증강현실 시스템 구현을 위한 단일 프레임 에서의 고속 카메라 위치추정

이범종[1], 박종승[2], 성미영[3], 노성렬[4]
인천대학교 컴퓨터공학과[1 2 3], 시리우스[4]
{leeyanga[1], jong[2], mysung[3]}@incheon.ac.kr, noh@sirius.co.kr[4]

Fast Camera Pose Estimation from a Single Frame for Augmented Reality Applications

Lee Bum-Jong[1], Jong-Seung Park[2], Mee Young Sung[3], Sung-Ryul Noh[4]
University of Incheon, Department of Computer Science and Engineering, [1 2 3], SIRIUS Co. Ltd.[4]

요 약

본 논문에서는 3D 복원과 카메라 측정과정 없이 정확하게 카메라 자세를 계산하고 가상객체를 비디오에 합성하기 위한 단일 프레임 기반의 고속 계산 기법을 제안한다. 객체의 로컬 좌표와 단일 이미지에서의 대응되는 이미지 좌표로부터 카메라 자세를 계산한다. 정사영 투영모델에서의 분해기법에 기반한 구조 계산 방법으로 카메라 자세의 고속 추정이 가능하다. 정사영 투영모델에 기반하기 때문에 참조점의 설정에 따라 정확도가 달라진다. 객체에 따라 참조점을 설정하여 정확한 카메라 자세를 계산하는 방법을 제안한다. 카메라 자세 및 물체의 형태는 단일 프레임 기반으로 수행되며 카메라 자세 추정 결과가 즉시 비디오 합성에 사용될 수 있도록 하였다. 제안하는 기법의 유효성 입증을 위해 실사 비디오에 기반한 증강현실시스템을 구현하고 카메라 자세 계산과 비디오 합성의 전체 과정을 단일 프레임에 기반하여 실험을 수행하고 제안 기법의 실용성을 보였다.

Keywords : Camera Pose from Orthography, Video Mixing, Non-coplanar/Coplanar Structure Recovery

1. 서론

본 논문에서는 단일 프레임으로부터 고속으로 카메라 자세를 계산하여 가상객체를 비디오에 합성하기 위한 기법을 제안한다. 증강현실 시스템은 실시간으로 처리할 수 있는 속도가 중요하다. 본 논문에서는 속도를 향상 시키기 위해 정사영 투영모델에서의 분해기법에 기반한 카메라 자세 계산 기법을 사용한다. 정사영 투영모델에서의 분해기법은 객체의 로컬 좌표와 단일 프레임에서의 대응되는 이미지 좌표로부터 카메라 자세를 계산한다.

Daniel 과 Larry 의 POSIT 알고리즘[11]은 동일 평면상이 아닌 Non-Coplanar 이어야 하고 네 개 이상의 점이 주어져야 한다. POSIT 알고리즘은 객체의 로컬 좌표와 대응되는 이미지 좌표를 통해 객체의 회전행렬과 이동벡터를 계산한다.

가상객체를 비디오에 합성하기 위한 방법은 크게 이미지 기반의 합성방법과 3D 기하정보를 이용한 합성방법의 두 가지로 분류할 수 있다. 이들 중 몇 가지 방법을 소개한다.

이미지 기반의 합성방법에는 비디오 프레임내의 특징점과 합성할 가상객체의 텍스처 이미지와의 이미지 변환 관계를 구하여 텍스처를 비디오 프레임에 합성하는 방법[1]이 있다. 이미지를 작은 패치들로 나눈 후 누빔(quilting)으로써 새로운 이미지를 만들어 내는 방법[2]도 있다. 같은 장면에 대한 두 장의 이미지를 이용하여 보간법으로 두 이미지 사이의 새로운 뷰를 만들어내는 방법[3], 3D 모델을 사용하지 않고 두 이미지로부터 동적 장면을 생성하는 방법[4], 투영 기하학 기반의 여러 대의 카메라를 사용하여 보간법으로 새로운 뷰를 생성하는 방법[5], 투영 기하학의 기본 원칙을 사용하여 3D 투영 카메라와 장면 변형을 정확하게 다

¶ 본 연구는 산업자원부 (MOCIE)의 지원으로 인천정보산업진흥원 (IIT)을 통하여 수행되었음.

뤄 새로운 뷰를 생성해 내는 방법[6]도 있다.

3D 기하정보를 이용한 합성 방법에는 여러 장의 이미지로 깊이 좌표를 계산하여 위치 관계를 고려한 합성방법[7]이 있다. 3D 객체의 기하정보를 바탕으로 3D 객체의 각 면의 이미지를 각 프레임의 해당 페이스 영역에 매핑하는 방법[8]도 제안되었다. 3D 메쉬를 비디오 프레임 내의 실객체로 표현하여 비디오 내의 실객체를 만드는 방법[9]도 제안되었다. 여러 장의 측정/비측정 이미지로부터 각 장면의 3D 정보를 모으고 복원하여 합성하는 방법[10]도 있다.

2. 정사영 투영모델에서의 카메라의 위치계산

POS(Pose from Orthography)는 객체의 회전행렬과 이동벡터를 구하는 알고리즘이다. 회전행렬 **R**은 객체 좌표 시스템 ($\boldsymbol{V}_0u$, $\boldsymbol{V}_0v$, $\boldsymbol{V}_0w$)에서 카메라 좌표 시스템의 단위 벡터 **i**, **j**, **k**의 좌표가 행 벡터로 이루어 진다. 회전 행렬은 아래와 같이 쓸 수 있다.

$$\mathbf{R} = \begin{bmatrix} i_u & i_v & i_w \\ j_u & j_v & j_w \\ k_u & k_v & k_w \end{bmatrix}$$

회전행렬을 계산하기 위해 객체 좌표 시스템에서의 **i**와 **j**를 계산하고 **k**는 **i**와 **j**의 외적으로 구한다. 이동벡터는 투사의 중심 O와 참조 점 $\mathbf{V}_0$사이의 벡터 OV_0 이다. 이동벡터 **T**는 Z_0/f*Ov_0와 같이 나타낼 수 있으므로 객체의 이동을 계산하기 위해서는 단지 Z_0만을 계산하면 된다. 즉, 카메라 차세를 계산하기 위해서는 **i**와 **j**와 $\mathbf{Z}_0$만을 계산하면 된다. 그림 1 에서 정사영 투영 모델의 기하도를 보여주고 있다.

Scaled Orthographic projection(SOP)는 객체의 참조 점 V_0의 깊이 Z_0가 객체의 다른 모든 점들의 깊이와 같다는 가정이 필요하다. SOP의 스케일링 요소 s는 f/Z_0로 나타낼 수 있다. 참조점 $\mathbf{V}_0$의 이미지 좌표는 SOP와 원근 투영에서 $\mathbf{v}_0$로 같다. SOP의 이미지 좌표 s_i는 식 (1)과 같다.

$$\begin{aligned} x_i^{'} &= fX_0/Z_0 + f(X_i - X_0)/Z_0 = x_0 + s(X_i - X_0) \\ y_i^{'} &= fY_0/Z_0 + f(Y_i - Y_0)/Z_0 = y_0 + s(Y_i - Y_0) \end{aligned} \quad (1)$$

객체 좌표 시스템에서 벡터 V_0V_i의 좌표와 이미지 점의 v_0, v_i의 좌표와 **i**, **j**, $\mathbf{Z}_0$와의 관계를 갖는 식은 식(2), 식 (3)과 같다.

$$\mathbf{V}_0\mathbf{V}_i.x = x_i(1+\varepsilon_i) - x_0 \quad (2)$$

$$\mathbf{V}_0\mathbf{V}_i.y = y_i(1+\varepsilon_i) - y_0 \quad (3)$$

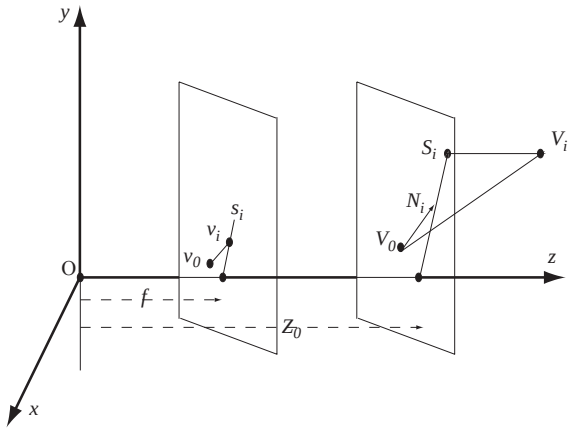


그림 1. 정사영 투영모델의 기하도

ε_i는 식 (4)로 정의한다.

$$\varepsilon_i = \frac{1}{Z_0}\mathbf{V}_0\mathbf{V}_i \cdot \mathbf{k} \quad (4)$$

벡터 $\mathbf{V}_0\mathbf{V}_i$는 벡터 $\mathbf{V}_0\mathbf{S}_i$와 $\mathbf{S}_i\mathbf{V}_i$의 합이다. 벡터 $\mathbf{V}_0\mathbf{S}_i$와 이미지 벡터 $\mathbf{v}_0\mathbf{s}_i$는 Z_0/f의 비율로 비례한다. 이를 식으로 나타내면 아래와 같다

$$\mathbf{V}_0\mathbf{V}_i = \frac{Z_0}{f}\mathbf{v}_0\mathbf{s}_i + \mathbf{S}_i\mathbf{V}_i$$

위의 벡터 $\mathbf{V}_0\mathbf{V}_i$에 카메라 좌표 시스템의 단위벡터 **i**를 내적한다. $\mathbf{S}_i\mathbf{V}_i$·**i**는 x축과 수직이므로 0 이 되고 $\mathbf{v}_0\mathbf{s}_i$·**i**는 벡터 $\mathbf{v}_0\mathbf{s}_i$의 x좌표인 x_i'-x_0가 된다. 이를 가지고 아래의 식 (5), 식 (6)을 얻을 수 있다.

$$\mathbf{V}_0\mathbf{V}_x.x = x_i' - x_0 \quad (5)$$

$$\mathbf{V}_0\mathbf{V}_i.y = y_i' - y_0 \quad (6)$$

위의 식 (5)와 식 (6)을 식 (2)와 식 (3)과 비교하면 s_i의 좌표는 $x_i'=x_i(1+\varepsilon_i)$와 $y_i'=y_i(1+\varepsilon_i)$로 나타낼 수 있다. 또한 $I=f/Z_0$·i, $J=f/Z_0$·j 를 이용하여 식 (2)와 식 (3)을 다시 정리 하면 아래의 식 (7)과 식 (8)이 된다.

$$\mathbf{V}_0\mathbf{V}_i.x = x_i(1+\varepsilon_i) - x_0 \quad (7)$$

$$\mathbf{V}_0\mathbf{V}_i.y = y_i(1+\varepsilon_i) - y_0 \quad (8)$$

정사영 투영모델에서 카메라의 자세를 계산하기 위해 ε_i가 정확하게 주어지지 않으면 근사값을 구하게 된다. 하지만 POSIT(POS with Iterations)알고리즘은 반복하며 점점 더 정확한 ε_i 값에 가까워지게 된다.

그림 2 는 N개의 점(참조점 $\mathbf{V}_0$를 포함한)을 사용하여 정사영 투영모델에서 카메라의 위치를 계산하는 알고리즘을 나타낸다.

3. 제안된 카메라의 위치 계산

카메라의 자세를 계산하기 위해서는 특징점을 추적하는 선행 작업이 필요하다. 본 논문에서는

특징점 추적을 위해서 Jean-Yves Bouguet 의 Lucas Kanade[13]방법을 사용한다. 본 논문에서 제안하는 카메라 자세 계산 및 가상객체의 비디오 합성의 전체 시스템 흐름도는 그림 3 과 같다.

Step 1 :
 N×3 의 행렬 A 를 만듦
 A의 각 행은 벡터 V_0V_i로 이루어짐 (i=0, … N-1)
 A 의 의사역행렬로 3×N 의 행렬 B 계산
Step 2 :
 $\varepsilon_{i(0)}$=0, (i=0…N-1), n=1;
Step 3 : 반복 시작, i, j, s 계산
 - x', y'계산(식(7), 식 (8))
 - I, J 계산(I=Bx′, J=By′)
 - scale s를 계산(s_1=(I·I)$^{1/2}$, s_2=(J·J)$^{1/2}$, s=(s_1+s_2)/2)
 - 단위 벡터 i, j계산(i=I/s_1, j=J/s_2)
Step 4 : k, Z_0 계산
 - 단위 벡터 k 계산(i 와 j 의 외적)
 - 이동벡터 Z_0의 z좌표 계산(Z_0=f/s)
Step 5 : ε_i계산
 - ε_i계산(식 (4))
 - If | $\varepsilon_{i(n)}$ - $\varepsilon_{i(n-1)}$ |>임계치 then n=n+1, Step 3 으로
Step 6 : 이동벡터, 회전행렬 계산
 - 이동벡터 OV_0 계산(Ov_0/s)
 - 회전행렬 계산(행 벡터 i, j, k)

그림 2. N 개의 점을 사용한 POSIT 알고리즘

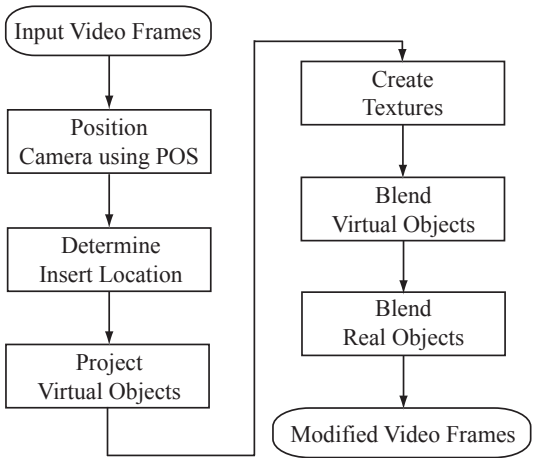


그림 3. 시스템 프레임워크

Daniel 과 Larry 의 알고리즘은 정사영 투영모델에 기반하기 때문에 참조점의 설정에 따라 정확도가 떨어질 수 있다. 본 논문에서는 참조점 설정의 문제를 해결하기 위해 주어진 객체의 점들 중 가장 정확한 카메라 자세를 계산할 수 있는 참조점을 설정하는 방법을 제안한다.

그림 4 는 본 논문에서 제안한 카메라 자세를 계산하는 알고리즘이다. 첫 번째 단계로 초점거리 f 를 설정한다. f 는 카메라 내부 파라메터인 **K** 행렬의 첫 번째 요소가 된다. 본 논문에서 제안한 방법은 초점거리를 정확하게 맞출 필요는 없고 유사한 값으로 설정한다.

Step 1: focal length 세팅
 - f = **K**[0](내부 파라메터의 첫 번째 요소나 그와 비슷한 값)
Step 2: 참조점 R 설정 (식 (9))
 - 물체의 각 점 P_i (i=0,..,N-1)에 대해서R_i←P_i
 - d_i=R_i로부터 모든 P_j (j≠i)와의 거리의 합을 계산
 - d_i가 최소이면 P_i를 R 로 지정
Step 3 : A 행렬 생성 (식 (10))
 - N×3 의 행렬 A 를 생성
 - A의 각 행은 벡터 V_0V_i로 이루어짐
Step 4: B 행렬 계산 (식 (11))
 - A 의 의사역행렬로 3×N 의 행렬 B 계산
Step 5 : i, j, s 계산 (식 (12))
 - x', y'계산(식 (7), 식 (8))
 - I, J 계산(I=Bx′, J=By′)
 - s_1=sqrt(I·I), s_2=sqrt(J·J), s=(s_1+s_2)/2
 - i, j계산(i=I/s_1, j=J/s_2)
Step 6: k, Z_0 계산
 - 단위 벡터 k 계산(i 와 j 의 cross-product)
 - Z_0 = f/s
Step 7: Translation, Rotation 계산
 - **T** = OV_0/s
 - $\mathbf{R} = \begin{bmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \end{bmatrix}$

그림 4. N 개의 점을 사용한 제안된 알고리즘

두 번째 단계는 참조점을 설정하는 단계로서 가장 정확하게 카메라 자세를 계산할 수 있는 참조점을 설정한다. 식 (9)는 참조점을 설정하는 식이다. 식 (9)를 사용하여 참조점을 설정함으로써 가장 정확한 카메라 자세를 계산할 수 있다.

$$\min_{i=0\ldots N-i}\left(\sum_{j=0,i\neq j}^{N-1}\left|\mathbf{V}_i-\mathbf{V}_j\right|\right) \tag{9}$$

세 번째 단계로는 행렬 **A**를 생성하는 단계로 식 (10)에 의해 참조점 $\mathbf{V}_0$와 $\mathbf{V}_i$의 벡터로 이루어진 행렬 **A**를 생성한다.

$$\mathbf{A}=\begin{bmatrix} \mathbf{V}_0-\mathbf{V}_0 \\ \cdots \\ \mathbf{V}_{N-1}-\mathbf{V}_0 \end{bmatrix} \tag{10}$$

네 번째 단계는 **A** 의 의사역행렬인 행렬 **B** 를 구하는 단계로 식 (11)에 의해 행렬 **B** 를 구한다.

$$\mathbf{B}=(\mathbf{A}^T\cdot\mathbf{A})^{-1}\cdot\mathbf{A}^T \tag{11}$$

다섯 번째 단계는 단위 벡터 **i**, **j** 와 스케일 요소 s 를 구하는 단계로 식 (12)에 의해 각각의 값을

구한다.

$$s_1 = ((\mathbf{Bx}') \cdot (\mathbf{Bx}'))^{\frac{1}{2}}, s_2 = ((\mathbf{By}') \cdot (\mathbf{By}'))^{\frac{1}{2}}, \quad s = (s_1 + s_2)/2 \tag{12}$$

여섯 번째 단계로는 단위 벡터 **k**와 $\mathbf{Z}_0$를 구한다. **k**는 **i**와 **j**의 외적으로 구하고 $\mathbf{Z}_0$는 f/s에 의해 구하게 된다.

다음은 마지막 단계로서 위에서 구해진 값들로 객체의 이동벡터와 회전행렬을 구함으로써 카메라의 자세 계산을 완료한다.

4. 가상객체 합성 응용 시스템 구현

가상객체를 합성하기 위해 실객체가 가상객체를 가릴 경우를 고려하여야 한다. 실객체가 가상객체보다 앞에 있는 경우 가상객체를 가릴 실객체에 대한 텍스처를 생성해야 한다. 실객체에 대한 텍스처를 생성하기 위해서는 실객체에 대한 특징점을 추적해야 한다. 추적된 각 프레임의 특징점을 x_0', x_1', x_2', x_3'라 한다. 생성될 텍스처의 좌표는 이미지의 크기로 이루어진 x_0, x_1, x_2, x_3이라 한다.

추적된 실객체의 네 개의 특징점의 이미지 좌표를 알고 있고 이미지의 크기로 이루어진 생성될 텍스처의 네 점에 대한 좌표를 알고 있다면 네 점의 쌍을 이용하여 식 (13)을 식 (14)의 **Am**=0 의 형태로 변형시킨다.

$$\begin{bmatrix} x_i't \\ y_i't \\ t \end{bmatrix} = \mathbf{R} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}, (i = 0,1,2,3) \tag{13}$$

식 (14)의 행렬 **A** 를 SVD(Singular Value Decomposition)를 사용하여 풀어내어 식 (13)의 행렬 **R** 을 구할 수 있다. 구해진 행렬 **R** 로 텍스처의 한 픽셀씩 채워 실객체에 대한 텍스처를 생성한다.

$$\mathbf{A} = \begin{bmatrix} x_0 & y_0 & 1 & 0 & 0 & 0 & -x_0'tx_0 & -x_0'ty_0 & -x_0't \\ 0 & 0 & 0 & x_0 & y_0 & 1 & -y_0'tx_0 & -y_0'ty_0 & -y_0't \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ x_i & y_i & 1 & 0 & 0 & 0 & -x_i'tx_i & -x_i'ty_i & -x_i't \\ 0 & 0 & 0 & x_i & y_i & 1 & -y_i'tx_i & -y_i'ty_i & -y_i't \end{bmatrix} \tag{14}$$

$$m = [m_{11} \quad m_{12} \quad \cdots \quad m_{33}]^T$$

본 논문에서 제안한 방법으로 계산된 회전행렬과 이동벡터를 사용하여 가상객체의 점들을 투사한다. 투사하여 얻어진 점들로 가상객체의 각 면에 대한 텍스처를 합성하여 가상객체를 합성한다. 가상객체의 텍스처를 합성하기 위해 각 면에 대한 최소 사각형 영역을 설정한다. 이 사각형 내의 픽셀들을 식 (15)로 검사하여 텍스처가 합성될 영역을 결정한다.

$$\mathbf{M}_{i,i+1} = \begin{vmatrix} x_i & y_i & 1 \\ x_{i+1} & y_{i+1} & 1 \\ x & y & 1 \end{vmatrix} \tag{15}$$

식 (15)의 행렬 **M**의 첫 번째 행 벡터는 기준점이 되는 점(i=0, 1, 2, 3)이고, 두 번째 행 벡터는 기준점의 시계방향의 오른쪽 점이다. 마지막 행 벡터는 검사할 픽셀의 좌표이다. 이렇게 얻어진 행렬 $\mathbf{M}_{0,1}$, $\mathbf{M}_{1,2}$, $\mathbf{M}_{2,3}$, $\mathbf{M}_{3,4}$, $\mathbf{M}_{4,0}$ 의 네개의 행렬식(determinant)이 모두 양수이면 점 x, y는 텍스처가 합성될 영역의 점이 된다. 텍스처가 합성될 부분의 점에 대해 해당 텍스처의 값을 채워 넣으며 차례로 텍스처를 합성해 나감으로써 가상객체 합성을 완료한다.

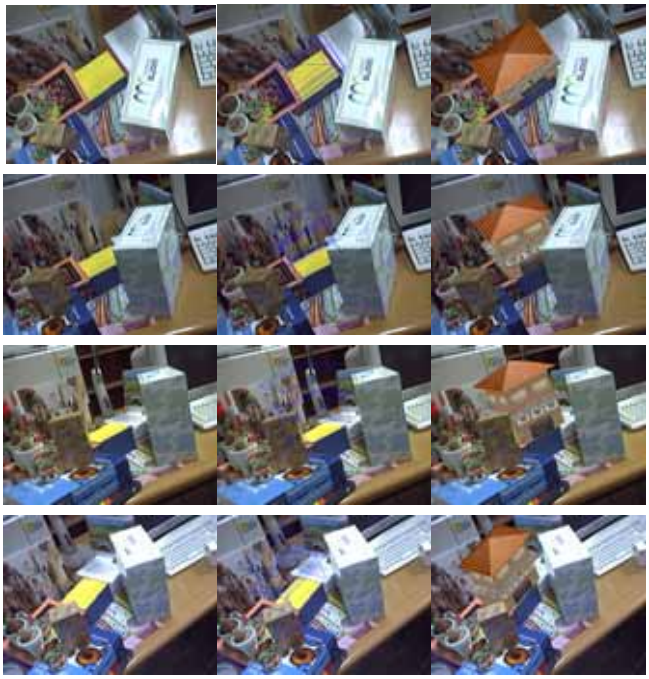
실객체가 가상객체를 가릴 경우에 대하여 실객체의 텍스처를 생성해 놓았다. 생성된 실객체에 대한 텍스처를 가상객체를 합성한 후에 다시 합성하게 된다. 이 실객체의 텍스처 합성은 텍스처를 생성하기 위해 구한 식 (13)의 행렬 **R** 을 사용한다. 행렬 **R**의 역 행렬 $\mathbf{R}^{-1}$을 구한 후, 식 (16)을 사용하여 실객체에 대한 텍스처를 합성한다.

$$\begin{bmatrix} x_it \\ y_it \\ t \end{bmatrix} = \mathbf{R}^{-1} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}, (i = 0,1,2,3) \tag{16}$$

5. 실험결과

본 논문에서 제안한 방법을 사용하여 카메라의 자세를 계산하고 비디오 프레임에 가상객체를 합성하는 실험을 하였다. 카메라의 자세를 계산하기 위해 가상객체가 합성될 부분의 동일 평면상에 존재하지 않는 5 개의 특징점을 추적하였다.

그림 5 는 비디오 프레임에서 100 프레임 단위로 캡쳐한 그림이다. (a)는 원본 프레임에서 캡처한 이미지로 이미지의 가운데 쪽에 있는 노란 상자 위에 가상객체를 합성하게 된다. (b)는 가상객체가 삽입될 모습을 미리 그려본 와이어 프레임이다. 이 와이어 프레임은 가상객체를 합성하기 전에 가상객체가 얼마나 정확하게 떨림이 없이 합성되는지를 미리 예측해 볼 수 있다. (c)는 가상 객체가 합성된 비디오 프레임이다. 노란 상자 위에 가상객체가 정확하게 합성되었으며, 양쪽에 서 있는 두 상자가 가상 객체를 가리는 모습을 볼 수 있다.



(a) (b) (c)

그림 5. 가상객체의 합성:(a)원본 프레임, (b)가상객체의 와이어 프레임, (c)가상객체가 삽입된 프레임

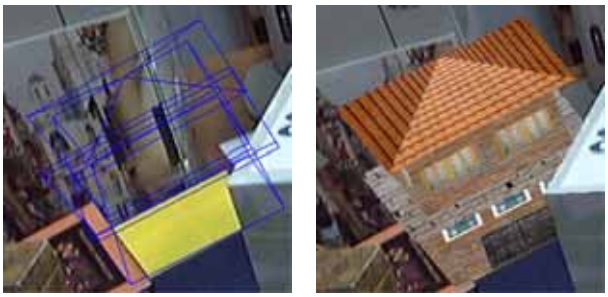


그림 6. 삽입된 가상객체의 모습

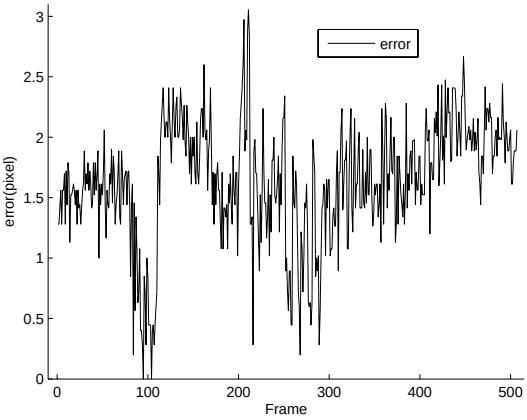


그림 7. 그림 5 의 비디오에서 카메라 위치 추정에 대한 투사 에러

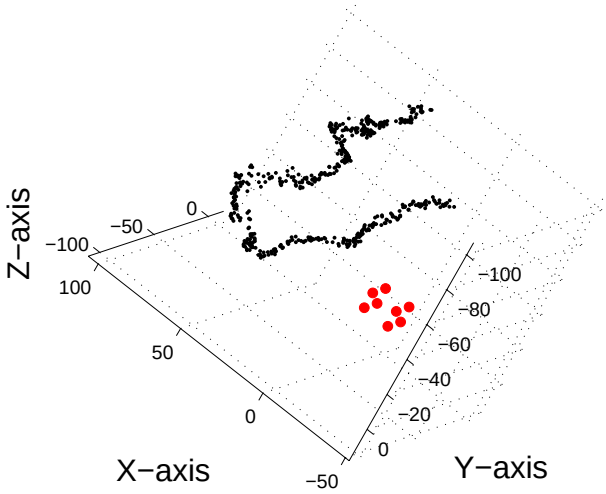


그림 8. 그림 5 의 비디오에서 카메라의 궤적

그림 6 의 왼쪽 이미지는 그림 5 의 (b)의 이미지를 확대한 것이고 그림 6 의 오른쪽 이미지는 그림 5 의 (c)의 가상객체가 합성된 프레임을 확대한 모습이다.

그림 7 은 그림 5 의 비디오에 대한 투사 에러 그래프로서 비디오 프레임 전체에서 평균 1.64 픽셀의 에러를 보인다. 정사영 투영 모델에서의 카메라 자세 계산이 객체의 모든 점의 깊이 좌표는 참조점의 깊이 좌표와 같다고 보기 때문에 생기는 에러이다. 에러를 구하는 식은 식 (17)과 같다. p_i는 가상객체를 합성할 부분의 특징 점이다. p_i'는 p_i를 입력으로 카메라 자세를 계산하여 얻어진 회전행렬과 이동벡터로 재 투사하여 얻어진 점이다.

$$e = \frac{1}{n}\sum_{i=0}^{n-1}\left|p_i - p_i'\right| \qquad (17)$$

그림 8 은 그림 5 의 비디오에서 카메라가 움직인 궤적을 나타낸 그림이다. 직육면체 모양을 이루는 8 개의 빨간 점이 가상객체의 위치를 나타내는 점이고 검정색 점들이 카메라가 움직인 궤적이다.

그림 9 는 6 개의 특징점 중 참조점을 바꿔가며 어떤 점을 참조점으로 설정했을 경우 에러가 가장 적게 나타나는지를 실험한 결과로 가장 적은 에러를 갖는 참조점으로 합성한 결과이다. 그림 10 은 그림 9 의 첫 번째 이미지들을 확대하고 카메라 자세를 계산하기 위해 사용된 특징점을 표시한 이미지 들이다.

그림 10 의 6 개의 점을 차례로 참조점으로 하였을 경우의 에러를 나타내는 표가 표 1 에 나타나 있다. 표 1 에서 점 2 와 점 3 을 참조점으로 했을 경우 가장 적은 에러가 나타난다. 또한 깊이 좌표가 같은 점 0 과 점 1, 점 2 와 점 3, 점 4 와 점 5 는 비슷한 에러를 갖는다. 표 1 에서 에러가 참조

점의 깊이 좌표와 관련이 있다는 것을 볼 수 있고 참조점과 나머지 점들 사이의 깊이 거리가 가장 작을 때 가장 정확한 카메라 자세 계산이 가능한 것을 볼 수 있다.

(a) (b)

그림 9. 6 개의 점을 추적하여 카메라 자세 추정을 통한 가상객체 합성 : (a) 원본 프레임 (b) 합성된 프레임

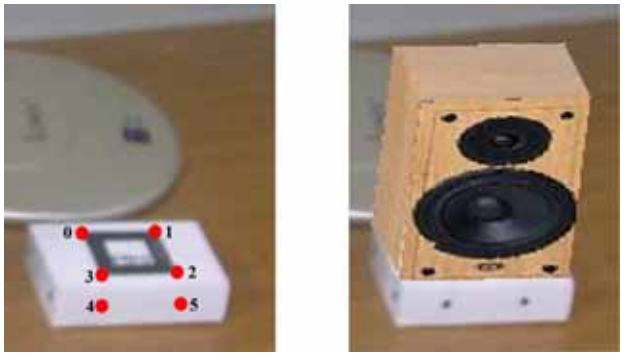


그림 10. : 왼쪽 – 그림 9 에서 카메라 자세 계산에 사용된 점, 오른쪽 – 합성된 결과를 확대한 이미지

그림 11 의 그래프에서 보듯이 깊이 좌표가 중간에 있는 점 2 를 참조점으로 했을 경우 에러가 가장 적게 나타난다. 표 1 과 그림 11 에 의해 참조점은 깊이 좌표가 중심에 가까운 점을 설정 할수록 정확한 카메라 자세 추정이 가능하다는 결론을 내릴 수 있다.

그림 12 는 참조점과의 거리 차이에 따른 에러를 나타내는 그래프이다. 실험은 점의 수를 하나씩 추가할 때마다 참조점에서 점차 멀리 있는 점들을 추가 하여 실험을 하였다. 그림을 보면 깊이 좌표의 차이가 큰 점의 수가 많아 질수록 에러가 늘어나는 것을 볼 수 있다.

표 1. 참조점에 대한 에러

Point	0	1	2	3	4	5
Error(pixel)	2.33763	2.46799	1.66688	1.75333	1.96703	1.95909

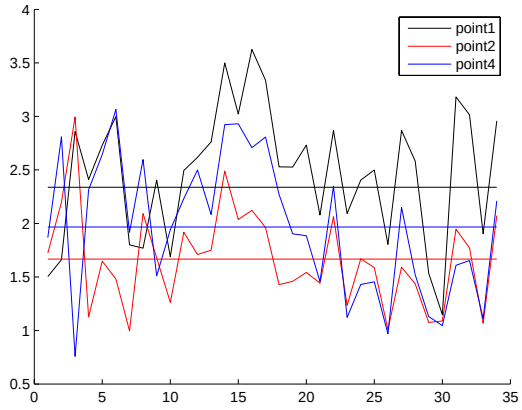


그림 11. 참조 점을 그림 10 의 점 1 과 점 2 와 점 4 를 참조 점으로 했을 경우의 에러 그래프

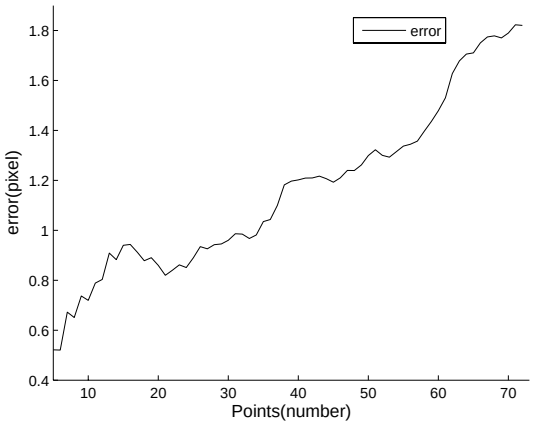


그림 12. 참조 점과의 거리 차이에 따른 에러

그림 13 은 점의 수에 따른 투사 에러를 나타내는 그래프이다. 점을 많이 사용할수록 정확하게 카메라 위치를 계산할 수 있다는 결론을 내릴 수 있다. 단, 참조점은 다른 점들과의 차이가 적게 나는 점을 설정 해야한다.

그림 14 는 점의 수에 따른 연산 시간을 나타내는 그래프이다. 점의 수가 증가함에 따라 수행 시간이 약간씩 증가 하는 것을 볼 수 있다. 72 개까지의 점에 대해서 실험을 해 보았을 때 가장 오래 걸린 시간이 3ms 이다. 70 개 이하의 점에 대해서는 1 초에 300 프레임 이상의 처리를 할 수 있는 증강현실 시스템에 적합한 고속의 알고리즘이다.

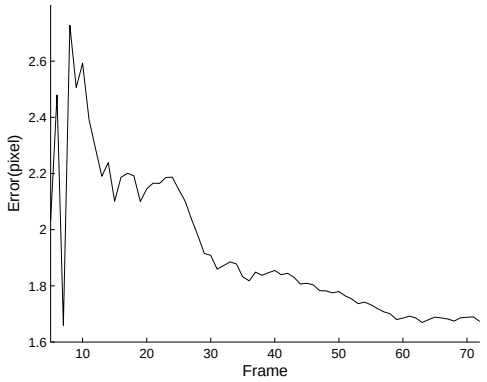


그림 13. 점의 수에 따른 에러율

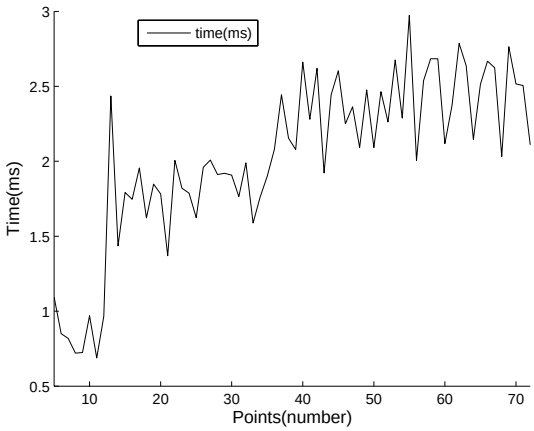


그림 14. 점의 수에 따른 연산 속도

(a) (b)

그림 15. (a) 원본 프레임, (b) 두 개의 가상객체를 합성한 프레임

그림 15 는 본 논문에서 제안한 방법으로 카메라의 위치를 계산하여 두 개의 가상객체를 합성한 결과를 보여주는 그림이다.

6. 결론

본 논문에서는 정사영 투영모델에서의 카메라 자세를 고속으로 계산해 내는 기법을 제안하였다. 정사영 투영모델에서 카메라 자세를 정확하게 계산하기 위해 참조점은 객체의 다른 점들과의 깊이 차이가 작은 점으로 설정을 해 주어야 한다. 구해진 카메라의 자세로 즉시 비디오의 단일프레임 별로 가상객체를 합성함으로써 실시간 기반의 증강현실 시스템에 적합한 카메라 자세 계산 기법을 제안하였다.

네 개의 점이 주어졌을 경우는 동일평면상이 아닌 경우 보다 동일평면상의 경우가 많다. 정사영 투영모델의 분해기법은 동일평면상이 아니어야 하기 때문에 카메라 위치를 계산하는데 있어 상당한 제약이 있다. 앞으로는 동일 평면상의 제약조건을 해결하여 동일평면상의 조건에서도 정확한 카메라 자세 계산을 위해 연구를 진행할 계획이다.

Reference

[1] Daniel G. Aliaga, Dimah Yanovsky, Thomas Funkhouse, Ingrid Carlbom, “Interactive Image-Based Rendering Using Feature Globalization,” Proc. of ACM SIGGRAPH 1995, pp.39-46, 1995.

[2] Alexei A. Efros, William T. Freeman, “Image quilting for texture synthesis and transfer,” Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, pp.341-346, August 2001.

[3] S. Seitz and C.R. Dyer, “Physically-Valid View Synthesis by Image Interpolation,” Proc. Workshop on Representation of Visual Scenes, pp.18-25, 1995.

[4] Jiangjian Xiao, Cen Rao and Mubarak Shah, “View Interpolation for Dynamic Scenes,” Proceedings of EUROGRAPHICS 2002, Germany, September 6-9, 2002.

[5] Saito H., Kimura M, Yaguchi S., Inamoto N., “View Interpolation of Multiple Cameras Based on Projective Geometry,” International Workshop on Pattern Recognition and Understanding for Visual Information Media, 2002.

[6] Seitz S., M., Dyer C. R., “View Morphing,” In Proc. SIGGRAPH 1996, pp.21-30, 1996.

[7] Max N. and Ohsaki K., “Rendering Trees from Precomputed Z-Buffer Views,” Rendering Techniques ’95 : Proceedings of the 6th Eurographics Workshop on Rendering, pp. 45-54, 1995.

[8] Matsushita K., Kaneko T., “Efficient and Handy Texture Mapping on 3D Surfaces,” In Proc. of Eurographics, pp. 349-358, 1999.

[9] Rocchini, C., Cignoni, P. and Montani, C., “Multiple textures stitching and blending on 3D objects,” Proc. 10th Eurographics Workshop on Rendering, pp. 127-138, 1999.
[10] Shou-Kang Wei, “Three-dimensional View Synthesis from Multiple Images,” Technical Report CITR-TR-42, University of Auckland, March 1999.
[11] Daniel F. DeMenthon and Larry S. Davis, “Model-Based Object Pose in 25 Lines of Code,” IJCV, Vol. 15, pp. 123-141, 1995.
[12] Jinhui Hu, Suya You, Ulrich Neumann, “Texture Painting from Video,” The Journal of WSCG, Vol. 13, pp. 119-125, 2005.
[13] Jean-Yves Bouguet, “Pyramidal Implementation of the Lucas Kanade Feature Tracker Description of the Algorithm,” Intel Corporation, Microprocessor Research Labs, 1999.