Morpheus3d

# light stage 캡처를 통한 렌더링 리소스 조정

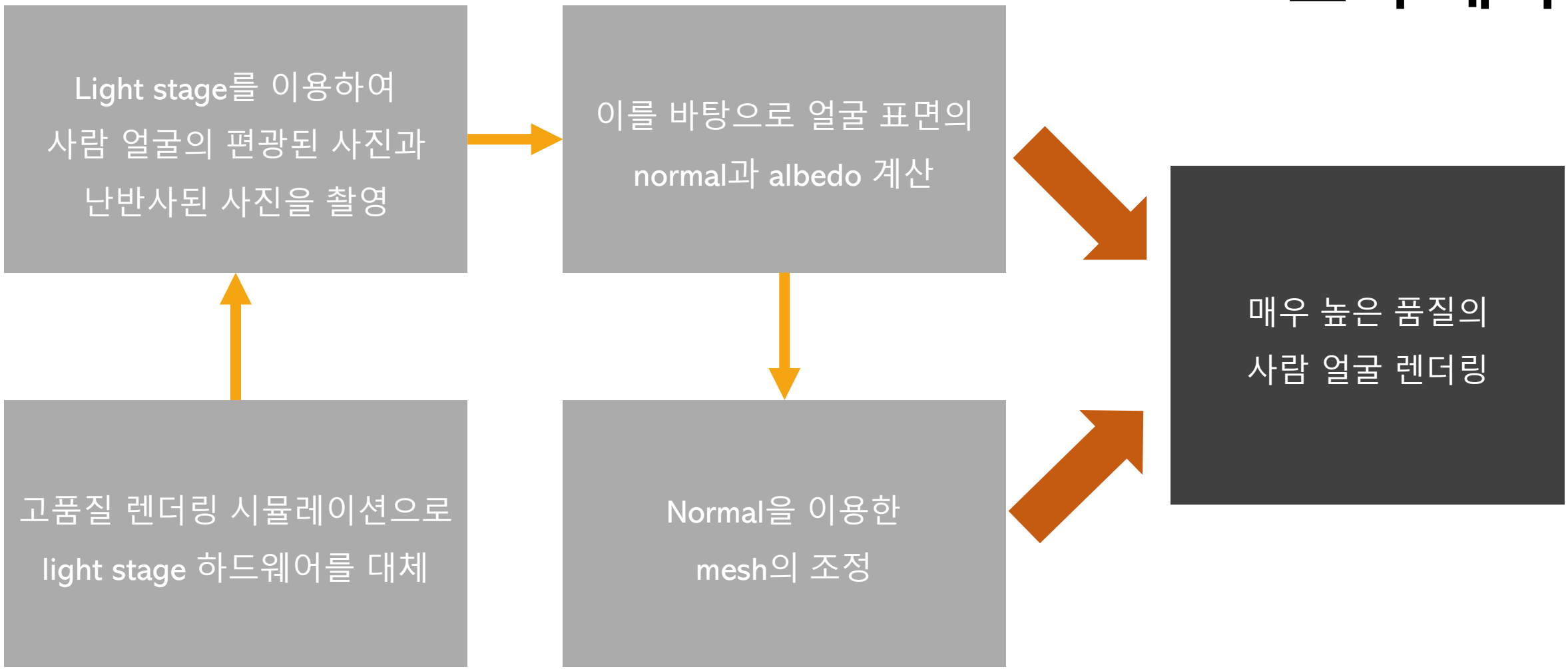

팀원

정유석
이준혁
이동학

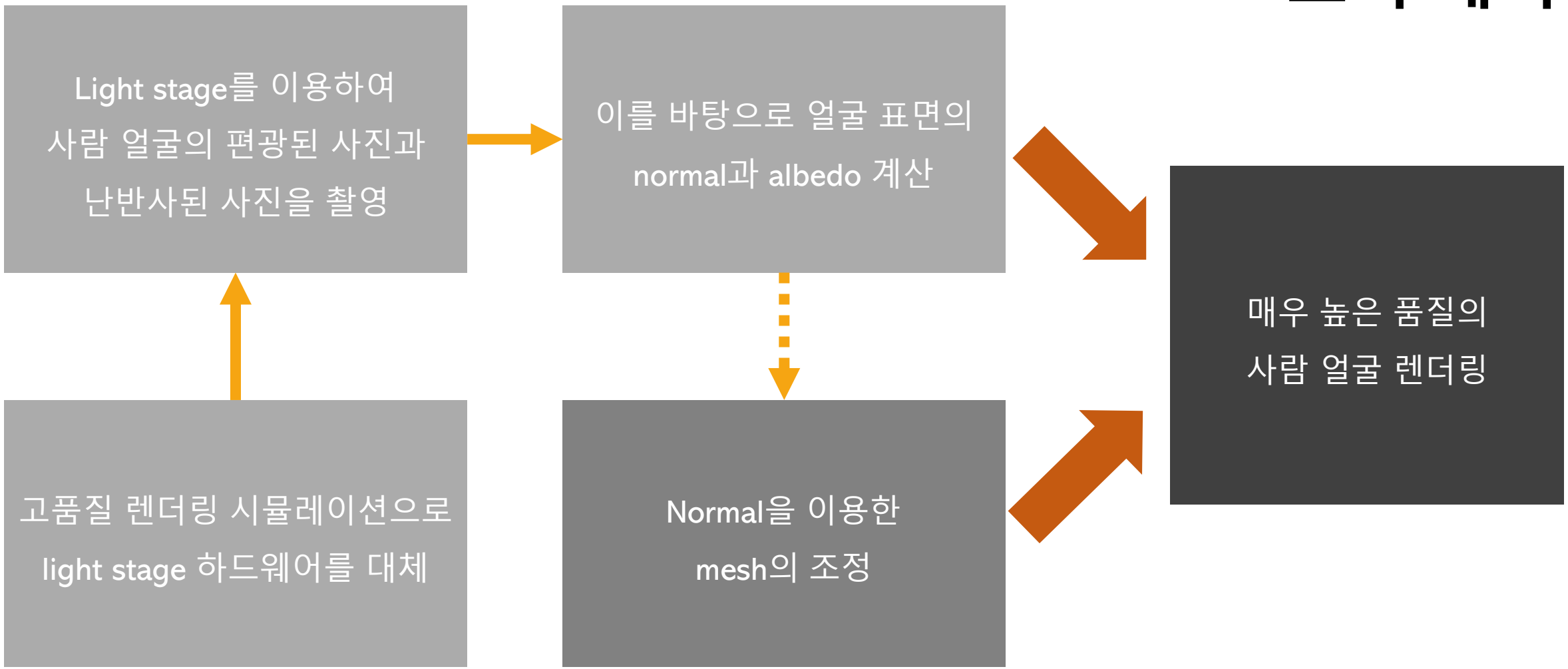

# INDEX

# 프로젝트 개요

# 초기 계획


Light stage를 이용하여
사람 얼굴의 편광된 사진과
난반사된 사진을 촬영
이를 바탕으로 얼굴 표면의
normal과 albedo 계산
매우 높은 품질의
사람 얼굴 렌더링
고품질 렌더링 시뮬레이션으로
light stage 하드웨어를 대체
Normal을 이용한
mesh의 조정

# 초기 계획


Light stage를 이용하여
사람 얼굴의 편광된 사진과
난반사된 사진을 촬영
이를 바탕으로 얼굴 표면의
normal과 albedo 계산
매우 높은 품질의
사람 얼굴 렌더링
고품질 렌더링 시뮬레이션으로
light stage 하드웨어를 대체
Normal을 이용한
mesh의 조정

# 변경된 계획

Normal 과 albedo 계산

Normal을 이용한
displacement map 구성

Light stage를 이용하여
사람 얼굴의 편광된 사진과
난반사된 사진을 촬영

고품질 렌더링 시뮬레이션으로
light stage 하드웨어를 대체

텍스처를 이용한
displacement map 구성

매우 높은 품질의
사람 얼굴 렌더링


Normal map

# 렌더링을 통한 하드웨어의 시뮬레이션

- 실제 하드웨어의 경우, 편광 필름을 통과한 빛의 세기가 최대일 때 한 번, 최소일 때 한 번 촬영한다.
- 난반사된 빛의 경우 편광 필름을 통과할 때 균일하게 그 세기가 1/2로 줄어든다
- 정반사된 빛의 경우 편광 필름을 통과함에 따라 온전히, 혹은 전혀 통과하지 못한다
- 셰이딩 단계에서 이를 시뮬레이션 하여 실제 하드웨어와 유사한 사진을 얻어냈다

$I_S + \frac{1}{2} I_D$ : 정반사된 빛이 최고 세기

$\frac{1}{2} I_D$ : 정반사된 빛이 최소 세기

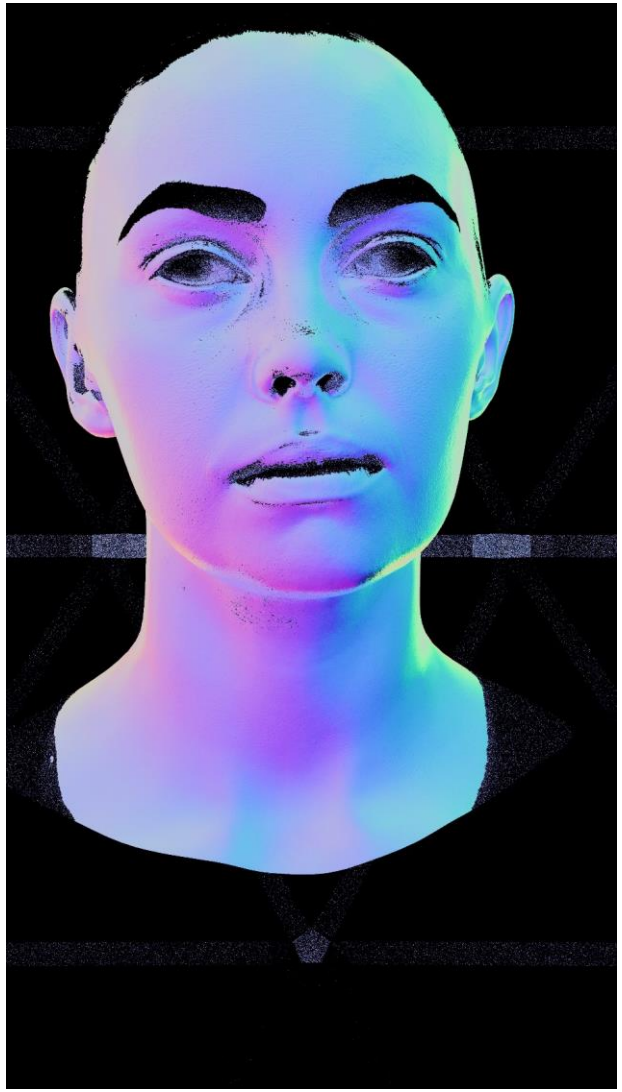
# Normal과 Albedo의 계산

- 광원의 위치에 dependent하게 적절한 광원 세기를 준 뒤 촬영한 사진을 바탕으로, 각 지점의 normal과 albedo를 계산할 수 있다
- 광원의 세기 분포를 설정하는 방법이 네 가지, 편광된 빛의 세가가 최소 최대로 두 가지: 총 8장의 사진을 바탕으로 계산한다
- 참고문헌에서 제시한 계산식을 openCV를 이용하여 구현
- 실시간으로 작동하는 시스템을 목표로 하기 때문에 C++를 통해 최적화

# light stage 캡처를 통한 렌더링 리소스 조정




난반사 항을 통해 계산한 normal

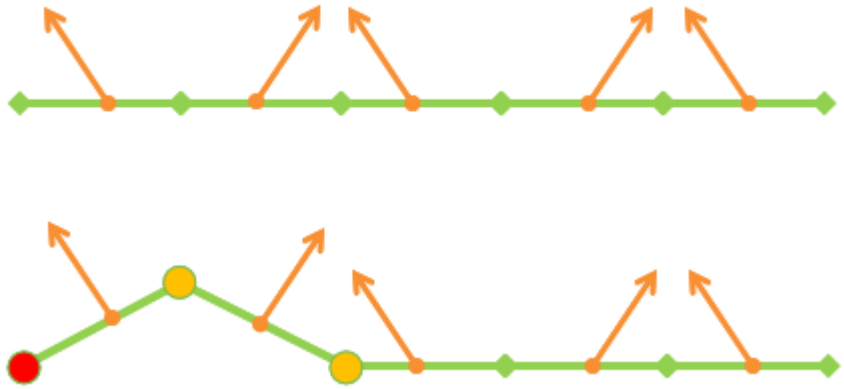


정반사 항을 통해 계산한 normal
(추후에 추가 개선)

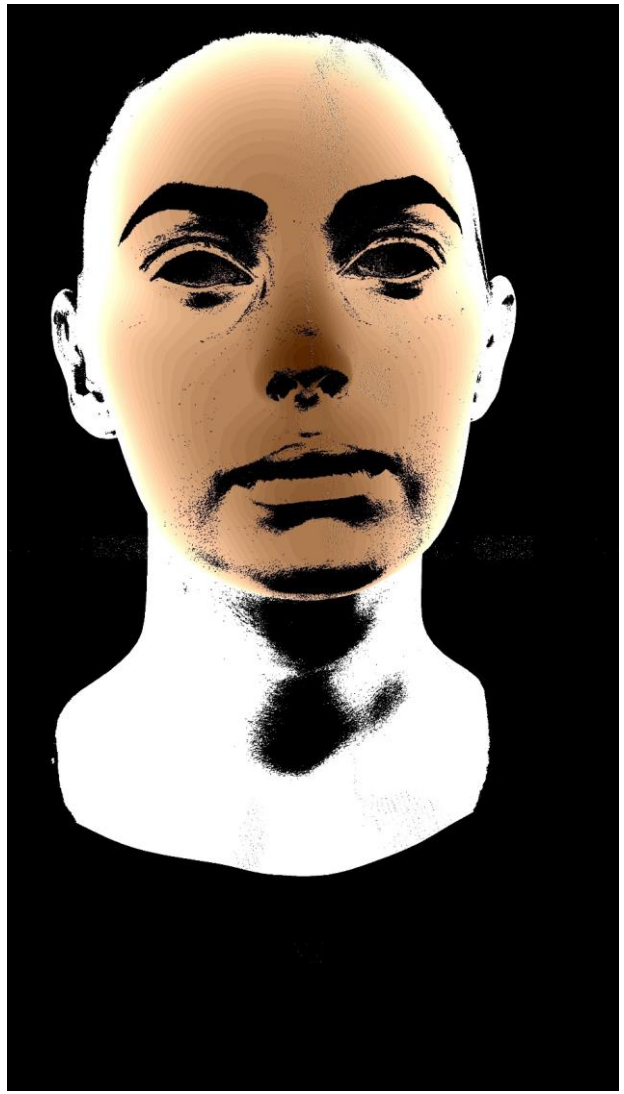

Displacement map

# Displacement map from normal map

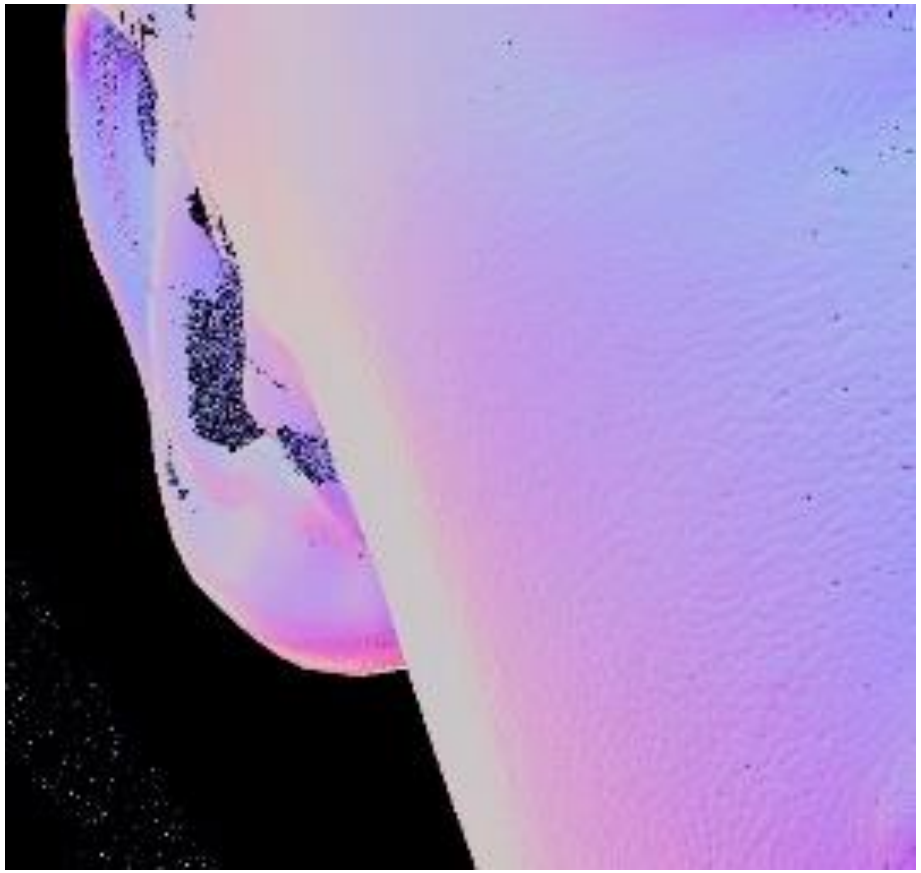
- displacement map: 근처 pixel과의 normal 차이에 의존

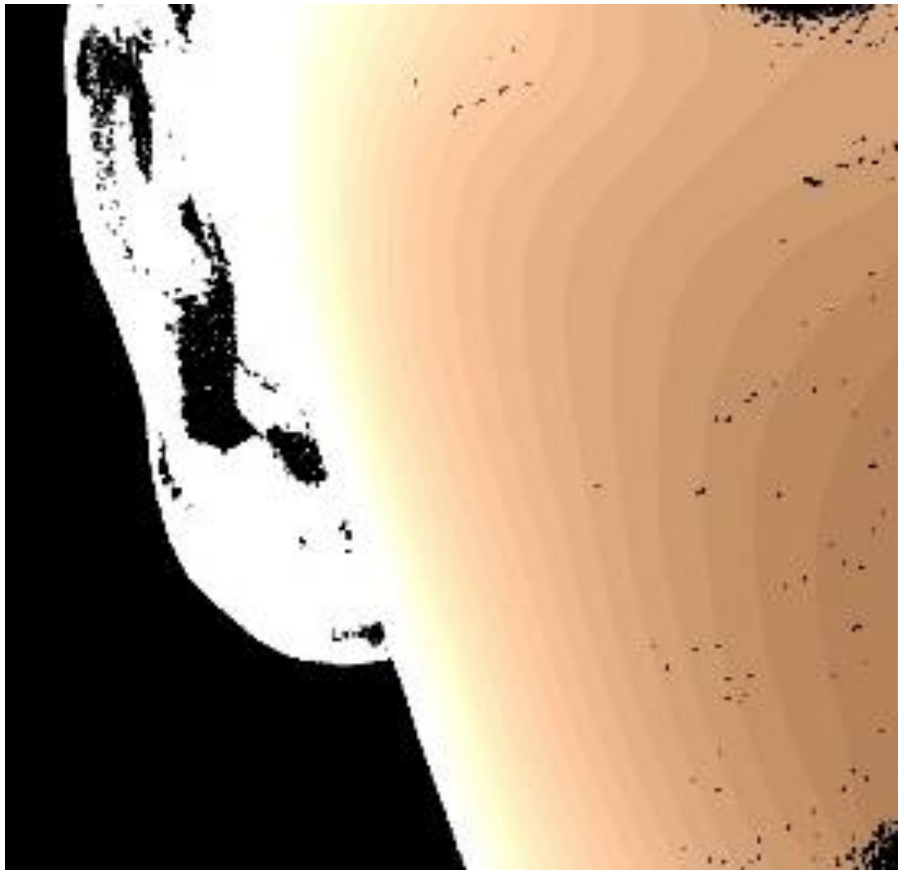


- 초기 normal / dismap (성능이 좋지 못함)



- 초기 normal / dismap(성능이 좋지 못함)





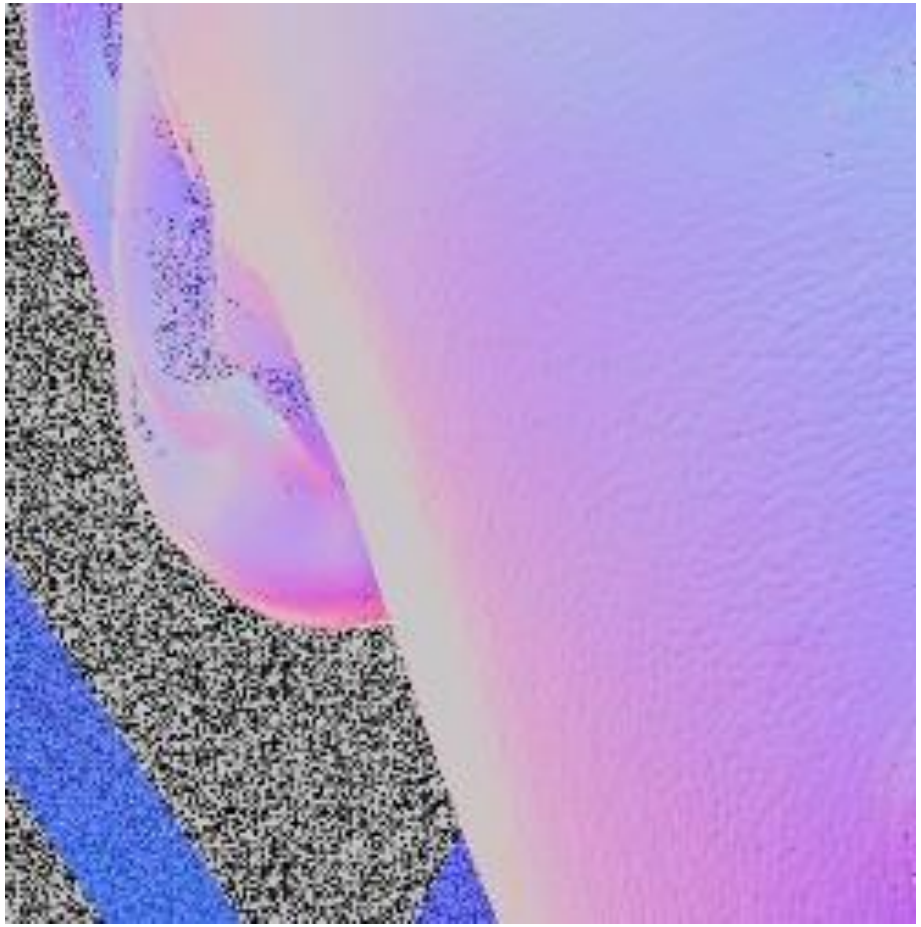
# Zero-point Error 발생

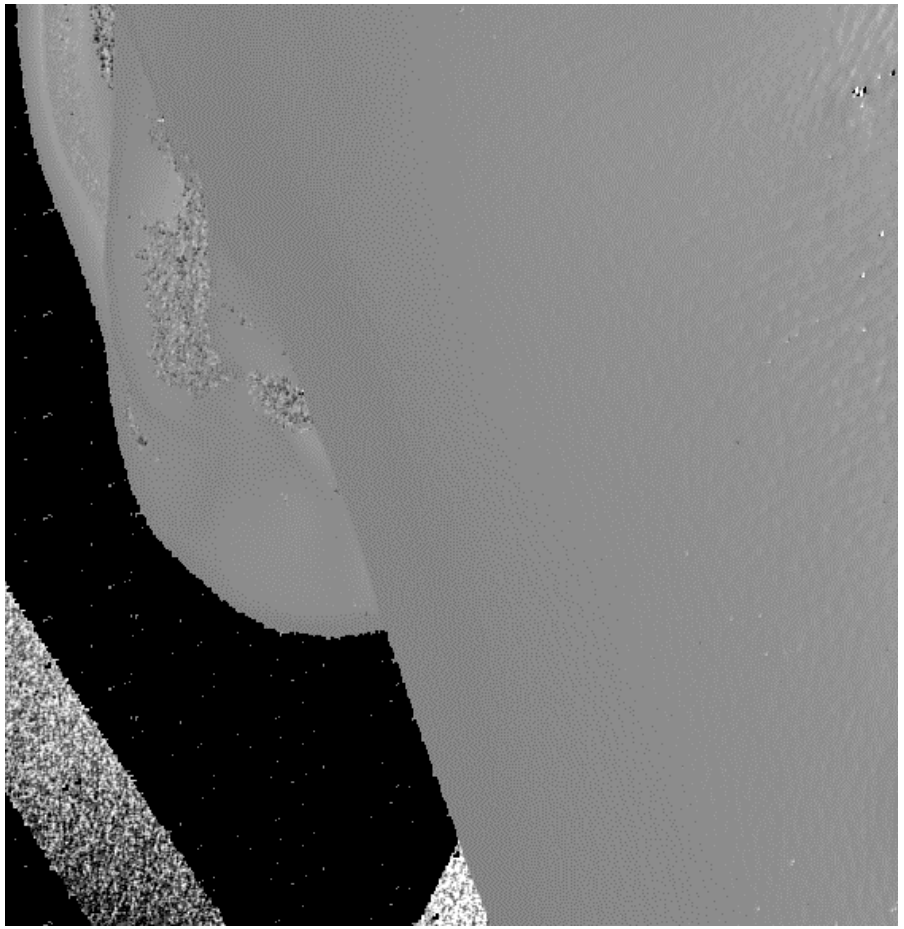
- 원인: image data가 RGB 0~255로 이산화 → 0으로 나누어지는 point 발생
- 주변 데이터를 이용해 계산 시도 → 개선에 도움되지 않음
- Why Critical?
  - 한 pixel의 error가 주변으로 전파됨 → 반드시 error reduce가 필요
- normal from diffuse 결과 사용

- Normal map - error reduce version

- 후기 normal map / dismap

- 후기 normal map / dismap





# 텍스처를 이용한 Displacement Mapping

- Normal을 이용한 향후 task들과는 별개로, 모델의 텍스처를 이용해 displacement mapping을 구성하고 렌더링 품질을 높이는 방안
  ➔ 이동학, 이준혁이 담당하여 진행

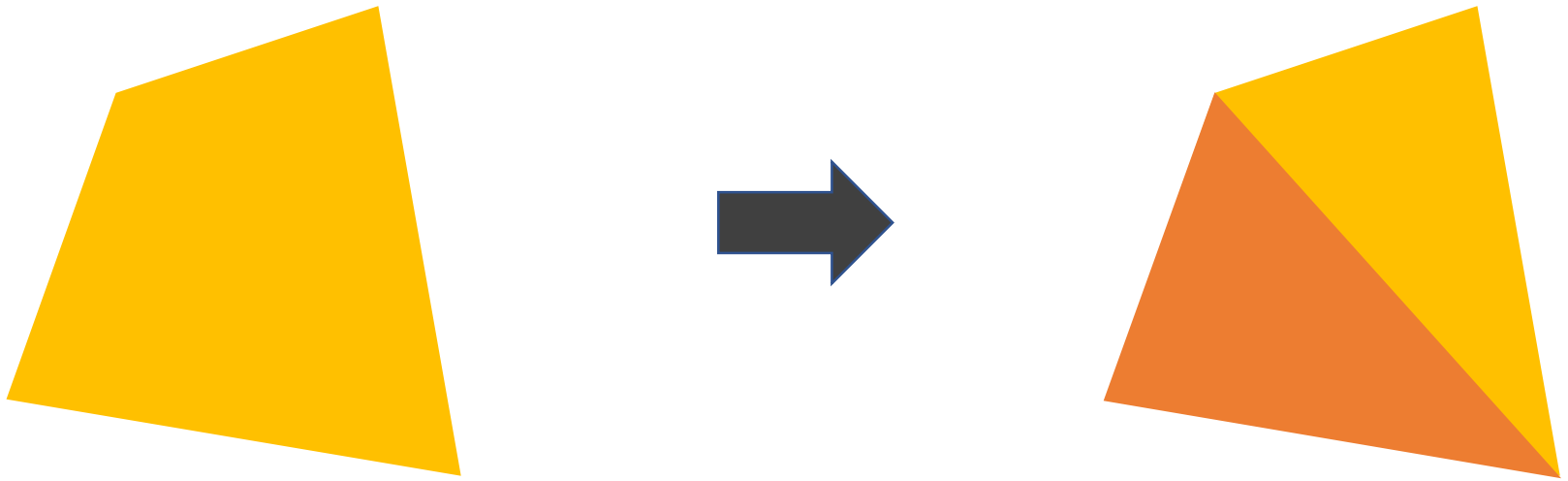
- 이를 구현해 보고, 실제 시스템에 포함시킬 수 있을지를 고려

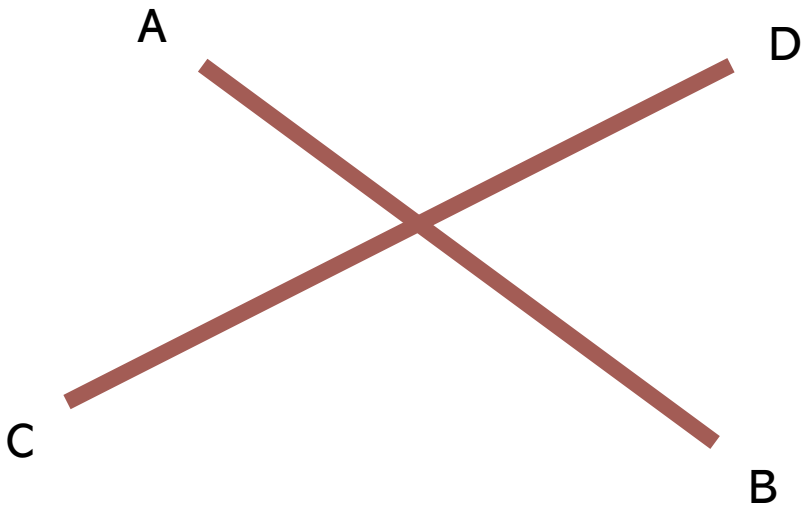
# mesh 분할하기

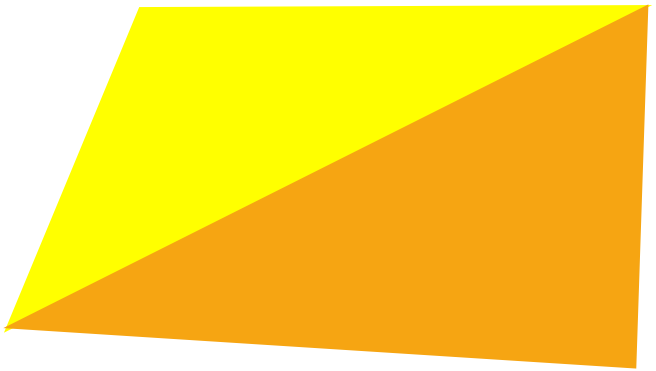
- Emily mesh: 준비된 것은 사각형 위주의 데이터

- 삼각형이 계산하기 편리

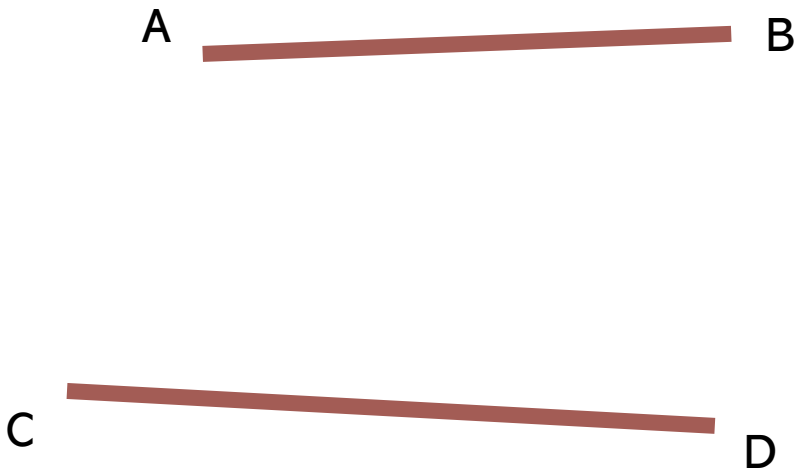
➔ 사각형 mesh를 삼각형 mesh로 변환하는 작업

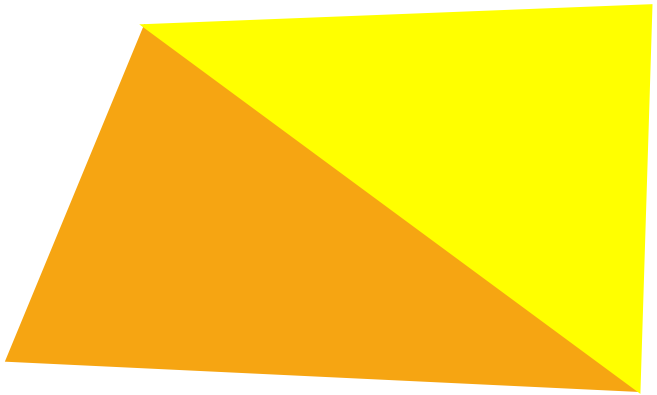




선분 AB와 CD가 교차할 경우 삼각형 ACD, BCD

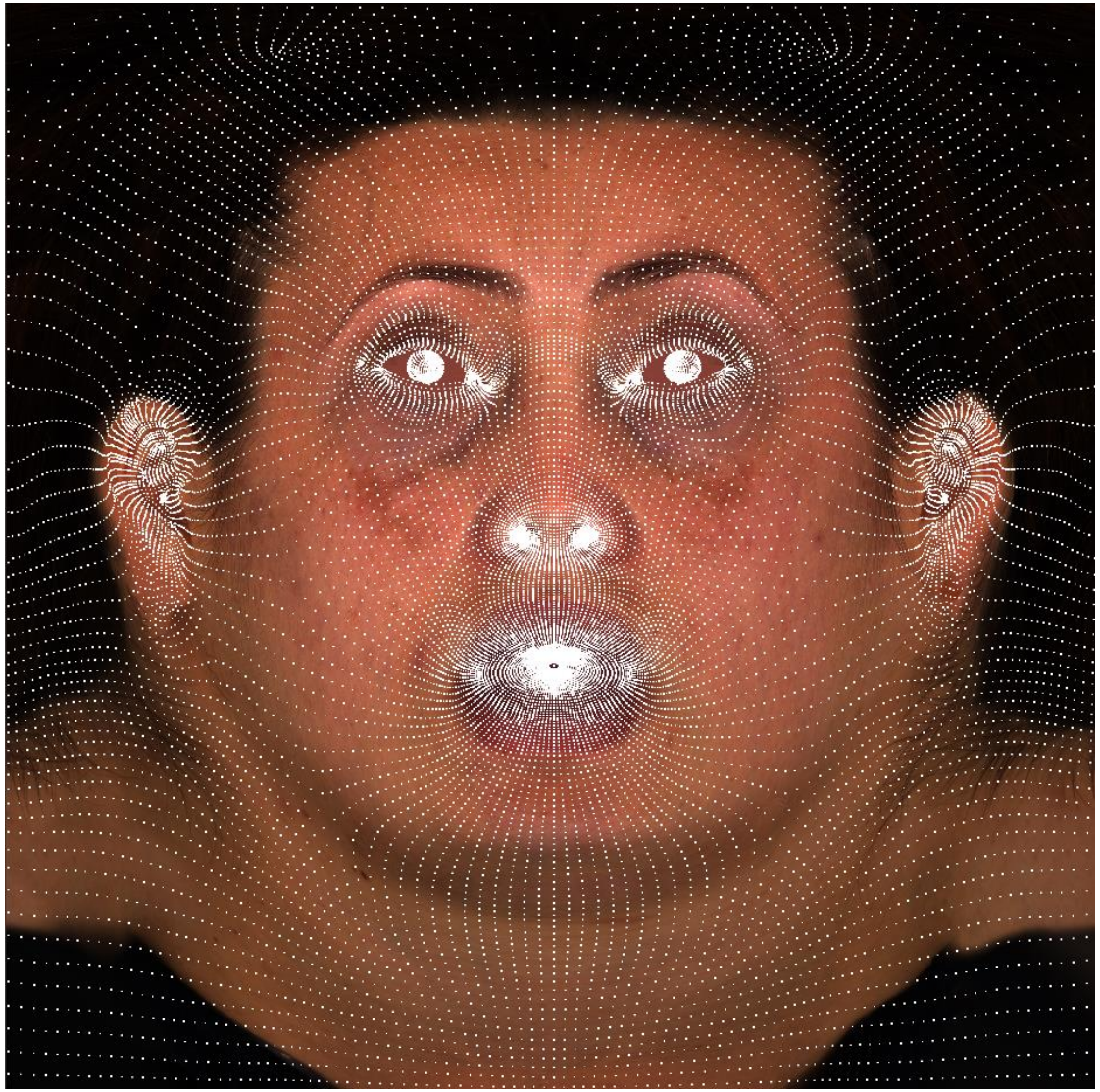


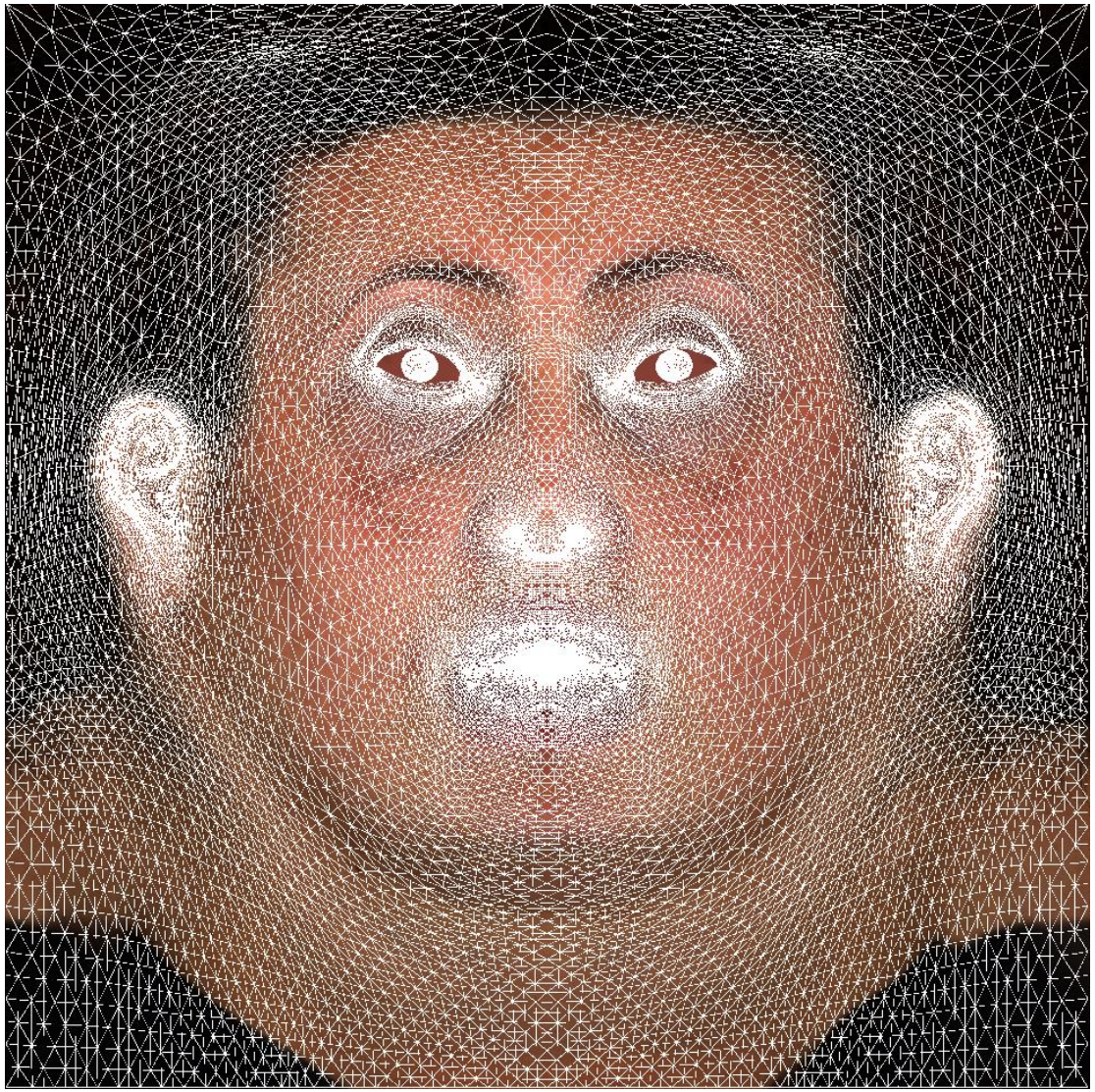



교차하지 않을 경우 삼각형 ABD, ACD

# Displacement map

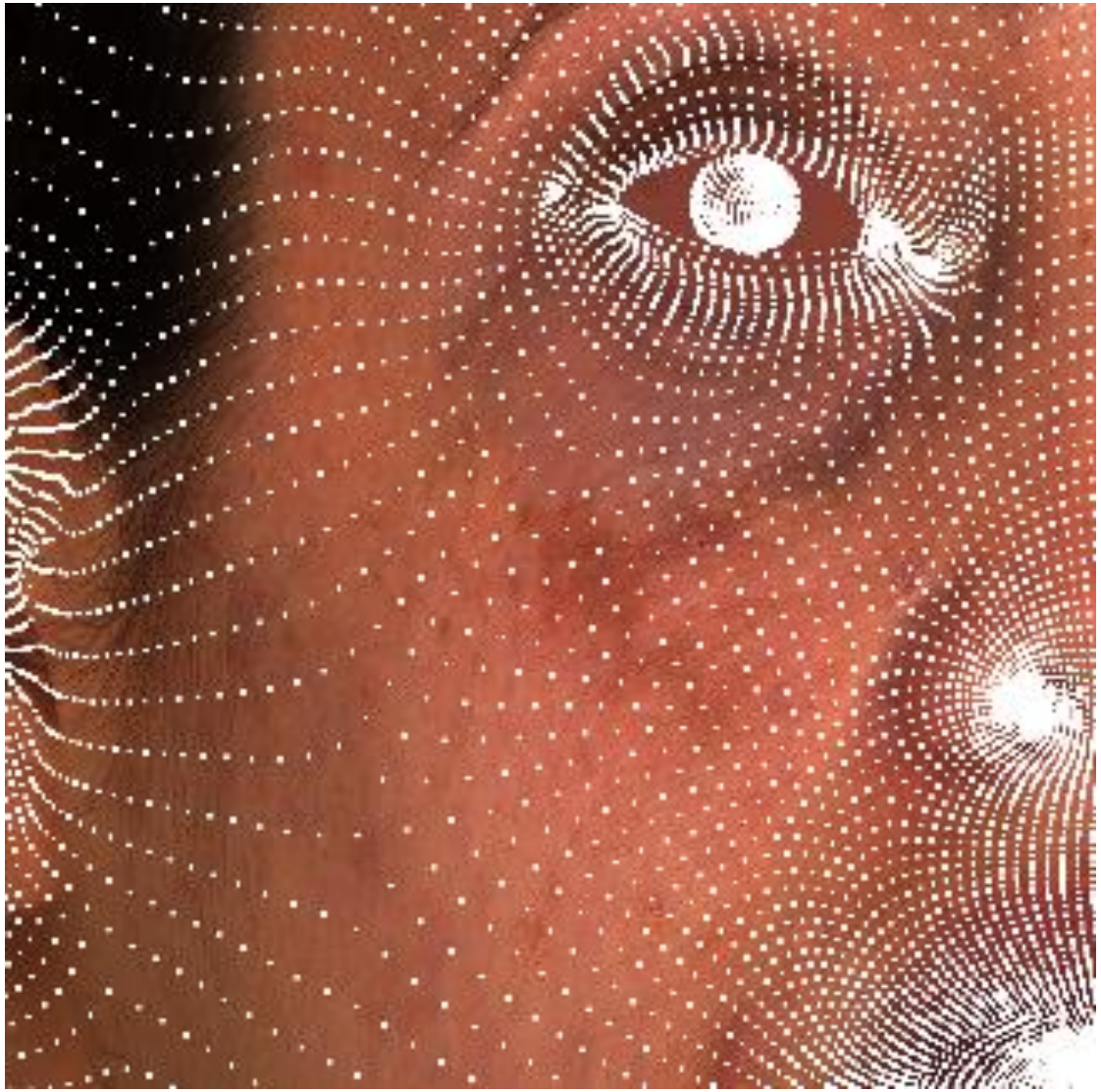
# light stage 캡처를 통한 렌더링 리소스 조정





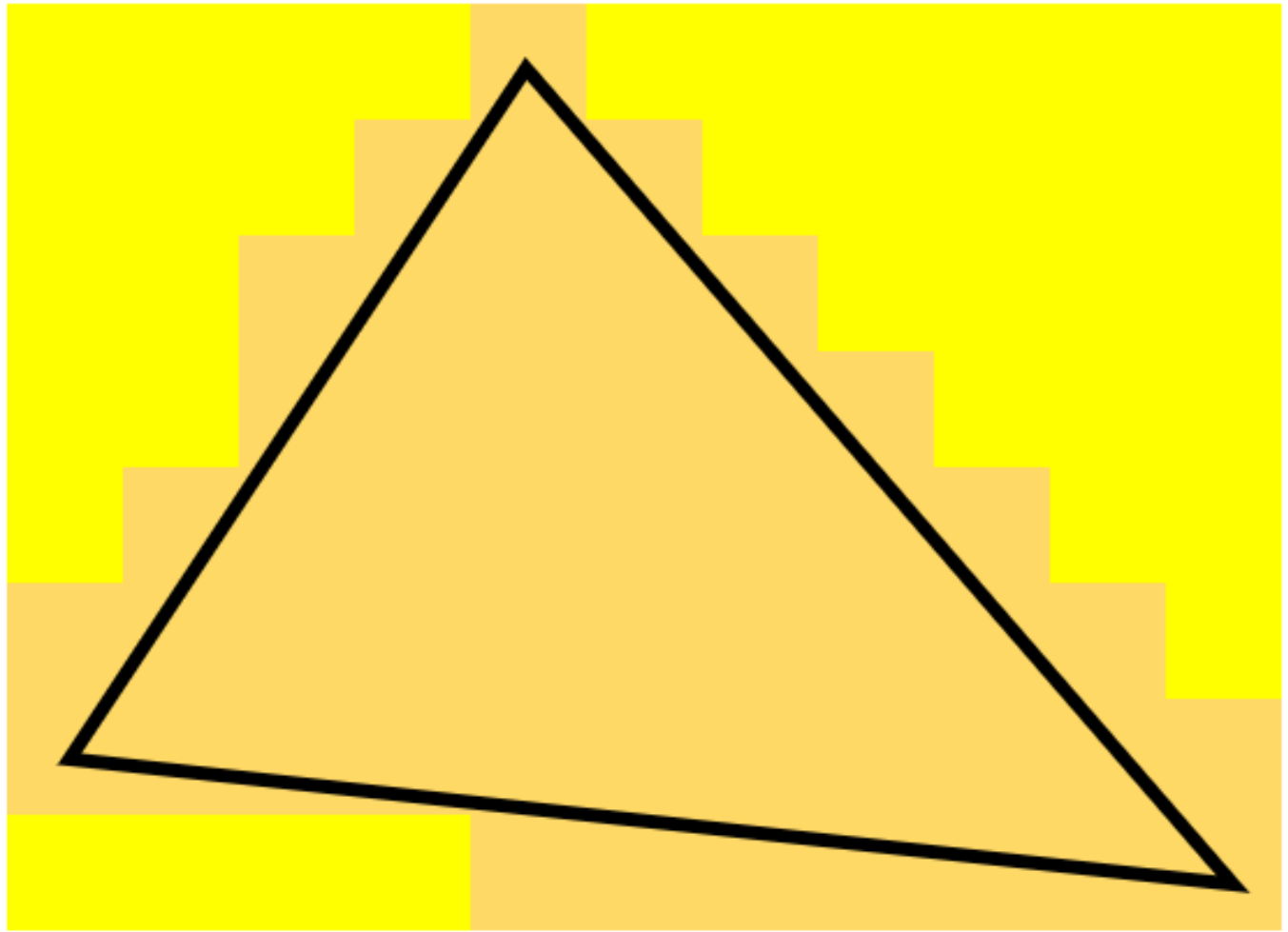
# Displacement map

# light stage 캡처를 통한 렌더링 리소스 조정



# Geometry 구하기

- texture 상의 geometry를 이용해 displacement map 계산
- polygon → texture: mesh에 명시되어 있다
- texture → polygon: 탐색 필요 (CCW)

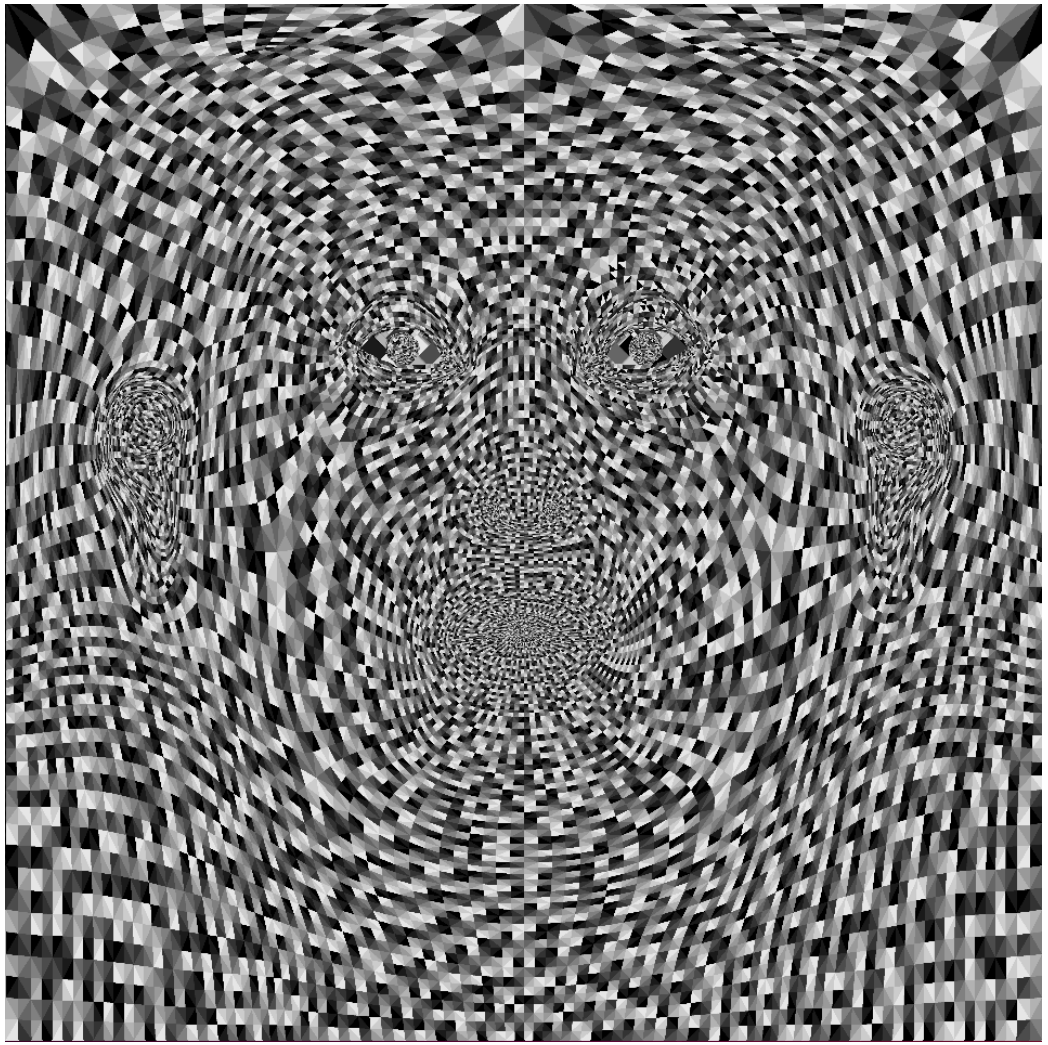


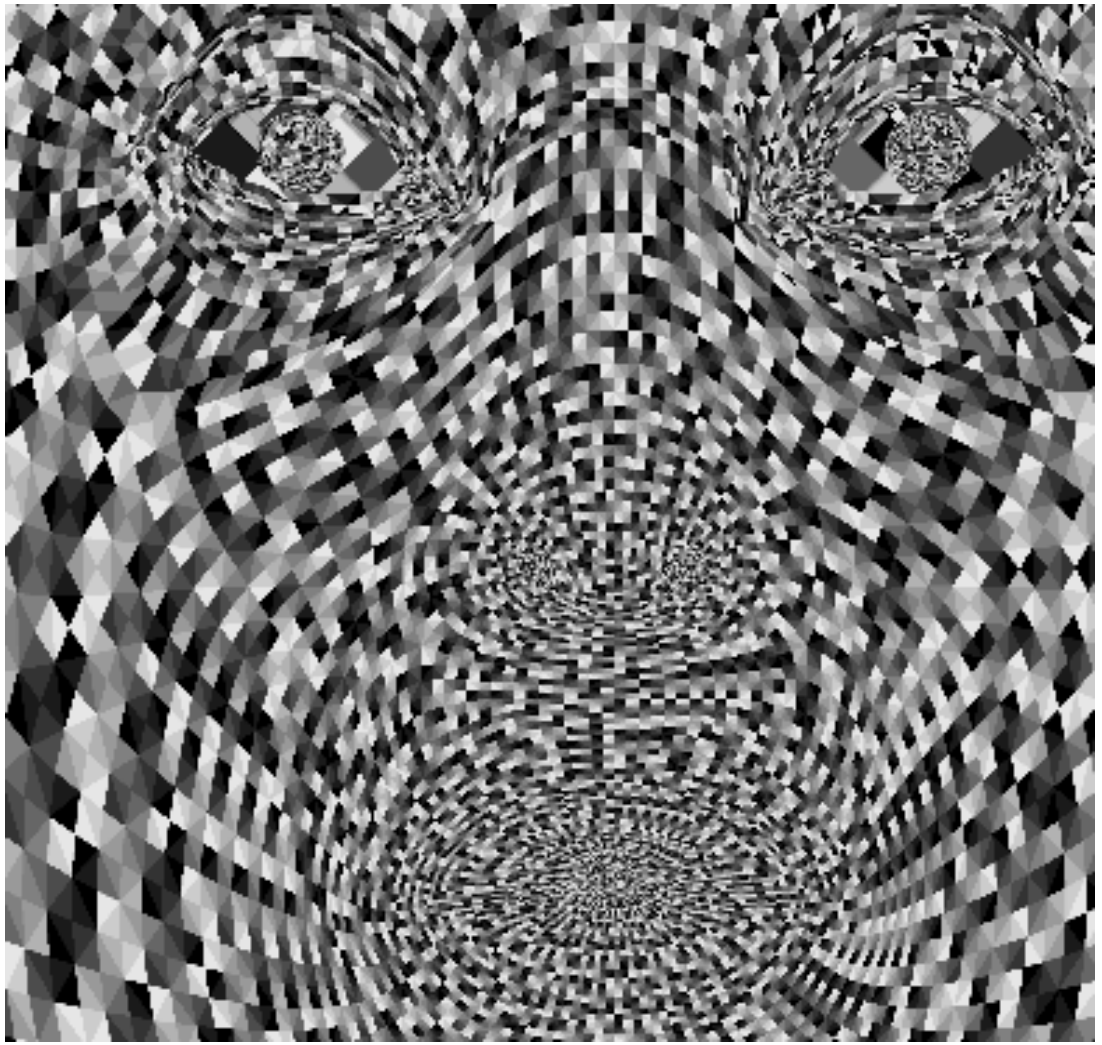
특정 삼각형 가로세로 근방의

모든 픽셀에 대해 삼각형 내부에

있는지의 여부를 판정

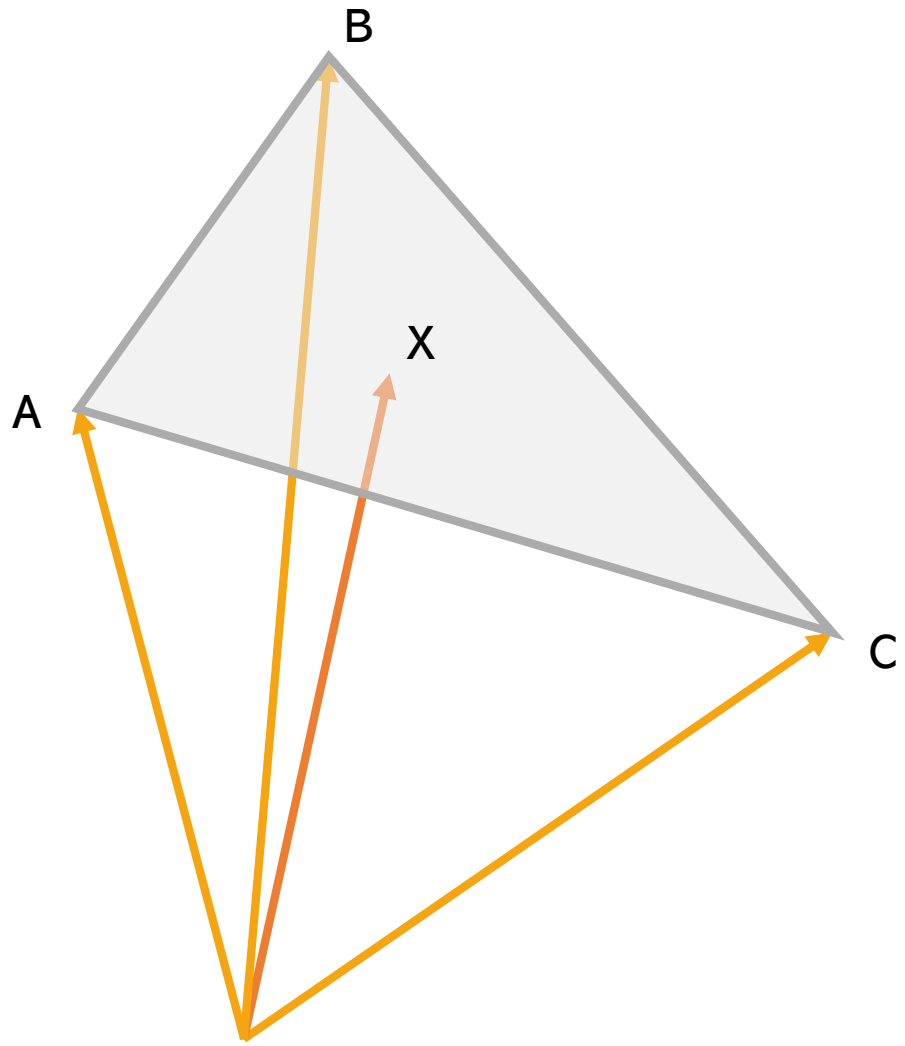
각각 점들의 소속 삼각형을 밝힌 뒤, 삼각형 번호에 따라 색을 입힌 사진





# Geometry interpolation

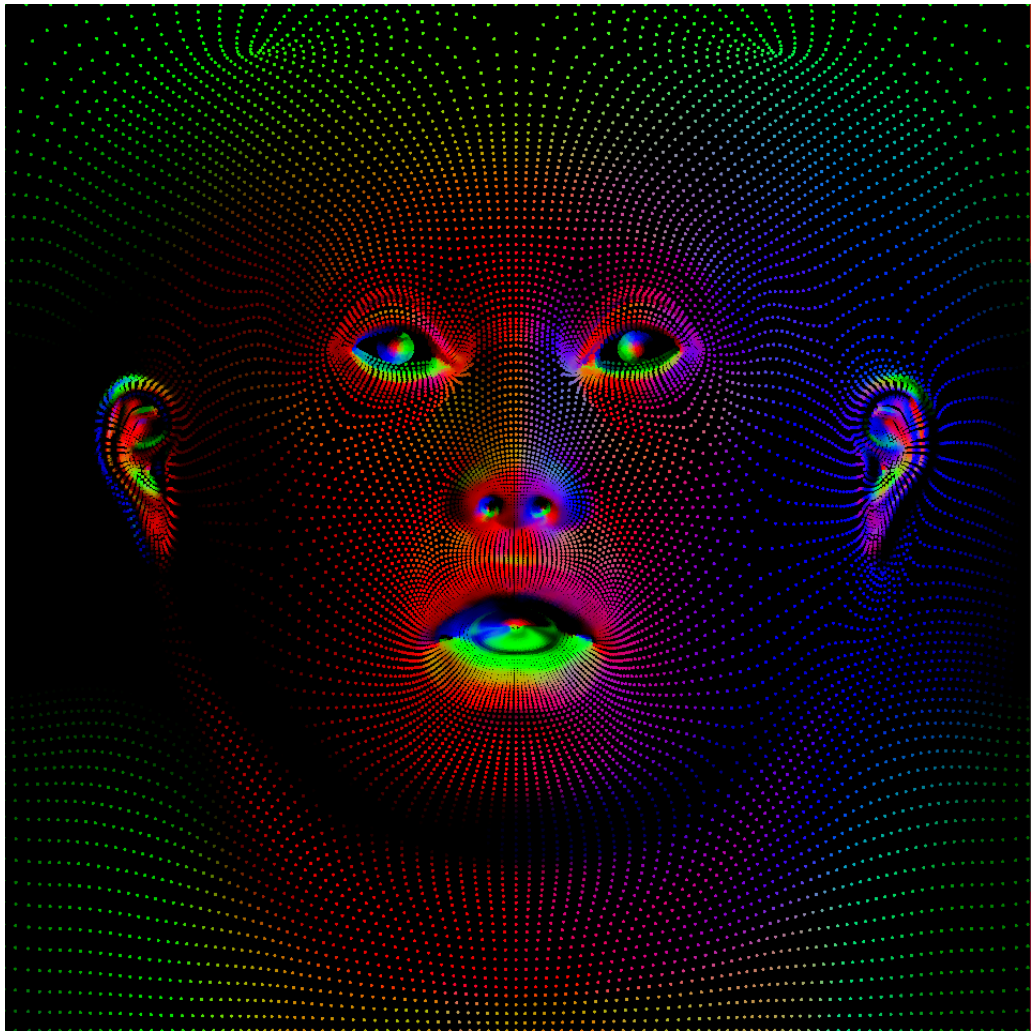
- vertex의 정보만을 알고 있음
- 삼각형 내부 점들의 정보 계산
- geometry는 시각화하기 어려우므로 normal을 예시로 사용



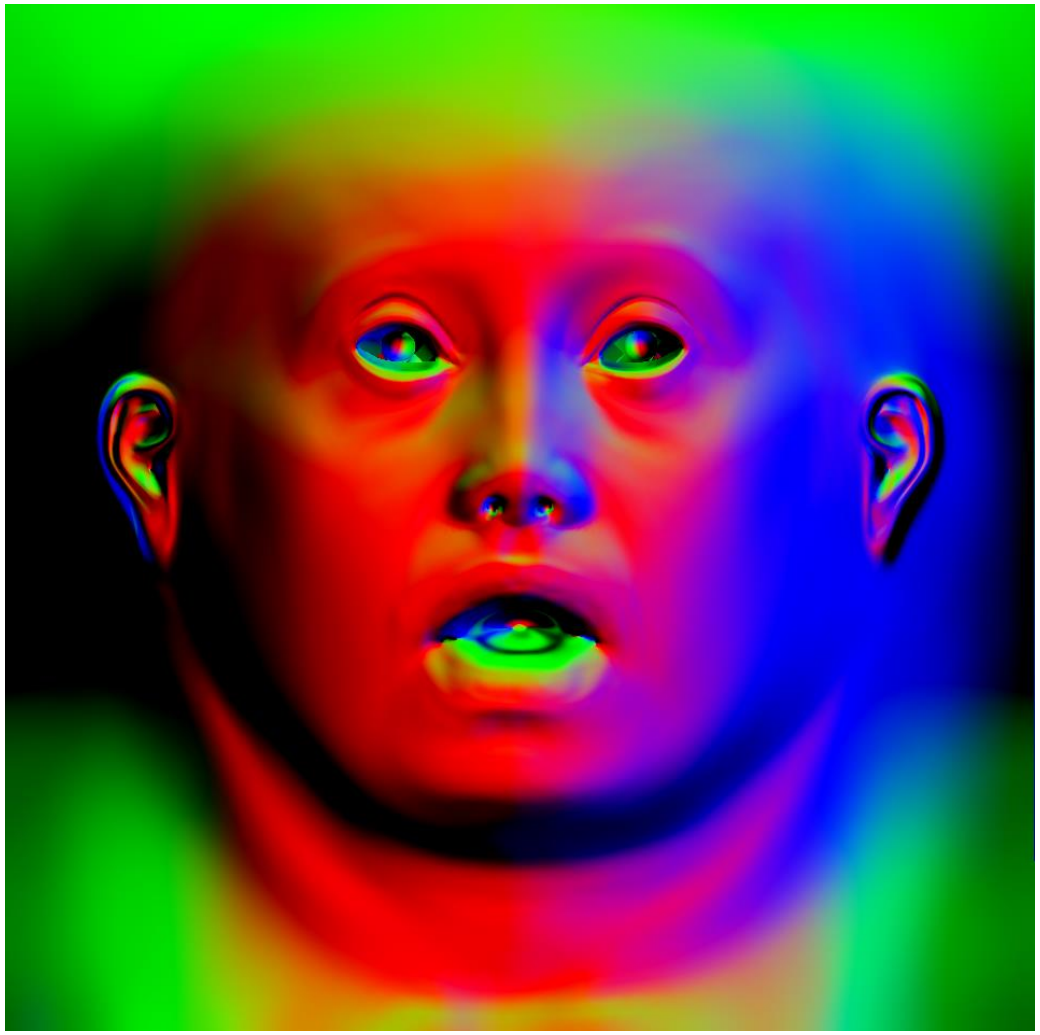

$$X = s \cdot A + t \cdot B + (1 - s - t) \cdot C$$

를 만족하는 실수 s, t를 찾는다

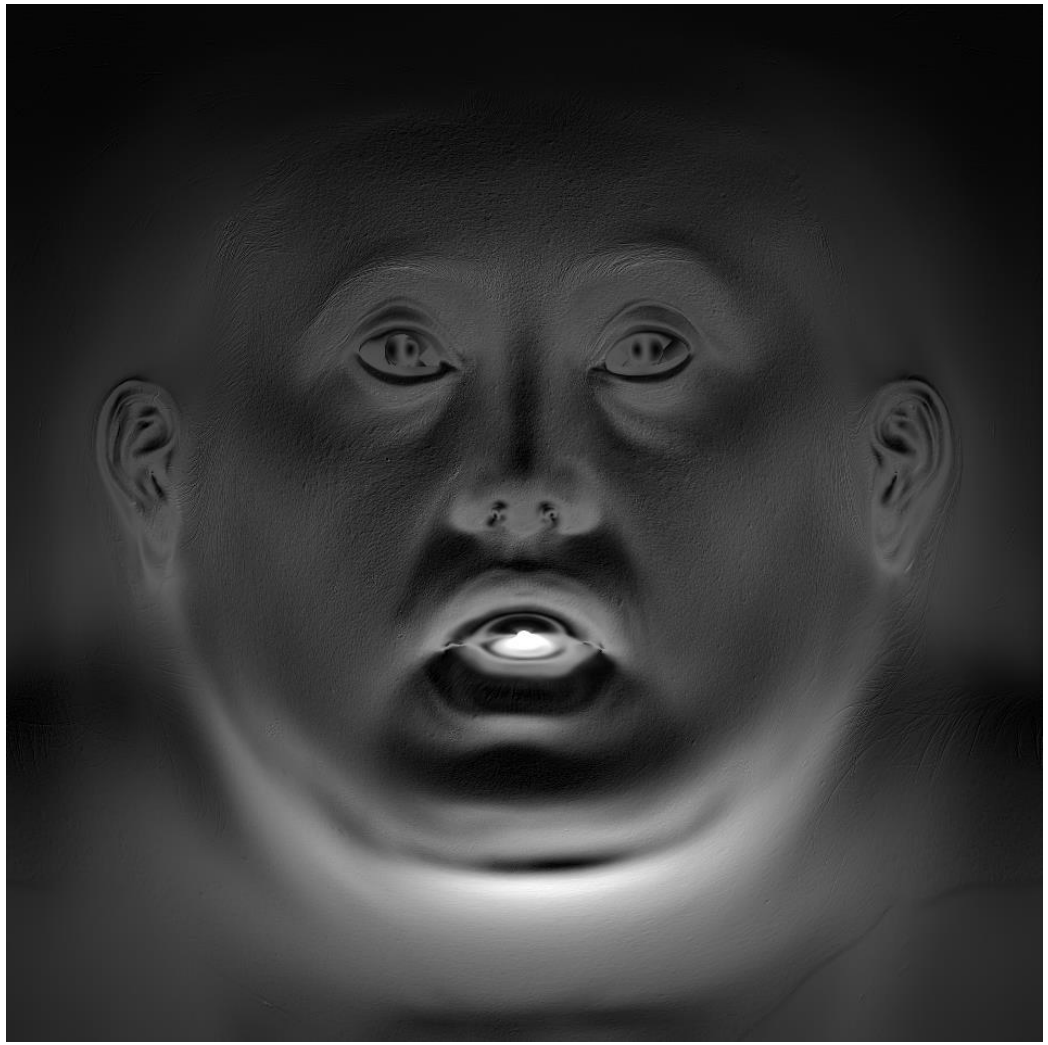
Mesh에 이미 normal이 주어져있던 지점들



이를 이용해 빈 곳을 채워넣은 normal map



최종 결과물인 displacement mapping

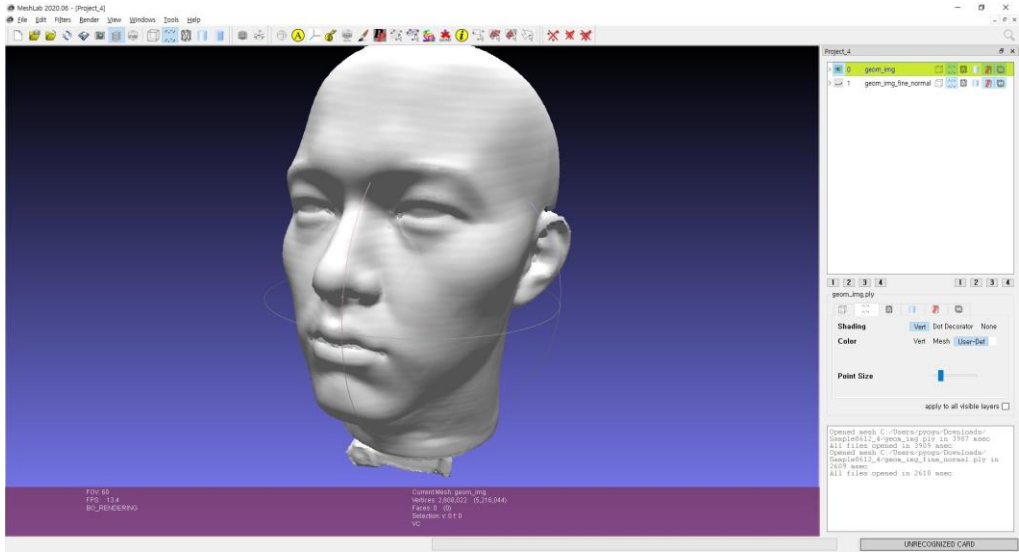


디테일이 드러난 모습

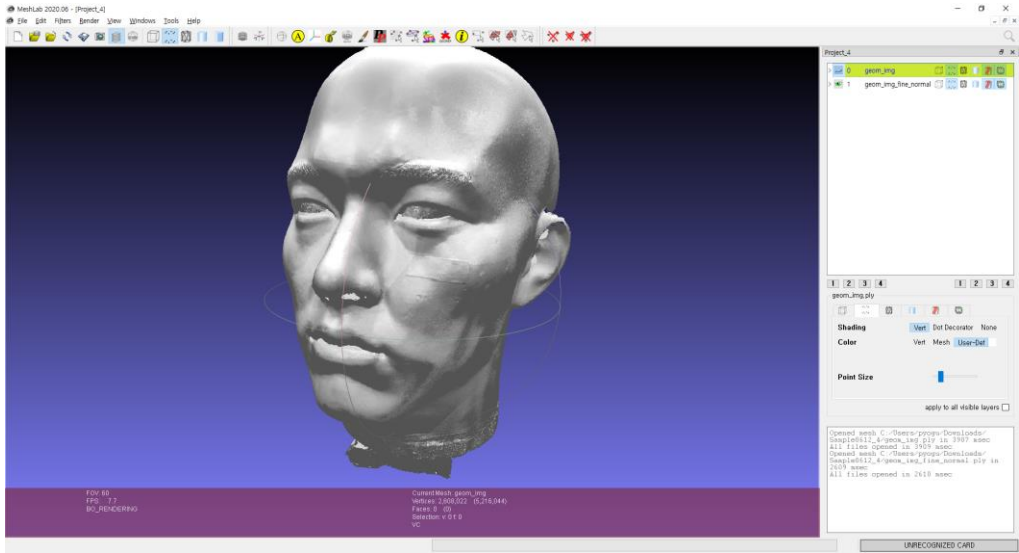
# 최종 결과

# 최종 결과

kNN normal



light stage normal



QnA