수가 늘어날 경우, 처리 속도가 크게 떨어질 수 있다. 따라서 고속으로 가상 시점을 합성해 내는 기술이 필요하다.

1. 깊이 지도를 이용한 영상 합성 기법

깊이 지도를 이용하여 영상 합성을 하기 위해서는 먼저 카메라 보정(camera calibration) 작업을 통해 얻은 기준 카메라 사이의 기하 관계와 스테레오 정합(stereo matching) 이나 혹은 Z-CAM과 같은 깊이 지도 추출 카메라를 이용해서 얻은 깊이 지도 영상이 있어야 한다.

본 논문에서는 영상 합성을 하기에 앞서 카메라 기하 관계와 깊이 지도 정보가 주어진다고 가정을 하고 실험을 하였으며, 다음 절에서는 본 논문에서 제안하는 영상 합성 기법의 원리와 방법을 설명한다.

1.1. 원리

그림 2에서 보는 것처럼, 전체적인 기준이 되는 좌표계를 X_0, 카메라 a의 좌표계를 X_a, 카메라 b의 좌표계를 X_b라 한다. X_0에서 X_a로의 평행이동은 T_{0a}, 방향은 R_{0a}라고 하고, T_{0b}와 R_{ob}는 X_0와 X_b사이의 관계이며 다음과 같이 표현된다.

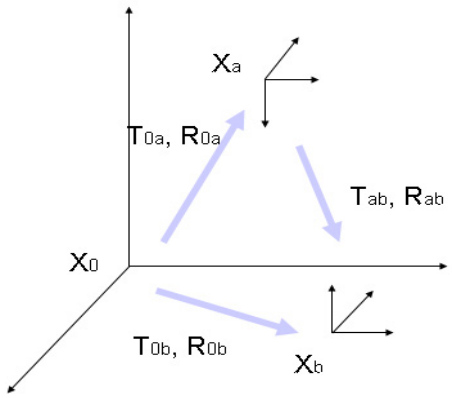


그림 2. 기준 좌표계와 카메라 좌표계
Fig 2. World coordinate system & camera coordinate system

$$X_a = R_{oa}X_0 + T_{oa}, \tag{1}$$

$$X_b = R_{ob}X_0 + T_{ob}. \tag{2}$$

T는 카메라의 평행이동 $T=(T_x, T_y, T_z)$를 의미하고, R은 3차원 회전 변환을 의미하며 다음과 같이 표현된다.

$$R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}. \tag{3}$$

X_a와 X_b 사이의 관계는 식 (1)로부터 아래와 같이 유도할 수 있다.

$$X_0 = R_{0a}^{-1}(X_a - T_{0a}). \tag{4}$$

식 (2)에 식 (4)를 대입하고 정리하면,

$$\begin{aligned} X_b &= R_{0b}R_{0a}^{-1}(X_a - T_{0a}) + T_{0b} \\ &= R_{0b}R_{0a}^{-1}X_a - T_{0b} - R_{0b}R_{0a}^{-1}T_{0a}, \end{aligned} \tag{5}$$

$$R_{0b}R_{0a}^{-1} = R_{ab},\ \ T_{0b} - R_{0b}R_{0a}^{-1}T_{0a} = T_{ab}. \tag{6}$$

최종적으로 아래와 같은 변환식을 얻을 수 있다.

$$X_b = R_{ab}X_a + T_{ab}. \tag{7}$$

카메라 a와 카메라 b의 관계를 임의 시점 생성과 관련해서 설명하면, 카메라 a는 참조 카메라가 되고, 카메라 b는 임의시점 카메라가 되므로, 카메라 a와 기준 좌표계와의 관계와 식 (7)로부터 임의 시점 카메라의 기준 좌표계와의 관계를 얻을 수 있다. 그림 3처럼 카메라 a의 좌표계가 기준 좌표계와 일치할 경우, 식은 더욱 간소화된다.

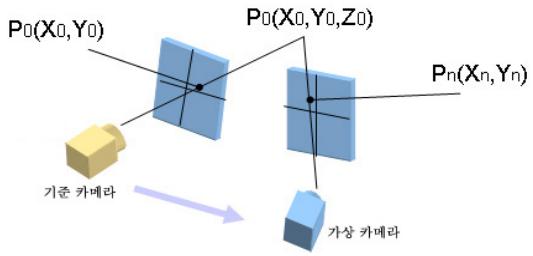


그림 3. 카메라 평행 이동과 회전
Fig 3. Translation and rotation of camera

참조 카메라의 좌표계를 X_0, 임의시점 카메라의 좌표계를 X_n이라고 하고, 카메라의 평행 이동을 T_{0n}, 회전을 R_{0n}로 하면 X_n은 식(7)로부터 아래와 같이 구해진다.

$$X_n = R_{0n}X_0 + T_{0n}. \tag{8}$$

좌표계 X_0의 점 $P_0(X_0, Y_0, Z_0)$는 아래의 식에 의해 참조 카메라에서 영상 평면상의 점 $p_0(x_0, y_0)$로 투영된다.

$$x_0 = F_0\frac{X_0}{Z_0},\ \ y_0 = F_0\frac{Y_0}{Z_0}. \tag{9}$$

여기서, F_0는 참조 카메라의 초점거리를 의미한다. 이와 동일하게 좌표계 X_n의 점 $P_n(X_n, Y_n, Z_n)$는 임의시점 카메라의 영상 평면상의 점 $p_n(x_n, y_n)$로 투영된다.

$$x_n = F_n\frac{X_n}{Z_n},\ \ y_n = F_n\frac{Y_n}{Z_n}. \tag{10}$$

식 (8)을 풀어보면,

$$\begin{cases} X_n = r_{11}X_0 + r_{12}Y_0 + r_{13}Z_0 + T_x \\ Y_n = r_{21}X_0 + r_{22}Y_0 + r_{23}Z_0 + T_y \\ Z_n = r_{31}X_0 + r_{32}Y_0 + r_{33}Z_0 + T_z \end{cases}. \tag{11}$$

식 (11)에 식 (10)을 대입한다.

$$\begin{aligned} x_n &= F_n\frac{X_n}{Z_n} = F_n\frac{r_{11}X_0 + r_{12}Y_0 + r_{13}Z_0 + T_x}{r_{31}X_0 + r_{32}Y_0 + r_{33}Z_0 + T_z}, \\ y_n &= F_n\frac{Y_n}{Z_n} = F_n\frac{r_{21}X_0 + r_{22}Y_0 + r_{23}Z_0 + T_y}{r_{31}X_0 + r_{32}Y_0 + r_{33}Z_0 + T_z}. \end{aligned} \tag{12}$$

여기서 F_n은 임의시점 카메라의 초점거리이다.

$$X_0 = x_0\frac{Z_0}{F_0},\ \ Y_0 = y_0\frac{Z_0}{F_0}. \tag{13}$$

식 (13)를 식 (12)에 대입하고 정리하면 최종적으로 다음과 같은 식을 얻을 수 있다.

$$\begin{aligned} x_n &= F_n\frac{r_{11}x_0 + r_{12}y_0 + r_{13}F_0 + \frac{F_0}{Z_0}T_x}{r_{31}x_0 + r_{32}y_0 + r_{33}F_0 + \frac{F_0}{Z_0}T_z}, \\ y_n &= F_n\frac{r_{21}x_0 + r_{22}y_0 + r_{23}F_0 + \frac{F_0}{Z_0}T_y}{r_{31}x_0 + r_{32}y_0 + r_{33}F_0 + \frac{F_0}{Z_0}T_z}. \end{aligned} \tag{14}$$

1.2 방법

기준 카메라로부터 가상 시점에서의 영상을 합성하기 위해서는 두 가지 문제를 해결해야 한다.

① 기준 카메라로 촬영한 기준 영상을 가상 카메라의 영상에 3차원 좌표 변환한다.
② 기준 영상에는 나타나지 않는 부분에 대해서는 다른 시점에 위치한 카메라로부터 정보를 얻는다.

식 (14)에서 나타냈던 것처럼 3차원 좌표 변환 계산에는 z축의 값이 필요하다. 이 깊이 정보는 깊이 지도에서 취득한다. 3차원 좌표 변환은 기준 영상을 가상 영상으로 사상하는 것이다. 사상의 방법으로 바람직한 것은 한 번의 계산으로 가상 영역의 모든 점에 정보를 보낼 수 있는 후방 사상이지만, 가상 영상의 깊이 정보는 원래 존재하지 않으므로 가상 영상으로부터 기준 영상을 사상을 계산할 수 없다. 그렇기 때문에 기준 영상에서 가상 영상으로의 3차원 좌표변환은 전방 사상으로 구한다.

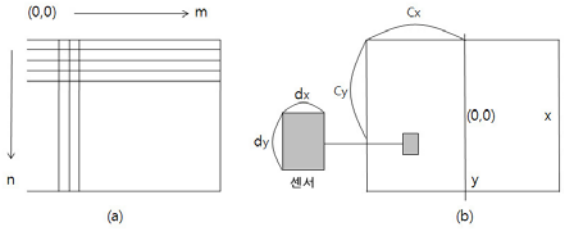


그림 4. (a) 메모리 상의 이미지 데이터, (b) CCD 카메라
Fig 4. (a) Image data in memory, (b) CCD camera

카메라 시스템을 구성하는 카메라는 CCD 카메라로, CCD 카메라의 센서(sensor) 영역은 그림 4(b)와 같고, 그것에 의해서 생성한 영상의 메모리 구조는 그림 4(a)와 같다. 이 둘 사이에는 다음과 같은 관계가 성립한다.

$$x = \frac{dx(m - C_x)}{sx}, \quad y = dy(n - C_y). \tag{15}$$

- C_x, C_y : 카메라의 광축이 촬영 평면상과 교차한 점 [pixel]
- dx, dy : 셀 하나의 상하 길이 [mm/pixel]
- sx : dx의 오차를 보정하기 위한 계수

식 (15)를 식 (14)에 대입하고 정리하면 아래의 식을 얻을 수 있다.

$$m_v = \frac{a_1 m_0 + a_2 n_0 + a_3 Z_0(m_0,n_0)^{-1} + a_4}{a_9 m_0 + a_{10} n_0 + a_{11} Z_0(m_0,n_0)^{-1} + a_{12}} + C_{x_v},$$

$$n_v = \frac{a_5 m_0 + a_6 n_0 + a_7 Z_0(m_0,n_0)^{-1} + a_8}{a_9 m_0 + a_{10} n_0 + a_{11} Z_0(m_0,n_0)^{-1} + a_{12}} + C_{y_v}.$$

$Z_0(m_0,n_0)$는 $p_0(m_0,n_0)$의 값으로 한다. 여기서 쓰인 각 계수는 아래와 같다.

$$a_1 = F_v \frac{sx_v}{dx_v} \frac{dx_0}{sx_0} r_{11},$$

$$a_2 = F_v \frac{sx_v}{dx_v} dy_0 r_{12},$$

$$a_3 = F_v \frac{sx_v}{dx_v} F_0 T_x,$$

$$a_4 = -F_v \frac{sx_v}{dx_v}(-r_{13}F_0 + \frac{dx_0}{sx_0} r_{11} C_{x_0} + dy_0 r_{12} C_{y_0}),$$

$$a_5 = F_v \frac{1}{dy_v} \frac{dx_0}{sx_0} r_{21},$$

$$a_6 = F_v \frac{1}{dy_v} dy_0 r_{22},$$

$$a_7 = F_v \frac{1}{dy_v} F_0 T_y,$$

$$a_8 = -F_v \frac{1}{dy_v}(-r_{23}F_0 + \frac{dx_0}{sx_0} r_{21} C_{x_0} + dy_0 r_{22} C_{y_0}),$$

$$a_9 = \frac{dx_0}{sx_0} r_{31},$$

$$a_{10} = dy_0 r_{32},$$

$$a_{11} = F_0 T_z,$$

$$a_{12} = r_{33}F_0 - (\frac{dx_0}{sx_0} r_{31} C_{x_0} + dy_0 r_{32} C_{y_0}).$$

위 식과 기준 영상 상의 좌표 $p_0(m_0,n_0)$를 이용해서 가상 영상 상의 좌표를 계산할 수 있다.

이런 과정을 통해 얻어진 관계식을 이용하여 가상 카메라의 위치로 전방 사상을 한다. 이렇게 구한 가상 시점의 깊이 지도는 새롭게 나타나는 영역(uncovered area)으로 인하여 틈(hole)을 가지게 되는데, 이는 보간법(interpolation)을 이용하여 채워줄 수 있다. 이 때, 깊이 지도 보간은 가상 카메라의 이동 방향을 고려해야 한다. 가상 카메라가 이동

하면 영상 안의 물체는 에피폴라 선(epipolar line)을 따라 반대로 이동하기 때문에 보간 역시 에피폴라 선의 방향에 따라 실시한다. 또한, 새롭게 나타나는 영역의 깊이 정보는 앞 쪽의 물체에 가려져 있던 영역이므로 뒤 쪽의 배경에 속할 것이라고 추정할 수 있다. 그렇기 때문에 그림 6과 같이 에피폴라 선 방향으로 뒤 쪽의 배경에서의 깊이 정보를 참고하여 가까운 점의 정보를 이용하여 보간한다. 보간의 방법은 여러 가지 방법이 있으나 본 실험에서는 고속화를 위해 단순 선형 보간을 하였다.

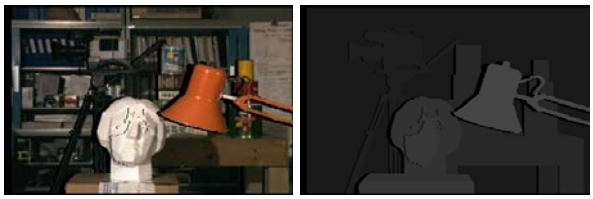


그림 5. 전방 사상된 영상과 노출 영역
Fig 5. Forward-mapped image & uncovered area

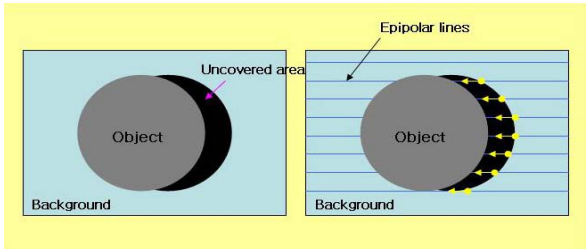


그림 6. 방향성 깊이 지도 보간
Fig 6. Directional depth interpolation

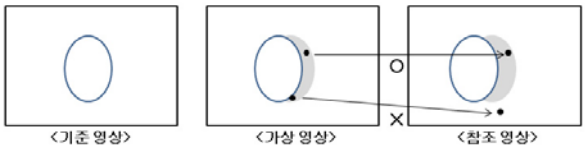


그림 7. 가상 시점 영상의 유효성 판정
Fig 7. Validation check of virtual view

방향성 깊이 지도 보간까지 이루어지면 보간된 깊이 정보를 이용하여 가상 시점의 영상에서 참조 영상으로의 사상 관계를 이용한 후방 사상(backward mapping)을 통해 텍스쳐 정보를 가져와서 새롭게 나타나는 영역을 채울 수 있다. 참조 영상의 좌표를 계산할 때, 계산된 좌표의 정보가 기준 영상에는 없고, 참조 영상에만 있는 정보인지 확인하는 유효성 검사를 해야 한다. 이를 확인하는 방법은 그림 7처럼 기준 영상의 깊이 지도를 참조 영상의 위치로 전방 사상했을 때 생기는 새롭게 나타나는 영역으로 사상이 되는 지를 확인하면 된다. 가상 시점 영상의 유효 판정을 통해 확인이 되면 계산된 점을 후방사상으로 채워주고, 확인되지 않은 점은 최종적으로 인접 화소 정보를 이용하여 보간을 해서 채워준다.

III. 고속 영상 합성

고속화를 하기 위해서는 먼저 현재 영상 합성 기법 과정을 분석하여 고속화가 가능한 부분을 찾아야 한다. 영상 합성 과정은 기본적으로 그림 8처럼 순차적인 구조로 이루어진다.

그림 8. 영상 합성 과정의 순차적인 구조
Fig 8. Sequential view synthesis

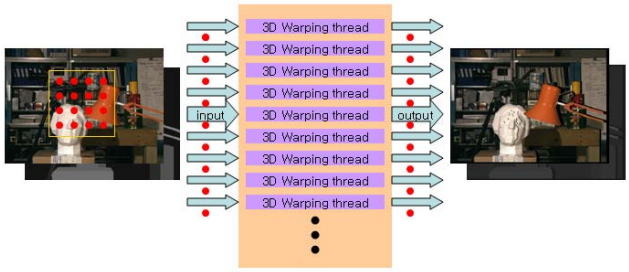


그림 9. 병렬처리 구조
Fig 9. Parallel view synthesis

그림 8의 왼쪽 그림의 점은 영상의 화소 하나를 의미한

다. 기준 영상의 각 화소 정보를 입력받으면, 미리 계산된 임의의 위치의 가상 시점 의로의 사상 관계식을 통해 계산된 목표 가상 시점으로 화소 정보를 복사함으로써 이루어지게 된다. 이때, 입력되는 각각의 화소 정보는 다르지만, 임의의 한 가상 시점으로의 사상에 쓰이는 관계식은 동일하기 때문에 SIMD(Single Instruction Multiple Data)구조로 볼 수 있으며, 이러한 SIMD 구조는 병렬처리를 통해 고속화가 가능하다.

그림 9는 영상 합성을 병렬처리로 할 때의 구조를 나타낸다. 동일한 작업(Single Instruction)을 하는 그룹 내의 쓰레드(thread)가 각각 화소(Multiple Data)를 하나씩 맡아서 동시에 처리를 한다면 순차적인 구조에 비해 월등한 속도 향상을 이룰 수 있다.

본 논문에서는 위의 개념을 기본으로 GPU를 이용한 병렬처리를 통해 고속화 실험을 하였고, 이를 위해 NVIDIA 사에서 범용 목적의 GPU연산을 위해 개발한 CUDA™를 사용하였다[10,11].

1. CUDA

CUDA로 프로그래밍을 하면 GPU는 많은 수의 쓰레드를 병렬로 실행시킬 수 있게 된다. 주 연산 장치인 CPU의 코프로세서(coprocessor)로서의 역할을 하는데, 호스트(host)에서 실행되는 응용 프로그램에서 병렬처리가 가능한 연산 집약적인 부분을 담당한다. 좀 더 자세히 설명하면, 각각 다른 데이터에 대해 독립적으로 여러 번 실행이 되는 부분은 GPU 디바이스(device)에서 많은 수의 다른 쓰레드로 실행되도록 하나의 함수로 정의가 될 수 있다. 이를 위해 함수는 디바이스에서 실행 가능하도록 컴파일 되고, 커널(kernel)이라고 하는 프로그램으로 디바이스로 넘겨져 각 쓰레드에 의해 실행이 된다.

하나의 커널을 실행하는 쓰레드 군(群)은 쓰레드 블록(thread block)으로 이루어진 그리드(grid)로 구성된다. 하나의 쓰레드 블록은 빠른 접근이 가능한 공유 메모리(shared memory)를 통해 효과적으로 데이터를 공유하고, 메모리 접근을 조절하기 위해 그것들의 실행을 동기화함으로써 함께 협력을 할 수 있는 쓰레드의 한 군을 말한다.

쓰레드 블록을 포함한 그리드는 멀티프로세서의 실행을 위해 스케쥴링(scheduling) 블록에 의해 디바이스에서 실행이 된다. 각 멀티프로세서는 일괄처리가 가능한 블록들의 군을 처리한 후에 다시 다음 군을 처리하는 식으로 블록들의 군을 처리한다. 한 블록은 하나의 멀티프로세서만으로 처리가 되며, 아주 빠른 메모리 접근을 위해 on-chip 공유메모리에 블록의 공유 메모리 공간이 실제로 상주하게 된다. 각 멀티프로세서가 한 번의 일괄처리로 처리할 수 있는 블록의 수는 실행될 커널 함수에 요구되는 쓰레드 하나당 레지스터의 개수와 블록 하나당 차지하는 공유메모리의 크기에 달려있다. 왜냐하면 멀티프로세서들의 레지스터와 공유메모리는 일괄처리 되는 블록들의 모든 쓰레드에 나뉘기 때문이다.

디바이스에서 실행되는 쓰레드는 디바이스 내의 DRAM과 그 밖에 다양한 범위의 메모리 영역을 통해 on-chip 메모리에 접근이 가능하다. 전역 메모리(global memory)는 호스트와 디바이스 사이의 읽기/쓰기가 가능한 데이터를 주고 받을 때 주로 쓰는 메모리이며 모든 쓰레드에 의해 접근이 가능하다. 텍스처 메모리(texture memory)와 상수 메모리(constant memory)는 호스트에 의해 초기화가 가능하며 on-chip 메모리이기 때문에 전역 메모리에 비해 접근 속도가 빠르다.

디바이스는 멀티프로세서(multiprocessor)의 세트로 구성된다. 각각의 멀티프로세서는 단일 명령으로 다양한 데이터를 처리한다. 각 멀티프로세서는 아래의 4종류의 on-chip 메모리를 갖는다.

① 프로세서 하나당 32비트 지역 레지스터(register)
② 공유 메모리 공간을 구성하며 모든 프로세서가 공유하는 병렬 데이터 캐시(cache)와 공유 메모리(shared memory)
③ 디바이스 메모리의 읽기 전용 영역으로 구성되고, 상수 메모리 공간으로부터 읽기 속도를 빠르게 하며, 모든 프로세서가 공유하는 읽기 전용 상수 캐시(constant cache)
④ 디바이스 메모리 읽기 전용 영역으로 구성되고, 텍스처 메모리 공간으로부터 읽기 속도를 빠르게 하며,

모든 프로세서가 공유하는 읽기 전용 텍스처 캐시 (texture cache)

지역 메모리와 전역 메모리 공간은 디바이스 메모리에서의 읽기/쓰기가 가능한 영역이지만 캐시가 되지 않는다.

위와 같이 커널 함수를 담당할 쓰레드의 개수와 각 쓰레드가 병렬로 처리할 영역을 지정하는 방식, 커널 함수의 구현할 때 여러 종류의 메모리 중에 어떤 메모리를 어떻게 활용할 지를 결정하는 것이 CUDA 프로그래밍에 있어서 매우 중요하다.

2. CUDA를 이용한 과정별 고속화 기법

CPU에서의 연산을 GPU에서 연산했을 때 속도가 개선되는 경우는 그 작업이 CPU보다는 GPU에서 더 적합하기 때문이다. SIMD 구조의 연산 작업을 병렬로 처리가 가능한 GPU의 이점을 이용하였기 때문에 속도가 개선이 되는 것이다. 그렇기 때문에 앞 절에서 언급했던 것처럼 기존의 CPU 기반의 알고리즘에서 어떤 부분이 GPU에서 속도 개선이 가능한지를 분석하는 것이 선행되어야 한다. 그런 후 호스트와 디바이스에 맡게 역할 분담을 하여 전체 프로그램의 구조를 구성하여야 한다. 본 논문에서는 고속화 효율을 극대화하기 위해서 CUDA에서 영상 합성의 모든 과정을 처리할 수 있도록 병렬화하였으며, 최적화 과정을 거침으로써 고속화 효율을 높일 수 있었다.

2.1 전방 사상의 고속화

영상 합성의 첫 번째 단계는 전방 사상이다. 여기서의 전방 사상은 두 기준 영상 사이의 임의의 가상 위치로 화소 정보를 복사하는 것인데, 이 때 한 기준 영상에서 가상 영상까지의 사상 계수를 미리 계산하고 그것을 이용하여 가상 영상의 위치로 화소 정보를 복사한다.

전방 사상에 필요한 정보는 기준 색상 영상과 깊이 지도 영상의 화소이다. 하나의 화소를 받아들여서 전방 사상을 담당하는 커널 함수를 디바이스 내의 쓰레드가 하나씩 맡아서 처리함으로써 병렬화가 가능하게 된다.

그림 10은 기준 영상을 디바이스 내의 하나의 멀티프로세서가 담당할 블록단위로 지역을 나누고 쓰레드 블록 내의 쓰레드들이 각각 화소 하나씩을 처리하는 것을 그림으로 나타낸 것이다.

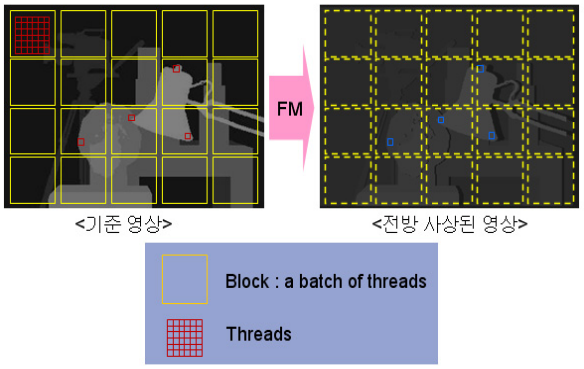


그림 10. 전방 사상의 병렬처리
Fig 10. Parallel processing of forward mapping

기본적으로 디바이스에서 실행되는 함수인 커널을 실행시키기 전에 병렬처리에 필요한 데이터를 호스트에서 먼저 계산을 하고 디바이스로 넘겨준다. 그리고 블록과 쓰레드의 차원의 수를 결정해야 하는데, 이것은 디바이스의 성능과 처리할 데이터의 양을 참고하여 적절하게 나누어야 한다.

영상은 2차원 구조의 배열이므로 그리드와 블록을 2차원으로 나누는 것이 좋다. 블록 안의 쓰레드가 영상의 화소를 하나씩 맡아서 병렬처리 하려면, 영상 화소의 개수만큼 쓰레드를 생성해야 한다. 실험에서 사용한 G80 그래픽 카드의 경우 한 블록당 쓰레드의 수는 최대 512개이다. 즉, 쓰레드를 모두 활용한다면 한 블록이 512개의 화소를 처리할 수 있다는 뜻이다. 한 블록이 영상의 512개의 화소까지만을 처리할 수 있기 때문에 영상의 해상도에 맞춰서 블록의 수를 만들면 된다. 예를 들어, 720x480 영상인 경우, 블록을 (16,32)로 구성한다면 한 블록당 최대 쓰레드의 개수가 512개를 만족하게 되고, 그리드를 (45,15)로 구성하면 모든 블록 안의 쓰레드들이 영상의 각 화소에 일대일 대응이 가능해지므로 각 화소당 쓰레드 하나의 방식으로 병렬처리가 가능해진다. 그렇다면 남은 문제는 각 쓰레드가 어떤 화소를 담당할지를 지정하는 것이다. 그것은 블록

과 쓰레드는 각자의 고유 식별번호가 있기 때문에 이를 이용하면 영상의 각 화소를 처리할 수 있도록 지정할 수 있다. 각 쓰레드가 이떤 화소를 처리할 것인지에 대한 지정이 끝나면 각 쓰레드가 병렬구조로 처리하는 동일한 함수, 즉 여기서는 한 화소의 전방 사상을 담당하는 커널을 구현하면 된다.

2.2 깊이 지도 보간의 고속화

전방 사상을 하면 새롭게 나타나는 영역이 생기기 때문에 이 부분에 대해 처리를 해야 한다. 그 영역의 정보는 다른 시점인 참조 영상에는 있지만, 새롭게 나타나는 영역의 3차원 깊이 정보가 없기 때문에 필요한 정보를 가지고 올 방법이 없다. 그렇기 때문에 깊이 지도 보간이 필요하며, 여러 보간 방법이 있지만, 여기서는 고속화와 병렬처리의 특성상 단순 선형 보간을 하였다.

여기서 사용한 보간 방법은 새롭게 나타나는 영역은 앞의 사물이 아닌 뒤쪽의 배경에 속할 것이라는 추정을 할 수 있기 때문에 카메라 이동의 반대 방향으로 뒤쪽의 배경에서의 깊이 정보를 참고하여 가까운 점의 정보를 이용하여 보간한다. 이는 방향성 보간으로서 먼저 보간이 된 결과가 나중에 보간이 될 결과에 영향을 미치게 된다. 이러한 방향성 보간은 CPU에서는 에피폴라 방향을 따라서 순차적 처리를 하면 되지만, 각 쓰레드가 동일한 함수를 독립적으로 연산하는 SIMD구조의 GPU에서는 어울리지가 않는다. 따라서 GPU의 SIMD구조에서도 방향성 보간을 효율적으로 연산할 수 있도록 변경할 필요가 있다.

이를 위해 우선 방향성 보간에서 병렬처리가 가능한 부분과 가능하지 않은 부분을 나눠야 한다. 병렬처리가 불가능한 부분은 순차적 처리로 할 수 밖에 없다. 보간 방향이 에피폴라 선을 따라서 영상의 오른쪽 끝부분에서 왼쪽으로 이동하며 점의 정보를 순차적으로 참조하는 것은 순차적으로 처리를 한다. 병렬처리가 가능한 부분은 같은 x축 좌표에 있는 점들, 즉 세로로 한 열을 이루는 화소들의 보간이다.

그림 11은 720x480 크기의 깊이 지도를 병렬처리로 보간하는 방법을 그림으로 표현한 것이다. 이를 CUDA에서 구현하려면 우선 480개의 쓰레드를 가지는 하나의 블록을 만든다. 그 블록은 우측 맨 끝을 시작으로 좌측으로 한 좌표씩 이동하면서 총719번의 보간을 한다. 그리고 그 블록에 속한 480개의 쓰레드는 블록이 이동할 때마다 입력 받은 화소에 대해 보간 작업을 독립적으로 실행하게 된다. 그리고 입출력 간의 데이터 동기를 맞추기 위해 블록이 이동할 때마다 모든 쓰레드의 작업이 끝났는지 확인을 하는 동기화 작업도 해야 한다.

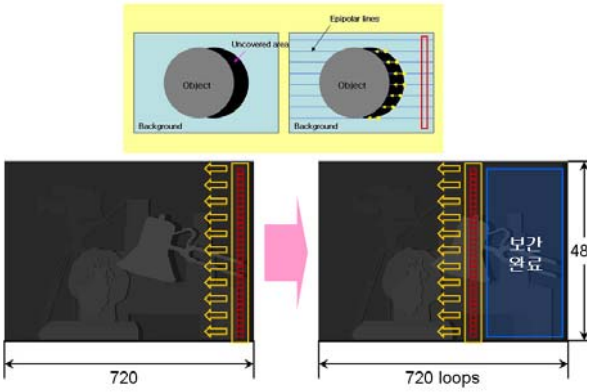


그림 11. 방향성 보간의 병렬처리
Fig 11. Parallel processing of directional interpolation

이러한 방법은 영상의 가로크기만큼 순환을 하는 방식이기 때문에 비효율적으로 보이지만 실제로 각 쓰레드는 새롭게 나타나는 영역에 대해서만 처리를 하기 때문에 실제 소요시간을 적다. 또한, G80 그래픽 카드의 경우 한 블록당 쓰레드의 최대 개수가 512개이기 때문에 영상의 세로크기가 512보다 클 경우 문제가 생길 수 있다. 하지만 이 경우에는 쓰레드가 처리하는 점의 개수를 하나 이상으로 늘리면 간단히 해결된다.

2.3 새롭게 나타나는 영역의 보간

깊이 정보 보간을 완료하면 이제 추정된 깊이 정보를 이용하여 가상 영상의 새롭게 나타나는 영역을 보간한다. 이 방법은 CPU에서의 처리와 크게 다르지 않으며, 병렬처리 구조는 앞서 설명한 전방 사상의 경우와 동일하다.

전방 사상으로 생긴 가상 영상의 새롭게 나타나는 영역에 대해서 각 쓰레드가 앞서 처리한 보간된 깊이 지도 정보

를 이용하여 참조 영상으로부터 후방 사상을 통해 필요한 색상 정보를 가지고 온다. 이때, 후방 사상에 쓰이는 깊이 지도 정보가 신뢰할 수 있는지에 대해 유효성 검사를 해야 한다. 앞서 보간된 깊이 지도 정보를 이용하여 보간할 가상 시점 영상의 화소를 참조 영상의 위치로 전방 사상을 해본다. 이 점이 기준 영상의 깊이 지도를 참조 영상의 위치로 전방 사상하였을 때 생기는 새롭게 나타나는 영역 안에 사상이 되면 유효한 것으로 판단을 하고 해당 점과 동일한 위치의 참조 영상의 색상 정보를 보간하는 데에 이용을 한다. CUDA에서의 구현은 전방 사상의 경우처럼 영상의 모든 화소를 각 쓰레드가 일대일로 처리할 수 있도록 그리드와 블록을 2차원으로 나누고, 새롭게 나타나는 영역에 대해서만 후방 사상을 하도록 하면 된다.

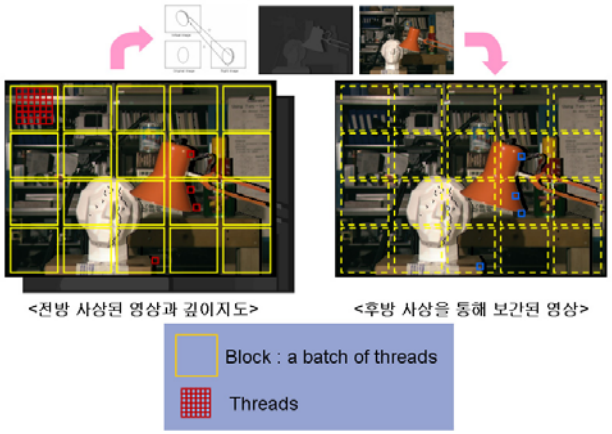


그림 12. 후방 사상의 병렬 처리
Fig 12. Parallel processing of backward mapping

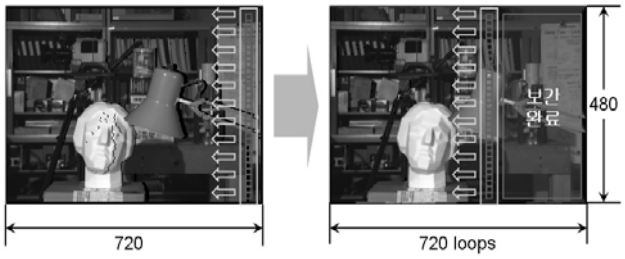


그림 13. 방향성 색상 보간
Fig 13. Directional color interpolation

후방 사상 과정을 거치면 대부분의 영역이 보간이 되지만, 유효성 검사에서 실패한 부분에 대해서는 여전히 정보가 없다. 그 부분에 대해서는 앞서 제안한 깊이 보간 때의 경우처럼 색상 영상에 대해서 방향성 보간법을 이용하여 최종 결과 영상을 만들어 낸다.

2.4 고속화 효율을 높이는 메모리 활용

GPU 디바이스에는 여러 종류의 메모리가 있다. 각 메모리는 크기, 접근권한, 접근속도, 생명주기 등이 다르기 때문에 용도에 맞게 잘 활용한다면 고속화 효율을 더욱 높일 수 있다. 고속화 효율을 높이기 위한 방법의 첫 번째로 우선 텍스쳐 캐시를 활용하였다. 중간 시점 생성에 참조 영상으로 쓰이는 좌우 두 기준 영상은 수정할 일이 없이 참조하는 데에만 쓰이기 때문에 읽기 전용의 텍스쳐 메모리를 활용하면 좋다. 텍스쳐 메모리는 off-chip메모리이기 때문에 접근 속도는 전역 메모리와 다르지 않지만, 두 기준 영상에서 필요한 정보를 참조할 때 시간적, 공간적 지역성을 보이는 경우 전역 메모리와 달리 on-chip메모리인 텍스쳐 캐시가 있기 때문에 메모리 접근 속도를 크게 줄일 수 있다.

두 번째 방안으로 상수 메모리(constant memory)를 활용할 수 있다. 상수 메모리의 구조와 역할은 텍스쳐 메모리와 유사하다. 텍스쳐 메모리처럼 상수 캐시(constant cache)가 있어서 시간적, 공간적 지역성을 보이는 경우 메모리 접근 속도를 크게 줄일 수 있다. 상수 메모리를 사용하는 데에 있어 유의해야 할 점은 텍스쳐 메모리와 달리 메모리 공간이 크지 않다는 것이다. 실험에 쓰인 G80 그래픽 카드의 경우 상수 메모리의 크기는 65536bytes이다. 영상 합성에서 각 시점마다 전방 사상과 후방 사상 연산을 할 때에 사상 계수(mapping coefficient)와 카메라 정보가 필요하다. 이때 쓰이는 사상 계수와 카메라 정보가 차지하는 메모리는 크지 않지만 자주 쓰이므로 상수 메모리에 할당을 하고 쓰레드에서 연산을 할 때 참조하면 메모리 접근 속도를 크게 줄일 수 있다.

고속화 효율을 보다 높이기 위한 방법으로 공유 메모리(shared memory)를 사용하는 방법도 있다. 공유 메모리는 전역 메모리에 담긴 데이터에 접근하는 시간을 줄이기 위해 쓰이는 캐시 개념의 on-chip메모리로 실제 주로 활용되

는 메모리이다. 이때 공유라는 뜻은 같은 그룹에 속한 쓰레드들만 공유가 가능하다는 뜻이다. 그렇기 때문에 쓰레드 그룹이 실행되기 전에 먼저 그 쓰레드 그룹에서 필요하고 자주 쓰이는 데이터들을 전역 메모리에서 공유 메모리에 옮겨놓게 된다. 이때도 마찬가지로 전역메모리에 접근을 하기에 데이터 복사 시간이 소요된다. 만약 그 한 번의 데이터 복사로 인해 전역 메모리의 접근 횟수가 많이 줄어든다면, 즉 공유 메모리에 복사된 데이터가 자주 쓰인다면 분명 고속화 효율이 크게 높아질 수 있다. 본 논문에서 실험을 할 때에 각 단계에서의 효율을 높이고자 공유 메모리를 사용하였으나, 실제로 계산에 쓰이는 데이터는 영상의 좌표, 색상 값, 깊이 지도 정보, 사상 계수 정도로 쓰레드 그룹 단위로 자주 쓰이는 데이터가 없었기 때문에 공유 메모리 사용으로 고속화 효율 개선을 기대하기에는 무리가 있었다.

IV. 실험 결과 및 분석

1. 실험 환경

영상 합성 실험을 위해 그림 14의 두 종류의 좌우 영상을 이용하였다. 두 영상은 영상 교정이 되어있으며, 크기는 가로 720, 세로 480 크기이다. 두 종류의 영상의 위쪽의 알로에 영상은 스테레오 비전 평가에 사용되는 실험 영상과 깊이 지도이며[8], 아래쪽 영상은 보정된 다시점 카메라 시스템을 이용하여 획득된 영상 중에서 3번째와 6번째 카메라의 영상과 계산을 통해 얻어진 깊이 지도이다[9]. 프로그램을 개발하고 실행한 환경은 표 1과 같다.

표 1. 실험 환경
Table 1. Experimental environment

CPU	**Intel Core 2 Duo Conroe 6300 @ 1.86 GHz**
RAM	2048 MB
VGA	NVIDIA Geforce 8800 GTS 512MB
Software	Microsoft Visual Studio 2005 CUDA Toolkit v1.1 CUDA SDK v1.1

2. 실험 결과 및 분석

기본적으로 3차원 디스플레이에 넘겨줄 영상의 개수를 총 9개로 정하는 것을 목표로 7개의 중간 시점 영상의 동시 생성을 목표로 실험을 진행하였다. 실험 영상의 좌측 기준 영상의 위치를 0, 우측 참조 영상을 1로 놓고, 좌측 기준 영상에서 우측의 참조 영상 방향으로 0.125씩 이동하면서 같은 간격으로 총 7개의 중간 시점 영상을 생성하였다.

그림 15는 영상 합성 과정의 첫 번째 과정인 전방 사상이 이루어진 중간 시점 중에 4번째 위치의 영상이다.

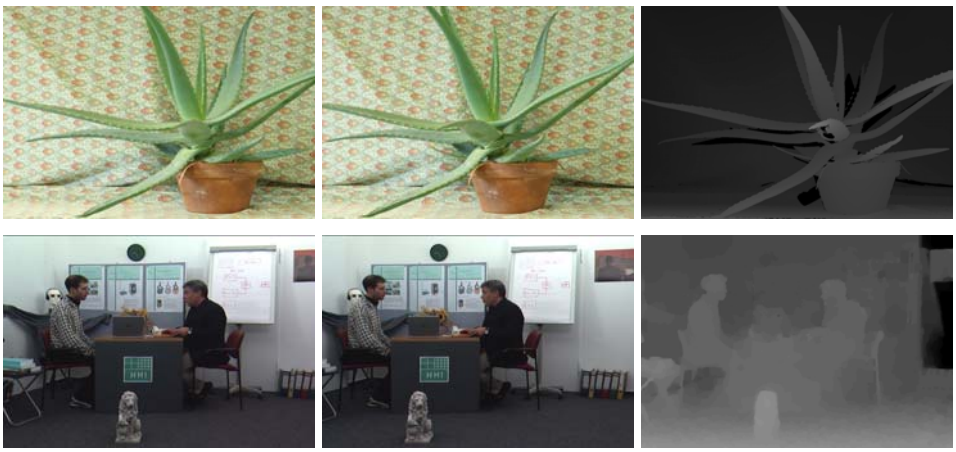
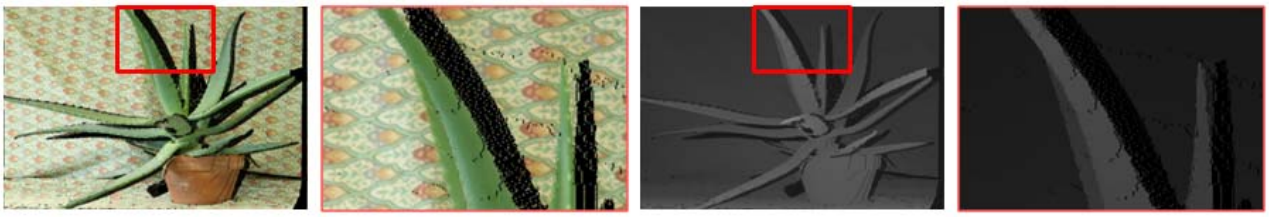


그림 14. 실험 영상
Fig 14. Stereo images and depth maps for experiments

그림 15의 확대된 영상을 보면 새롭게 나타나는 영역(uncovered area)에 없어야 할 점들이 보인다. 이러한 점들은 기준 영상의 깊이 지도의 물체의 경계부분에서의 시차값(disparity value)이 불연속적이어서 생기는 현상이다. 이러한 점들은 전방 사상의 다음 과정인 방향성 깊이 지도 보간을 할 때에 영향을 미치기 때문에 없애는 작업을 해주는 것이 좋다. 이러한 점들은 아무 정보가 없는 영역에 이산적으로 분포를 하기 때문에 영상의 각 화소에 대해 한 화소 간격으로 상하좌우에 위치한 화소가 모두 정보가 없는 경우 없애는 방법으로 정제 작업을 하였다. 상하좌우만 했을 경우에는 대각선 방향으로 분포하는 경우가 남아서 연이어 대각선 네 방향으로 방향을 바꿔서 정제 작업을 하였으며 그 결과 영상의 다른 부분에 영향 없이 효과적으로 점을 없앨 수 있었다. 그림 16은 정제 작업 후의 부분 영상을 보여준다.

그림 17의 좌측 영상은 정제 작업을 거친 후에 방향성 깊이 보간을 거친 영상이며, 우측 영상은 보간된 좌측 깊이 지도를 이용하여 후방 사상을 한 결과 영상이다. 그림을 통해 새롭게 나타나는 영역이 효과적으로 메워졌음을 알 수 있다. 후방 사상 과정에서 유효성 검사를 통과하지 못한, 즉 방향성 깊이 지도 보간 방법으로 보간이 된 깊이 지도 정보를 신뢰할 수 없는 경우에는 참조 영상에서 색상 정보를 가지고 오지 않기 때문에 그림 17의 우측 영상처럼 여전히 검은 부분이 남아있는 것을 볼 수 있다. 이러



(a) (b)

그림 15. 전방 사상 결과와 부분 확대 영상. (a) 전방 사상된 색상 영상, (b) 전방 사상된 깊이 영상
Fig 15. Result images after forward mapping. (a) Forward-mapped color image, (b) Forward-mapped depth map

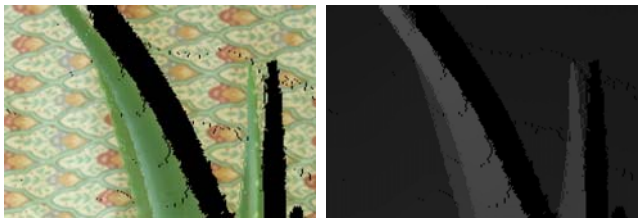


그림 16. 새롭게 나타나는 영역 정제 작업 후의 부분 영상
Fig 16. Uncovered area after refinement

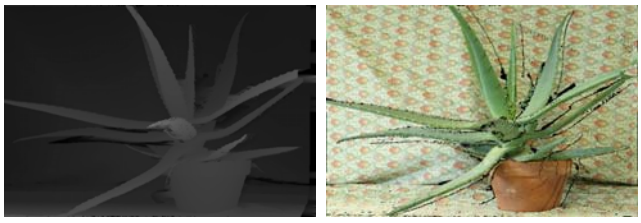


그림 17. 방향성 깊이 보간 결과(좌), 후방 사상 결과(우)
Fig 17. Result image after directional interpolation(Left), Result image after backward mapping(Right)

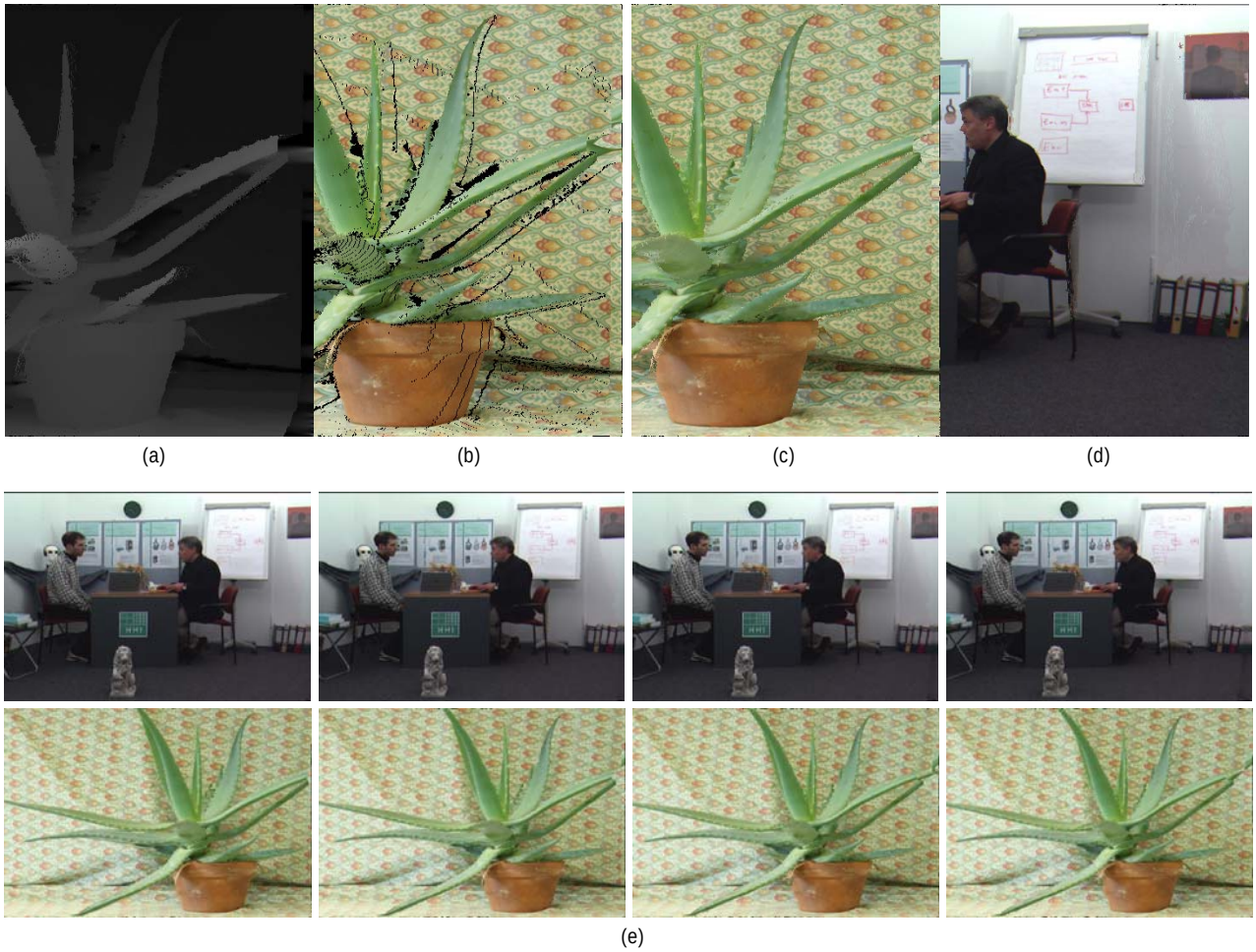


그림 18. 과정별 부분 영상과 중간 시점 영상 합성 결과. (a) 방향성 보간 결과 부분 깊이 영상, (b) 후방 사상 결과 부분 색상 영상, (c) 최종 결과의 부분 확대 영상, (d) 최종 결과의 부분 확대 영상, (e) 4시점 위치에서의 중간 시점 영상

Fig 18. Step wise enlarge images and result images of intermediate view synthesis. (a) Enlarged image after directional interpolation, (b) Enlarged image after backward mapping, (c) Enlarged final result image, (d) Enlarged final result image, (e) Final result images

한 부분을 처리하기 위해 최종적으로 색상 보간 과정을 거친다.

그림 18은 7개의 중간 시점 영상 중에서 과정별 부분 영상과 각 시점 위치에 따른 4개의 영상을 보여준다.

물체가 왼쪽 방향으로 이동함에 따라 물체의 오른편에 새롭게 나타나는 영역이 대체적으로 약간 흐릿하게 보인다. 이 부분이 깊이 지도 보간과 후방 사상을 통해 채워진 영역이다. 결과 영상의 우측 가장자리에서 어긋난 부분이 다른 부분보다 눈에 띄는 것은 보간을 할 때에 참고할 정보가 없기 때문이다. 이것은 기준 영상은 두 시점이지만, 깊이 지도 영상은 한 시점만을 이용하기 때문에 참조할 정보 부재로 인해 어쩔 수 없이 생기는 영역이다. 이러한 부분은 시스템적으로 보완을 하면 어렵지 않게 해결될 수 있다. 본 논문에서는 가능성 실험을 위해 양안 시점과 기준 시점의 깊이 지도만을 이용하였지만, 양안이 아닌 가운데 기준 시점을 중심으로 하여 양쪽으로 참조 시점이 존재하는 3시점 카메라 시스템의 경우, 중간의 기준 시점을 중심으로 좌우로 근접한 위치에서 가상 영상을 생성한다면 가장자리의

어긋난 부분이 크게 줄어들 수 있다.

3. 영상 합성 소요시간 결과 분석

영상 합성의 전체 과정은 그림 19와 같이 이루어진다. 먼저 호스트에서는 영상 입력과 출력에 필요한 공간을 디바이스에 할당하고, 영상 합성에 필요한 정보를 준비하거나 디바이스에 할당된 메모리로 복사하는 역할은 호스트의 CPU가 처리한다. 모든 정보를 디바이스로 넘겨주면 디바이스에서는 각 중간 시점 영상을 순차적으로 그리고 영상 합성 과정의 순서에 따라 순차적으로 처리를 한다. 마지막으로 디바이스에서 최종 결과가 생성이 되면 다시 호스트로 복사를 하거나 디스플레이 장치로 출력을 해주면 된다.

그림 19. 영상 합성 개략도
Fig 19. Overview of parallel view synthesis

표 2는 가로 720, 세로 480 크기의 실험 영상을 CPU에서 영상 합성을 했을 때와 CUDA를 이용하여 GPU에서 연산했을 때의 소요 시간 비교를 나타낸다. 약 9배의 개선이 이루어졌음을 알 수 있다.

표 3은 영상 합성 과정별 소요 시간과 고속화 효율 개선 시간 비교를 나타낸다. 소요 시간을 보면 개선 전과 개선 후로 나뉘어져 있다. 개선된 결과는 앞 절에서 언급했던 고속화 효율을 높이기 위해 다른 메모리를 활용한 경우이다. 사상 계수와 카메라 정보를 전역 메모리가 아닌 상수 메모리에 선언하여 참조한 경우인데, 각 시점의 전방 사상과 후방 사상 연산에 필요한 사상 계수와 카메라 정보는 같기 때문에 시간적 지역성으로 인해 상수 메모리에 접근하지 않고 상수 캐시를 이용하므로 메모리 접근 시간이 눈에 띄게 줄어들었다.

표 2. CPU와 GPU의 소요 시간 비교
Table 2. Computation time (CPU vs. GPU)

CPU(ms)	GPU(ms)	비율
2203	237.857	1:9.262

표 3. 영상 합성 과정별 소요 시간과 고속화 효율 개선 시간 비교
Table 3. Step wise computation time and its improvement by memory management

영상 합성 과정	소요 시간 (ms)		비율 (%)	
	개선 전	개선 후	개선 전	개선 후
데이터 복사(H to D)	37.664	33.696	15.83%	26.38%
전방 사상	119.312	25.175	50.16%	19.71%
정제 과정	7.424	7.172	3.12%	5.62%
방향성 깊이 지도 보간	13.519	13.737	5.68%	10.76%
후방 사상	14.332	4.548	6.03%	3.56%
방향성 색상 영상 보간	33.01	31.542	13.88%	24.70%
데이터 복사(D to H)	12.569	11.842	5.30%	9.27%
데이터 복사 소요 시간	50.260	45.538	21.13%	35.66%
프로세스 소요 시간	187.597	82.174	78.87%	64.34%
전체 소요 시간	237.857	127.712	100%	100%

영상 합성 과정별 소요 시간을 살펴보면 대체적으로 전역 메모리에 접근하는 횟수가 많은 과정일수록 비율이 높음을 알 수 있다. 전방 사상의 경우 중간 시점 영상의 모든 화소에 대하여 사상을 하게 되므로 전역 메모리로의 접근하는 횟수가 가장 많기 때문에 비율이 가장 높다. 새롭게 나타나는 영역의 정제와 후방 사상의 경우 특정 조건에 맞는 화소에 대해서만 처리를 하기 때문에 전역 메모리로의 접근 횟수가 적다. 방향성 깊이 보간의 경우 전역 메모리에 접근하는 횟수는 후방 사상과 비슷하지만 비율이 3배 이상 차이가 난다. 그 이유는 후방 사상은 여러 블록 안의 쓰레드들이 화소와 일대일 대응하여 처리를 하지만, 방향성 깊이 보간은 한 블록 안의 영상의 세로 크기와 같은 개수의 쓰레드들이 영상의 가로 크기만큼 순환을 하면서 처리를 하기 때문이다.

영상 합성이 최후에 이루어지는 방향성 색상 보간이 총 프로세스 소요 시간 중에서 가장 많은 비율을 차지한다. 방향성 깊이 보간과 같은 방법으로 이루어지고, 깊이 보간보다 훨씬 적은 부분을 처리하는데 이와 같이 비율이 높은 이유는 한 채널의 깊이 지도 보간과 달리 세 채널의 색상 영역을 보간하기 때문에 깊이 지도 보간에 비해 3배 많은 메모리에 접근을 하기 때문이다.

상수 메모리를 활용함으로써 전역 메모리의 접근 횟수를 줄인 전방 사상과 후방 사상의 개선 결과와 방향성 색상 보간의 비율에서 알 수 있듯이 off-chip 메모리인 전역 메모리에 접근하는 횟수를 줄이는 것이 고속화 효율을 높이는 데에 큰 관건이라 할 수 있다.

또한, 호스트에서 디바이스로, 디바이스에서 호스트로 데이터를 복사하는 데에만 소요되는 시간이 전체 영상 합성 소요 시간의 35.66%를 차지하는 것을 알 수 있다. 이것은 호스트의 메모리와 디바이스의 메모리 간의 데이터 이동 시간이 디바이스 내의 실행 시간에 비해 상대적으로 많이 걸리기 때문인데 이것은 전체 실행 시간의 병목 현상을 야기할 수 있다. 이는 하드웨어의 한계이기 때문에 이동하는 데이터의 양을 줄이는 방법 외에는 소프트웨어 수준에서 개선할 방법은 없다.

표 4는 가로 720, 세로 480 크기의 실험 영상을 CPU에서 영상 합성을 했을 때와 CUDA를 이용하여 GPU에서 연산했을 때의 시점 개수에 따른 소요 시간 비교를 나타낸다.

표 4. 시점의 개수에 따른 소요 시간
Table 4. Computation time w.r.t. the number of viewpoints

총 시점	CPU(ms)	GPU(ms)	비율
9	2203	127.712	1:17.250
16	4375	250.116	1:17.492
25	7234	399.856	1:18.092

9개의 시점을 동시 생성하는 영상 합성을 CPU에서 연산을 하면 초당 0.5프레임 이하의 속도밖에 나오지 않지만 GPU에서 했을 때는 초당 8프레임의 속도로 약 17배의 속도 개선이 이루어졌다.

시점의 수가 늘어날수록 소요시간이 늘어나는 것은 GPU에서도 마찬가지이지만, CUDA에서는 그래픽 카드, 즉 디바이스의 개수를 늘려서 디바이스마다 각기 다른 시점의 생성을 맡기는 식의 병렬처리가 가능하다. 여러 디바이스를 쓰는 시스템을 구성한다면 각 시점 생성의 병렬처리를 할 수 있기 때문에 더욱 개선된 결과를 얻을 수 있을 것으로 기대한다.

V. 결 론

본 논문에서는 깊이 지도 기반의 영상 합성 기법을 3차원 디스플레이로의 적용을 하기 위해 고속화 작업을 진행하였다. SD급인 가로 720, 세로 480크기의 실험 영상으로 총 9개의 시점 영상을 동시에 실시간으로 생성하는 것을 목표로 실험을 하였다. 고속화 작업을 위해 GPU기반의 병렬처리를 하였으며 이를 CUDA™로 구현하였다. 또한, GPU프로그래밍에 있어서 고려사항인 호스트와 디바이스의 대역폭 문제로 인한 병목 현상을 줄이기 위해 영상 합성의 모든 과정을 GPU가 병렬처리를 할 수 있도록 함으로써 고속화 효율을 높였다.

CUDA의 메모리의 활용도에 따라 다른 결과를 보임으로써 메모리 구조의 효과적인 활용이 고속화 효율과 직결된다는 것을 알 수 있었다. 마지막으로 CUDA를 이용한 GPU에서 처리한 결과가 CPU에서의 소요 시간과 비교하여 약 17배의 속도 개선이 이루어졌으며, 고속화 작업으로 3차원 디스플레이로의 적용 가능성을 충분히 가늠해볼 수 있었다.

본 논문에서는 SD급의 영상으로 총 9개의 시점 생성을 목표로 실험을 하였지만, HD급 화질의 디스플레이가 출시되고 있는 요즘 시대 변화에 발맞추어 HD급 혹은 그 크기 이상의 영상으로의 적용에 대한 연구도 지속적으로 이루어져야 할 것이다. 또한, 본 논문에서는 기준 시점과 참조 시점 사이에 위치한 가상 시점들의 간격을 균일하게 두었지만, 향후 인간의 시각 특성을 기반으로 하여 보다 입체감이 느껴지는 3차원 영상이 생성되도록 시점의 위치를 계산, 조절하는 연구도 이루어져야할 것이다.

참 고 문 헌

[1] O. Schreer, P. Kauff, and T. Sikora, 3D Videocommunication,

John Wiley & Sons, 2005.
[2] C. L. Zitnick et al, "High-quality video view interpolation using a layered representation," ACM Transactions on Graphics, vol. 23, no. 3, pp. 600-608, 2004.
[3] J. Park, G. Um, C. Ahn, and C.-T. Ahn, "Virtual control of optical axis of the 3DTV camera for reducing visual fatigue in stereoscopic 3DTV," ETRI Journal, pp. 597-604, 2004.
[4] A.-R. Mansouri and J. Konrad, "Bayesian winner-take-all reconstruction of intermediate views from stereoscopic images," IEEE Trans. Image Processing, vol. 9, pp. 1710-1722, 2000.
[5] N. Cornelis and L. Van Gool, "Fast scale invariant feature detection and matching on programmable graphics hardware," Proc. of Workshop on Visual Computer Vision on GPUs, June, 2008.
[6] Y. Luo and R. Duraiswami, "Canny edge detection on NVIDIA CUDA," Proc. of Workshop on Visual Computer Vision on GPUs, June, 2008.
[7] V. Vineet and P. J. Narayanan, "CUDA cuts: fast graph cuts on the GPU," Proc. of Workshop on Visual Computer Vision on GPUs, June, 2008.
[8] http://vision.middlebury.edu/stereo/data/
[9] http://www.hhi.fraunhofer.de/
[10] http://www.nvidia.com/object/cuda_home.html
[11] http://developer.download.nvidia.com/compute/cuda/1_1/NVIDIA_CUDA_Programming_Guide_1.1.pdf

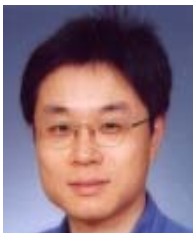
저 자 소 개

신 홍 창

- 2005년 : 세종대학교 컴퓨터공학과 학사 졸업
- 2008년 : 한양대학교 전자통신컴퓨터공학과 석사 졸업
- 주관심분야 : 가상현실, 3차원 영상처리, HCI, 영상 합성, 3차원 디스플레이

박 한 훈

- 2000년 : 한양대학교 전자통신전파공학과 학사 졸업
- 2002년 : 한양대학교 전자통신전파공학과 석사 졸업
- 2007년 : 한양대학교 전자통신전파공학과 박사 졸업
- 2008년 현재 : NHK 연구소 연구원
- 주관심분야 : 가상현실, 증강현실, HCI, 3차원 영상처리



박 종 일

- 1987년 : 서울대학교 전자공학과 학사 졸업
- 1989년 : 서울대학교 전자공학과 석사 졸업
- 1995년 : 서울대학교 전자공학과 박사 졸업
- 1992년 ~ 1994년 : 일본 NHK방송기술연구소 객원연구원
- 1995년 ~ 1996년 : 한국방송개발원 선임연구원
- 1996년 ~ 1999년 : 일본 ATR지능영상통신연구소 연구원
- 1999년 ~ 현재 : 한양대학교 전자통신컴퓨터공학과 교수
- 주관심분야 : 가상현실, 증강현실, HCI, 3차원 영상처리, 컴퓨터그래픽스/비전