LightMap구현전략

유영천

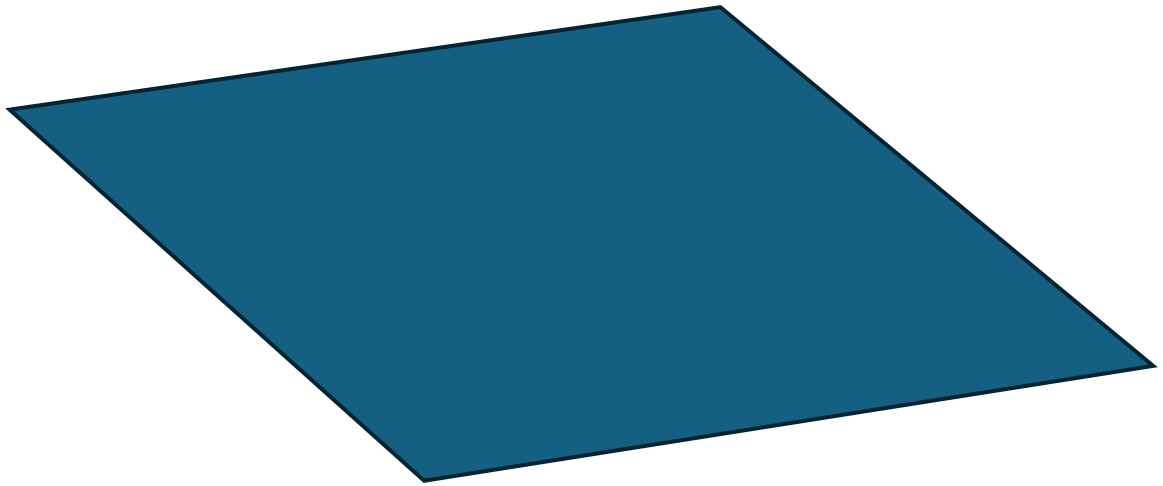
https://youtube.com/megayuchi

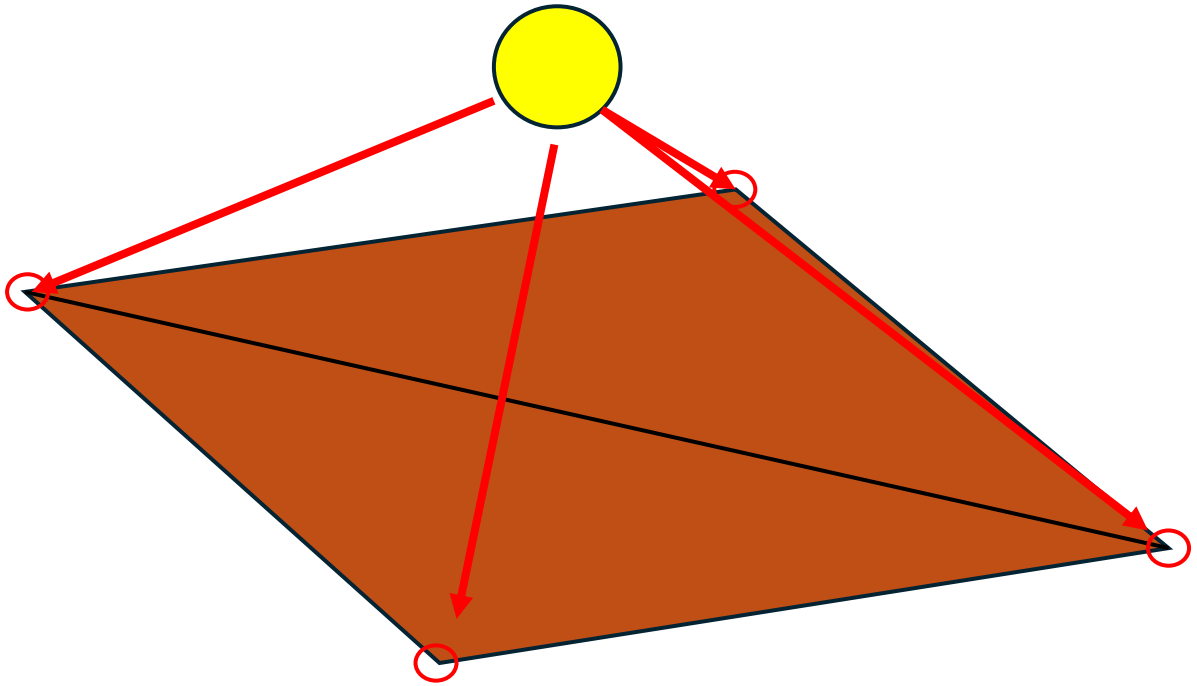
https://megayuchi.com

Lightmap

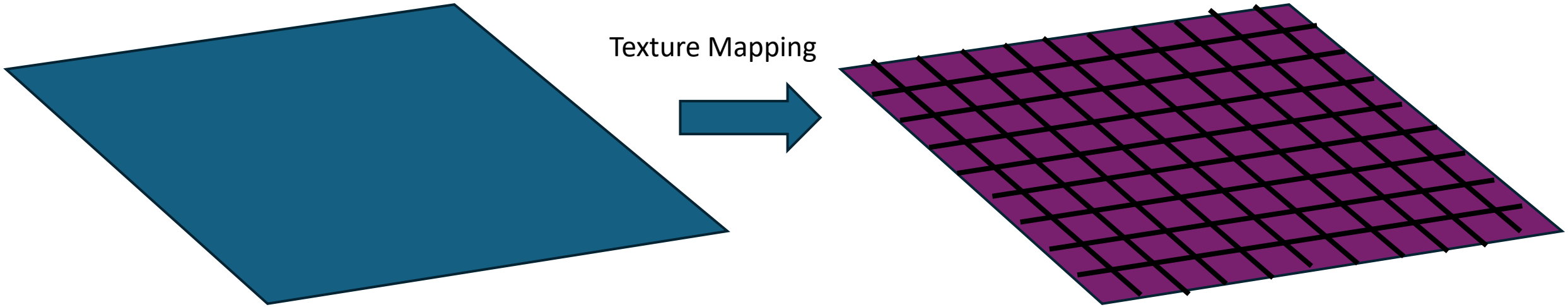
- 폴리곤에 Light Texture를 맵핑
- 폴리곤의 vertex가 아닌 Light Texture의 texel단위로 광도를 저장
- Shader가 없던 시절 적은 수의 폴리곤으로도 화려한 그래픽을 표현할 수 있었음.
- Shader를 이용한 Per pixel lighting이 가능해진 이후로도 성능상의 이유로 많이 사용됐음.
- 막상 직접 구현은 쉽지 않아서 한 때 3dsmax등 전문 모델링 툴에서 데이터를 뽑아서 쓰기도 했음.
  - 솔직히 이 방식으로 정상적으로 출시한 게임 별로 없음.
  - 유지보수 대단히 어려움

Per Vertex Lighting

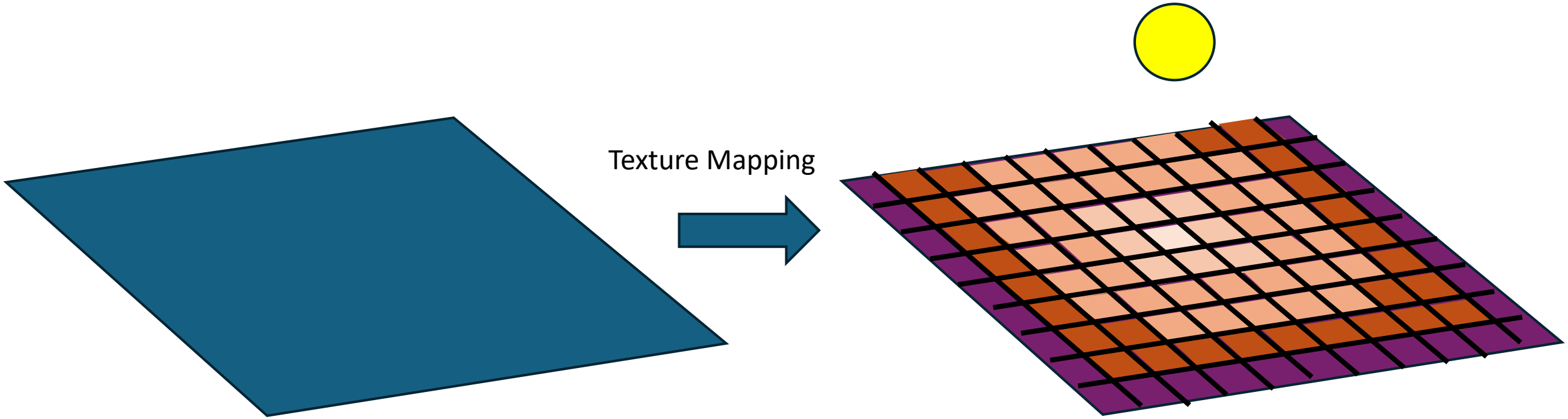




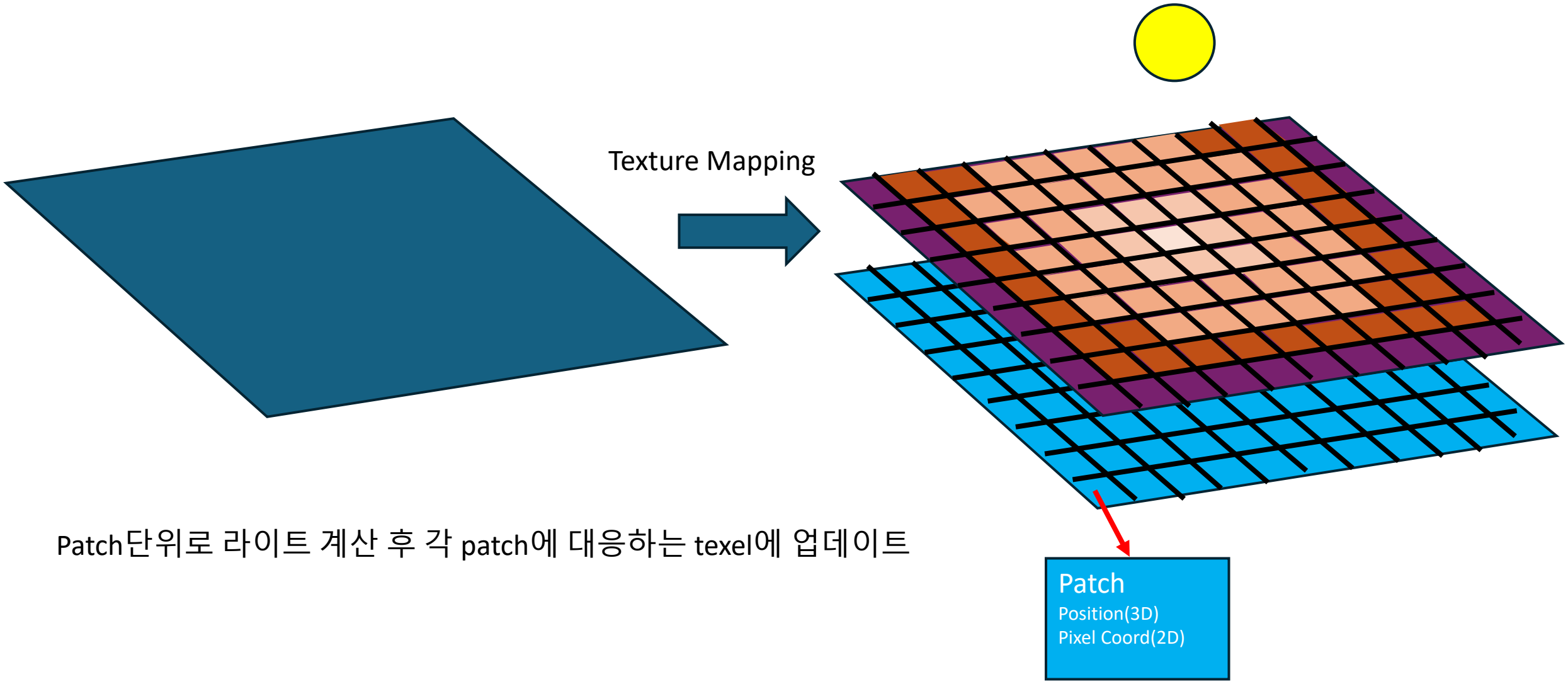
Per Texel Lighting(Not Per Pixel Lighting)



Per Texel Lighting(Not Per Piexl Lighting)

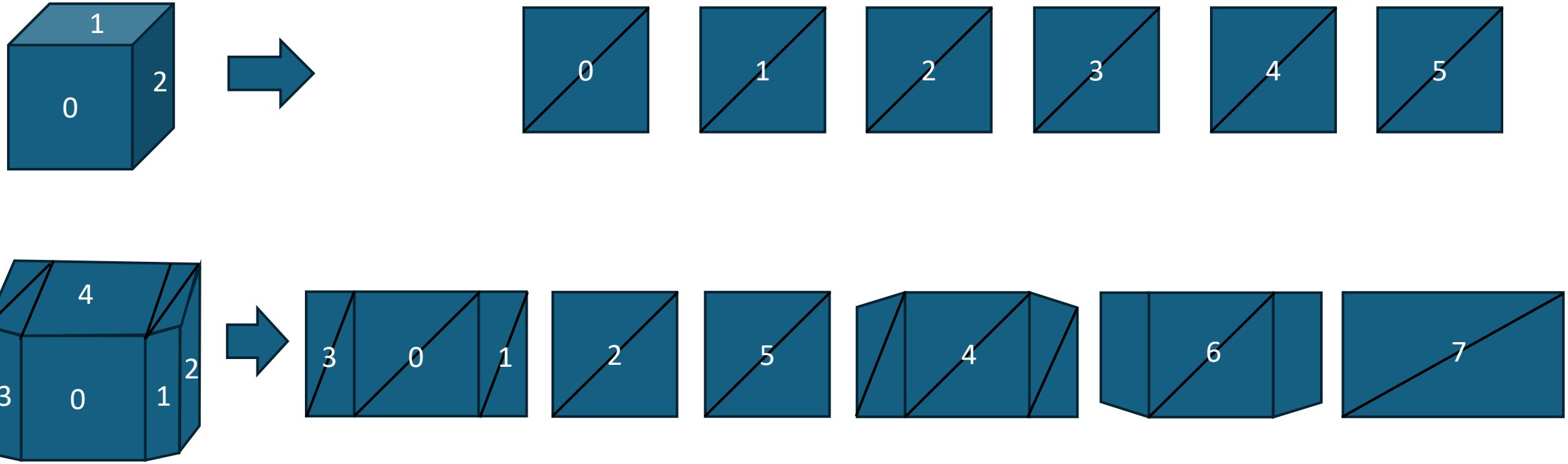


Per Texel Lighting(Not Per Piexl Lighting)



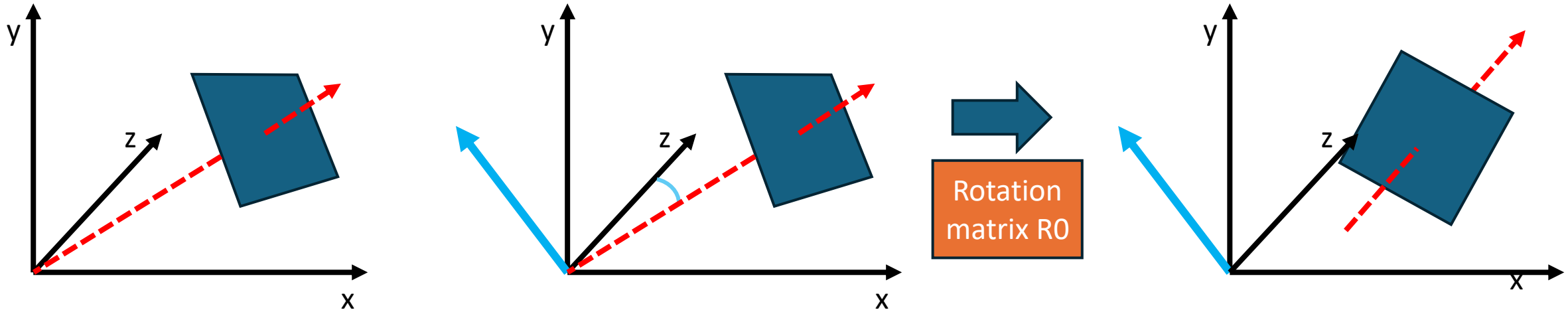
Patch단위로 라이트 계산 후 각 patch에 대응하는 texel에 업데이트

삼각형 그룹분리

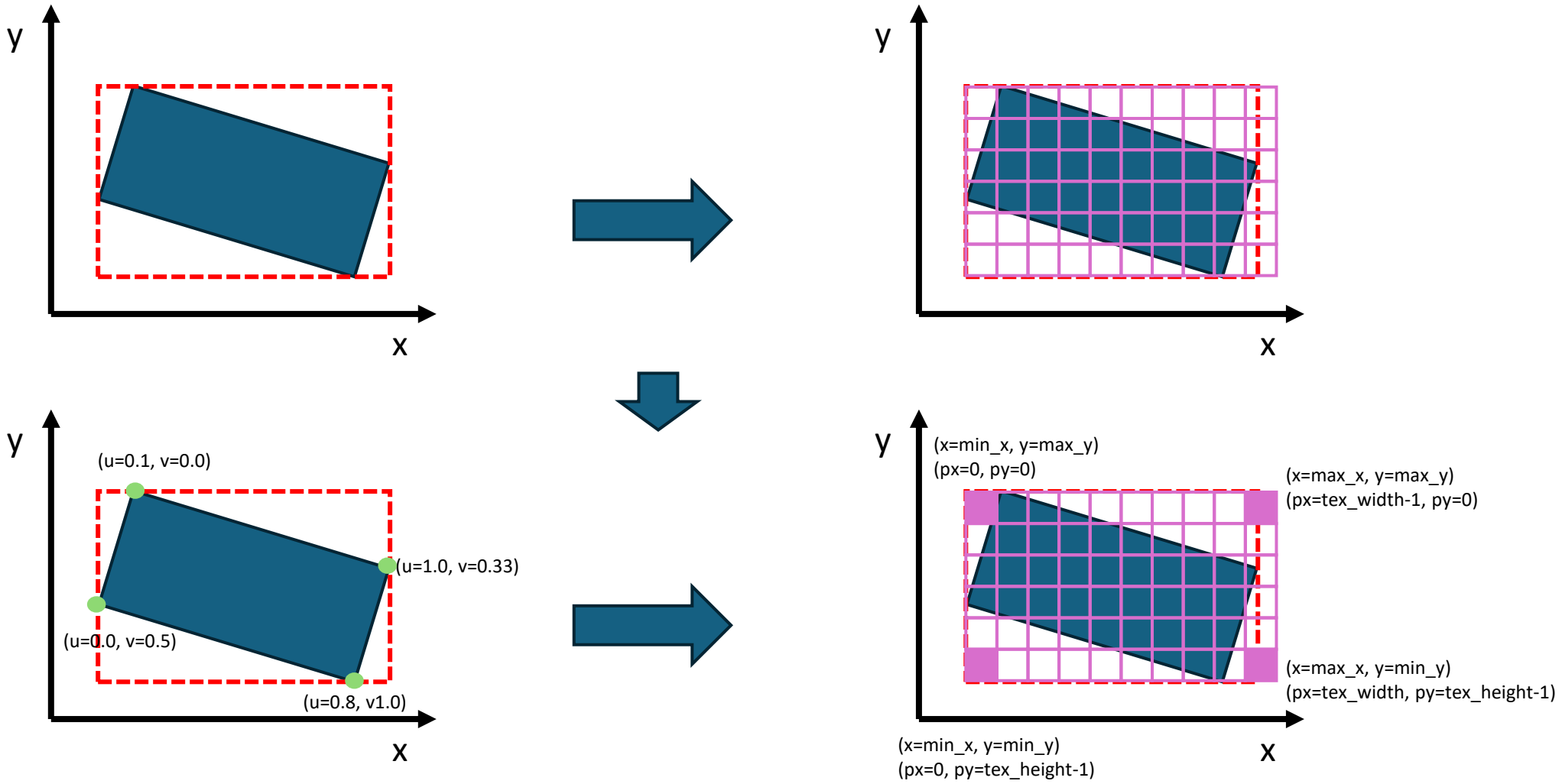


1. 인접한 삼각형들끼리 그룹을 짓는다.
2. 최초로 선택한 면이 해당 그룹의 비교 대상 평면이 되고 이 평면과 이후에 선택하는 평면의 방향 벡터가 일정 각도를 넘어가면 다른 그룹으로 분리한다.

삼각형 그룹을 xy평면에 투영(3D –> 2D)

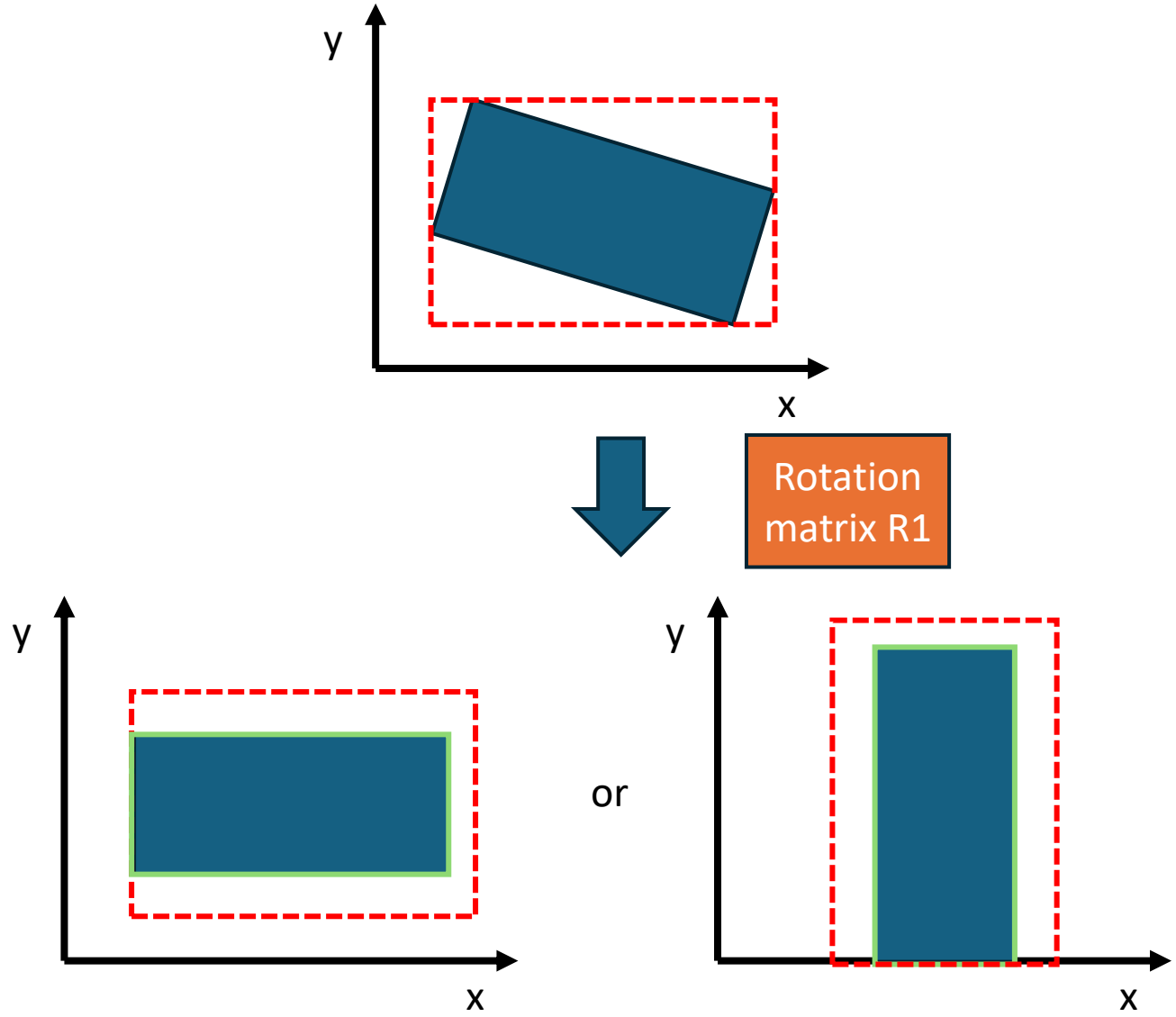


1. z=1인 벡터와 삼각형 그룹의 면방향 벡터를 cross해서 축 벡터를 얻는다.
2. Z=1인 벡터와 삼각형 그룹의 면방향 벡터의 사이각을 구한다.
3. Z=1인 벡터와의 사이각이 0이 되도록 회전시키는 행렬 R0을 구해서 삼각형 그룹을 회전시킨다.
4. 이제 삼각형 그룹의 모든 점의 z성분이 같은 값을 가진다. 즉 xy평면에 평면에 투영된 것과 같다.



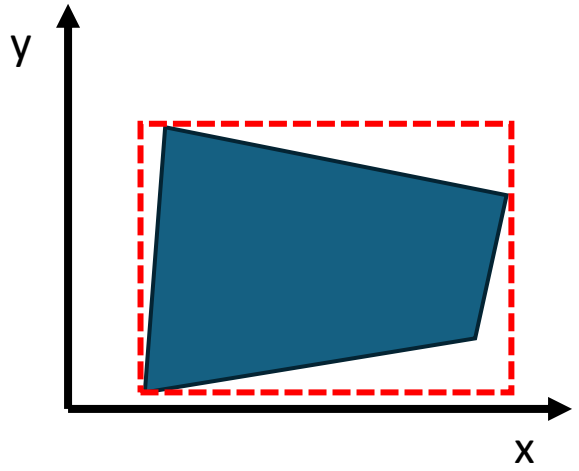
1. 2D공간에 투영된 삼각형 그룹을 감싸는 AABB(min,max)를 구한다.
2. 2D공간에 투영된 AABB를 텍스처에 대응시킨다.
3. AABB만으로도 uv좌표 및 patch데이터 생성 가능
4. AABB의 위치로부터 patch 각각의 2D에서의 위치를 구한다.

품질개선

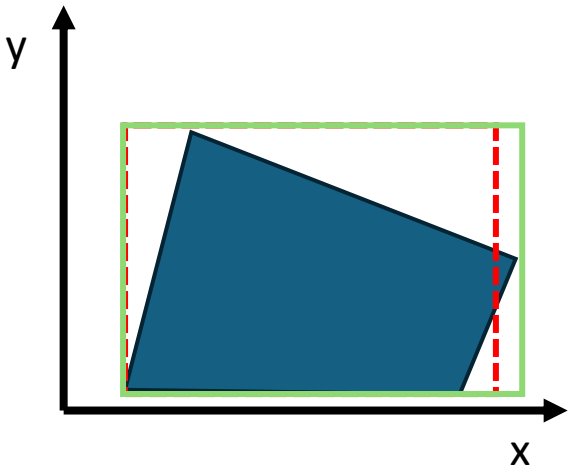


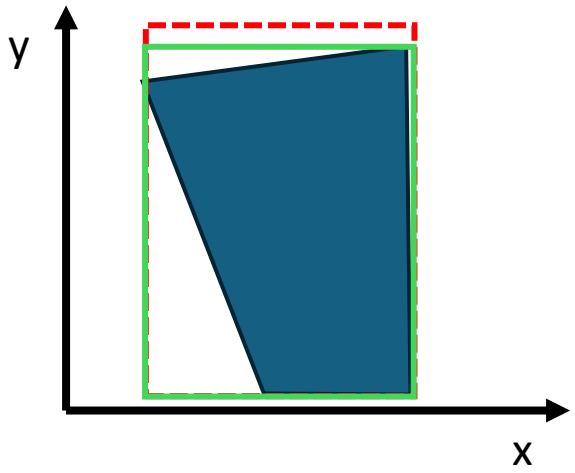
1. 삼각형 그룹에 더 많은 텍셀을 할당하고
2. 렌더링 시 도형의 엣지와 라이트텍스처의 결을 맞추기 위해
3. 삼각형 그룹을 x or y축에 대해 정렬
4. 정렬 후 AABB의 면적이 가장 적은 케이스를 선택한다.

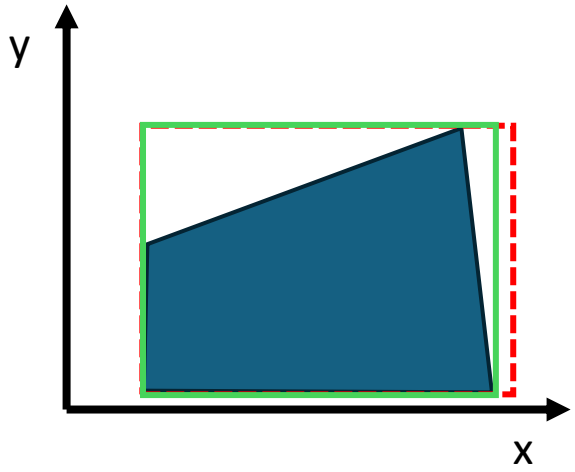
N개의 엣지를 가진 삼각형 그룹에 대한 정렬

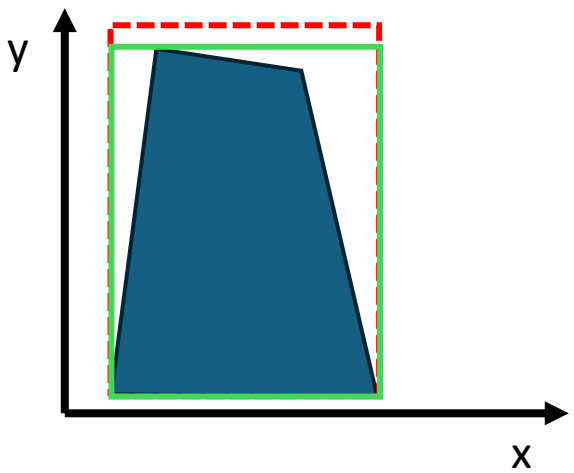


Rotation matrix R1

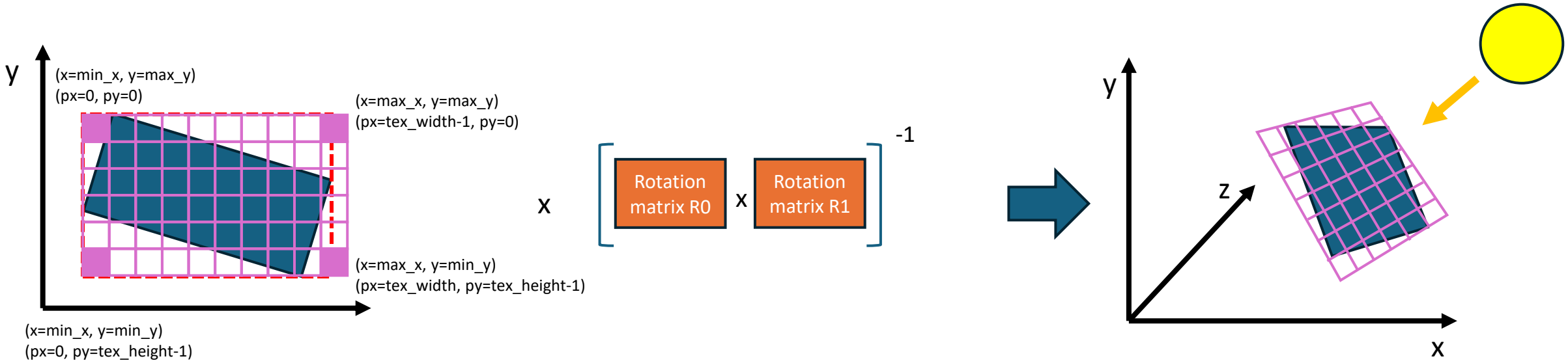






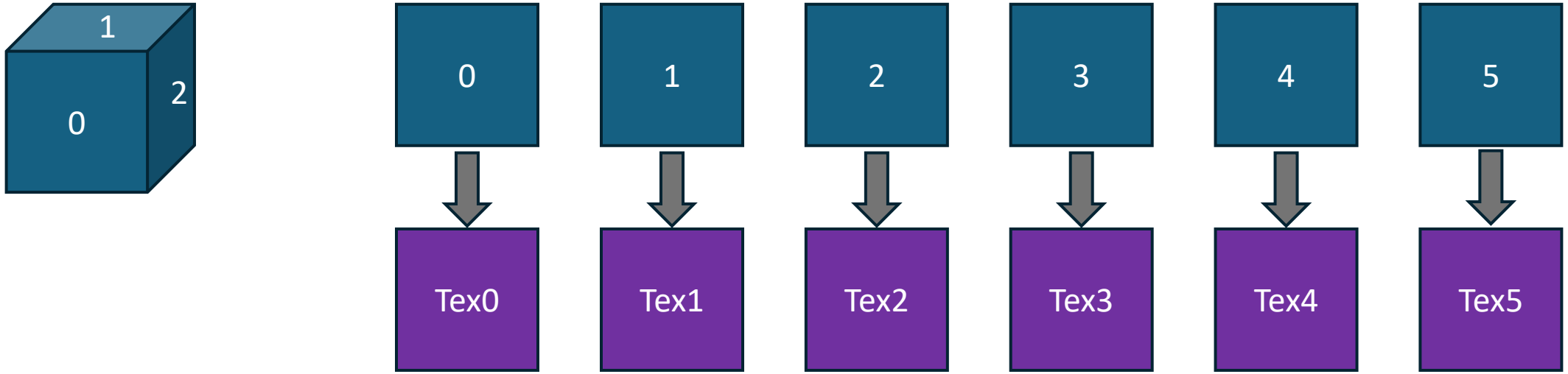


2D -> 3D



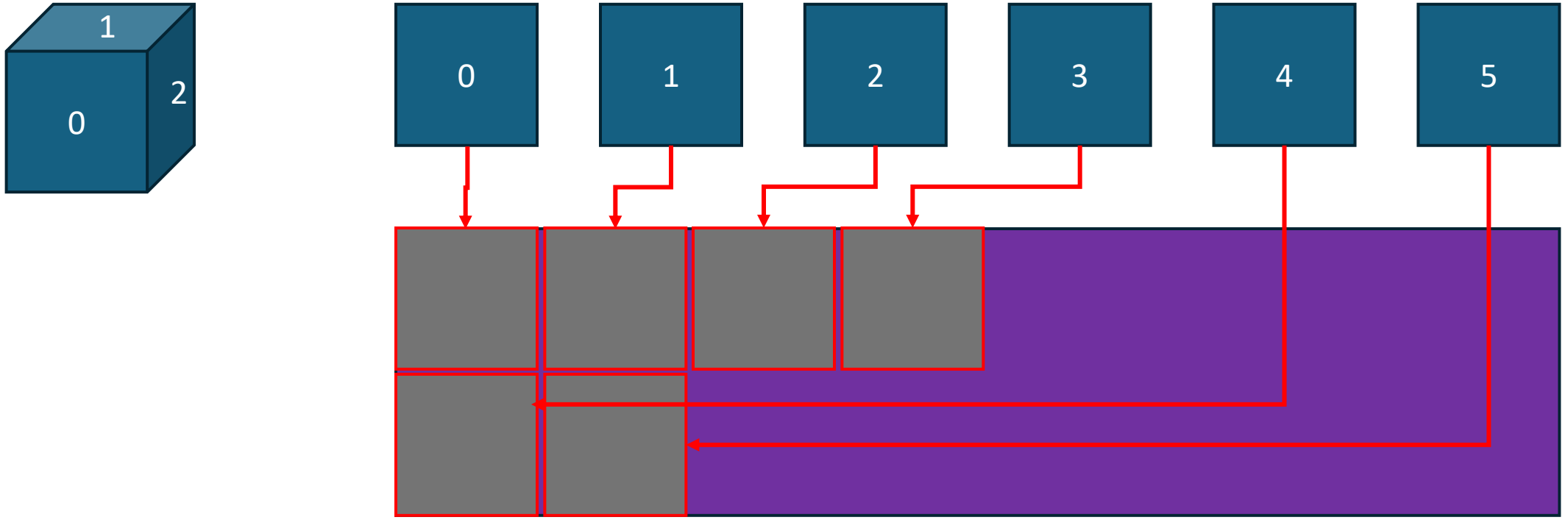
1. 2D공간에서 생성된 Lightmap Patch들을 처음의 3D좌표계로 돌린다.
2. 각 patch들의 3D좌표를 알고 있으므로 patch마다 라이트를 계산할 수 있다.
3. 각 patch들은 텍스처의 픽셀 좌표를 가지고 있으므로 2에서 계산한 광도를 텍스처에 업데이트 할 수 있다.

Texture Packing

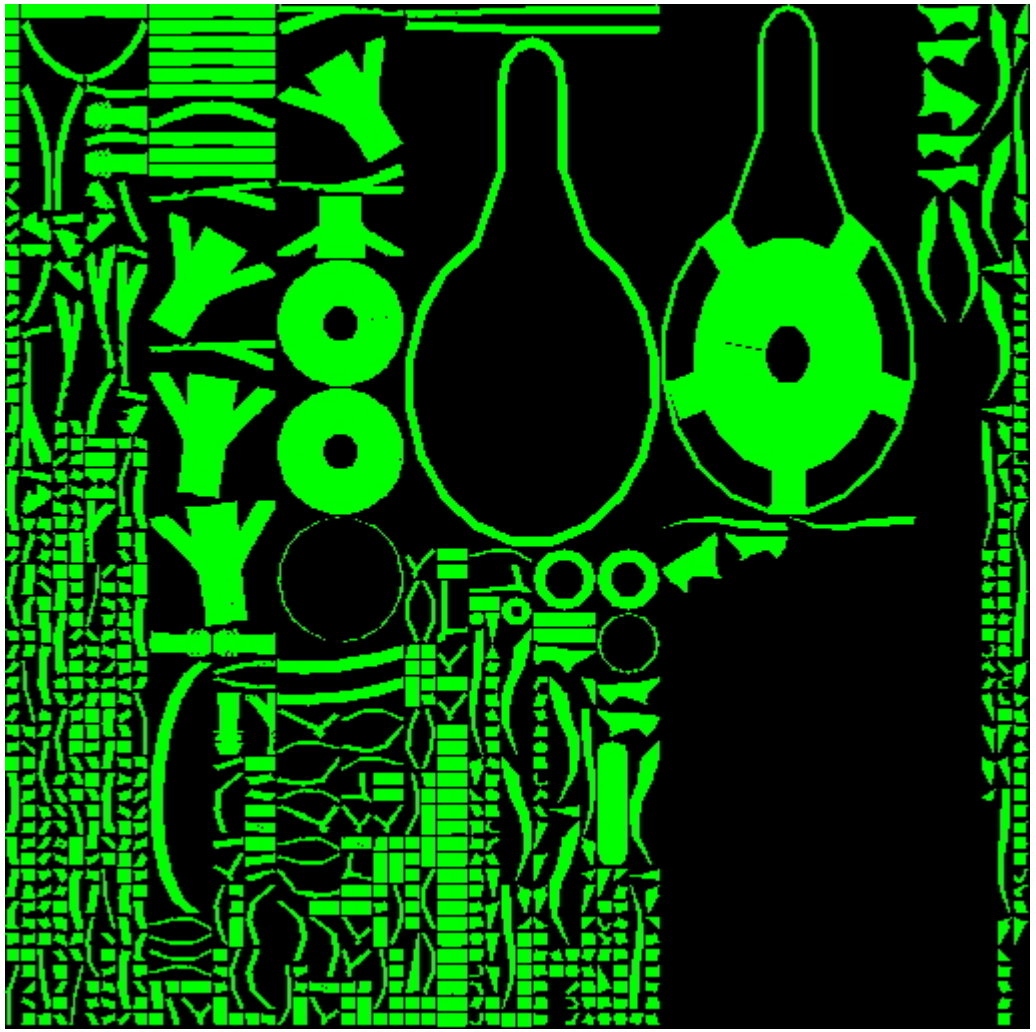


- 평면(그룹)당 하나의 텍스처를 할당하는 경우 너무나 많은 텍스처를 사용하게 된다.
- 성능저하는 물론 안정성에 심각한 문제를 일으킬 수 있다.

Texture Packing



- 최대한 하나의 텍스처에 여러 개의 평면그룹을 맵핑시킨다.
- 텍스처의 텍셀간 보간을 고려하여 2픽셀씩 여유를 둔다.



baking

- Radiosity등 여러가지 방법이 있음.
- 기본적으로는 N개의 patch에서 M개의 라이트에 대해 ray테스트 하는것으로 광도와 그림자 적용 여부를 판단.
- 많은 시간이 소요될 수 있으므로 멀티스레드 필수.
- 그림자 처리를 위해서는 ray 교차 테스트를 위한 자료구조(KD Tree등) 필수.
- CUDA를 사용하면 성능을 10배이상 향상 시킬 수 있다.