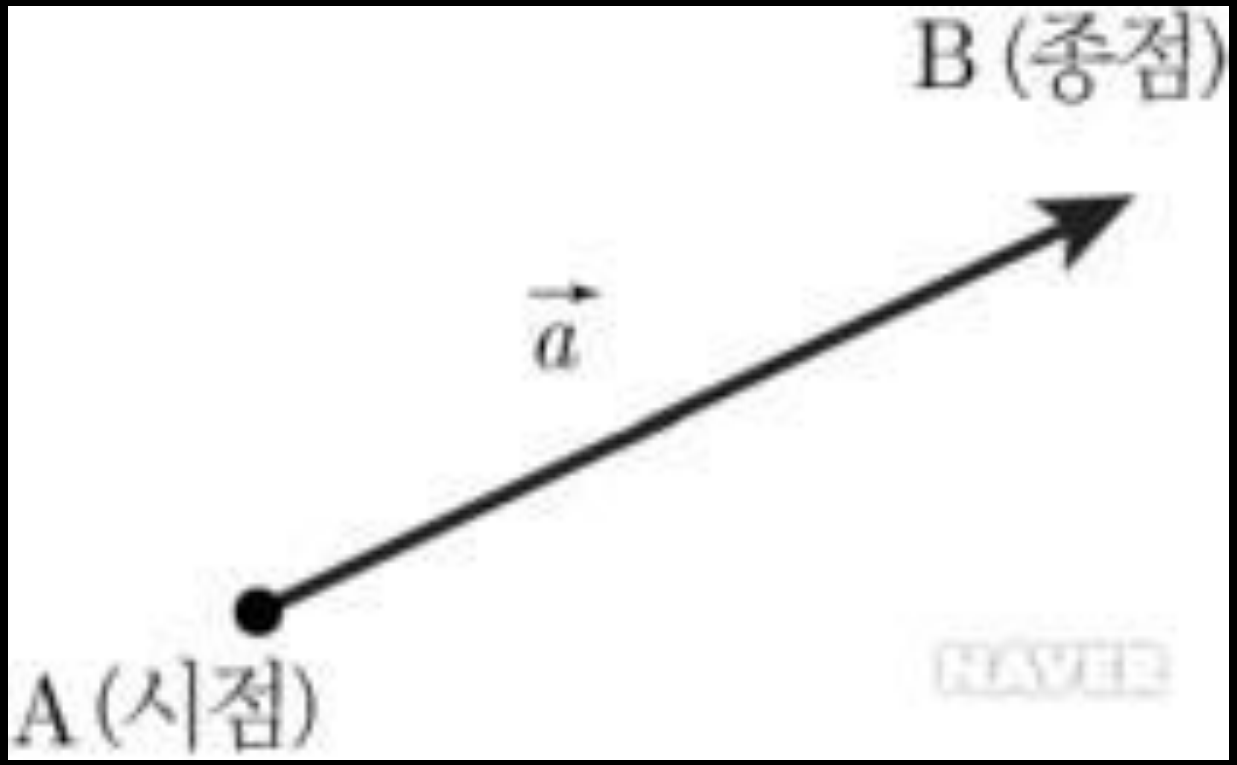
Normal mapping
Parallax mapping
Displacement mapping

쿠재아이
김재경

벡터(Vector)

- 크기와 방향을 가지는 양



http://terms.naver.com/entry.nhn?docId=2073863&cid=47324&categoryId=47324

법선(Normal)

- 평면곡선 위의 1점 P를 지나고, 그 점에서의 접선에 수직인 직선 및 곡면 위의 한 점 P를 지나고 그 점에서의 접평면에 수직인 직선이다.

http://terms.naver.com/entry.nhn?docId=1100941&cid=40942&categoryId=32223

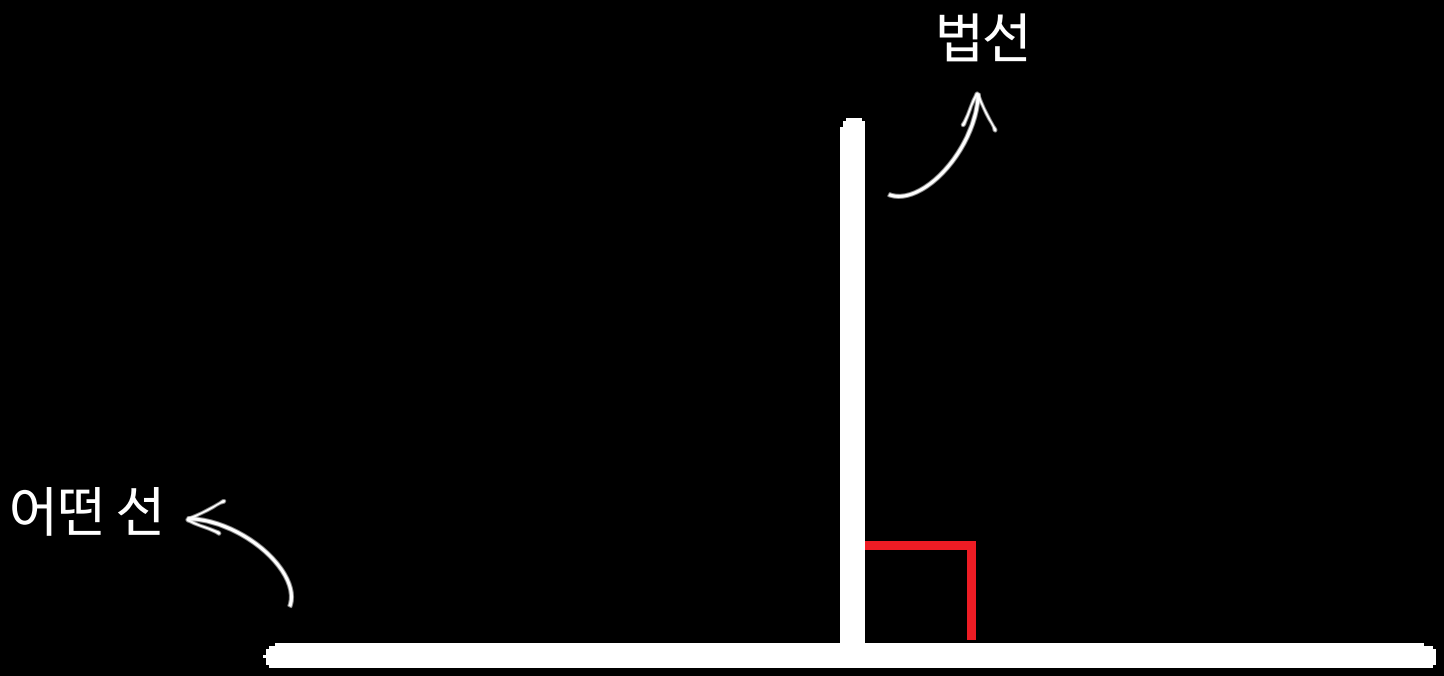
법선(Normal)

- 평면곡선 위의 1점 P를 지나고, 그 점에서의 접선에 수직인 직선 및 곡면 위의 한 점 P를 지나고 그 점에서의 접평면에 수직인 직선이다.

? ? ?

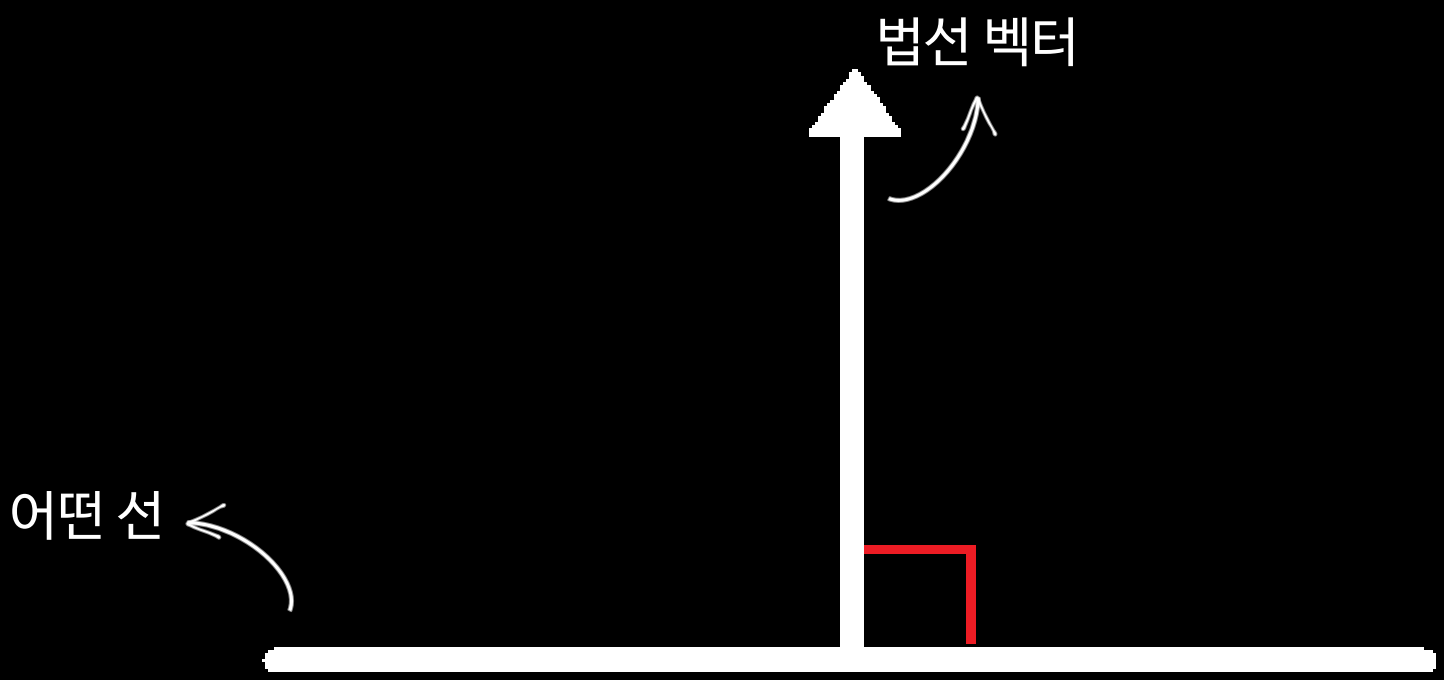
법선(Normal)

- 선이나 면에 수직인 직선



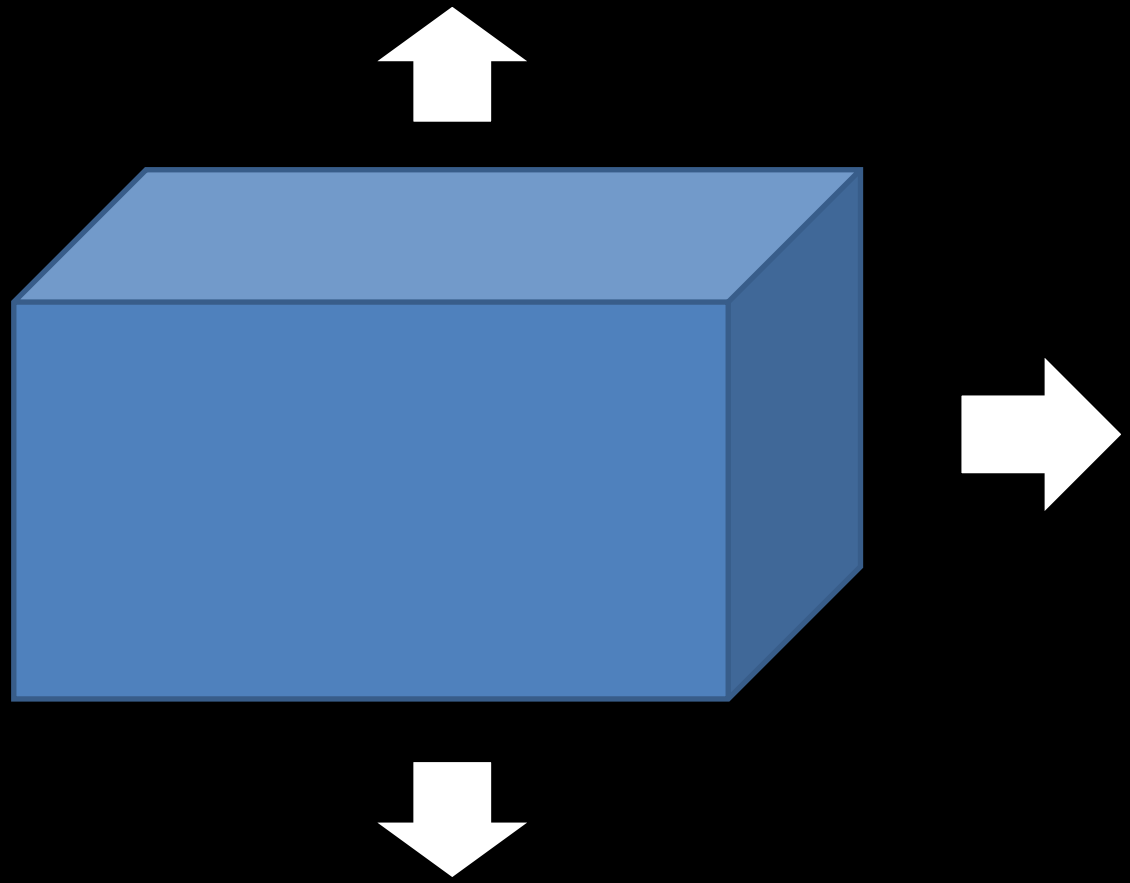
법선 벡터(Normal vector)

- 선이나 면에 수직인 벡터



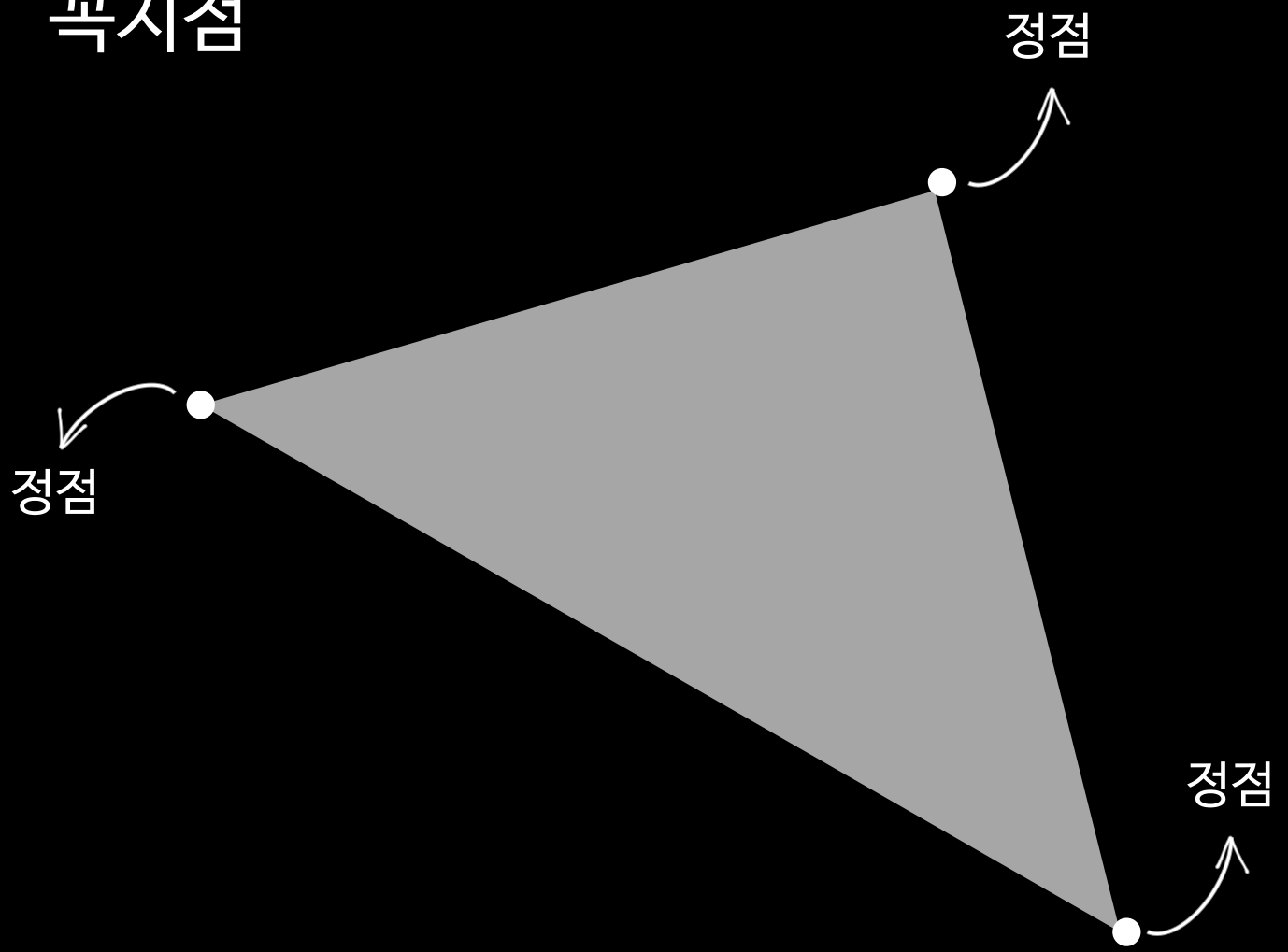
법선 벡터(Normal vector)

- 물체의 표면의 방향을 결정



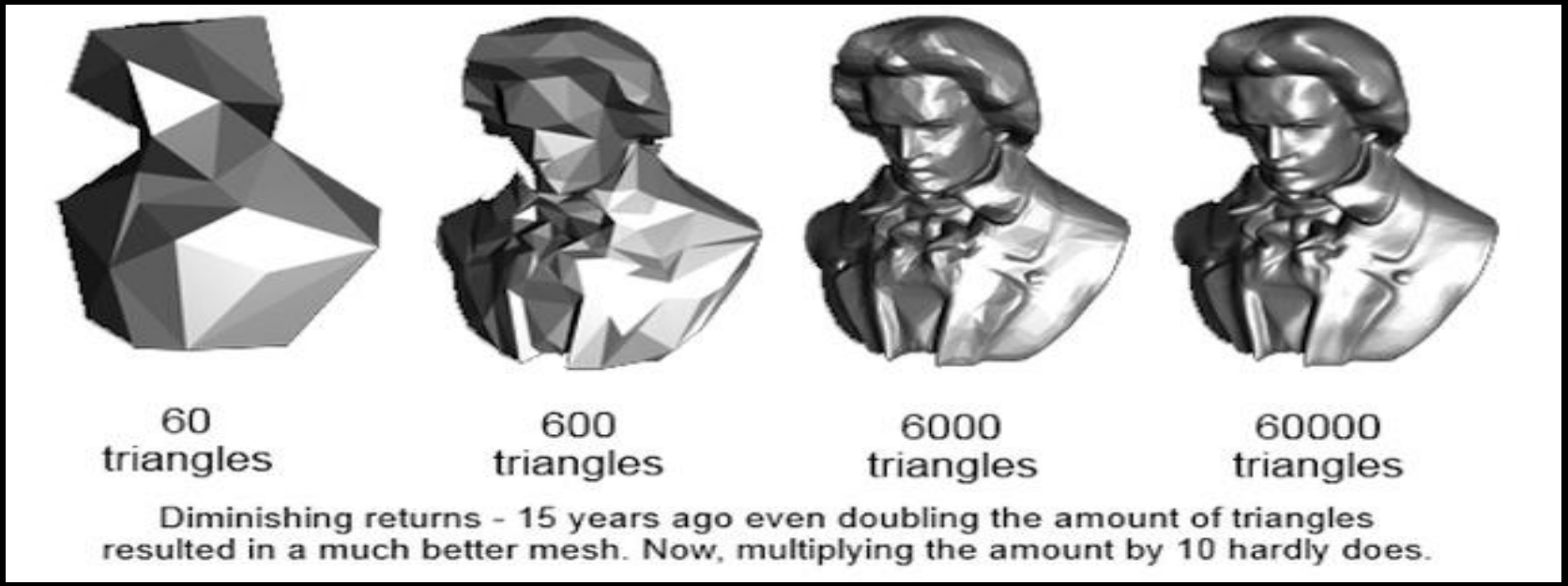
정점(Vertex)

- 꼭지점



폴리곤(Polygon)

- 다각형을 의미
- 3D 모델을 구성하는 기본 단위
- 일반적으로 삼각형을 사용



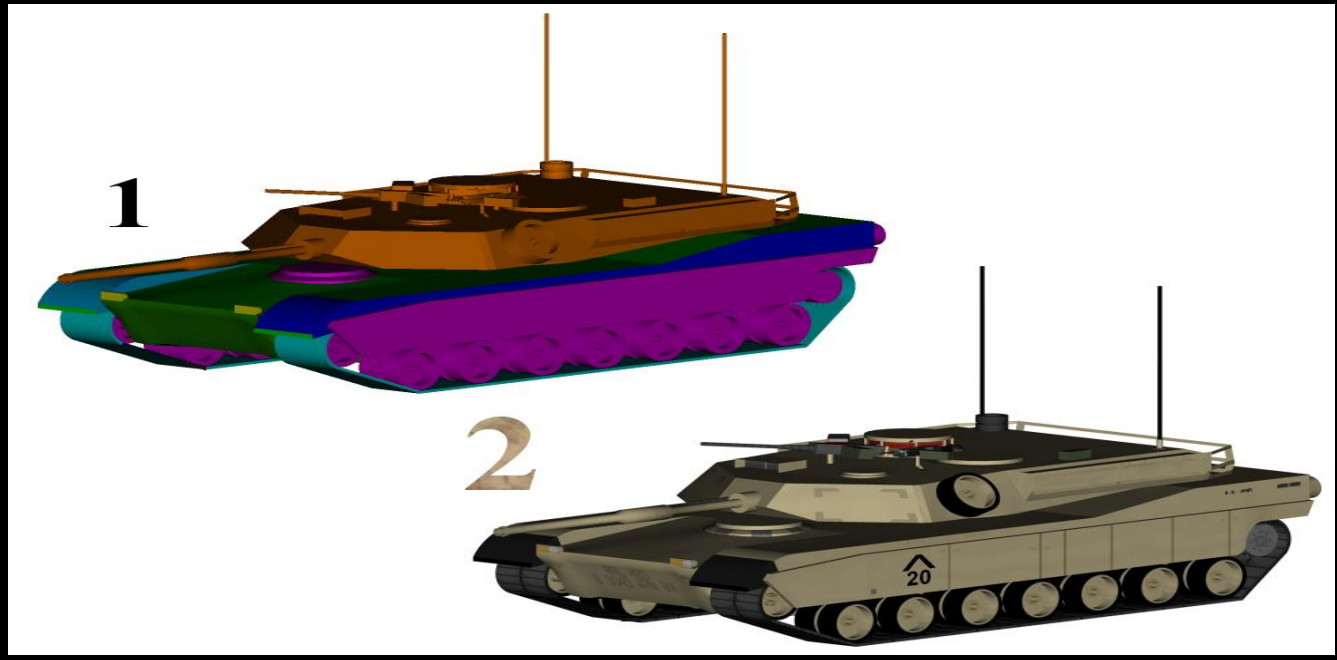
http://chzbk.com/xe/UCreate/228066

텍스처(Texture)

- 폴리곤 겉에 씌우는 이미지(RGB 값이 저장 됨)
- 실제의 물체처럼 느끼게 함

1. 텍스처 無

2. 텍스처 有

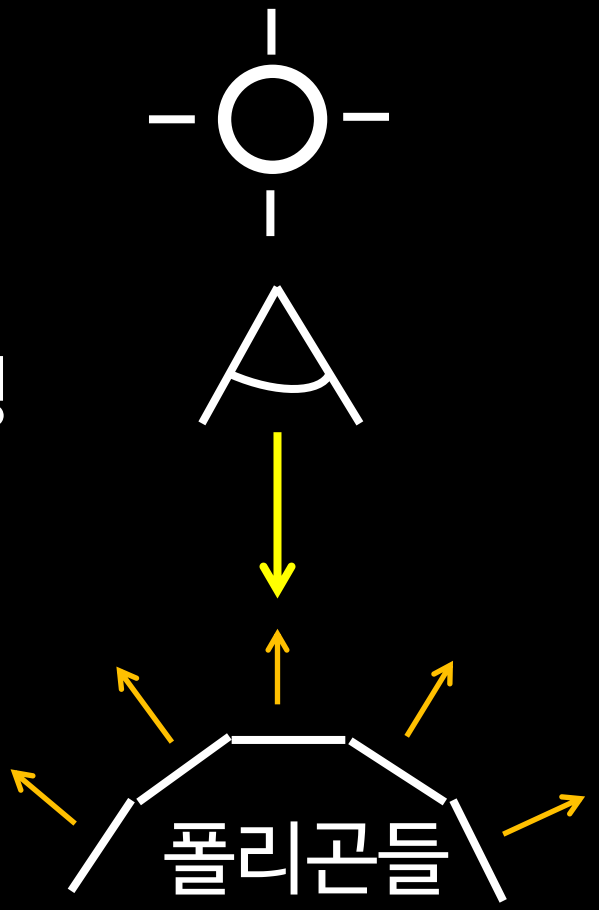


https://ko.wikipedia.org/wiki/%ED%85%8D%EC%8A%A4%EC%B2%98_%EB%A7%A4%ED%95%91

Normal mapping

들어가기 전에…

- 3D에서 음영(그림자)이 생기는 원리
- 광원 벡터(빛) –
- 법선 벡터(표면의 방향) –
- 2가지의 벡터를 계산하여 결정



문제점

- 더 많은 굴곡, 음영을 표현하기 위해서는 더 많은 폴리곤이 필요
- 그래서 폴리곤을 늘리면?

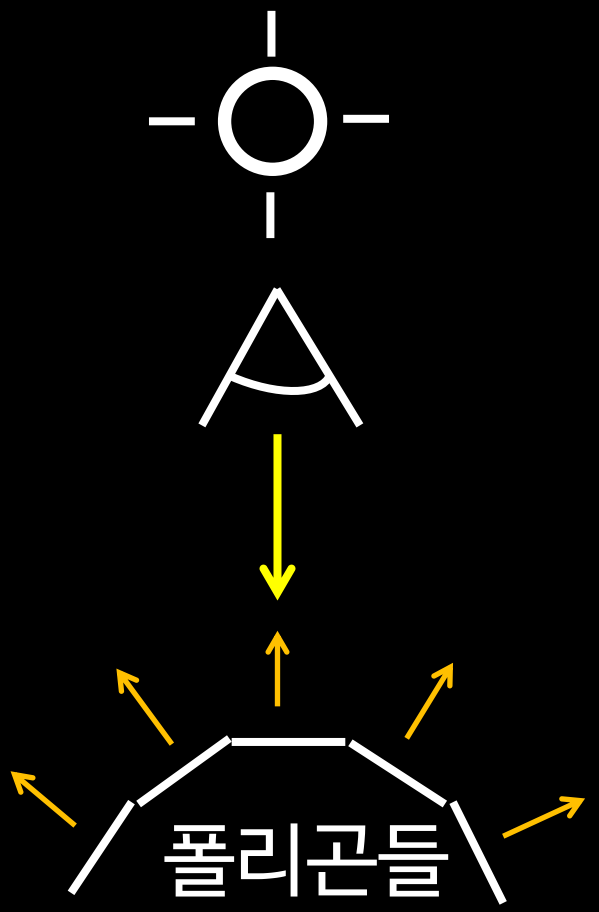
연산량 증가

⬇

렉이 걸림

⬇

망한다

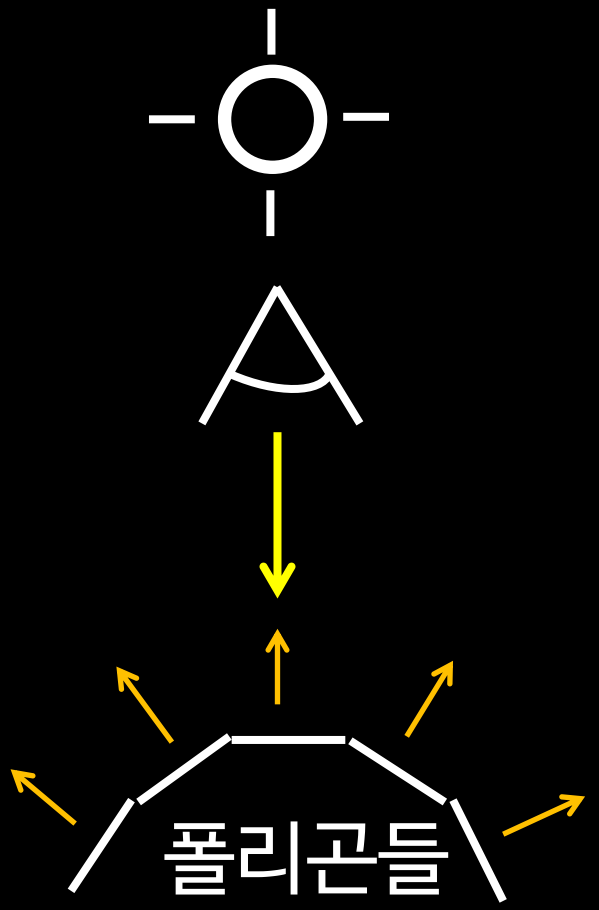


생각해보자

- 사실 게임에서 모든 물체의 음영을 세밀하게 묘사할 필요는 없다
- 그렇다면 세밀하게 묘사된 척(!)만 하면 어떨까?
- 어떻게? How?

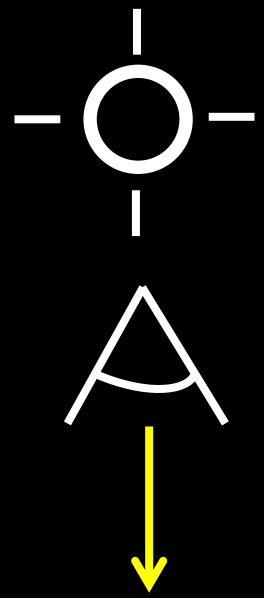
생각해보자

- 음영을 계산할 때 광원벡터는 고정된다
- 법선 벡터에 따라 달라보일뿐
- 그렇다면…

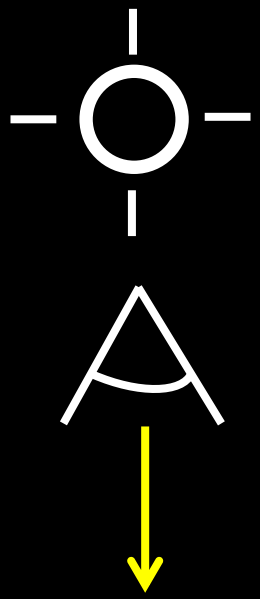


생각해보자

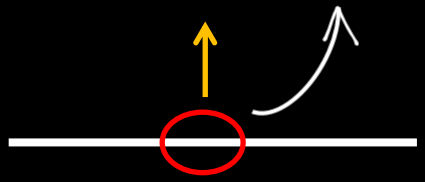
- 아까와는 달리 적은 수의 폴리곤에



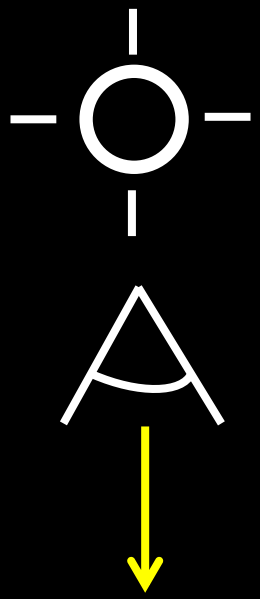
생각해보자



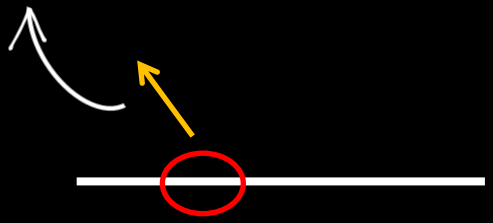
여기 노말은 이렇게 주고



생각해보자

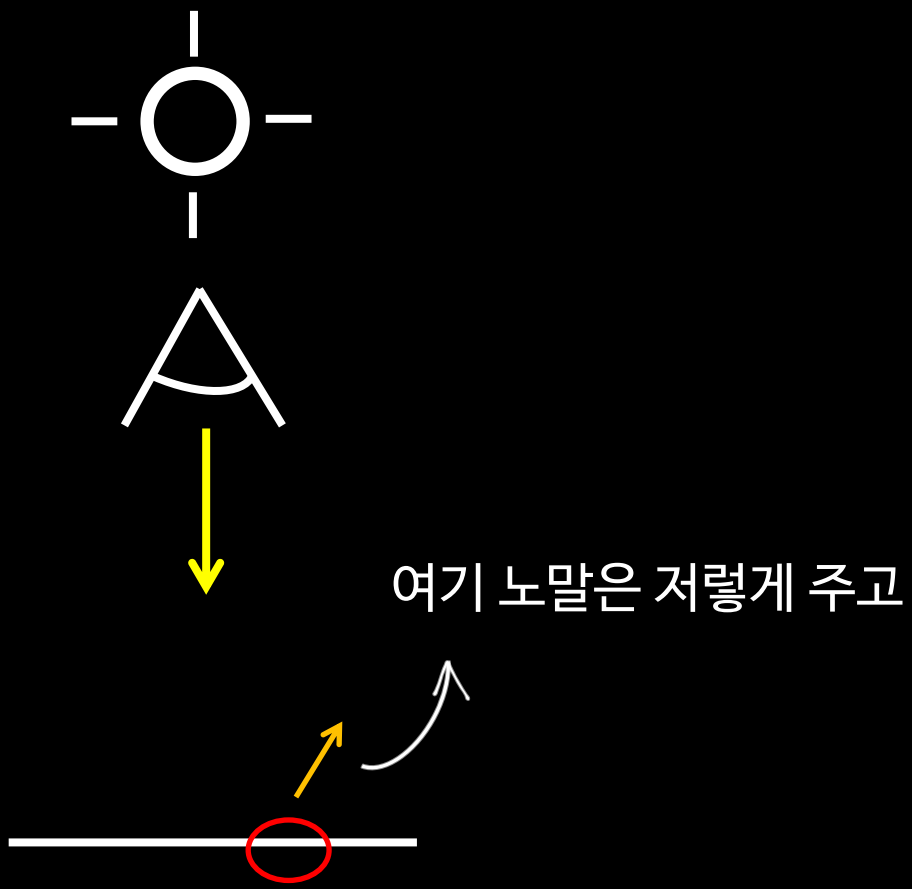


여기 노말은 요렇게 주고



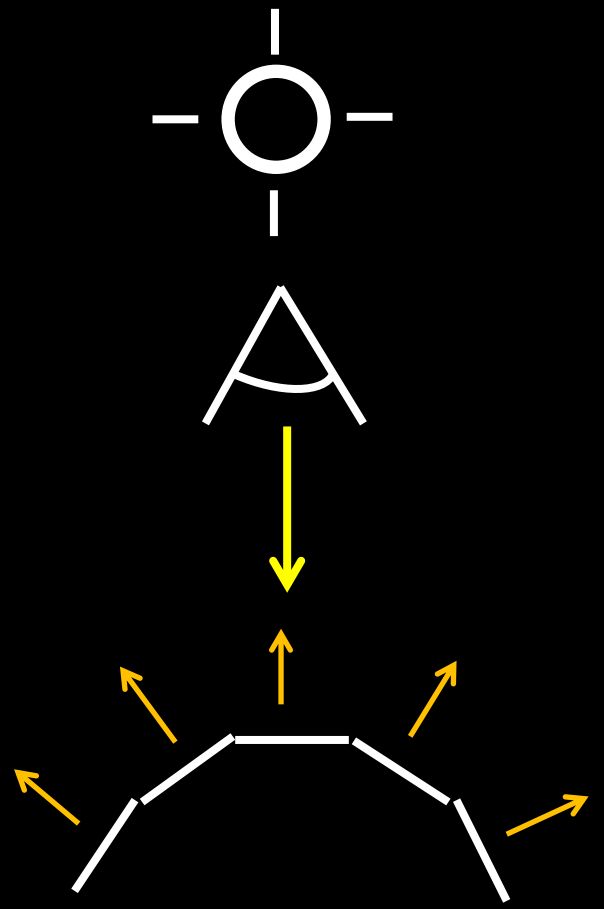
생각해보자

- 이렇게 노말 값을 다르게 준다면



생각해보자

- 결국 이렇게 보이지 않을까?



생각해보자

- 노말 값을 저장할 공간이 필요
- 텍스처를 이용
- RGB 값 대신 노말 값(x,y,z)를 저장한다

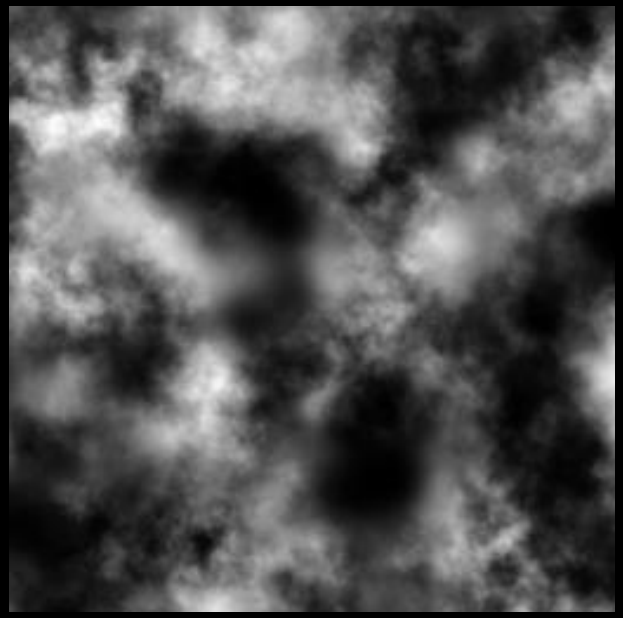
Normal map 생성 방법

어떻게 노말 값을 생성할 것인가?

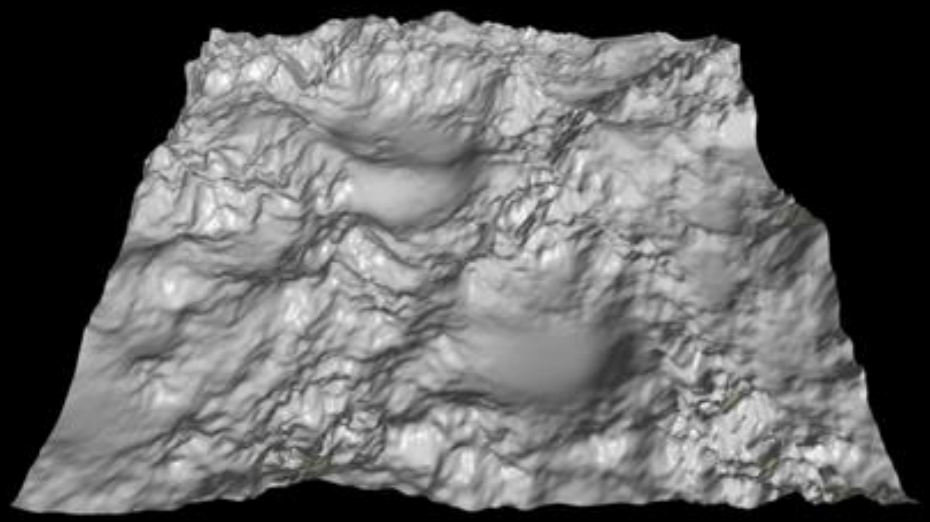
- 높이 맵 이용
- 오브젝트 비교

높이 맵?

- Height map
- 높이를 0~255 값으로 저장한 텍스처



높이 맵

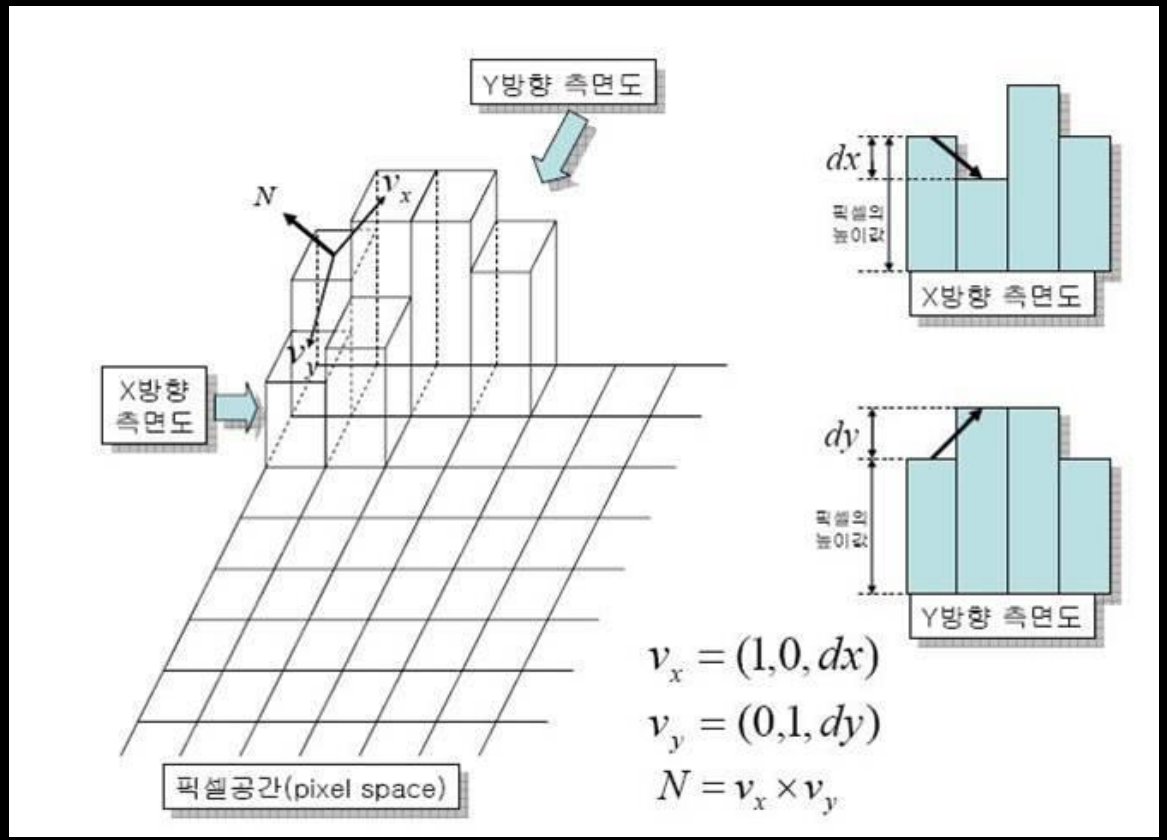


높이 맵으로 생성된 지형

https://mooneegee.blogspot.kr/2015/03/directx9_8.html

높이 맵을 이용한 생성

- 구하고자 하는 UV좌표의 한 칸씩 옆에 있는 높이 값의 평균을 이용해 두 개의 벡터를 만들고 그 벡터들의 외적을 구하는 것이다.



높이 맵을 이용한 생성

$$\frac{\partial h}{\partial x}(x_j, y_i) \approx \frac{h(x_j + \Delta x, y_i) - h(x_j - \Delta x, y_i)}{2\Delta x}$$

$$\frac{\partial h}{\partial y}(x_j, y_i) \approx \frac{h(x_j, y_i + \Delta y) - h(x_j, y_i - \Delta y)}{2\Delta y}$$

$$\vec{T}_x(x_j, y_i) \approx \left(1, 0, \frac{\partial h}{\partial x}(x_j, y_i)\right)$$

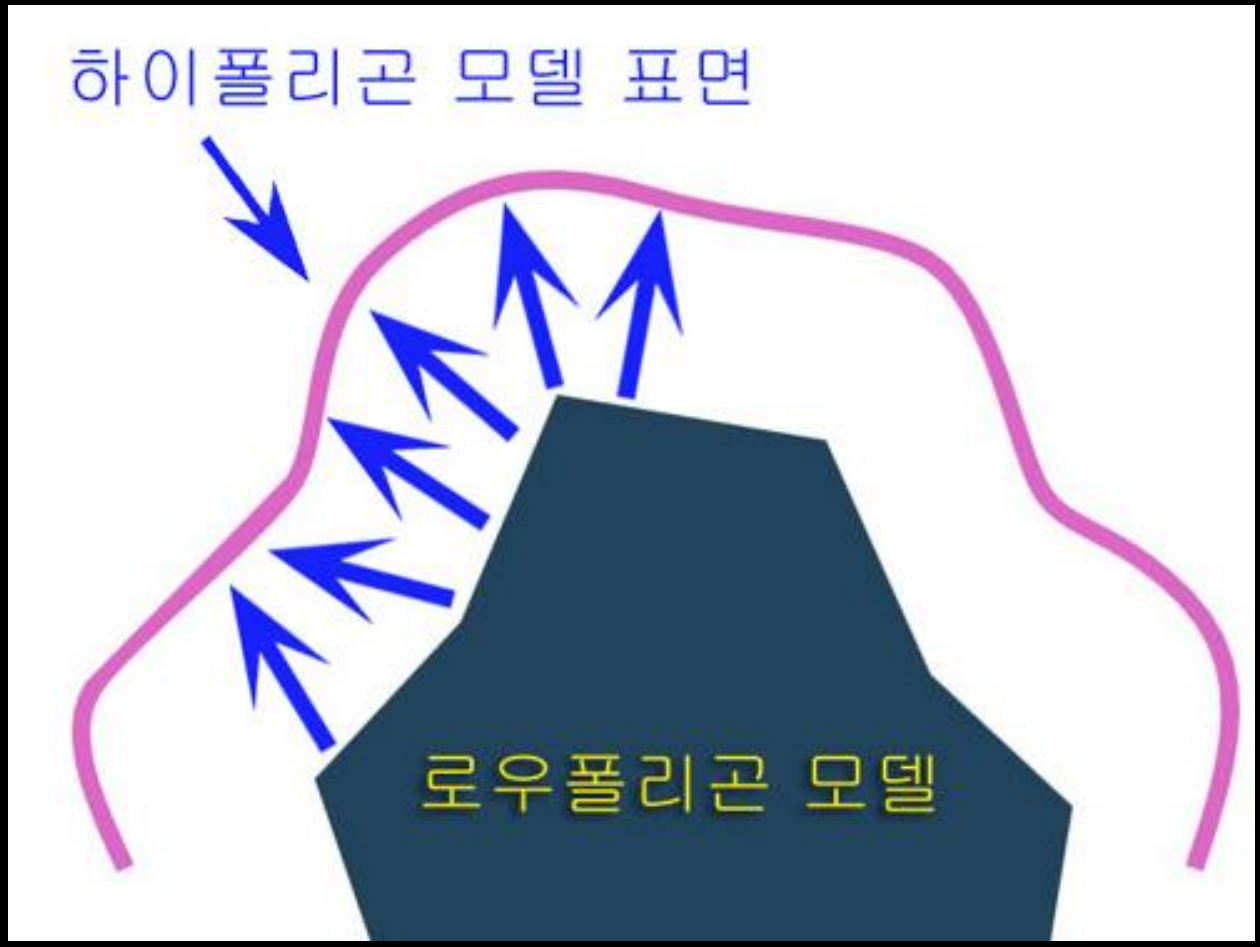
$$\vec{T}_y(x_j, y_i) \approx \left(0, 1, \frac{\partial h}{\partial y}(x_j, y_i)\right)$$

$$\vec{n}(x_j, y_i) = normalize\left(\vec{T}_x \times \vec{T}_y\right)$$

D3DXComputeNormalMap() 함수

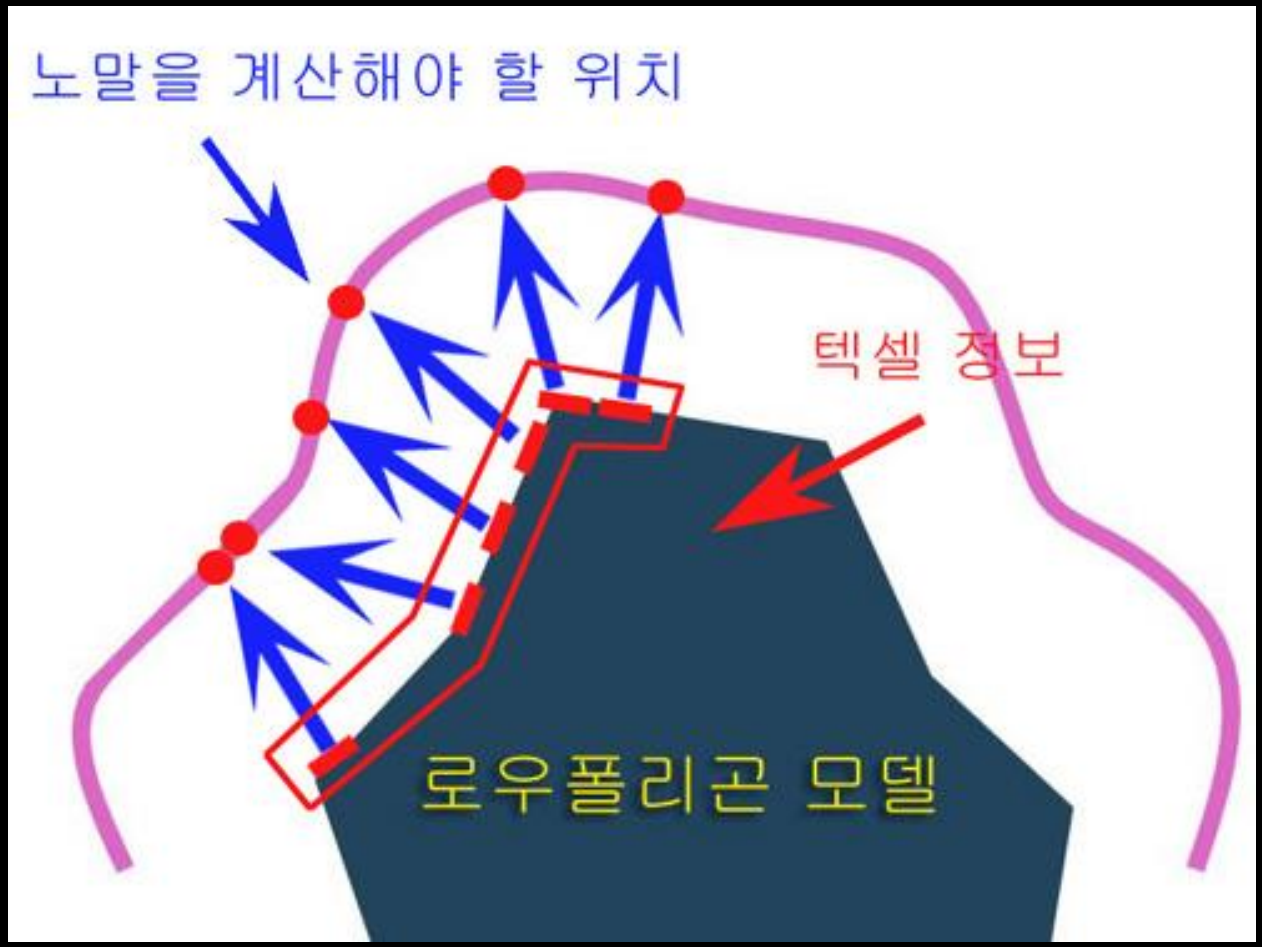
오브젝트 비교

- 로우폴리곤으로부터 하이폴리곤을 향해 반직선을 투영



오브젝트 비교

- 교점(빨간색 점)의 노말 값을 계산하여 텍스처에 저장



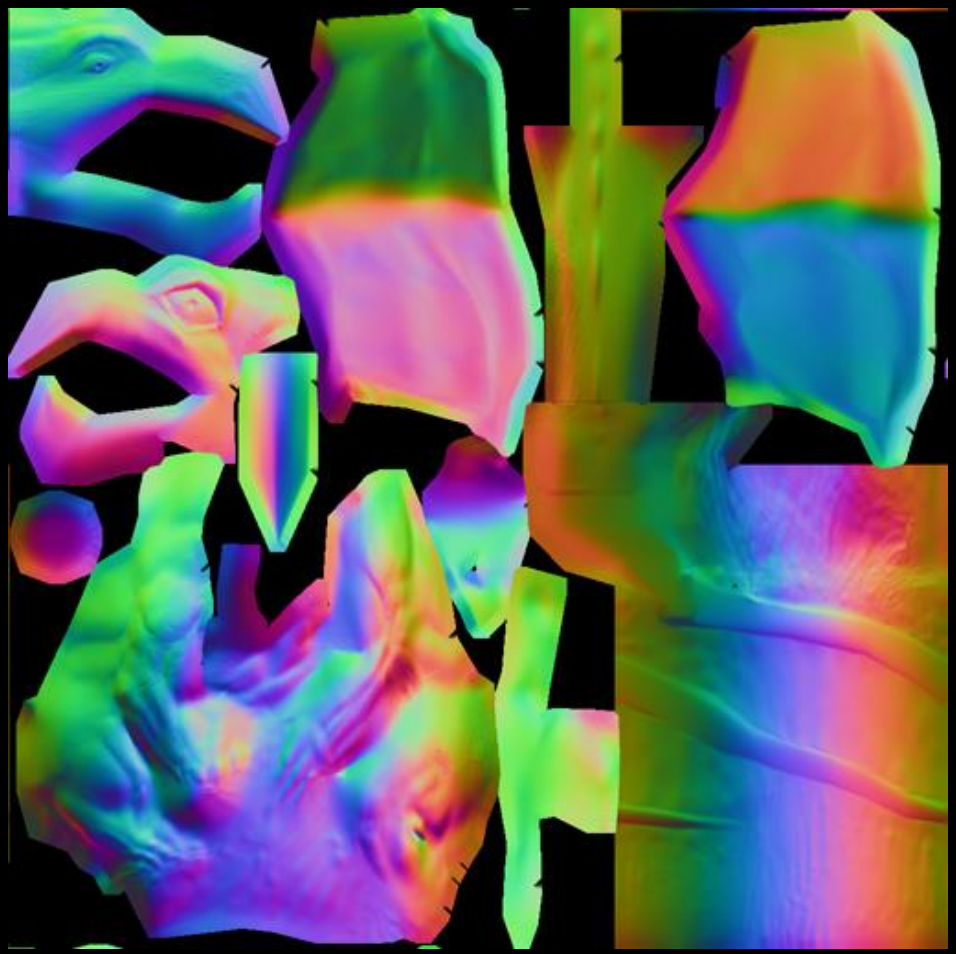
Normal map 종류

- Object space normal map

- Tangent space normal map

Object space normal map

- Local space 좌표계 상에서의 노말을 저장

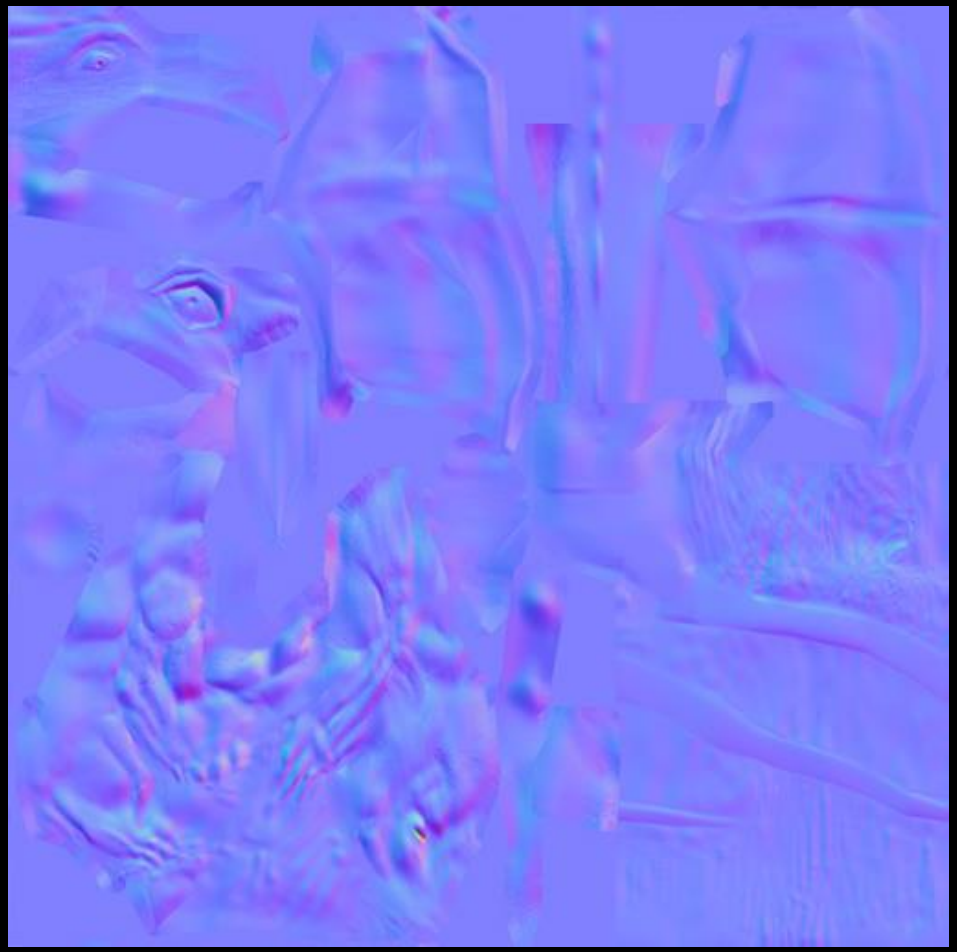


Object space normal map

- 장점
  - 구현이 쉽다
  - 외곽 실루엣을 제외하면 하이폴리곤과 동일한 효과
  - 스키닝(정점 변화) 되지 않는 건물, 자동차에 적절
- 단점
  - 움직이는 물체에 부적절

Tangent space normal map

- 탄젠트(접선) 좌표계 상에서의 노말을 저장



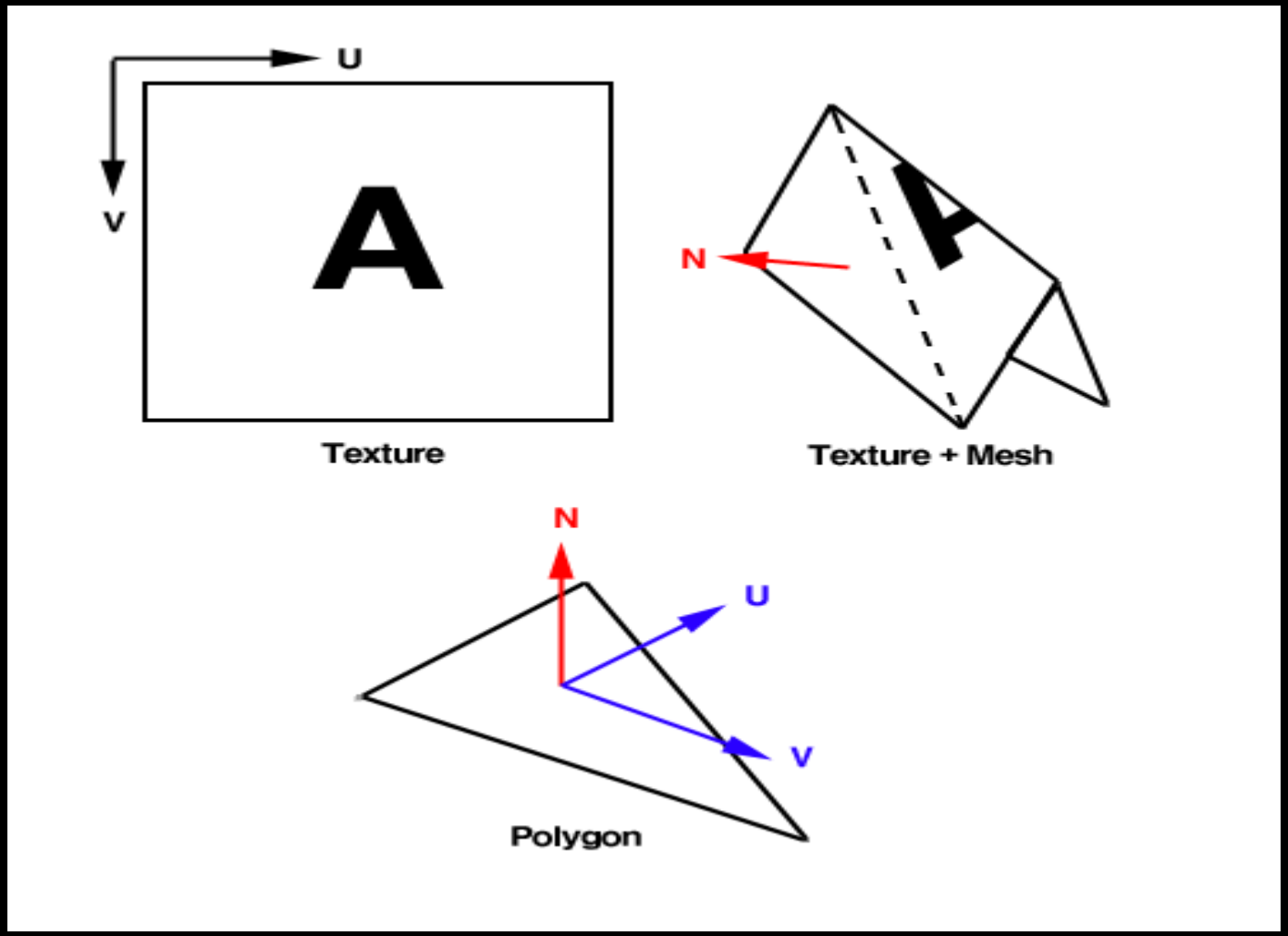
Tangent space?

- 물체의 표면을 기준으로 법선 벡터 방향을 Z축으로 놓는 공간

- T : Tangent vector (접선 벡터)
- B : Binormal vector (종법선 벡터)
- N : normal vector (법선 벡터)

- 이 3개의 벡터를 기준으로 되어있다

Tangent space?



Tangent space?

- 로컬 공간에서 탄젠트 공간으로 변환이 필요
- X -〉 U, Y -〉 V로 변환되어야 한다.
- 변환 행렬 과정(클릭)

Tangent space normal map

- 장점
  - 스키닝 되어도 적용 가능
  - 재사용 가능
- 단점
  - 탄젠트 공간으로 변환하는데 비용 소모
  - Object space normal map보다 퀄리티가 떨어짐

노말 저장

- 텍스처는 R, G, B 공간이 있으므로 각각 X, Y, Z 값을 대입
- 법선 벡터를 정규화하여 단위 벡터(크기가 1인 벡터)로 저장
- 단위 벡터는 (0,0,1)이 될 수도 있지만 (0,0,-1)이 될 수도 있음
- 즉, 정규화된 법선 벡터의 X, Y, Z는 -1~1이다
- 그런데 텍스처에 저장되는 값의 범위는 0~1
- 변환이 필요하다

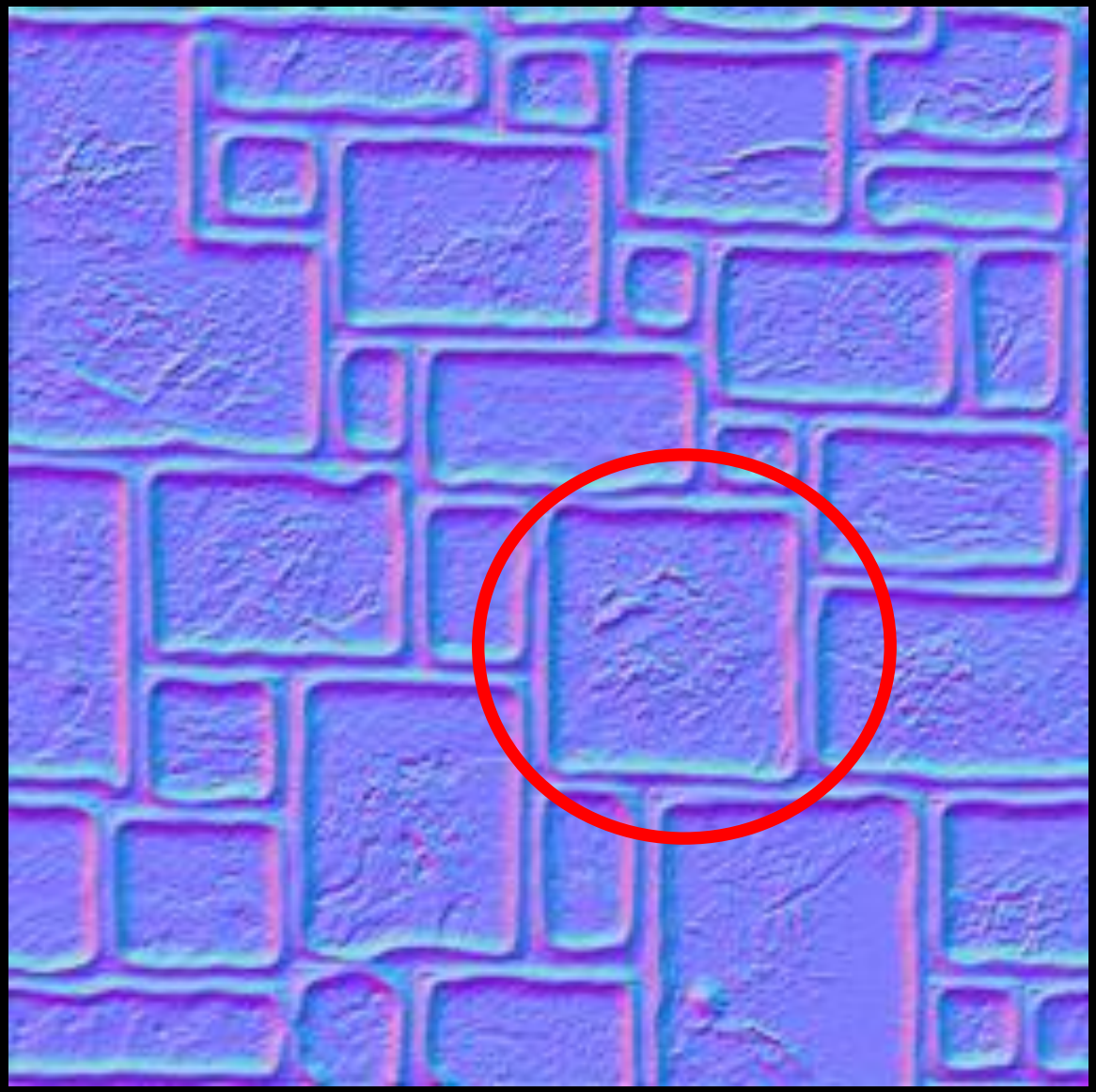
노말 값 변환

- 법선 벡터(XYZ) = -1~1
- 법선 벡터(XYZ) * 0.5 = -0.5~0.5
- 법선 벡터(XYZ) * 0.5 + 0.5 = 0~1
  = 법선맵(RGB)

계산해보자

계산해보자

원 부분을 계산해보자

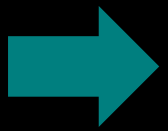


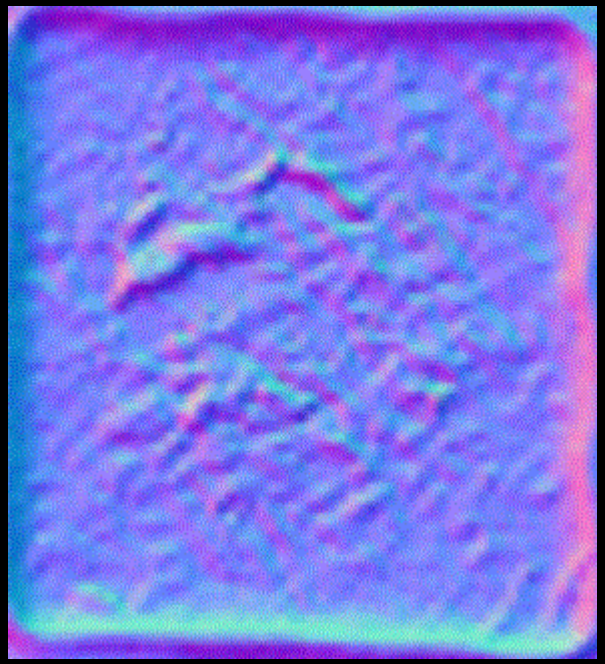
계산해보자

왼쪽 법선 벡터(-1,0,0)

변환 후 RGB(0, 0.5, 0.5)

RGB(0, 127, 127)



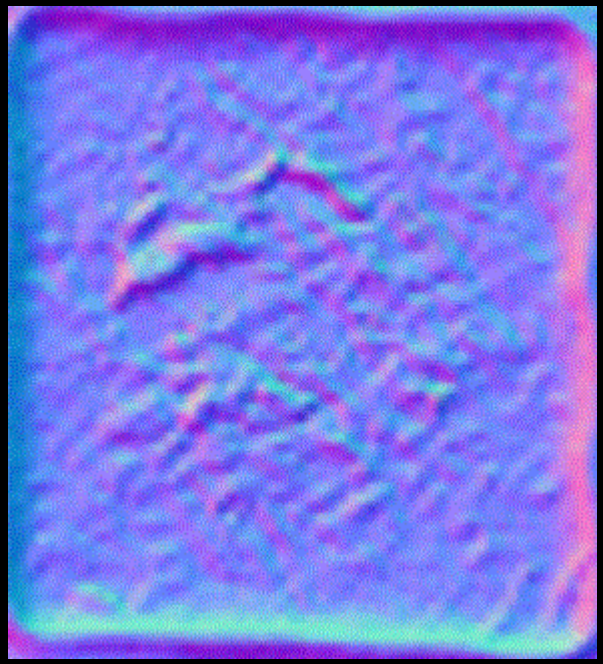


계산해보자

오른쪽 법선 벡터(1,0,0)

변환 후 RGB(1, 0.5, 0.5)

RGB(255, 127, 127)

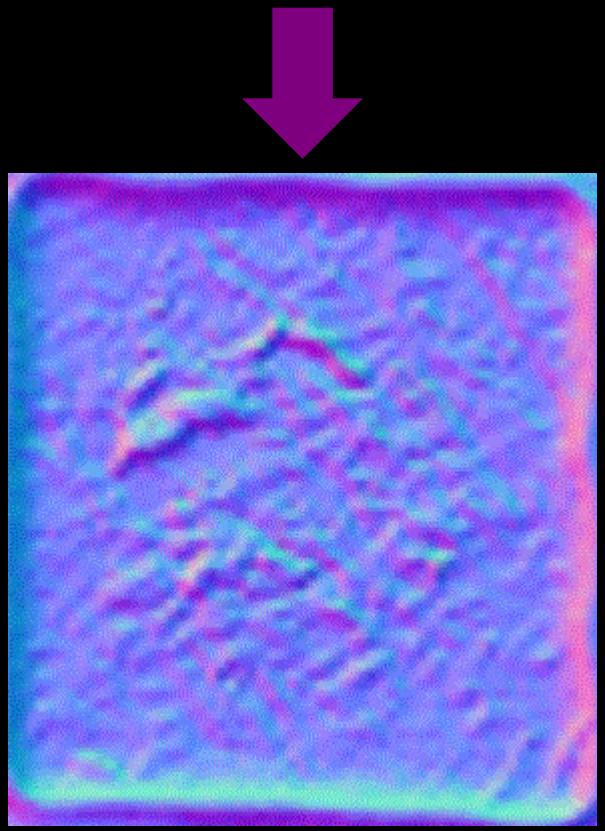


계산해보자

위쪽 법선 벡터(0,-1,0)

변환 후 RGB(0.5, 0, 0.5)

RGB(127, 0, 127) 색상

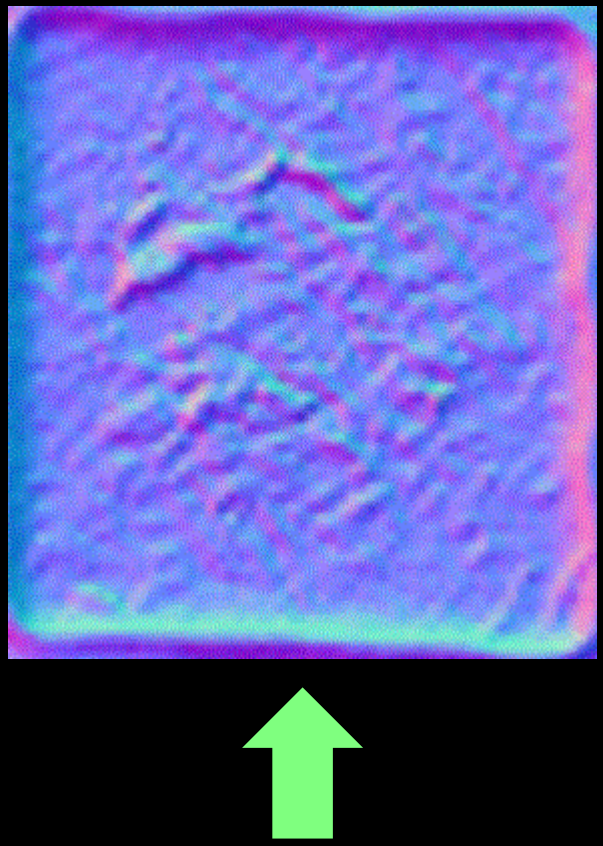


계산해보자

아래쪽 법선 벡터(0,1,0)

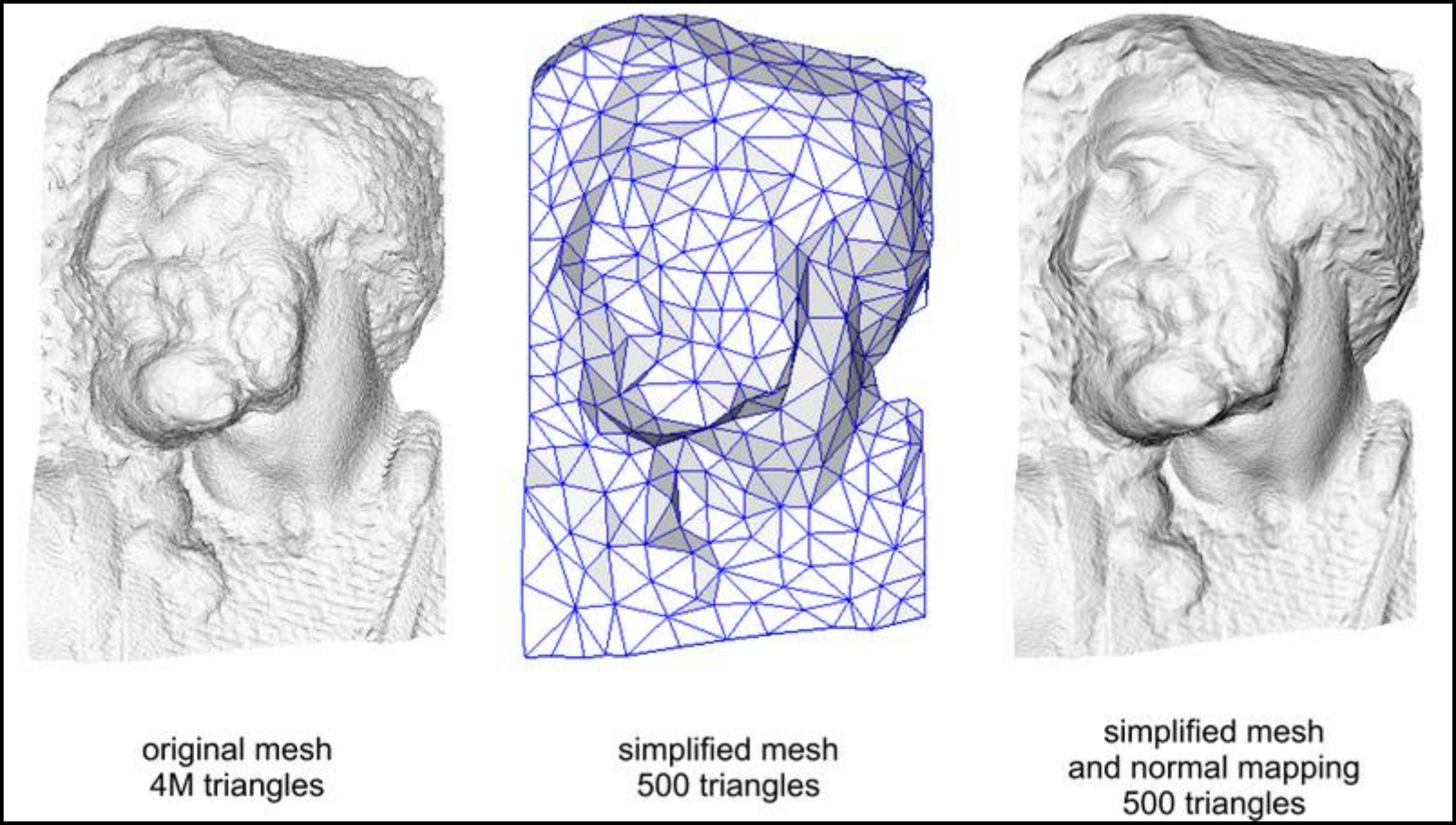
변환 후 RGB(0.5, 1, 0.5)

RGB(127, 255, 127) 색상



Parallax mapping

노말 맵핑의 효과


original mesh
4M triangles
simplified mesh
500 triangles
simplified mesh
and normal mapping
500 triangles

노말 맵핑의 효과

그런데 말입니다

노말 맵핑의 단점

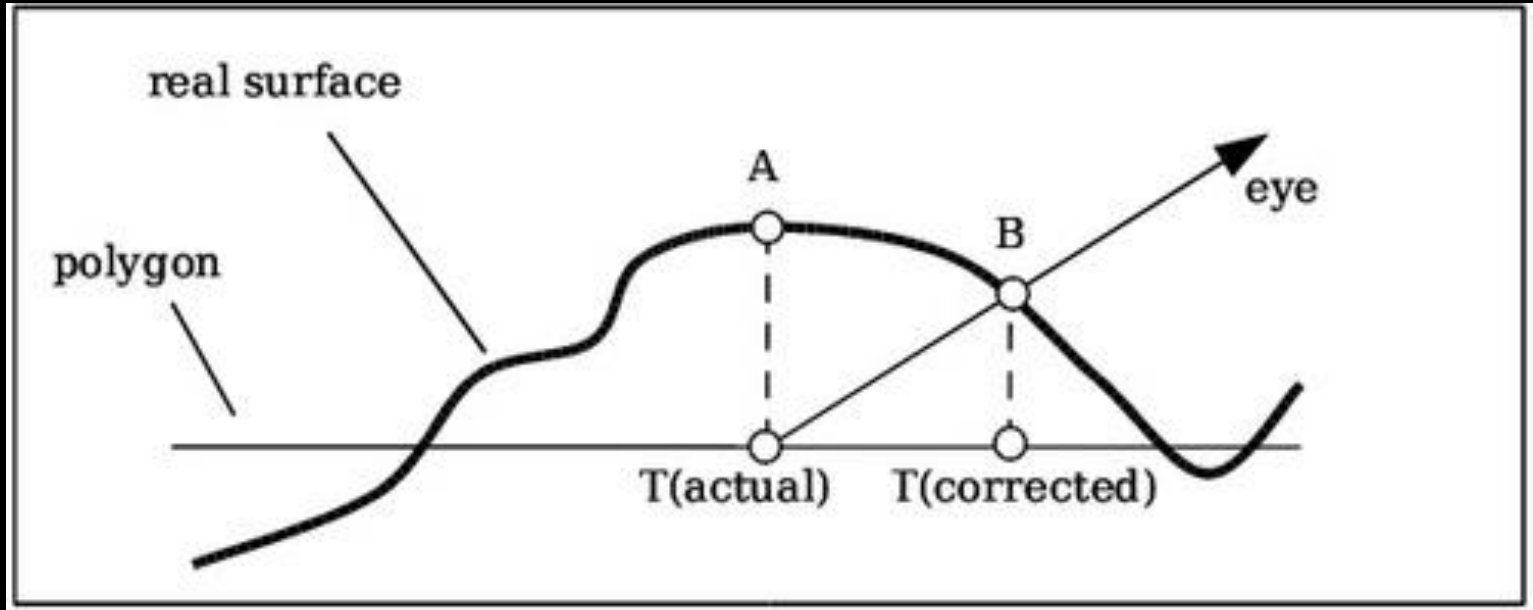


Figure 1. The observed texel is incorrect because the texture map has been flattened onto the polygon.

- 실제로는 평면이기 때문에 굴곡의 높이를 고려할 수 없다
- B가 보여야 하는데 A가 보임
- 거리가 가까울수록 or 사각으로 갈수록 부자연스럽게 보인다

Parallax mapping

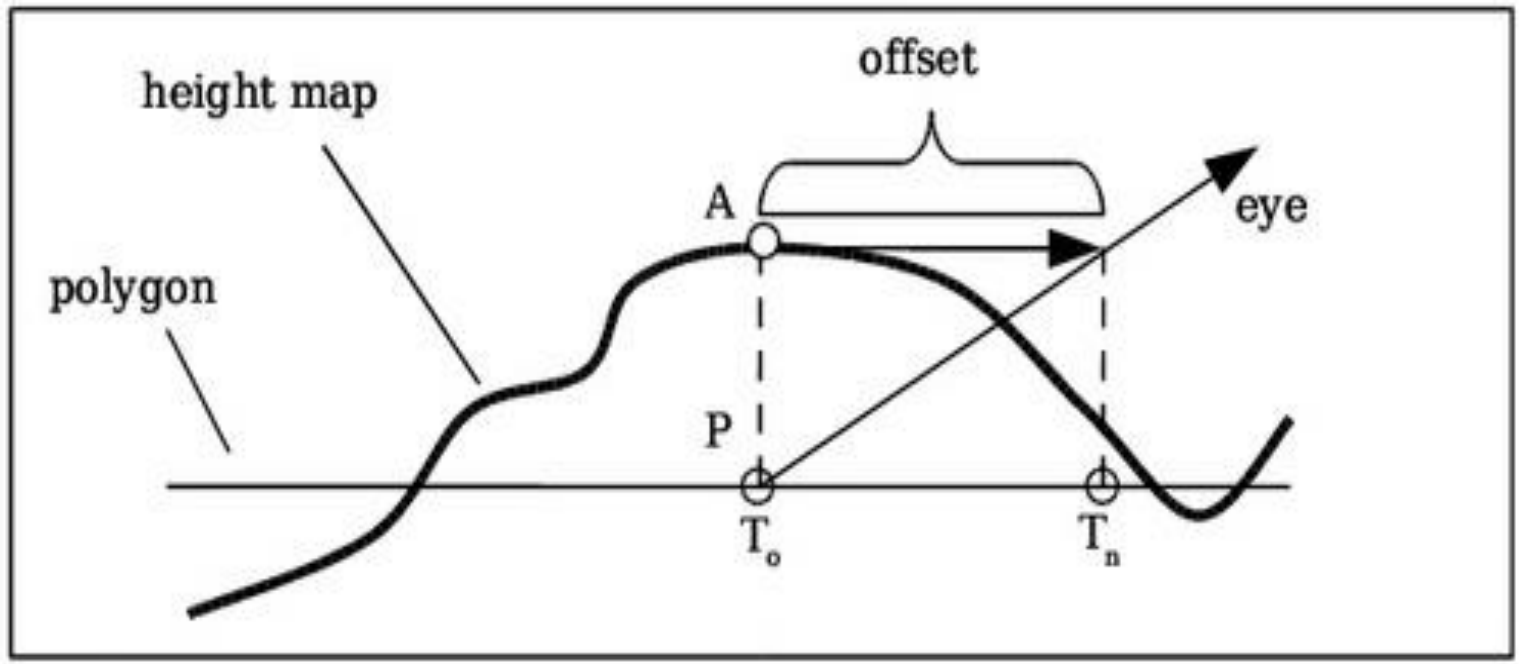
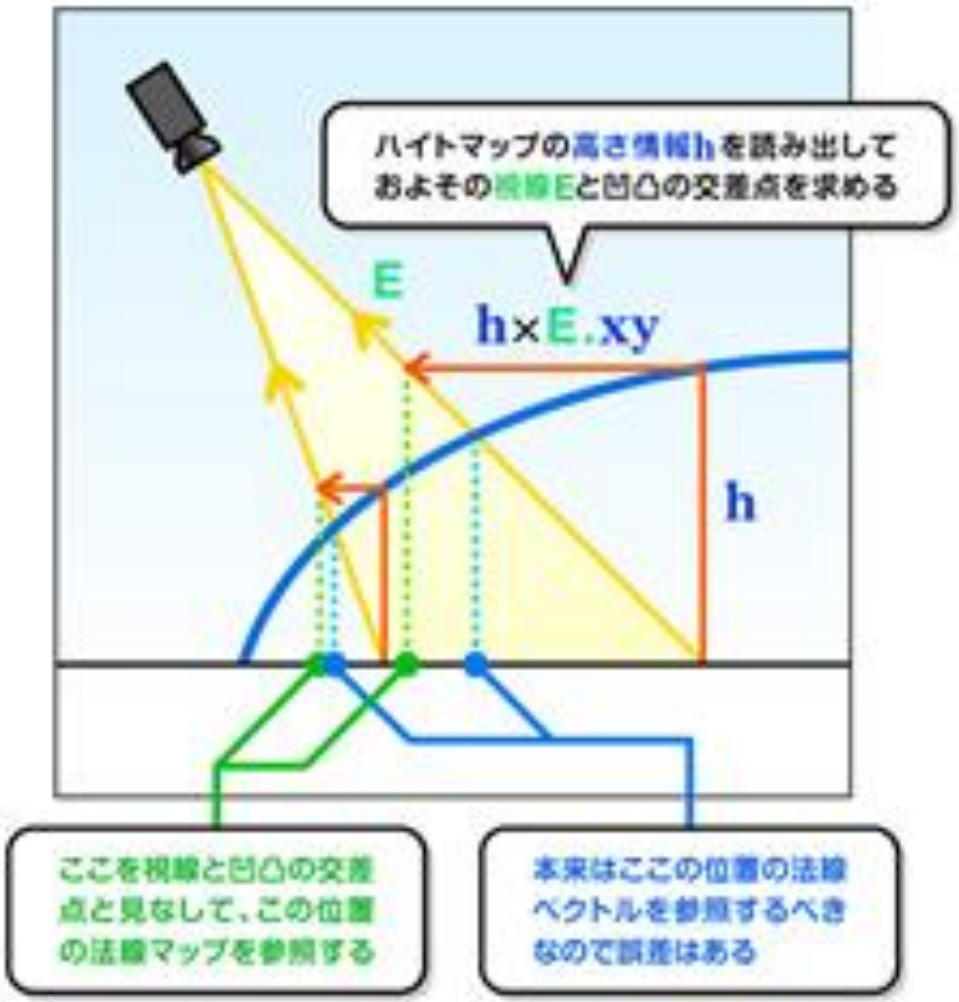


Figure 2. Calculating the offset.

- 시차 맵핑이라 불림
- 높이 맵을 이용하여 대략적으로 시선과 요철(볼록,오목)이 교차하는 지점을 구한다
- 오차가 있다

Parallax mapping



- 여기를 시선과 요철의 교차점으로 보고, 이 위치의 법선맵을 참조한다.
- 원래는 여기 위치의 법선 벡터를 참조해야 하지만 오차는 있다.
- H x E.xy
- 높이맵의 높이정보 h를 읽어서 대략적으로 시선 E와 요철이 교차하는 지점을 구한다.

노말 맵핑과 비교

노말 맵핑만 적용

시차 맵핑 적용

노말 맵핑과 비교

노말 맵핑만 적용

시차 맵핑 적용

알아야할 점

- 노말 맵핑과 시차 맵핑 모두 픽셀 당 계산일 뿐 버텍스에는 어느 영향도 끼치지 않는다. 즉 버텍스가 많아지거나 위치가 변화하는 것은 아니다

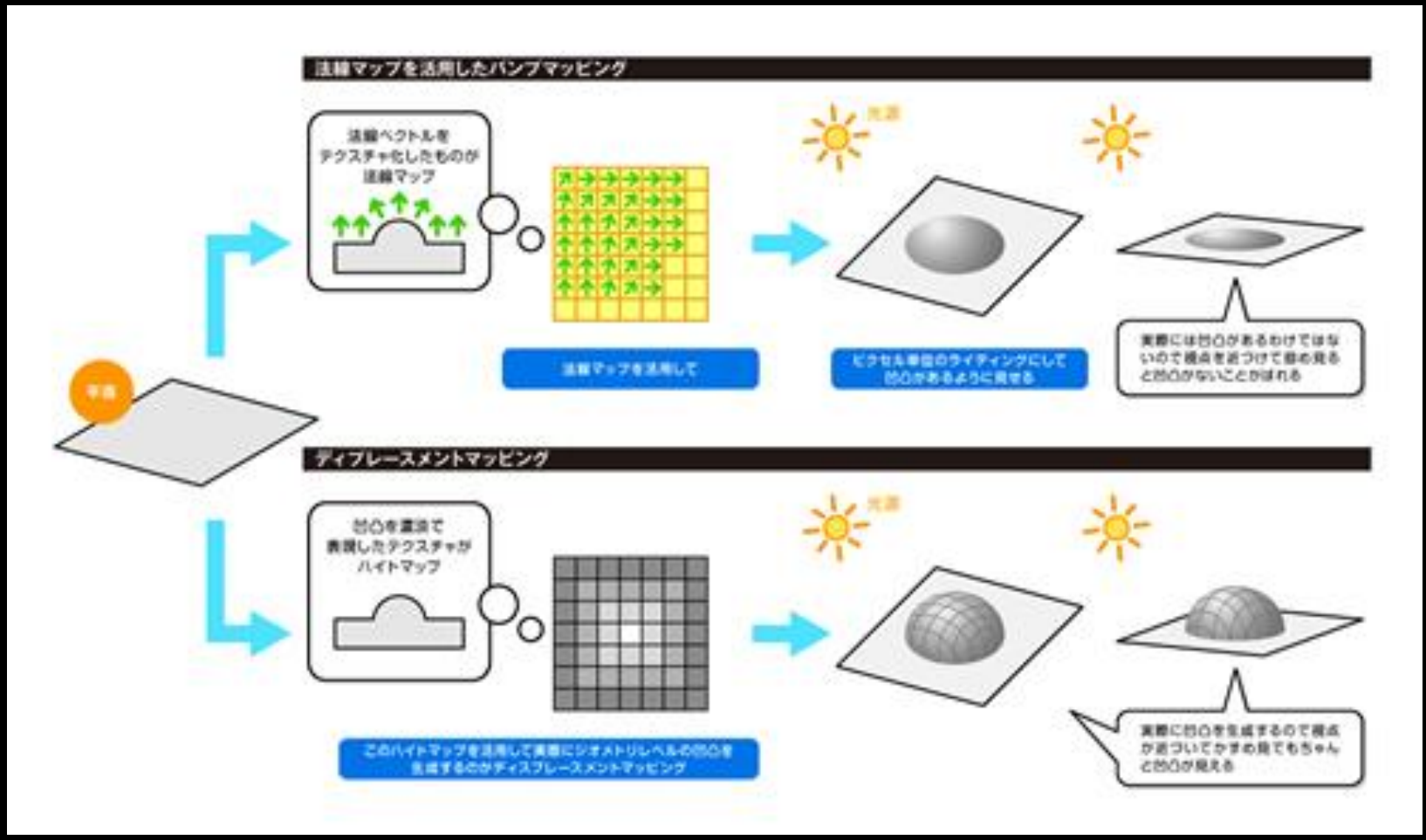
- 개량형으로 Parallax Occlusion Mapping 이란 것도 있음

Displacement mapping

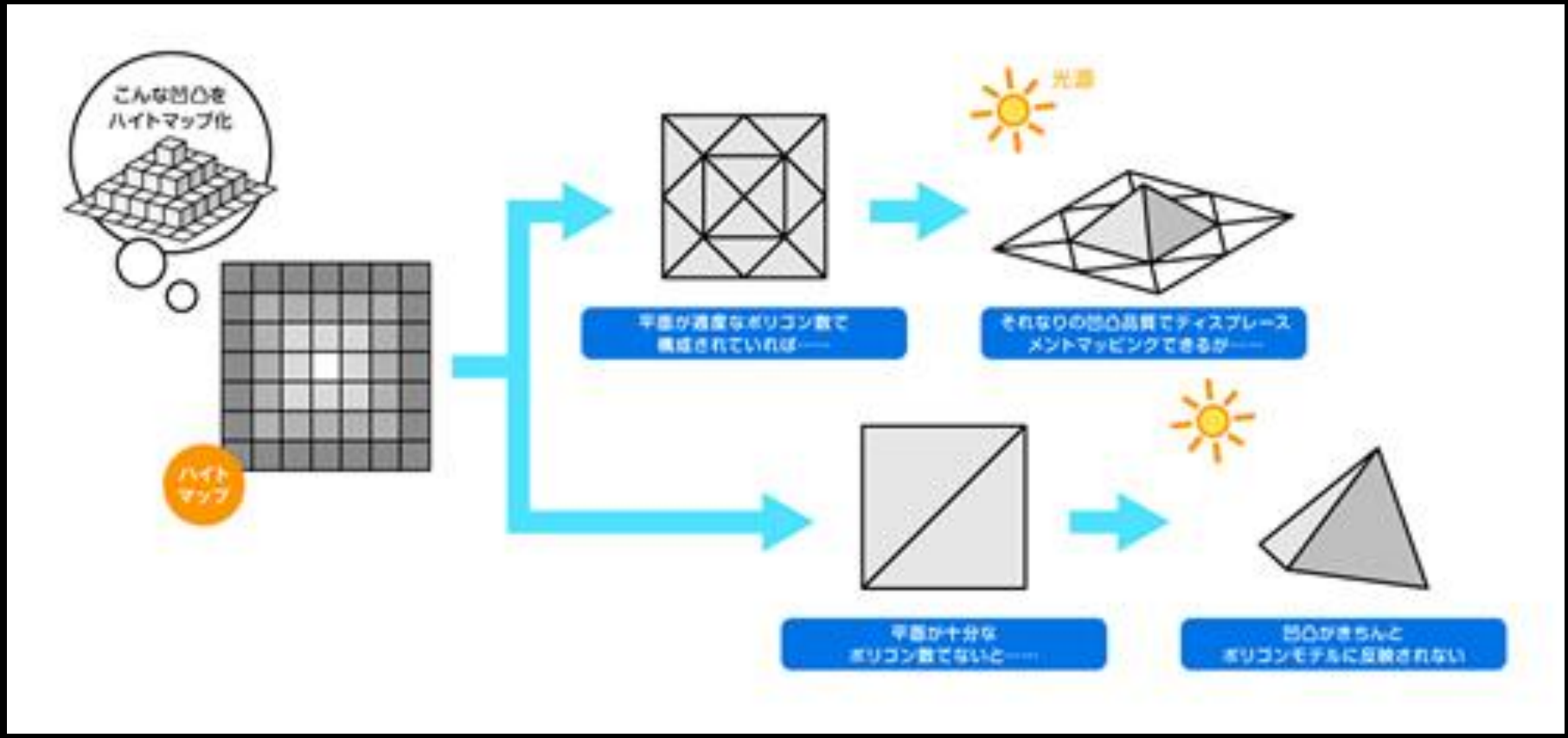
Displacement mapping

- 변위 맵핑이라 불림
- 노말 맵핑과 시차 맵핑은 아무리 북치고 장구쳐도 실제로는 평면이기에 굴곡이 없는 것이 드러난다
- 높이 맵에 따라서 3D 모델을 변위(displace) 시키는 기술
- 정점 셰이더에서 텍스처를 읽어내 정점을 변위시키는 작업을 수행

노말 맵핑과 차이

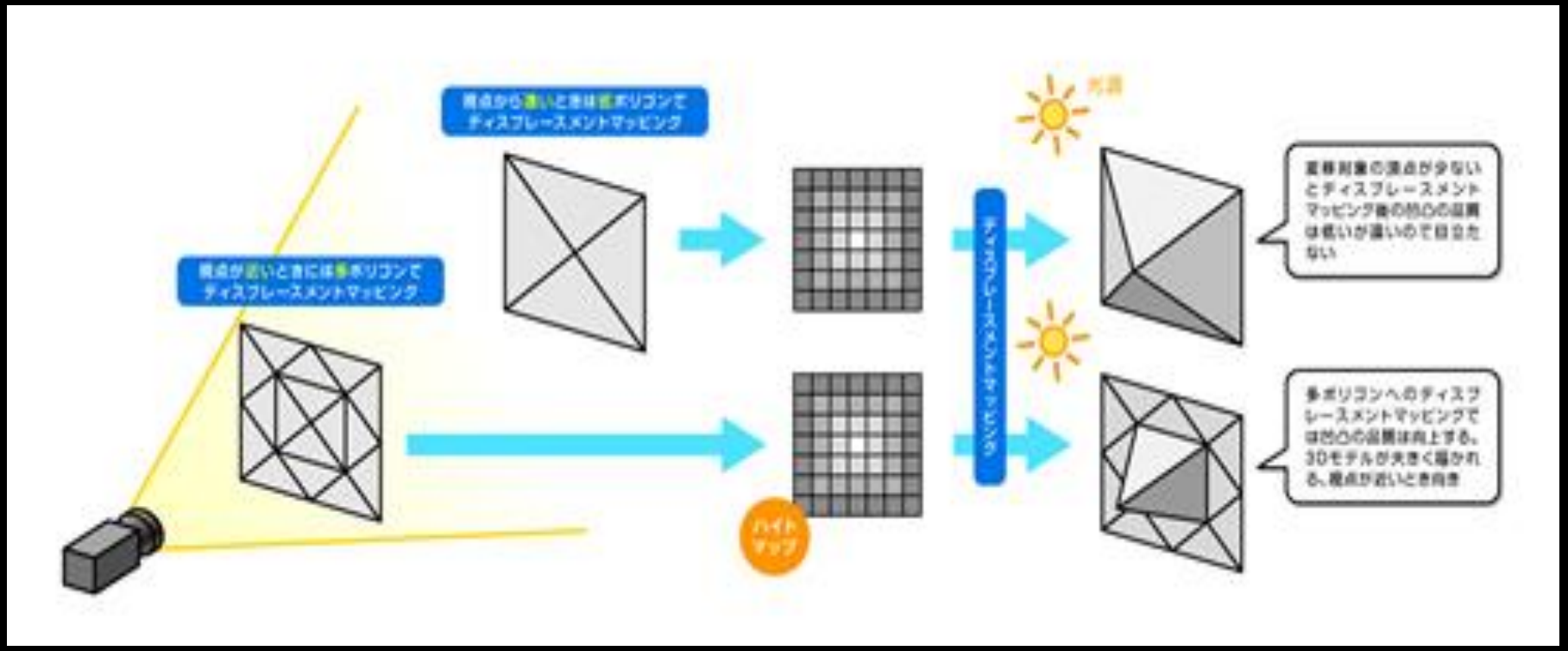

法線マップを活用したバンプマッピング
法線ベクトルを
テクスチャ化したものが
法線マップ
光源
平面
ディプレースメントマッピング
凹凸を濃淡で
表現したテクスチャが
ハイトマップ
光源

조건



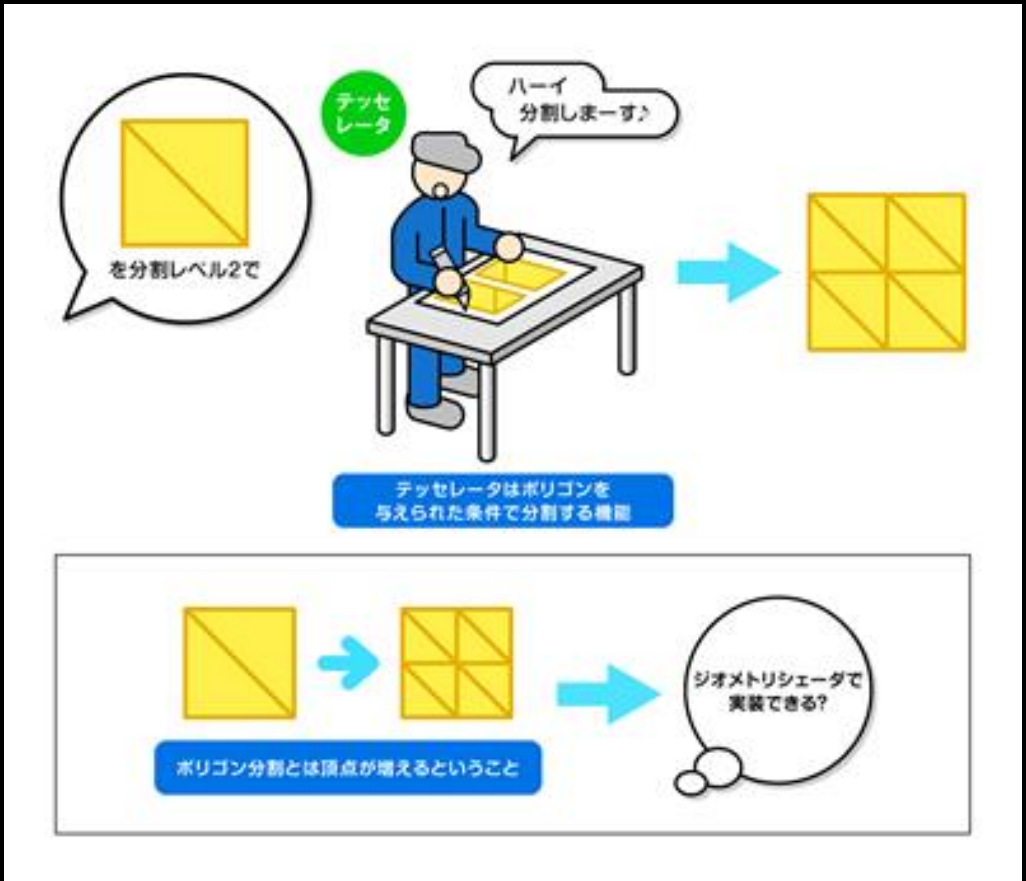
- 높이 맵과 폴리곤 수의 밸런스가 중요
- 밸런스가 맞지 않으면 정확하게 폴리곤 모델에 반영되지 않음

이상적 상황



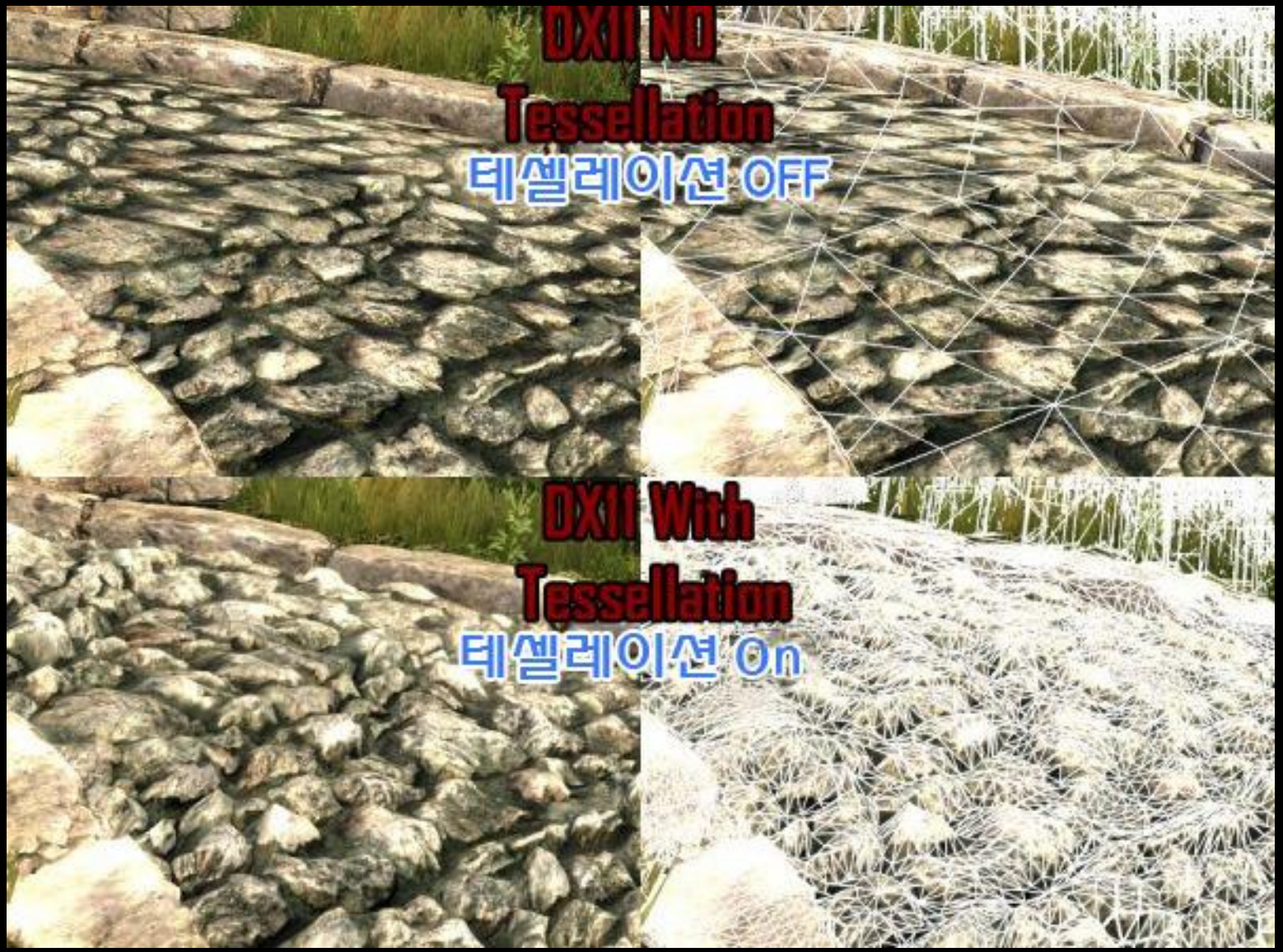
- 시점으로부터의 거리에 따라 필요한 폴리곤 분할을 실시한 후 변위 맵핑을 하는 것이 이상적

테셀레이션(Tessellation)



- 폴리곤을 쪼개는 기술. 버텍스가 증가한다
- DirectX 11, OpenGL 4.0부터 지원(그 이전부터 있었긴 한데 비표준)

비교


DX11 NO
Tessellation
테셀레이션 OFF
DX11 With
Tessellation
테셀레이션 On

궁금한 점

- 쪼갠 후 다시 합치지는 않나??

출처

- Normal mapping
  - https://namu.wiki/w/%EB%85%B8%EB%A7%90%EB%A7%B5%ED%95%91
  - http://dolphin.ivyro.net/file/shader9.0/tutorial04.html
  - http://rapapa.net/?p=2419
  - http://egloos.zum.com/piglove/v/4834666
  - http://mgun.tistory.com/1342
  - http://www.3dkingdoms.com/tutorial.htm
  - http://blog.naver.com/jungjin02/20040252755
  - http://zho.pe.kr/view.html?file_name=doc/normalmap.txt
  - http://ppparkje.tistory.com/35
  - http://cafe.naver.com/devrookie/4470
  - http://cafe.naver.com/devrookie/1210

출처

- Parallax mapping
  - http://egloos.zum.com/chulin28ho/v/3749030
  - http://x66vx.egloos.com/3705266
  - http://egloos.zum.com/wrice/v/4953158
  - http://cybershin.tistory.com/53
  - https://docs.unrealengine.com/udk/Three/DevelopmentKitGemsParallaxOccludedMappingKR.html
- Displacement mapping
  - http://kr.nvidia.com/object/tessellation_kr.html
  - http://allosha.tistory.com/37
- Tangent space
  - http://blog.naver.com/elsacred/50015540325

출처

- Texture or Texel
  - http://kblog.popekim.com/2011/12/03-part-1.html
  - http://cafe.naver.com/devrookie/1169

Q / A

Tangent space?

삼각형의 각 버텍스를 P0 , P1 , P2로 생각하자.

그리고 각각 버텍스의 UV좌표를(u0 , v0) , (u1 , v1) , (u2 , v2)로 생각하자.

이 삼각형 안의 한 점을 Q로 보고 UV좌표를(u , v)로 생각하자.

탄젠트 공간으로 변환해주는 행렬은 다음과 같다.

이 행렬을 구해보자.

$$M_{\Delta} = \begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \\ N_x & N_y & N_z \end{bmatrix}$$

Tangent space?

$$\mathbf{Q} - \mathbf{P}_0 = (u - u_0)\mathbf{T} + (v - v_0)\mathbf{B},$$

$$\mathbf{Q}_1 = \mathbf{P}_1 - \mathbf{P}_0$$

$$\mathbf{Q}_2 = \mathbf{P}_2 - \mathbf{P}_0$$

$$(s_1, t_1) = (u_1 - u_0, v_1 - v_0)$$

$$(s_2, t_2) = (u_2 - u_0, v_2 - v_0)$$

$$\mathbf{Q}_1 = s_1\mathbf{T} + t_1\mathbf{B}$$

$$\mathbf{Q}_2 = s_2\mathbf{T} + t_2\mathbf{B}$$

Tangent space?

$$\begin{bmatrix} (\mathbf{Q}_1)_x & (\mathbf{Q}_1)_y & (\mathbf{Q}_1)_z \\ (\mathbf{Q}_2)_x & (\mathbf{Q}_2)_y & (\mathbf{Q}_2)_z \end{bmatrix} = \begin{bmatrix} s_1 & t_1 \\ s_2 & t_2 \end{bmatrix} \begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \end{bmatrix}$$

$$\begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \end{bmatrix} = \frac{1}{s_1 t_2 - s_2 t_1} \begin{bmatrix} t_2 & -t_1 \\ -s_2 & s_1 \end{bmatrix} \begin{bmatrix} (\mathbf{Q}_1)_x & (\mathbf{Q}_1)_y & (\mathbf{Q}_1)_z \\ (\mathbf{Q}_2)_x & (\mathbf{Q}_2)_y & (\mathbf{Q}_2)_z \end{bmatrix}$$

Tangent space?

정점들의 T , B 벡터들의 평균을 이용해서 계산하면 직교하지 않을 수 있다.

안전하게 그램 - 슈미트 방법으로 직교하는 기저를 만들어 준다.

$$T' = T - (N \cdot T)N$$
$$B' = B - (N \cdot B)N - (T' \cdot B)T'/T'^2$$

D3DXComputeTangent() 사용

Tangent space?

The Gram–Schmidt process

$$\operatorname{proj}_{\mathbf{u}}(\mathbf{v}) = \frac{\langle \mathbf{v}, \mathbf{u} \rangle}{\langle \mathbf{u}, \mathbf{u} \rangle} \mathbf{u},$$

$$\begin{aligned}
\mathbf{u}_1 &= \mathbf{v}_1, & \mathbf{e}_1 &= \frac{\mathbf{u}_1}{\|\mathbf{u}_1\|} \\
\mathbf{u}_2 &= \mathbf{v}_2 - \operatorname{proj}_{\mathbf{u}_1}(\mathbf{v}_2), & \mathbf{e}_2 &= \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|} \\
\mathbf{u}_3 &= \mathbf{v}_3 - \operatorname{proj}_{\mathbf{u}_1}(\mathbf{v}_3) - \operatorname{proj}_{\mathbf{u}_2}(\mathbf{v}_3), & \mathbf{e}_3 &= \frac{\mathbf{u}_3}{\|\mathbf{u}_3\|} \\
\mathbf{u}_4 &= \mathbf{v}_4 - \operatorname{proj}_{\mathbf{u}_1}(\mathbf{v}_4) - \operatorname{proj}_{\mathbf{u}_2}(\mathbf{v}_4) - \operatorname{proj}_{\mathbf{u}_3}(\mathbf{v}_4), & \mathbf{e}_4 &= \frac{\mathbf{u}_4}{\|\mathbf{u}_4\|} \\
&\vdots & &\vdots \\
\mathbf{u}_k &= \mathbf{v}_k - \sum_{j=1}^{k-1} \operatorname{proj}_{\mathbf{u}_j}(\mathbf{v}_k), & \mathbf{e}_k &= \frac{\mathbf{u}_k}{\|\mathbf{u}_k\|}.
\end{aligned}$$