고품질 6DoF VR 영상을 위한 Dense Point Cloud 생성

□ 김성민, 한종기 / 세종대학교

요 약

다수의 이미지로부터 6DoF VR 영상을 얻기 위해서 먼저 각 이미지에 대해 특징점 추출과 매칭 과정을 수행하고, 이 정보를 바탕으로 각 이미지의 Camera Pose를 구해야 한다. 이 과정에서 특징점들을 3차원 좌표로 옮긴 Sparse Point Cloud를 얻을 수 있다. 하지만, Sparse Point Cloud만으로는 품질 높은 6DoF VR 영상을 얻을 수 없는데, 한 이미지에서 특징점은 전체 픽셀 수에 비해 매우 적기 때문이다. 따라서, 품질이 좋은 6DoF VR 영상을 위해서 모든 픽셀을 3차원 좌표로 옮긴 Dense Point Cloud를 얻는 과정이 필수적이다. 본고에서는 다수의 이미지와 이에 대한 Camera Pose를 활용해 모든 이미지의 Depth Map을 추정하는 Multi-View Stereo 알고리즘을 사용하여 최종적으로 Dense Point Cloud를 얻는 과정에 대해 설명한다.

Ⅰ. 서 론

6DoF VR 영상을 얻기 위해서는 한 장소에서 찍은 다수의 이미지들을 Structure from Motion (SfM) 등 [1]의 알고리즘을 활용해 각 이미지에 대한 Camera Pose를 얻어야 한다. SfM 알고리즘에서는 먼저, 각 이미지에 대한 특징점을 추출하고, 각 특징점의 descriptor 정보를 바탕으로 유사한 특징점끼리 매칭 관계를 계산한다. 그리고, 이 매칭 관계를 활용하여 각 이미지의 Camera Pose를 계산한다. Camera Pose와 특징점의 매칭 관계를 활용하면, 이미지 위의 특징점들을 3차원 월드 좌표계에서 표현할 수 있고, 모든 특징점들을 3차원 월드 좌표계에 표현한 3D Point의 집합을 Sparse Point Cloud라고 한다. 특징점이 아닌 픽셀은 매칭 관계가 없기 때문에 3차원 월드 좌표계로 표현할 수 없다.

특징점은 이미지의 전체 픽셀 수에 비하면 매우 적기 때문에, Sparse Point Cloud로는 품질 좋은 6DoF VR 영상을 얻을 수 없다. 따라서, 다수의 입력 이미지를 활용하여 3차원 물체나 공간을 복원하는 Multi-View Stereo 알고리즘을 사용하여 더욱 품질이 좋은 6DoF VR 영상을 얻는 것

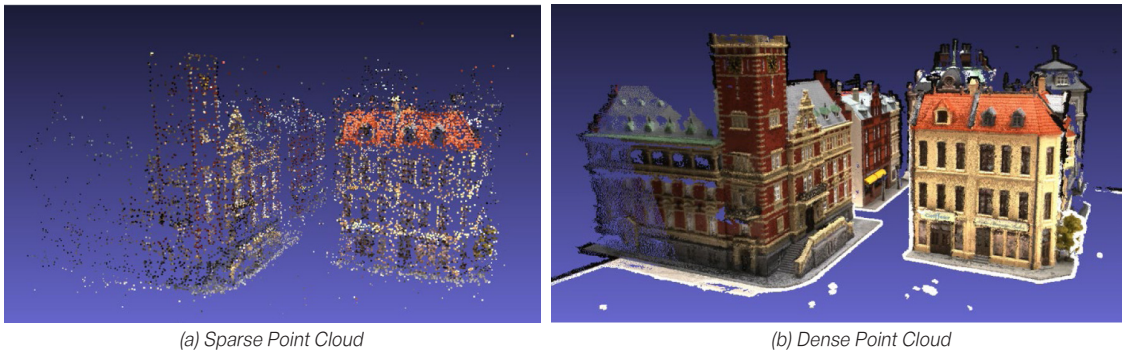
이 일반적이다. Multi-View Stereo를 활용하여 3차원 물체나 공간을 복원하는 방법은 여러 가지가 있지만, 그중에서도 Depth Map을 사용하는 방법이 활용도가 가장 높고, 가장 많이 사용된다. 하나의 물체나 공간을 표현하는 여러 장의 이미지에 대해 Multi-View Stereo 알고리즘을 사용하여 얻은 Depth Map과 Camera Pose를 이용하면 Dense Point Cloud를 만들 수 있고, 후처리 작업을 거쳐 더 품질이 좋은 6DoF VR 영상을 얻을 수 있다.

한 편, 한 장의 이미지만 입력으로 받아, 입력 이미지의 Depth Map을 계산하는 Monocular Depth Estimation 알고리즘을 통해서도 Depth Map을 얻을 수 있다. 하지만, Monocular Depth Estimation 알고리즘의 경우, Scale에 강인하지 않기 때문에 여러 장의 이미지에 대해 각각의 Depth Map을 얻어도, 서로의 정보가 3차원상에서 일치하지 않는 문제가 생긴다. 즉, Monocular Depth Estimation 알고리즘으로 얻은 Depth Map들은 6DoF VR 영상을 얻기 위해 활용하기에는 적절하지 않다. 따라서, 6DoF VR 영상을 얻을 때에는 여러 장의 이미지를 통해 얻은 주변 정보를 활용하여 더 정확하고 Scale에 강인한 Depth Map을 출력하는 Multi-View Stereo 기법을 사용한다.

Multi-View Stereo에서 Depth Map을 추정하는 전통적인 방법으로는 대표적으로 PatchMatch 알고리즘[2]이 있다. PatchMatch 알고리즘은 성능이 좋으며, 최근까지도 많이 사용되는 Depth Map 추정 알고리즘 중 하나이다. 하지만, PatchMatch 알고리즘은 특징점이 거의 없는 textureless region에서 Depth를 잘 추정하지 못한다는 단점을 가지고 있기 때문에, 이를 보완하기 위해 최근에도 많은 연구들이 진행되고 있다[3, 4, 5].

최근에는 전통적인 방법이 아닌, 딥러닝 기반의 Multi-View Stereo 기법들이 많이 등장하고 있다. 딥러닝을 기반으로 Depth Map을 추정하는 Multi-View Stereo 알고리즘들은 일반적으로 다수의 이미지를 입력으로 받아 3D Cost Volume을 만들고, 3D 컨볼루션 연산을 통해 Depth Map을 얻는다. 딥러닝 기반의 알고리즘은 전통적인 PatchMatch 알고리즘에서 잘 추정하지 못했던 textureless region에서 Depth를 더 잘 추정한다는 장점을 가지고 있지만, 3D Cost Volume을 만들고 이를 3D 컨볼루션 연산하는 과정에서 계산량이 증가하고 메모리를 너무 많이 소모한다는 단점이 있다. 따라서, 메모리 사용량을 줄이기 위해 차선책으로 해상도가 낮은 Depth Map을 얻은 후 이를 업샘플링하여 사용하는 등의 방법을 사용한다. 따라서, 최근에는 이러한 점을 보완하기 위해 더 적은 메모리를 사용하고도 정확한 Depth Map을 추정하는 딥러닝 기반의 Multi-View Stereo에 대한 연구들이 진행되고 있다[6, 7, 8].

본고에서는 대표적인 전통적인 알고리즘을 2장에서 설명하고, 최근의 딥러닝 기반의 알고리즘을 3장에서 설명하며 4장에서는 2, 3장에서 설명한 알고리즘들의 성능을 비교한다.



(a) Sparse Point Cloud (b) Dense Point Cloud

<그림 1> DTU Dataset [9]의 Scan 29 데이터의 Sparse Point Cloud와 Dense Point Cloud

Ⅱ. 전통적인 Multi-View Stereo

본 장에서는 전통적인 Multi-View Stereo 알고리즘인 PatchMatch 알고리즘에 대해 설명하고자 한다. PatchMatch 알고리즘은 총 네 단계로 구성된다. 첫 번째로, Depth를 구하고자 하는 이미지의 이웃 이미지를 찾는 Stereo Pair Selection 과정을 수행하고 두 번째로, 이미지의 이웃 관계를 이용하여 Depth Map을 계산한다. 세 번째로, Depth Map Refinement 과정을 진행하는데, 각 이미지에서 얻은 Depth Map이 3차원상에서 서로 일치하는지 확인하는 과정이다. 마지막으로 Depth Map Fusing 과정을 수행하는데, 여러 장의 이미지에서 얻은 Depth Map을 이용하여 해당 픽셀들을 3차원에 표현하여 Dense Point Cloud를 생성하는 과정이다.

1. Stereo Pair Selection

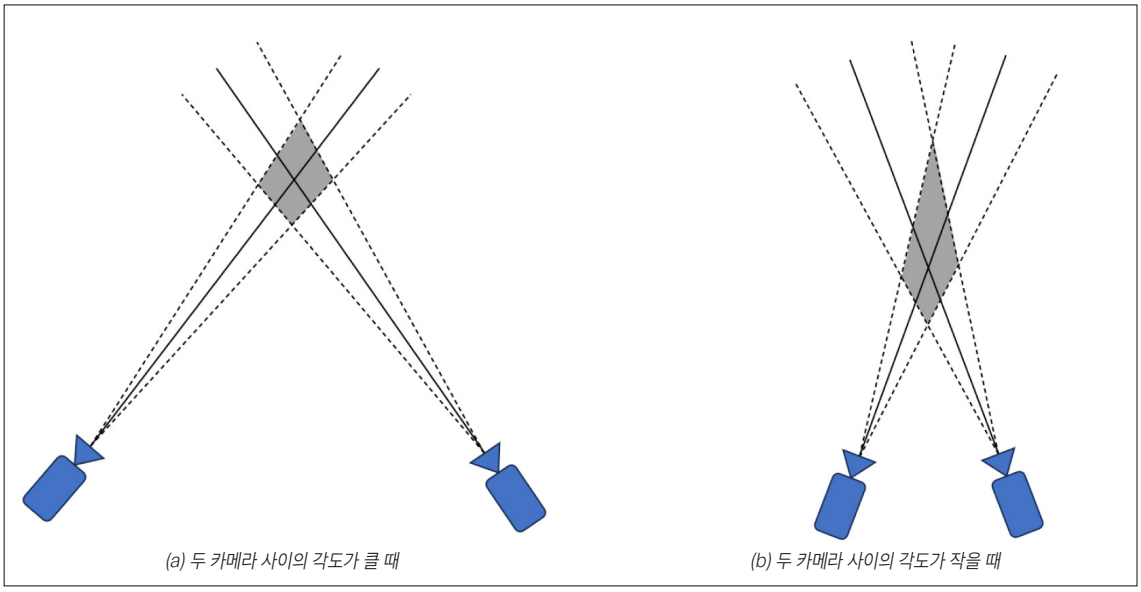
Stereo Pair Selection은 Depth Map을 계산하기 전에 수행하는 과정으로 기준 이미지와 비슷한 뷰를 바라보는 이웃 이미지들을 선택하는 과정이다. 먼저, 기준 이미지와 나머지 이미지들에 대해 카메라 중심 사이의 거리와 카메라 사이의 각도를 식 (1), (2), (3)을 이용하여 계산한다.

$$\theta_{ij} = \frac{\sum_{k=1}^{n} cos^{-1} \frac{\overrightarrow{X_k C_i} \cdot \overrightarrow{X_k C_j}}{\left\|\overrightarrow{X_k C_i}\right\| \left\|\overrightarrow{X_k C_j}\right\|}}{n} \quad (1)$$

$$d_{ij} = \sqrt{\left(C_i - C_j\right)^2} \quad (2)$$

$$C_i = -R_i^T t_i \quad (3)$$

여기서 θ_{ij}는 카메라 사이의 각도, d_{ij}는 카메라 중심 사이의 거리, C_i는 카메라 중심이며 i는 기준 이미지의 인덱스, j는 이웃 이미지의 후보의 인덱스이다. R_i, t_i은 i번째 이미지의 회전 행렬과 이동 벡터이며 X_k은 i번째 이미지와 j번째 이미지 모두에서 보이는 k번째 3D Point, n은 X_k의 개수이다. 카메라 사이의 각도는 i번째 이미지와 j번째 이미지 보이는 모든 3D Point에 대해 $\angle C_i X_k C_j$를 계산하여



(a) 두 카메라 사이의 각도가 클 때 (b) 두 카메라 사이의 각도가 작을 때

<그림 2> 두 카메라 사이의 각도에 따른 3D Point의 오차 범위

그 평균을 취하고, 카메라 중심 사이의 거리는 유클라디안 거리를 활용하여 얻는다.

카메라 중심 사이의 거리가 너무 가깝고 카메라 사이의 각도가 너무 작으면 Camera Pose에 약간의 에러만 존재하더라도 픽셀을 3차원상으로 옮길 때 〈그림 2〉(b)처럼 오차 범위가 커진다. 하지만, 카메라 중심 사이의 거리가 너무 멀고 카메라 사이의 각도가 너무 크면 두 이미지가 중첩되는 영역이 적어 바라보는 뷰가 서로 다를 수 있다. 카메라 중심 사이의 거리가 너무 멀고 카메라 사이의 각도는 작은 경우에는 두 이미지의 Scale이 달라 이웃 이미지로 선택되기 적합하지 않을 수 있다. 따라서, 카메라 사이의 각도와 거리가 너무 작지 않은 선에서 이웃 이미지를 선택해야 한다.

이에 따라서 PatchMatch 알고리즘 [2]에서는 j번째 이미지가 i번째 이미지의 이웃 이미지가 되기 위해서는 카메라 사이의 각도와 거리가 식 (4)와 식 (5)를 만족해야 한다고 제시한다.

$$5° < \theta_{ij} < 60° \tag{4}$$

$$0.05\bar{d} < d_{ij} < 2\bar{d} \tag{5}$$

$\bar{d}$는 $d_{i0}, d_{i1}, \cdots, d_{ij}$ 중 중간 값이다. 식 (4), (5)를 만족하지 못하는 이미지는 기준 이미지인 i번째 이미지의 이웃 이미지가 될 수 없으므로 이웃 이미지 후보에서 탈락시킨다. 이후 남은 이웃 이미지 후보들에 대해서 θ_{ij}, d_{ij}를 계산한 후, 이 값을 기준으로 오름차순 정렬한다. 정렬된 값들 중에서 첫 N장이 i번째 이미지의 이웃 이미지가 된다.

2. Depth Map Computation

이웃 이미지 선별이 끝나면, 이를 활용해 기준 이미지의 Depth Map을 추정한다. Depth를 추정하기 위한 주요 아이디어는 입력 이미지의 각 픽셀 p가 월드 좌표계의 3차원 공간에서 어떤 평면 f위에 있다고 가정하고, 그 평면을 찾는 것이다. 평면 f를 찾기 위해서는 Depth 뿐만 아니라 법선 벡터도 추정해야 한다. 즉, Depth와 법선 벡터를 정의하면, 평면 f가 특정된다.

Depth Map을 계산하는 첫 단계는 랜덤 초기화로, 이미지 내 각 픽셀을 임의의 Depth $\lambda \in [\lambda_{min}, \lambda_{max}]$를 만족하는 λ 중 랜덤으로 초기화한다. 일반적으로, λ_{min}는 이미지에서 바라보는 Sparse Point Cloud의 3D Point 중 가장 Depth가 작은 값, λ_{max}는 가장 Depth가 큰 값으로 설정한다. Sparse Point Cloud를 통해 특징점의 Depth는 알고 있으므로 특징점의 Depth는 알고 있는 값으로 작성한다. 법선 벡터 $\vec{n}$는 구 좌표계에서 정의되며 식 (6)과 같이 θ, φ 두 변수로 구성된다. θ는 0°에서 360° 사이의 값을 가지며 φ는 0°에서 90° 사이의 값을 가진다. 이 역시 랜덤으로 추정한다.

$$\vec{n} = \begin{bmatrix} \cos\theta \sin\phi \\ \sin\theta \sin\phi \\ \cos\phi \end{bmatrix} \tag{6}$$

만약, 랜덤 초기화 과정에서 평면 f를 잘 찾으면 입력 이미지와 이웃 이미지의 호모그래피 행렬이 잘 구해지고, 그렇지 않으면 잘못된 호모그래피 행렬이 구해진다. 즉, 호모그래피 행렬을 통해 Depth가 잘 구해졌는지 평가할 수 있다. 호모그래피 행렬은 수식 (7)로 구할 수 있다.

$$H = K_j(R_j R_i^{-1} + \frac{R_j(C_i - C_j)\vec{n}^T}{\vec{n}^T X})K_i^{-1} \tag{7}$$

K_i, R_i, C_i는 각각 기준 이미지의 카메라 내부 행렬, 회전 행렬 그리고 카메라 위치 벡터이며, K_j, R_j, C_j는 각각 이웃 이미지의 카메라 내부 행렬, 회전 행렬 그리고 카메라 위치 벡터이다. $\vec{n}$은 픽셀 p에서 추정한 법선 벡터, X는 카메라 좌표계에서 픽셀 p의 좌표이며 현재 추정한 Depth λ를 사용하여 $X=\lambda K_i^{-1}p$를 이용하여 계산할 수 있다.

이를 활용하여 Depth를 평가하기 위해서는 먼저 Depth를 평가하고자 하는 픽셀 p 주변에 $w \times w$ 크기의 윈도우 B를 만든다. 그 후, 픽셀 p의 Depth λ와 법선

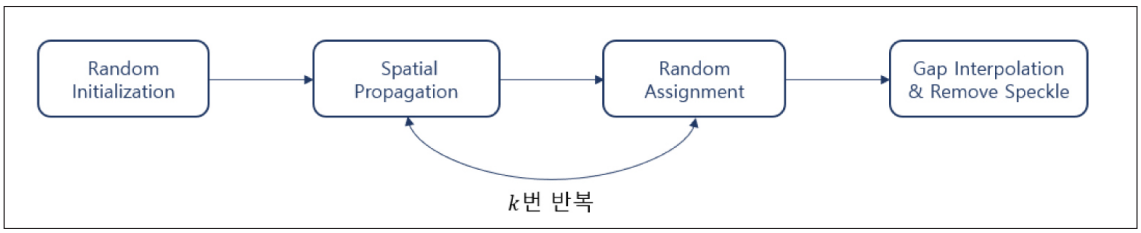
$$m(p,\lambda,\vec{n}) = 1 - ZNCC = 1 - \frac{\sum_{p\in B}(I(p)-\overline{I(p)})(I(H(p))-\overline{I(H(p))})}{\sqrt{\sum_{p\in B}(I(p)-\overline{I(p)})^2 \sum_{p\in B}(I(H(p))-\overline{I(H(p))})^2}} \quad (8)$$

벡터 $\vec{n}$를 활용하여 식 (7)과 같이 호모그래피 행렬 H를 계산하고, 이를 이용하여 윈도우 B 내의 모든 픽셀을 이웃 이미지로 투영시킨다. 이웃 이미지에 투영된 윈도우를 B'라고 한다면, B와 B' 사이의 밝기 분포를 ZNCC로 측정한 후, 식 (8)과 같이 최종적으로 픽셀 p의 Depth λ의 불확실성을 계산한다. Depth λ와 법선 벡터 $\vec{n}$가 정확하여 호모그래피 행렬 H가 잘 구해졌으면, B와 B'이 같은 신호 정보를 담고 있을 것이므로 밝기 분포 역시 유사할 것이고, 그렇지 않다면 B와 B'이 다른 신호를 담고 있으며 Depth λ와 법선 벡터 $\vec{n}$가 부정확하다는 것을 의미한다.

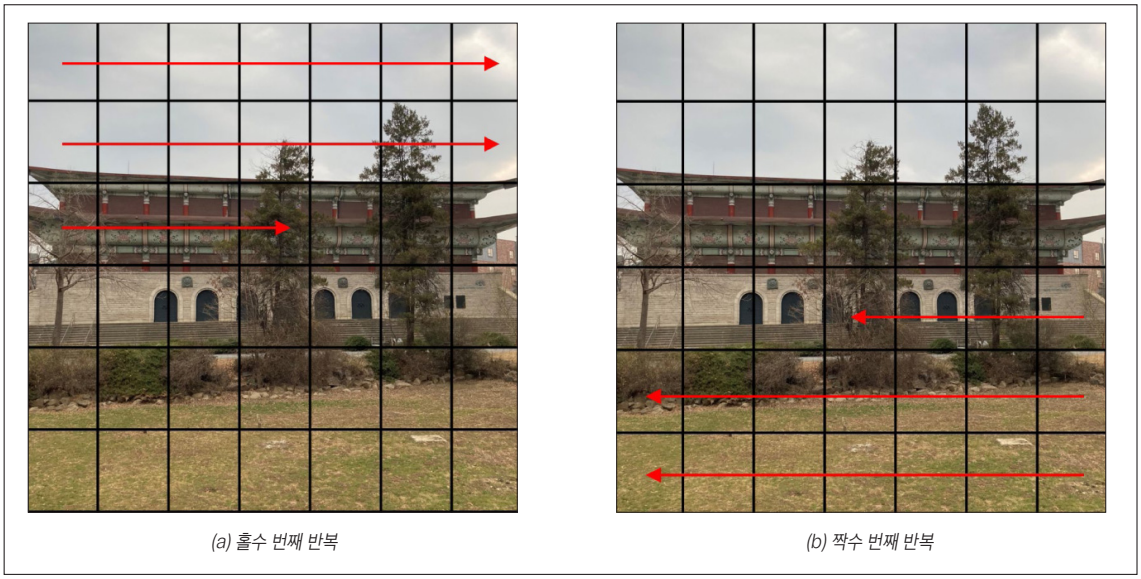
$m(p,\lambda,\vec{n})$은 픽셀 p에서 추정한 Depth λ와 법선 벡터 $\vec{n}$의 불확실성을 뜻하며, 낮을수록 Depth λ와 법선 벡터 $\vec{n}$의 신뢰도가 높다. B는 기준 이미지에서 픽셀 p 주변의 $w\times w$ 크기의 윈도우이며, $I(\cdot)$는 입력 픽셀 위치에서의 밝기를 뜻하며, $H(\cdot)$는 식 (7)로 구한 호모그래피 행렬을 이용해 입력 픽셀을 이웃 이미지로 투영시키는 연산을 의미한다. $\overline{I(\cdot)}$는 윈도우 B 내의 밝기 평균을, $\overline{I(H(\cdot))}$는 윈도우 B를 이웃 이미지로 투영시킨 윈도우 B'의 밝기 평균을 의미한다. 만약, 이웃 이미지가 N장인 경우, 각 이웃 이미지를 사용하여 픽셀 p의 불확실성을 N개 구한 후, 이를 평균하거나 중간 값을 사용하는 등의 방법을 사용할 수 있다.

랜덤 초기화 과정을 통해 각 픽셀의 Depth와 법선 벡터를 초기화 했다면, 각 픽셀의 Depth를 Spatial Propagation과 Random Assignment 과정을 반복 수행하여 추정하며, 결과적으로 Depth Map을 얻게 된다. 즉, 전체적인 Depth Map 계산 과정은 〈그림 3〉과 같다.

Spatial Propagation은 인접한 픽셀들은 3차원상에서 같거나 비슷한 평면에 존재하여 Depth와 법선 벡터 역시 비슷할 것이라는 가정하에 이루어진다. 현재 Depth를 구하고자 하는 픽셀이 p이고, p의 Depth가 λ, 법선 벡터가 $\vec{n}$이며, p 주변의 이웃 픽셀 p_n이 가진 Depth를 λ_n, 법선 벡터 $\vec{n}_n$라 하자. 픽셀 p와 픽셀 p_n는 가까이 있는 픽셀이므로 같은 평면 위에 있을 확률이 높다. 따라서, 픽셀 p_n의 Depth λ_n와 법선 벡터 $\vec{n}_n$를 픽셀 p로 가져와 불확실성을 측정한다. 만약, 불확실성 $m(p,\lambda,\vec{n})$ 보다 $m(p,\lambda_n,\vec{n}_n)$이 더 낮게 측정된다면, 픽셀 p의 Depth와 법선 벡터로 기존의 $(\lambda,\vec{n})$ 대신 $(\lambda_n,\vec{n}_n)$를 사용한다. 이 과정을 모든 픽셀에 대해 수행하는 것을 Spatial Propagation이라고 하며, 이는 〈그림 4〉처럼 여러 번 반복된다. 홀수 번째 반복에서는 〈그림 4〉(a)처럼 전체 이미지의 좌상단부터 우하단 방향으로 Spatial Propagation하며, 이웃 픽셀은 〈그림 5〉에서 파란 부분에 해당하는 부분이 이웃 픽셀이 되고, 짝수 번째 반복에서는 〈그림 4〉(b)처럼 우하단부터 좌상단 방향으로 Spatial Propagation하

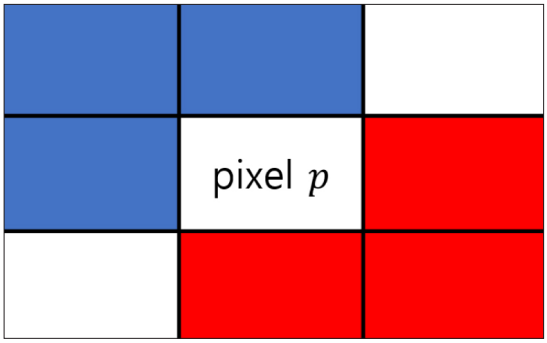


<그림 3> Depth Map Computation의 전체 과정



(a) 홀수 번째 반복　　(b) 짝수 번째 반복

<그림 4> 반복 횟수에 따른 Spatial Propagation의 진행 방향



<그림 5> 현재 픽셀 p와 인접한 8개의 픽셀

며, 〈그림 5〉에서 빨간 부분에 해당하는 부분이 이웃 픽셀이 된다.

Random Assignment는 현재 픽셀 p의 Depth λ와 법선 벡터 $\vec{n}$의 두 파라미터 θ, ϕ의 주변 값들 중에서 랜덤으로 Depth와 법선 벡터를 취한 뒤, 불확실성을 계산하여 불확실성 $m(p,\lambda,\vec{n})$ 보다 $m(p,\lambda',\vec{n'})$이 더 낮게 측정된다면, 현재 픽셀 p가 가진 Depth λ와 법선 벡터 $\vec{n}$ 대신 Depth λ'와 법선 벡터 $\vec{n'}$를 취하는 알고리즘이다.

구체적으로, 현재 픽셀 p의 Depth λ와 법선 벡터 $\vec{n}$의 두 파라미터 θ, ϕ의 주변에서 랜덤으로 새로운 Depth λ'와 새로운 θ', ϕ'로 계산되는 새로운 법선 벡터 $\vec{n'}$를 얻는다. 이때, 식 (9)와 같이 $\Delta\lambda$, $\Delta\theta$, $\Delta\phi$ 이 주변 값의 경계값이 된다. 만약, 새로운 Depth λ'와 법선 벡터 $\vec{n'}$로 계산한 불확실성 $m(p,\lambda',\vec{n'})$이 기존의 Depth와 법선 벡터의 불확실성 $m(p,\lambda,\vec{n})$보다 낮다면, λ'가 현재 픽셀 p의 Depth, $\vec{n'}$가 새로운 법선 벡터가 된다. 그리고, $\Delta\lambda$, $\Delta\theta$, $\Delta\phi$ 값을 절반으로 줄인 후 위 과정을 반복한다. 만약, 랜덤으로 선택한 Depth와 법선 벡터의 불확실성 $m(p,\lambda',\vec{n'})$이 기존 값 $m(p,\lambda,\vec{n})$ 보다 높다면, 다음 픽셀로 넘어간다.

앞서 언급한 대로, Depth Map은 위에서 설명한 Spatial Propagation과 Random Refinement를 〈그림 3〉과 같이 여러 번 반복하면 얻을 수 있다. 하지만, Spatial Propagation과 Random Refinement의 반복 횟수를 늘

$$\lambda' \in [\lambda - \Delta\lambda,\ \lambda + \Delta\lambda],\ \theta' \in [\theta - \Delta\theta,\ \theta + \Delta\theta],\ \phi' \in [\phi - \Delta\phi,\ \phi + \Delta\phi] \tag{9}$$

린다고 해서, Depth Map의 정확도가 선형적으로 올라가지는 않기 때문에 계산 효율성을 위해서 적당한 반복 횟수를 찾는 것이 좋다. 이렇게 얻은 Depth Map 중에서 불확실성이 임계 값보다 높은 픽셀의 경우는 Depth를 찾지 못한 것으로 정의한다.

마지막으로 Gap Interpolation이나 Speckle 제거 단계 등 후처리 작업을 통해 Depth Map을 수정할 수 있다. 두 알고리즘 모두 근접한 픽셀은 같은 평면 위에 있을 확률이 높으며, 이에 따라 유사한 Depth를 가질 확률이 높다는 가정을 가지고 있다. Gap Interpolation은 Depth를 찾지 못한 픽셀들의 세그먼트가 일정 크기 이하이면, 주변의 Depth 정보를 이용하여 선형 보간시키는 작업이며, Speckle 제거는 비슷한 Depth를 가지는 픽셀들로 이루어진 세그먼트에 대해서 그 세그먼트의 크기가 너무 작으면 해당 세그먼트의 Depth를 모두 잘못 구한 것으로 판단하는 것이다. 인접한 픽셀들끼리 Depth가 비슷하다는 것은 곧 식 (10)을 만족한다는 것이며, 만약 그렇지 않으면 서로 다른 세그먼트로 분리된다. 임계 값은 주로 0.01을 사용한다.

$$\frac{|d_0 - d_1|}{d_0} < threshold \tag{10}$$

3. Depth Map Refinement

모든 이미지에 대하여 Depth Map을 얻고 나면, 각 Depth Map들이 3차원상에서 일치하는 정보를 가지고 있는지를 확인하여야 한다. 만약, 그렇지 않다면, Dense Point Cloud를 구성할 때 품질이 떨어진다. 먼저, 식 (11)과 같이 기준 이미지의 각 픽셀 p에 대해 p의 depth λ와 카메라 파리미터를 통해 3차원 월드 좌표계로 이동시킨다.

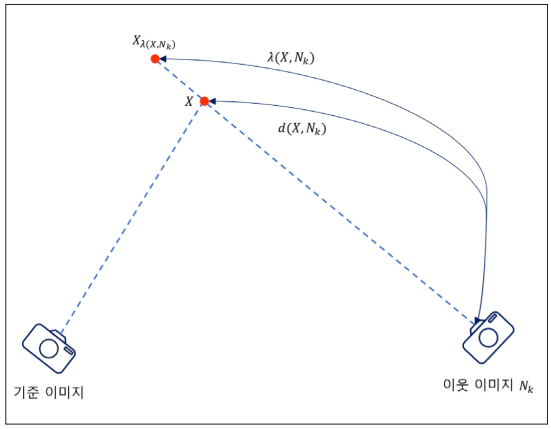
$$X = \lambda R_i^T K_i^{-1} p + C_i \tag{11}$$

X는 픽셀 p를 3차원 월드 좌표계로 이동시킨 점이며, K_i, R_i, C_i는 각각 기준 이미지의 카메라 내부 행렬, 회전 행렬 그리고 카메라 위치 벡터이다.

그 다음, X를 이웃 이미지로 투영시킨다. N_k가 k번째 이웃 이미지이고, X를 N_k에 투영하여 얻은 점을 X_k^N이라고 하면, 이웃 이미지 N_k의 Depth Map의 X_k^N에 해당하는 depth를 $\lambda(X, N_k)$라고 한다. 그리고, X를 이웃 이미지 N_k의 3차원 카메라 좌표계로 옮긴 뒤 얻은 Z축 좌표, 즉, N_k의 카메라에서 바라본 depth를 $d(X, N_k)$라고 표기한다. 만약, $\lambda(X, N_k)$와 $d(X, N_k)$이 식 (12)를 만족하면, 기준 이미지에서 구한 픽셀 p의 depth λ를 활용하여 얻은 X가 이웃 이미지 N_k와 일관성이 있다고 할 수 있다. 만약, X가 적어도 n개의 이웃 이미지와 일관성이 있다면, 신뢰할 수 있는 포인트로 여겨지며, 기준 이미지의 픽셀 p의 depth λ는 유지되고, 그렇지 않으면 지워진다. n은 일반적으로 2이다.

$$\frac{|d(X, N_k) - \lambda(X, N_k)|}{\lambda(X, N_k)} < threshold \tag{12}$$

이 과정을 모든 이미지의 모든 픽셀에 대하여 진행한다. Depth Map Refinement를 진행하고 나면, 대부분의 에러는 제거되며 상대적으로 각 이미지에서 깨끗한 Depth Map을 얻을 수 있게 된다.



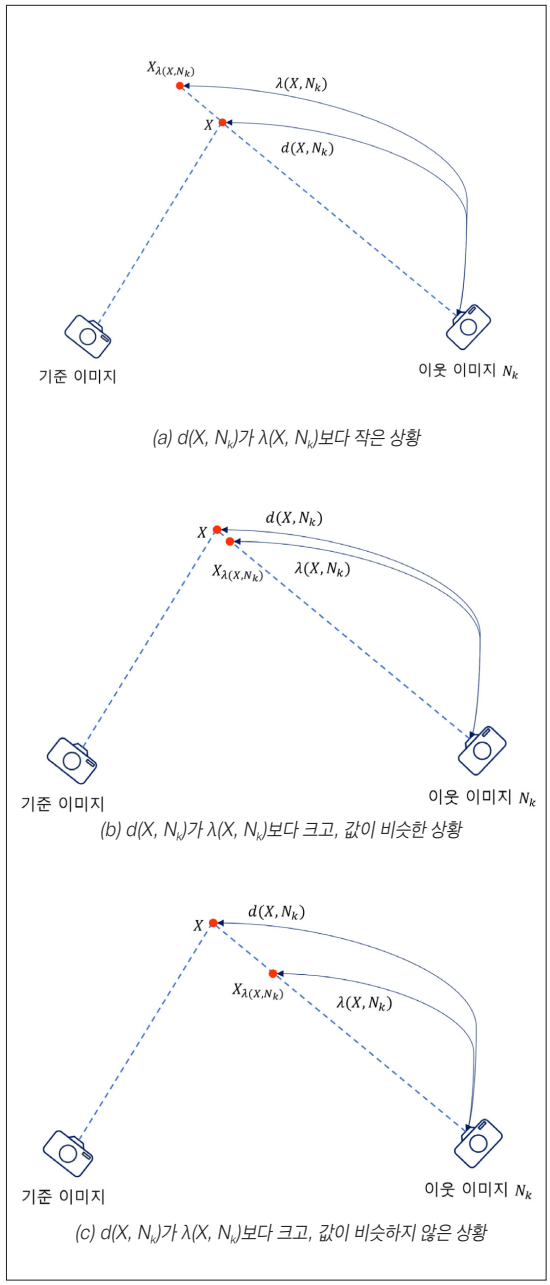
<그림 6> 이웃 이미지 N_k에서 X를 바라봤을 때의 Depth $d(X, N_k)$와 X를 N_k에 투영시킨 후 Depth Map을 통해 얻은 Depth $\lambda(X, N_k)$

4. Depth Map Fusing

Depth Map Refinement까지 마치면, Depth Map의 정보를 활용하여 Dense Point Cloud를 바로 만들 수 있다. 하지만, 이는 상당히 많은 중복 계산을 포함하는데, 각 이미지들이 중복되는 뷰를 공유하기 때문이다. 따라서, 이러한 반복을 제거하기 위한 알고리즘이 필요하다.

먼저, 현재 이미지 I_i에서 식 (11)을 이용하여 각 픽셀을 3차원 월드 좌표계의 3D 포인트 X로 투영시키고, 이를 이웃 이미지로 다시 재투영한다. 이때, 세 가지 상황이 발생한다. 첫 번째는 〈그림 7〉(a)와 같이 $d(X, N_k)$가 $\lambda(X, N_k)$보다 작은 상황으로, 이런 경우 해당 포인트는 이웃 이미지 N_k에서 계산하기로 하고 건너뛴다. 두 번째는 〈그림 7〉(b)와 같이 $d(X, N_k)$가 $\lambda(X, N_k)$보다 크고, 식 (12)를 만족하는 경우이다. 이 경우는 X에 대해 현재 이미지 I_i에서 구한 Depth와 이웃 이미지 N_k에서 구한 Depth의 불확실성을 반영하여 가중치 평균함으로써 최종 Depth를 결정하고 3D 포인트의 좌표를 결정한다. 그리고, 이웃 이미지 N_k에서 X가 투영된 위치의 Depth는 지워서 중복으로 계산되는 것을 막는다. 마지막은 〈그림 7〉(c)와 같이 $d(X, N_k)$가 $\lambda(X, N_k)$보다 크지만, 식 (12)를 만족하지 않는 상황으로 앞의 두 조건을 만족하지 않는 경우이다. 이 경우에는 현재 이미지 I_i에서 구한 X와 이웃 이미지 N_k의 Depth Map에서 얻은 $\lambda(X, N_k)$ 정보를 활용하여 얻은 $X_\lambda(X, N_k)$가 같은 포인트가 아닌 서로 다른 포인트라고 가정한다. 따라서, Point Cloud에는 현재 이미지에서 얻은 Depth 정보만 활용하여 X를 표시하고, 이웃 이미지에서 $\lambda(X, N_k)$를 지우지 않는다.

이 과정을 모든 이미지의 모든 픽셀에 대해 수행하며, Depth를 구하지 못하거나 중복 계산을 방지하기 위해 지운 픽셀에 대해서는 연산을 수행하지 않고 건너뛴다. Depth Map Fusing 과정이 끝나면, Sparse Point Cloud보다 훨씬 촘촘한 Dense Point Cloud를 얻을 수 있다. 이를 이용하면 Mesh Reconstruction[10]이나 Texturing[11] 등의 후처리 과정을 거쳐 더욱 현실성 있는 6DoF VR 영상을 제작할 수 있다.



<그림 7> Depth Map Fusing 시 만나게 되는 세 가지 상황

Ⅲ. 딥러닝 기반의 Multi-View Stereo

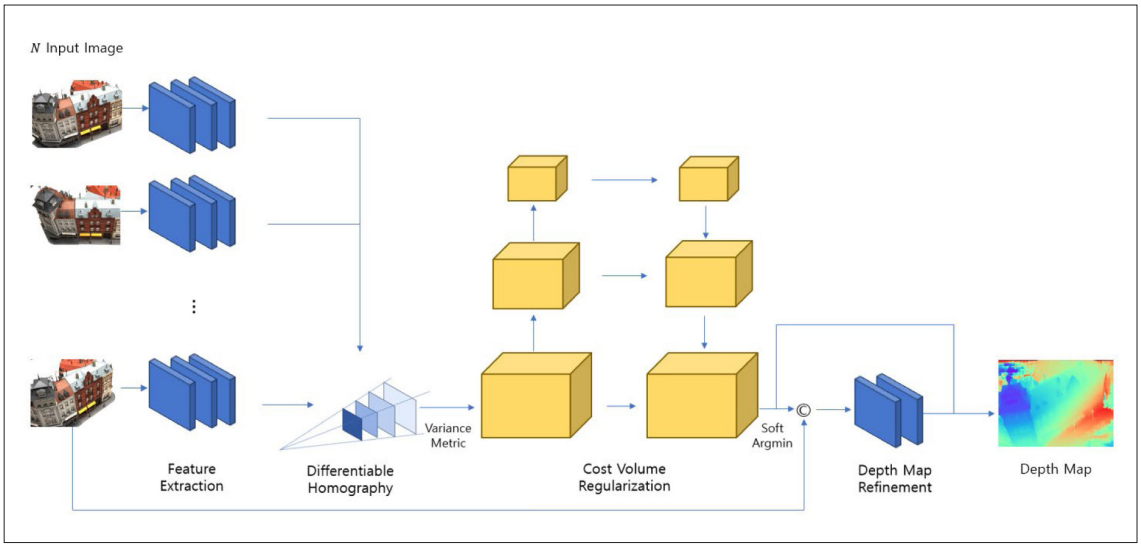
PatchMatch 기반의 전통적인 Multi-View Stereo는 뛰어난 Depth Map 추정 능력을 가지지만, 특징점이 적은 textureless region이나 반복적인 무늬가 나타나는 지역, 투명한 물체가 있는 곳 등에서는 Depth를 잘 추정하지 못한다는 단점이 있다. 이를 개선하기 위한 연구가 많이 진행되고 있는데, 딥러닝을 활용한 Multi-View Stereo에 대한 연구 역시 활발히 진행되고 있다.

이번 장에서는 3D Cost Volume을 활용하면서 end-to-end 학습이 가능한 딥러닝 모델 MVSNet[12]과 MVSNet의 3D Cost Volume의 단점을 개선한 Cascade Cost Volume[13]에 대해 설명하고자 한다. 두 알고리즘 모두 기준 이미지와 기준 이미지의 이웃 이미지를 입력으로 넣으며, 모든 이미지의 Camera Pose가 필요하다. 출력으로 기준 이미지의 Depth Map을 얻을 수 있으며, Dense Point Cloud를 얻기 위해서 수행해야 하는 Depth Map Refinement와 Depth Map Fusing은 기존의 알고리즘을 활용한다.

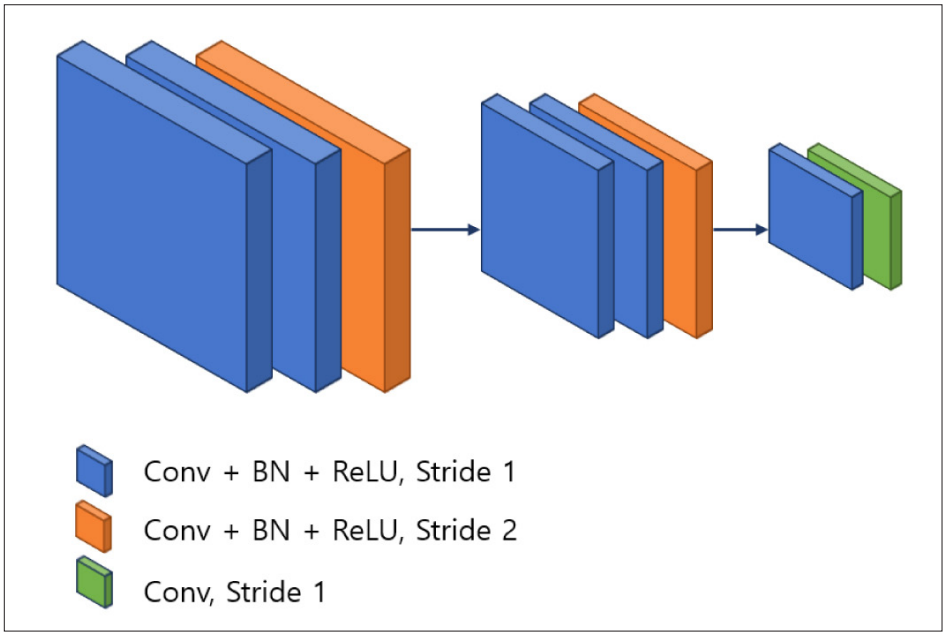
1. MVSNet

MVSNet[12]는 3D Cost Volume을 활용하여 Depth Map을 얻는 end-to-end 학습이 가능한 딥러닝 모델을 제안하였다. 최근의 많은 딥러닝 기반의 Multi-View Stereo 알고리즘은 3D Cost Volume을 활용하는 MVSNet의 아이디어를 활용한다. 전통적인 PatchMatch 알고리즘은 Depth를 추정하고 평가할 때 해당 픽셀의 주변 $w \times w$ 윈도우의 정보밖에 활용하지 못한다. 즉, receptive field의 크기가 작다. 반면에, 딥러닝 기반의 MVSNet은 3D Cost Volume과 컨볼루션 연산을 사용함으로써 receptive field가 PatchMatch 알고리즘보다 훨씬 커지기 때문에 MVSNet이 반복적인 무늬가 나타나는 지역이나 특징점이 적은 textureless region에서 전통적인 PatchMatch 알고리즘보다 Depth를 더 잘 추정한다.

MVSNet의 전체적인 구조는 〈그림 8〉과 같다. MVSNet의 가장 첫 부분은 2D CNN 네트워크를 통한 Feature Extraction이다. *N*장의 입력 이미지를 각각 2D CNN 네트워크에 통과시켜 *N*장의 Feature Map을 얻는다. 이때, 한 장은 Depth를 구하고자 하는 기준 이미지이며, 나머지



<그림 8> MVSNet의 구조



<그림 9> MVSNet의 Feature Extraction을 위한 CNN 네트워크 구조

N-1장은 기준 이미지에 대한 이웃 이미지이다. Feature Extraction을 위한 2D CNN 네트워크는 〈그림 9〉와 같이 8개의 2D CNN 구조로 이루어진다. 세부적으로 마지막 레이어를 제외한 모든 CNN 레이어 이후에는 배치 정규화 [14]와 ReLU 함수가 적용된다. 3번째와 6번째 CNN 레이어의 보폭은 2로 설정한다. 출력으로 얻는 Feature Map은 32개의 채널을 가지며, 입력 이미지에 비해 너비와 높이가 1/4배 감소한다. 즉, *N*장의 입력 이미지를 Feature Extraction 네트워크에 입력시키면 32채널 Feature Map을 *N*장 얻는다.

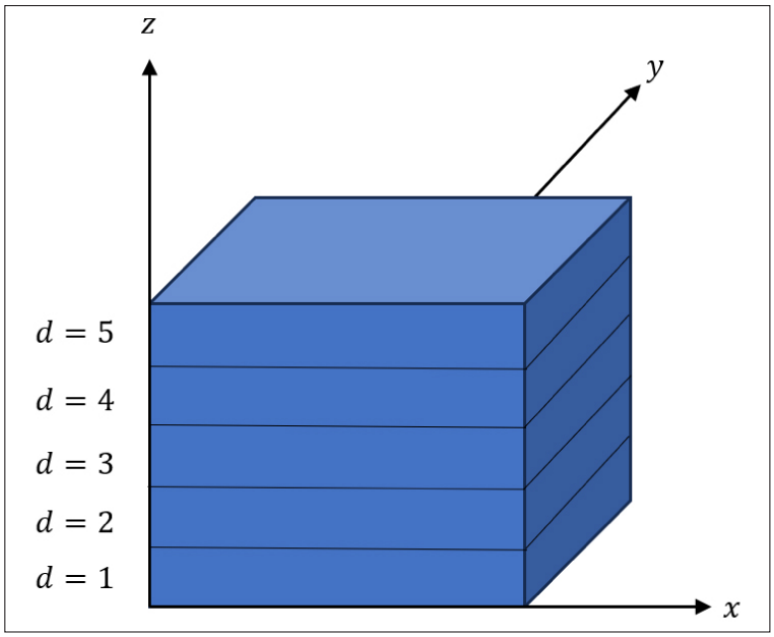
다음으로, Feature Extraction을 통해 얻은 *N*장의 Feature Map을 사용해 하나의 3D Cost Volume을 얻는다. 먼저, Sparse Point Cloud를 참조하여 Depth 범위인 [λ_{min}, λ_{max}]를 정하고, Depth 범위를 몇 개의 값으로 샘플링할지 정한다. 그리고, Feature Map을 기준 카메라 좌표계의 z= λ 평면에 식 (13)을 이용해 warping한다.

$$H_i(d) = K_i R_i (I - \frac{(t_1 - t_i) n_1^T}{d}) R_1^T K_1^T \qquad (13)$$

K_i, R_i, t_i는 warping을 하려는 Feature Map의 카메라 내부 행렬, 카메라 회전 행렬, 카메라 이동 벡터이며, K_1, R_1, t_1는 기준 이미지의 카메라 내부 행렬, 카메라 회전 행렬, 카메라 이동 벡터이다. d는 warping 하려는 평면의 Depth 값이며, n_1은 warping 하려는 평면의 법선 벡터이다.

예를 들어, 〈그림 10〉과 같이 λ_{min}=1, λ_{max}=5이고, 해당 범위를 균일하게 5개의 값으로 샘플링한다면, Feature Map을 기준 카메라 좌표계의 z=1, z=2, …, z=5 평면에 warping하여 Feature Volume을 얻는다. Feature Map이 *N*장이므로, Feature Volume 역시 *N*개 생성되며, 각 Feature Volume은 $\frac{W}{4} \times \frac{H}{4} \times D \times F$ 차원을 가진다. W, H는 각각 입력 이미지의 너비와 높이, D는 Depth 샘플링 수, F는 Feature Map의 채널 수이다. 이렇게 *N*장의 Feature Volume을 얻는 과정을 Differentiable Homography라고 한다.

이제, *N*장의 Feature Volume을 하나의 3D Cost Volume C로 합치기 위해 variance-based Cost Metric



<그림 10> Depth 샘플링에 따른 Feature Volume

$\mathcal{M}$을 사용한다. Variance-based Cost Metric을 통해 입력 이미지의 수 N과 상관없이 하나의 3D Cost Volume을 얻을 수 있으므로, MVSNet의 전체 입력 이미지 수에 구조적인 제한이 없어진다. Variance-based Cost Metric $\mathcal{M}$은 식 (14)를 통해 계산된다.

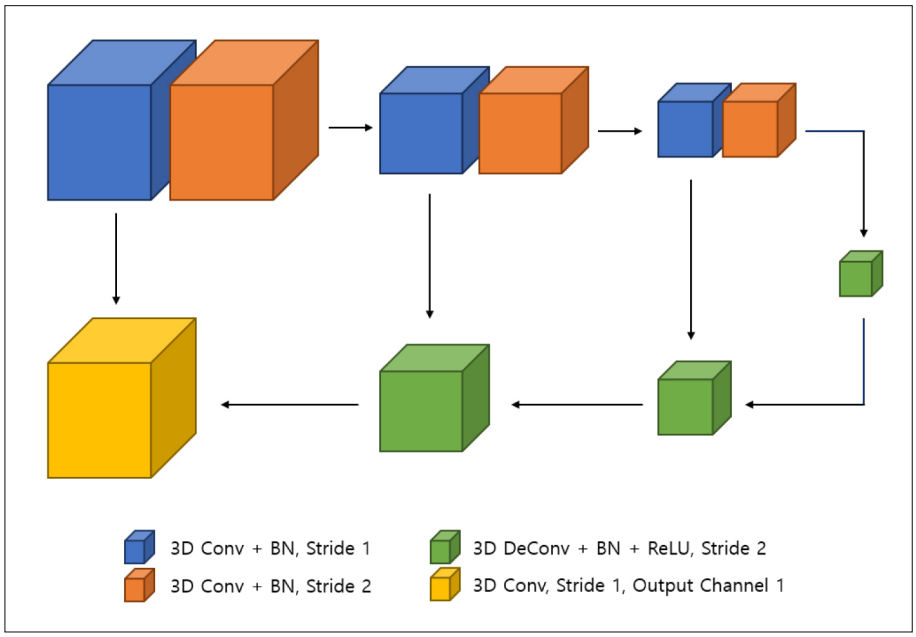
$$C = \mathcal{M}(V_1 \cdots V_N) = \frac{\sum_{i=1}^{N}(V_i - \bar{V})^2}{N} \qquad (14)$$

C는 3D Cost Volume, V_i는 i번째 입력 이미지의 Feature Volume, $\bar{V}$는 모든 Feature Volume의 평균, N은 Feature Volume의 개수이다. 모든 연산은 Feature Volume의 원소별로 이루어진다.

이렇게 얻은 Cost Volume C를 사용하여 Cost Volume Regularization을 수행한다. 이는 Cost Volume C를 사용하여 Probability Volume P를 만드는 과정이다. 먼저, Cost Volume C를 3D CNN 네트워크에 통과시킨다. 이는 반사 물질이나 가려지는 물체 등에 의한 노이즈를 제거함과 동시에 Cost Volume C를 활용하여 Depth Map을 추론하기 위함이다.

3D CNN 네트워크는 〈그림 11〉과 같이 구성되며, 3D 버전의 UNet[15]과 유사하다. 먼저, 첫 3D CNN 레이어에서 32채널 Cost Volume을 8채널로 줄이는데, 이를 통해 계산량을 감소시킨다. 마지막 3D CNN 레이어의 출력은 1채널 Volume이며, 모든 3D CNN 레이어를 거치고 나면, Depth 방향으로 Softmax를 적용하여 확률 정규화한다.

Cost Volume Regularization 과정을 마치면, 각 픽셀에 대해 맨 처음에 샘플링한 모든 Depth 값의 확률을 얻게 된다. 이 확률 값이 높을수록, 해당 픽셀의 Depth가 될 확률이 높음을 의미한다. Depth Map을 얻는 가장 간단한 방법은 각 픽셀마다 가장 높은 확률을 가지는 Depth를 취하여 Depth Map을 구성하는 것이지만, 이는 미분 불가능해서 역전파가 불가능해지며, Depth Map을 업샘플링할 때, 새로 생기는 픽셀에 대해 Depth를 추정하기 어려워진다. 따라서, 식 (15)와 같이 각 픽셀에 대해 Depth 방향을 따라 확률 기반으로 기대값을 계산



<그림 11> MVSNet의 Cost Volume Regularization을 위한 3D CNN 네트워크

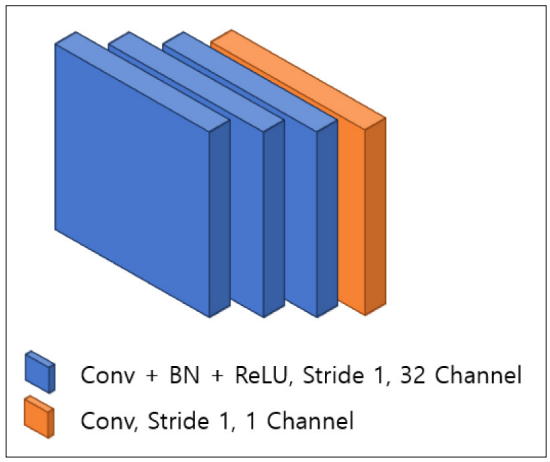
하여 Depth Map을 구한다. 이렇게 얻은 Depth Map은 Feature Extraction 후에 얻은 Feature Map과 같은 크기, 즉, $\frac{W}{4} \times \frac{H}{4}$를 가진다.

$$D = \sum_{d=d_{min}}^{d_{max}} d \times P(d) \tag{15}$$

D는 최종적으로 구한 Depth이며, d는 처음에 샘플링한 Depth 값이며, $P(d)$는 d가 실제 Depth일 확률이다. 이 과정을 soft argmin이라고 한다. Soft argmin을 통해 Depth를 추정하면 완벽하게 미분 가능하며, Depth가 Depth 범위에서 균일하게 샘플링되어도, 출력 Depth의 값은 연속값을 가지기 때문에 더 정확한 Depth를 추정할 수 있다.

이렇게 얻은 Depth Map은 꽤 품질이 좋지만, MVSNet의 너무 넓은 receptive field 때문에 Depth Map이 과하게 smoothing되는 문제가 존재한다. 이는 물체의 경계 부분의 Depth 품질을 떨어뜨리며, 딥러닝 기반의 Image Segmentation 등의 알고리즘에서 나타나는 문제와 같은 맥락이다. 따라서, Depth Map을 refinement하는 과정이 필요하다.

기존의 이미지에 물체의 경계 정보가 존재하므로, 처음에 입력한 기준 이미지를 refinement에 활용한다. Depth Residual을 학습하기 위해, 앞에서 얻은 초기 Depth Map



<그림 12> MVSNet의 Depth Refinement를 위한 CNN 네트워크

과 기준 이미지를 Concatenate하여 4채널 Input Image를 만든 후, 〈그림 12〉와 같은 간단한 2D CNN 네트워크를 통과시킨다. 2D CNN 네트워크는 32채널을 가지는 보폭 1의 2D CNN 레이어 3개와 마지막 보폭 1의 1채널 2D CNN 레이어로 구성되며 첫 3개의 레이어 뒤에는 배치 정규화와 ReLU 함수가 적용되지만, 마지막 레이어 뒤에는 아무것도 적용하지 않는다. Depth가 아닌 Depth Residual 학습이므로 음의 값도 필요하기 때문이다. 이렇게 얻은 출력 Depth Residual Map을 초기 Depth Map과 더하여 최종 Depth Map을 얻는다.

MVSNet의 손실 함수는 식 (16)와 같다. 손실 함수에 최종 Depth Map과 정답 Depth Map 사이의 손실뿐만 아니라, 초기 Depth Map과 정답 Depth Map 사이의 손실도 반영된다.

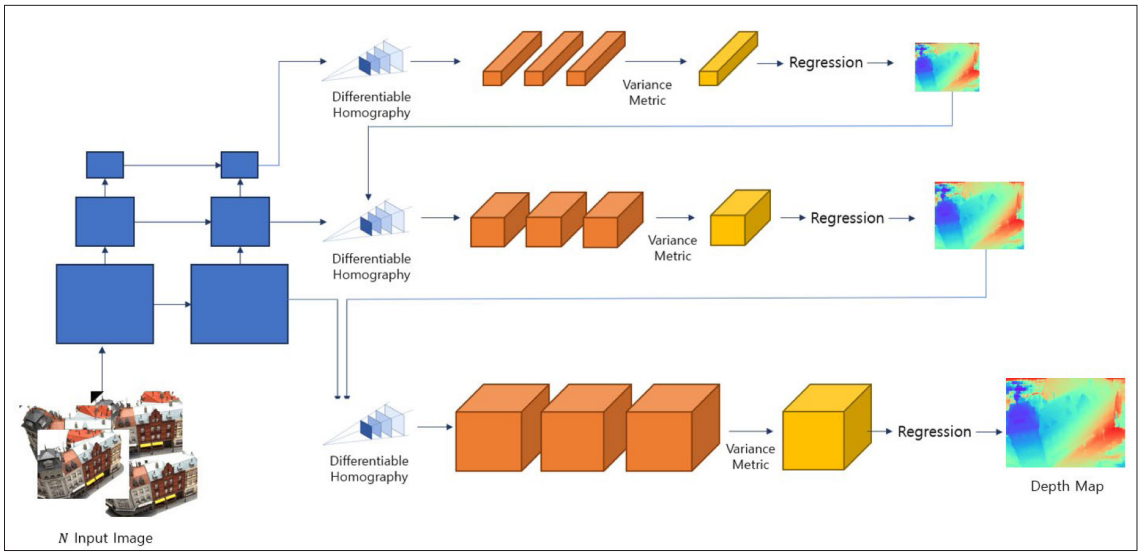
$$Loss = \sum_{p \in P_{valid}} \left\| d(p) - \hat{d}_i(p) \right\|_1 + \lambda \left\| d(p) - \hat{d}_r(p) \right\|_1 \quad (16)$$

$d(p)$는 픽셀 p의 정답 Depth 값, $\hat{d}_i(p)$는 초기 Depth Map에서 추정한 Depth 값, $\hat{d}_r(p)$는 최종 Depth Map에서 얻은 Depth 값, λ는 두 항의 비율을 조절하는 가중치 상수로 기본값은 1.0이며, 손실은 L1 Norm으로 계산한다. 어떤 픽셀에 대해서는 정답 Depth 값이 없는 경우도 존재할 수 있는데, 그런 경우는 제외하고, 정답 Depth 값이 있는 픽셀에 대해서만 손실을 계산한다.

MVSNet은 반복적인 무늬가 있는 지역이나 특징점이 적은 textureless region에서도 Depth를 잘 추정하여 전통적인 PatchMatch 기반의 알고리즘의 단점을 극복하였지만, 3D Cost Volume과 3D 컨볼루션 연산이 상당한 메모리와 계산량을 요구하기 때문에, 고해상도의 Depth Map을 직접적으로 얻기 어렵다는 단점이 있다.

2. Cascade Cost Volume for Multi-View Stereo

상당한 메모리와 계산량을 요구하는 MVSNet의 단점을 개선하기 위해, Cascade Cost Volume[13]이 제안되었다. 단일 크기의 Feature Map을 이용하는 3D Cost Volume과 다르게, Cascade Cost Volume은 CNN 네트워크를 통해 다양한 크기의 Feature Map으로 이루어진 Feature Pyramid를 활용하여 처음에는 대략적인 Depth만 추정



<그림 13> Cascade Cost Volume의 구조

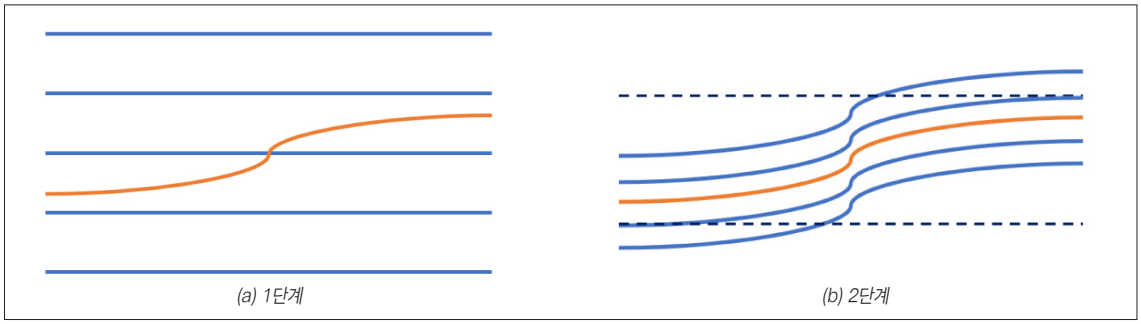
하고, 이후 단계에서 앞에서 구한 Depth 정보를 바탕으로 Depth를 정교하게 구하는 과정을 수행한다. 이번 장에서는 3D Cost Volume을 보완하는 Cascade Cost Volume의 구조를 설명하며 이를 위해 MVSNet을 Backbone Network로 사용한다.

Cascade Cost Volume의 전체적인 구조는 〈그림 13〉과 같다. Cascade Cost Volume의 주요 아이디어는 단일 Feature Map이 아닌, Multi-Scale의 Feature Pyramid를 활용하는 것이다. 2D CNN 네트워크를 통해 Multi-Scale의 Feature Map을 얻고, 작은 Feature Map을 사용하여 Cost Volume을 만들어 대략적인 Depth를 추정한다. 그 다음, 한 단계 더 큰 Feature Map과 이전 단계에서 추정한 Depth 정보를 활용하여 더 정확한 Depth를 추정하며, 이 과정을 Multi-Scale의 Feature Map 수만큼 반복한다. 이때, Depth를 추정하는 방법은 Backbone Network의 방법을 따른다.

Cascade Cost Volume의 가장 첫 부분은 기존의 Cost Volume과 마찬가지로, 2D CNN 네트워크로 Feature Map을 얻는 것이다. 하지만, 기존의 Cost Volume과 다르게 Cascade Cost Volume은 Feature Pyramid Network [16]을 사용하여 Multi-Scale의 Feature Map으로 이루어진 Feature Pyramid를 얻는다. Feature Pyramid가 *N*개의 Scale을 가졌다면, Cascade Cost Volume을 활용해 Depth Map을 얻는 과정은 *N*단계로 진행된다. *N*의 기본 값은 3이다.

1단계에서는 Feature Pyramid에서 가장 작은 크기를 가지는 Feature Map을 가져와 Depth Map을 추정한다. 1단계에서는 기존의 Cost Volume을 활용하는 것과 동일하게 Depth Map을 만든다. 기존의 Cost Volume을 활용한 MVS 딥러닝 모델은 여기서 Depth Map 추정이 종료되기 때문에 여기서 더 큰 Cost Volume을 만들어 더 정확한 Depth Map을 얻을 수 있도록 해야 하지만, Cascade Cost Volume 구조에서는 여기서 추정한 Depth Map을 바탕으로 이후 단계에서 더 정확한 Depth Map을 추정할 수 있으므로 비교적 더 작은 Cost Volume을 사용하여도 된다.

2단계부터 *N*단계까지는 이전 단계에서 추정한 Depth 정보를 활용하여, 이전 단계에서 얻은 Depth 근처에서 Depth 범위를 설정한다. 예를 들어, 〈그림 14〉(a)는 1단계에서의 Depth 샘플링을 통해 얻은 Depth Hypothesis를 파란 실선으로, Depth Map 추정 결과를 주황 실선으로 나타낸다. 〈그림 14〉(b)는 2단계에서의 Depth Hypothesis를 설명하는데, 점선은 1단계의 Depth Hypothesis을 나타내고, 주황 실선은 1단계에서의 Depth Map 추정 결과를 나타낸다. 2단계에서는 이를 기준으로 residual을 설정하여 Depth 범위를 새롭게 설정하고 샘플링하며, 〈그림 14〉(b)의 파란 실선이 이를 통해 얻은 2단계에서의 Depth Hypothesis이다. 이렇게 하면, 전체적인 Depth 범위가 이전 단계의 범위보다 줄어들며, Depth 샘플링 간격 역시 현저히 줄어들어 이전 단계보다



<그림 14> Cascade Cost Volume의 단계별 Depth 범위

더 정교한 Depth 추정이 가능해진다.

1단계에서는 기존의 Cost Volume과 동일하게, Feature Volume을 생성하기 위해 식 (13)을 활용하지만, 2단계부터는 이전 단계의 Depth의 residual을 활용하므로 식 (17)을 활용하여 Feature Volume을 생성한다.

$$H_i(d_k^m + \Delta_{k+1}^m) = K_i R_i (I - \frac{(t_1 - t_i)n_1^T}{d_k^m + \Delta_{k+1}^m}) R_1^T K_1^T \qquad (17)$$

d_k^m은 k단계에서 구한 픽셀 m의 Depth이며 Δ_{k+1}^m은 k+1 단계에서 픽셀 m에 대해 설정한 residual이다.

이후 Cost Volume을 만들기 위해 variance-based Cost Metric을 사용하는 것은 이전과 동일하며, 각 단계에서 Cost Volume Regulariazation이나 soft argmin, Depth Map Refinement 등을 활용하여 Depth Map을 구하는 것은 기존 MVSNet과 동일하다. 즉, Cost Volume으로부터 Depth Map을 얻는 과정은 Backbone Network의 방법을 그대로 따른다. 또, 손실 함수는 식 (18)과 같이 정의되며, 손실 함수 역시 Backbone Network의 것을 활용하는데, 단계별로 얻은 Depth Map에 대해 손실을 구하고, 이를 가중치 합하여 최종 손실을 구한다.

$$Loss = \sum_{k=1}^{N} \lambda^k \cdot L^k \qquad (18)$$

L^k는 k단계의 Depth Map과 정답 Depth Map을 비교하여 Backbone Network의 손실 함수를 사용하여 구한 손실이며, λ^k는 각 단계별 손실에 대한 가중치이다.

기존의 Cost Volume을 활용한 MVSNet에서는 더 정확한 Depth를 얻기 위해서는 더 큰 Cost Volume이 필요하므로 더 많은 메모리가 필요했지만, Cascade Cost Volume 알고리즘을 사용하면, Depth 범위 [λ_{min},λ_{max}]을 비교적 덜 촘촘하게 샘플링하여 더 작은 Cost Volume을 만들더라도 여러 단계를 거쳐 추가적인 Depth 회귀를 진행하므로 더 적은 메모리를 사용하고도 더 정확한 Depth Map을 얻을 수 있다.

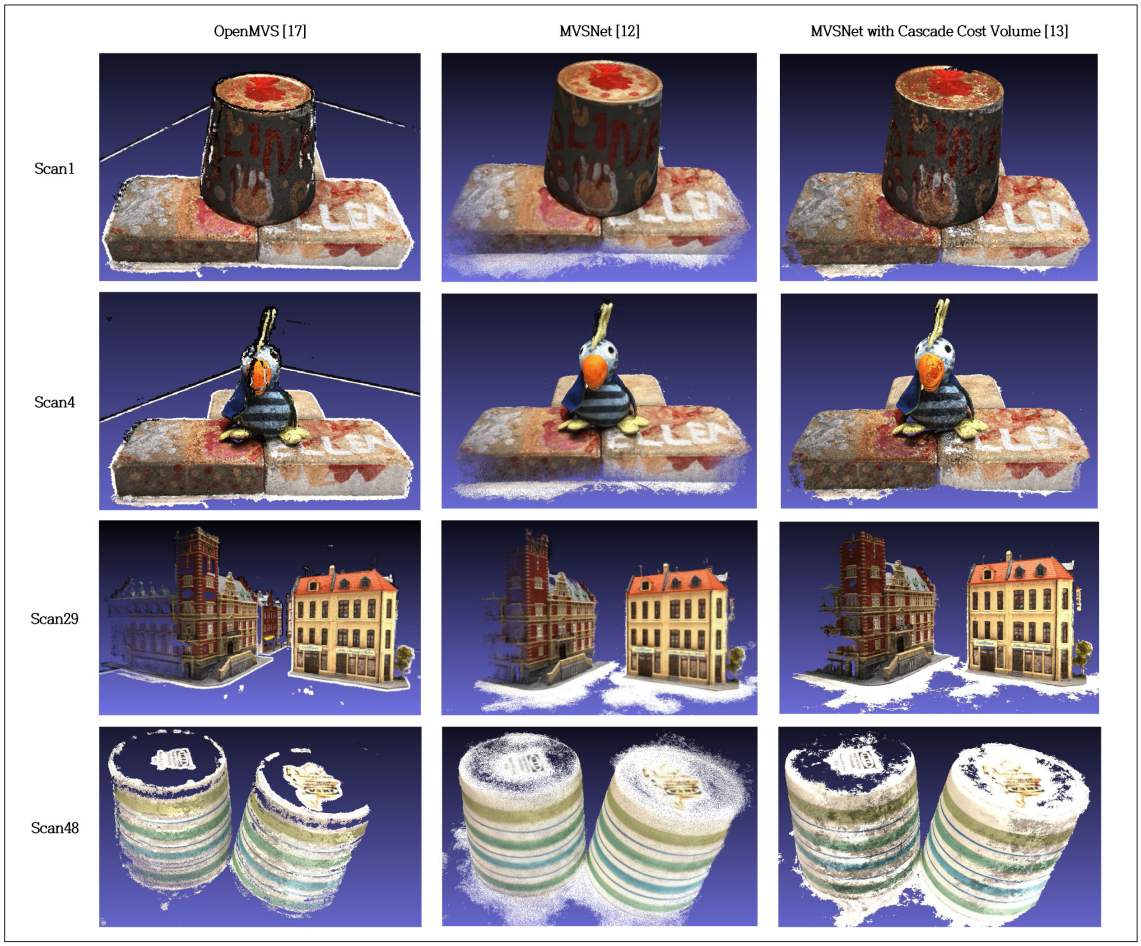
Ⅳ. 성능 비교

2장과 3장에서 알아본 전통적인 PatchMatch 알고리즘과 MVSNet 그리고 Cascade Cost Volume을 적용한 MVSNet의 성능을 비교하기 위해, 각 알고리즘을 DTU Dataset[9]에 적용하여 〈그림 15〉와 같이 Dense Point Cloud를 만들었다. PatchMatch 알고리즘의 경우 대표적인 라이브러리인 OpenMVS[17]을 사용하였다.

PatchMatch 알고리즘의 경우, Scan29 데이터에서 딥러닝 기반의 알고리즘보다 더 많은 객체를 복원했지만, Scan29와 Scan48에서 특징점이 적은 textureless region에서 Depth를 잘 추정하지 못하는 것을 관찰할 수 있었으며, Scan1과 Scan4 데이터에서는 전통적인 알고리즘을 사용하였을 때, 더 많은 노이즈가 생성되는 것을 확인할 수 있었다. Scan1과 Scan4에서 나타난 노이즈는 PatchMatch 알고리즘이 입력 이미지의 배경인 검은색 픽셀 혹은 하얀색 픽셀에 대해 Depth를 추정한 것이 원인이다.

MVSNet의 경우 기존의 Cost Volume을 활용한 모델은 Depth Sampling을 192개로, Cascade Cost Volume의 경우 1단계에서는 48개, 2단계에서는 32개, 3단계에서는 8개로 설정했다. 기존의 MVSNet과 Cascade Cost Volume을 활용한 MVSNet 모두 Scan1과 Scan4 데이터셋에서 노이즈가 거의 없으며, Scan48의 textureless region에서도 PatchMatch 알고리즘에 비해 비교적 객체를 잘 복원했다. 기존의 MVSNet의 경우 PatchMatch 알고리즘이나 Cascade Cost Volume을 활용한 경우보다 선명도가 더 떨어졌는데, 이는 기존의 Cost Volume의 메모리 사용량을 줄이기 위해 저해상도의 Depth Map을 출력했기 때문이다. 이 문제를 해결한 Cascade Cost Volume은 더 선명한 Dense Point Cloud를 생성했다. Dense Point Cloud를 이루는 3D Point 역시 Cascade Cost Volume을 활용한 경우 각 데이터마다 1500만 개에서 3000만 개를 생성한 반면, 기존의 Cost Volume을 활용한 경우 200만 개 이하의 3D Point를 생성했다.

실행 시간에 있어서도 각 알고리즘별로 큰 차이를 보였는데, PatchMatch 알고리즘의 경우 이미지 한 장의 Depth Map을 얻는데 약 9초가 소요되었고, 기존의 MVSNet의 경우 이미지 한 장의 Depth Map을 얻는데 약 11초가 소요되었으며, Cascade Cost Volume을 활용한 MVSNet의 경우 이미지 한 장의 Depth Map을 얻는데 약 0.4초가 소요되었다. 기존의 Cost Volume을 활용할 경우 Depth Sampling의 수가 많아져 Cost Volume이 커지고, 이에 따라 3D 컨볼루션 연산을 진행할 때, 계산량이 상당히 커지는 반면, Cascade Cost Volume을 활용할 경우, Depth Sampling을 작게 설정하여 여러 단계를 걸쳐 Depth를 추정하더라도, 3D 컨볼루션 연산이 적게 소요되어 계산량이 상당히 줄어들기 때문에, 기존의 MVSNet과 Cascade Cost Volume을 활용한 MVSNet의 실행 시간에 큰 차이가 생긴다.



<그림 15> DTU Dataset [9]을 사용하여 OpenMVS와 MVSNet, Cascade Cost Volume을 적용한 MVSNet으로 제작한 Dense Point Cloud 영상

참 고 문 헌

[1] J. L. Schönberger and J. -M. Frahm, "Structure-from-Motion Revisited," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 4104-4113, doi: 10.1109/CVPR.2016.445.

[2] S. Shen, "Accurate Multiple View 3D Reconstruction Using Patch-Based Stereo for Large-Scale Scenes," in IEEE Transactions on Image Processing, vol. 22, no. 5, pp. 1901-1914, May 2013, doi: 10.1109/TIP.2013.2237921.

[3] Wang, Yuesong, Zhaojie Zeng, Tao Guan, Wei Yang, Zhuo Chen, Wenkai Liu, Luoyuan Xu, and Yawei Luo. "Adaptive Patch Deformation for Textureless-Resilient Multi-View Stereo." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 1621-1630. 2023.

[4] Xu, Qingshan, and Wenbing Tao. "Planar prior assisted patchmatch multi-view stereo." In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, no. 07, pp. 12516-12523. 2020.

[5] Li, Zhaoxin, Wangmeng Zuo, Zhaoqi Wang, and Lei Zhang. "Confidence-based large-scale dense multi-view stereo." IEEE Transactions on Image Processing 29 (2020): 7176-7191.

[6] Zhang, Jingyang, Yao Yao, Shiwei Li, Zixin Luo, and Tian Fang. "Visibility-aware multi-view stereo network." arXiv preprint arXiv:2008.07928 (2020).

[7] Yu, Zehao, and Shenghua Gao. "Fast-mvsnet: Sparse-to-dense multi-view stereo with learned propagation and gauss-newton refinement." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 1949-1958. 2020.

[8] Wang, Shaoqian, Bo Li, and Yuchao Dai. "Efficient multi-view stereo by iterative dynamic cost volume." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 8655-8664. 2022.

[9] Aanæs, Henrik, Rasmus Ramsbøl Jensen, George Vogiatzis, Engin Tola, and Anders Bjorholm Dahl. "Large-scale data for multiple-view stereopsis." International Journal of Computer Vision 120 (2016): 153-168.

[10] Jancosek, Michal, and Tomas Pajdla. "Exploiting visibility information in surface reconstruction to preserve weakly supported surfaces." International scholarly research notices 2014 (2014).

[11] Waechter, Michael, Nils Moehrle, and Michael Goesele. "Let there be color! Large-scale texturing of 3D reconstructions." In Computer Vision-ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13, pp. 836-850. Springer International Publishing, 2014.

[12] Yao, Yao, Zixin Luo, Shiwei Li, Tian Fang, and Long Quan. "Mvsnet: Depth inference for unstructured multi-view stereo." In Proceedings of the European conference on computer vision (ECCV), pp. 767-783. 2018.

[13] Gu, Xiaodong, Zhiwen Fan, Siyu Zhu, Zuozhuo Dai, Feitong Tan, and Ping Tan. "Cascade cost volume for high-resolution multi-view stereo and stereo matching." In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 2495-2504. 2020.

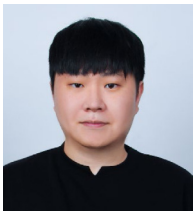
[14] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." In International conference on machine learning, pp. 448-456. pmlr, 2015.

[15] Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." In Medical Image Computing and Computer-Assisted Intervention-MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18, pp. 234-241. Springer International Publishing, 2015.

[16] Lin, Tsung-Yi, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. "Feature pyramid networks for object detection." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2117-2125. 2017.

[17] OpenMVS. Accessed: 2019. [Online]. Avaliable : https://github.com/cdcseacave/openMVS

저 자 소 개



김 성 민

- 2017년 3월 ~ 현재 : 세종대학교 전자정보통신공학과 학부
- 주관심분야 : AI, VR 영상 제작, Computer Vision

한 종 기

- 1992년 : KAIST 전기및전자공학과 공학사
- 1994년 : KAIST 전기및전자공학과 공학석사
- 1999년 : KAIST 전기및전자공학과 공학박사
- 1999년 3월 ~ 2001년 8월 : 삼성전자 DM연구소 책임연구원
- 2001년 9월 ~ 현재 : 세종대학교 전자정보통신공학과 교수
- 2008년 9월 ~ 2009년 8월 : University California San Diego (UCSD) Visiting Scholar
- ORCID : https://orcid.org/0000-0002-5036-7199
- 주관심분야 : VR, 3D 영상신호 구현, 비디오 코덱, 영상 신호처리, 정보 압축, 방송 시스템