

Real-Time Rendering

230127 19101188 고은수

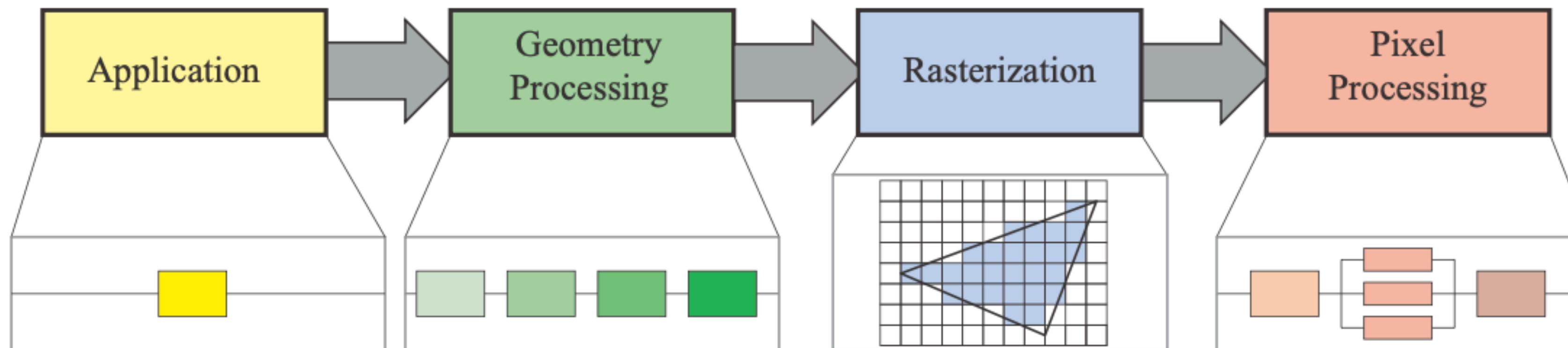
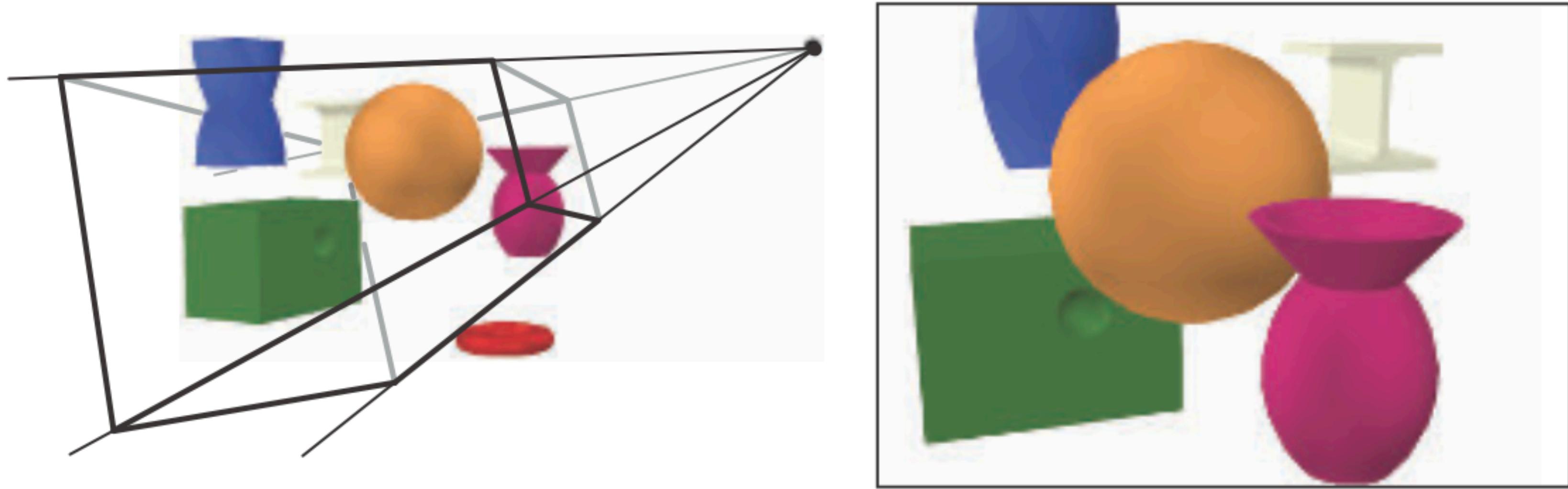
Contents

1. Introduction
2. The Graphics Rendering Pipeline
3. The Graphics Processing Unit
4. Transforms

Introduction



The Graphics Rendering Pipeline



Optional Vertex Processing

1. Tessellation

가까우면 많이, 멀면 적게 생성

2. Geometry shader

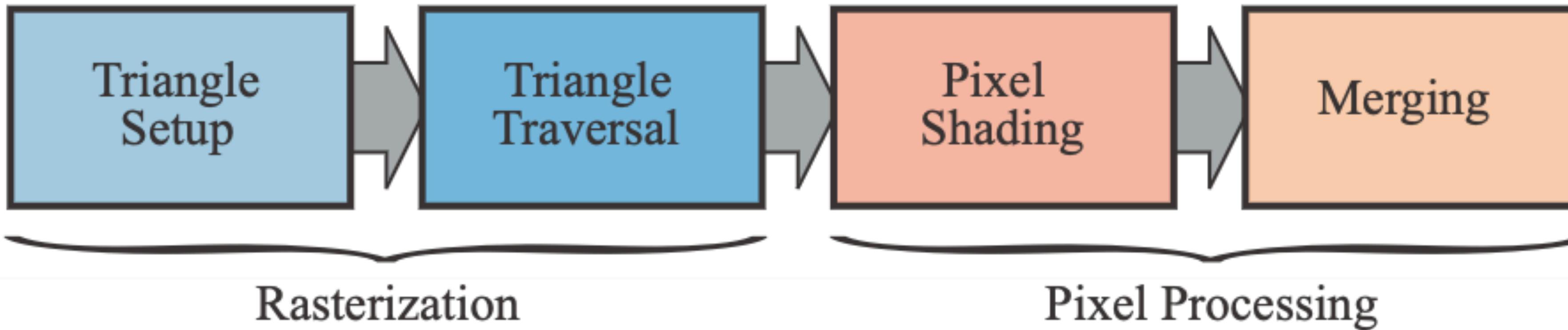
vertex를 입력으로 받아 다음 쉐이더로 보내기 전에 적절한 형태로 변환, 추가 생성

3. Stream output

파이프라인의 다음 부분으로 보내는 대신 추가 처리를 위해 배열로 출력 가능

Rasterization

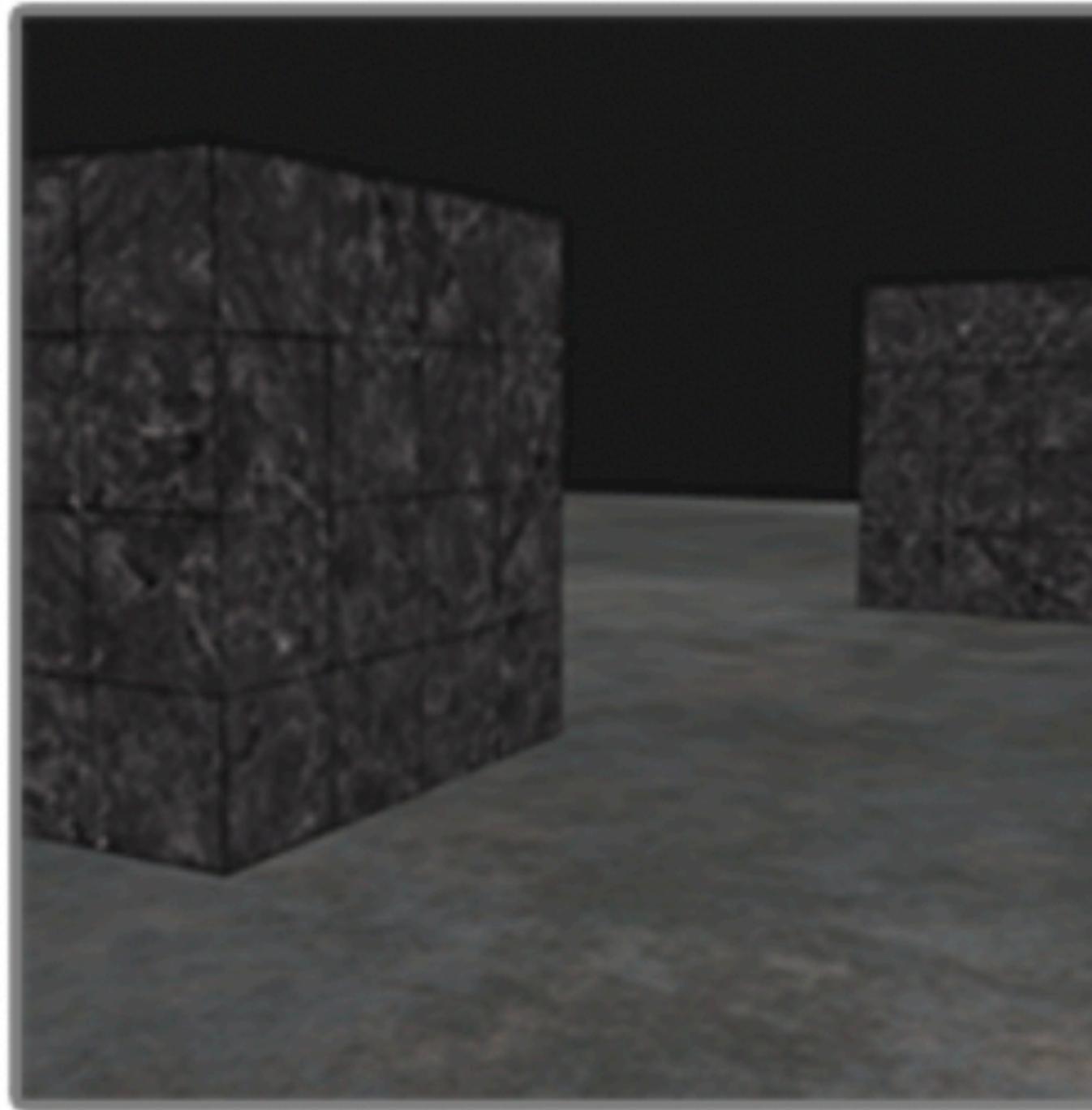
1. Triangle Setup - 삼각형의 미분, 엣지 방정식 등 데이터 계산
2. Triangle Traversal - 삼각형 안의 샘플이나 픽셀을 찾음
3. Pixel Shading - 픽셀에 색상 넣음 (텍스쳐링)
4. Merging - color buffer, z-buffer, alpha-buffer, stencil buffer, frame buffer 등



Stencil Buffer

Fragment가 폐기될지 안될지 테스트

일반적으로 8비트의 stencil value를 가지고 이 값은 pixel마다 256개의 값으로 나타남

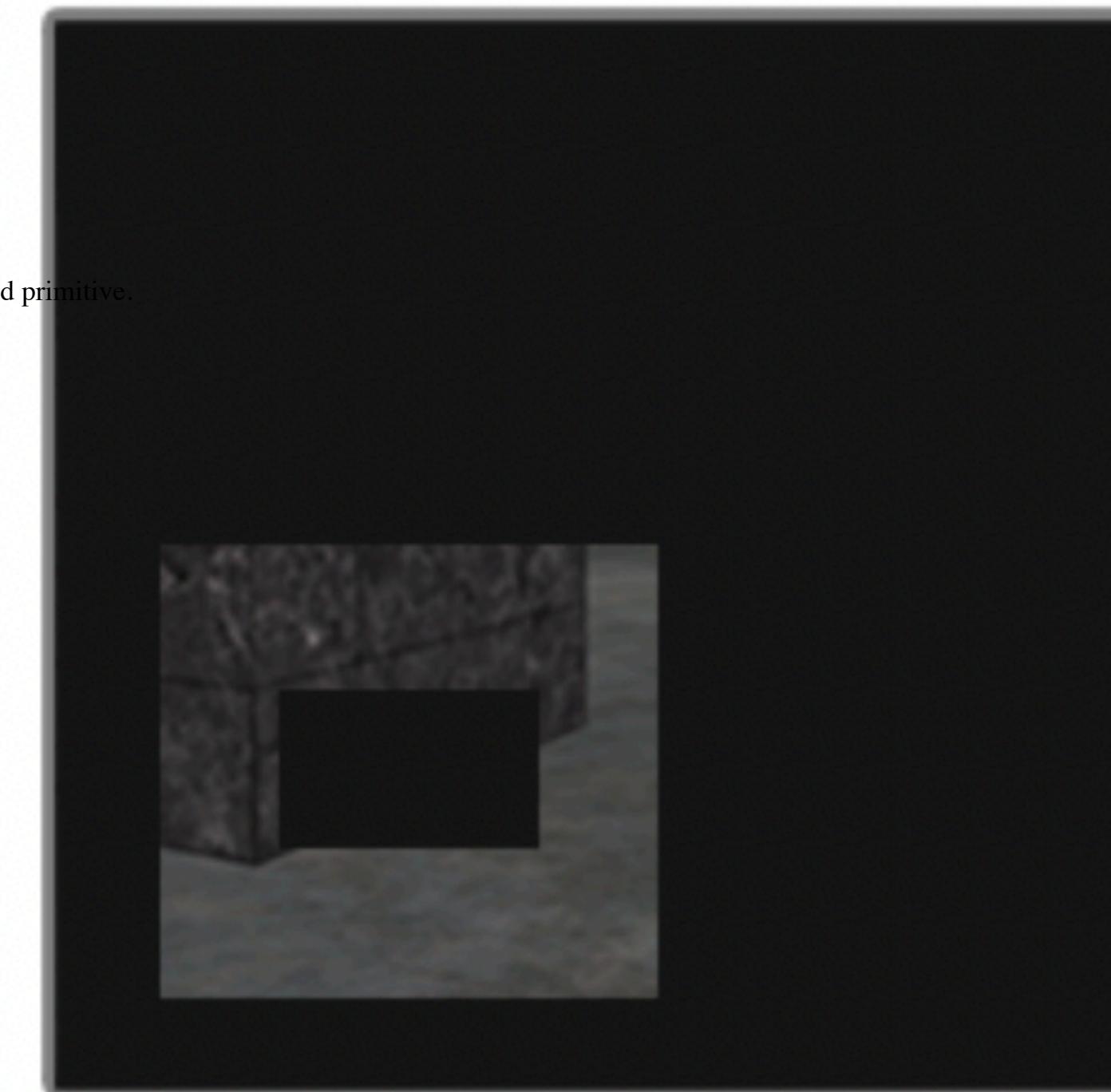


Color buffer

0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0111111110000000
0111111110000000
0110000110000000
0110000110000000
0111111110000000
0111111110000000
0000000000000000

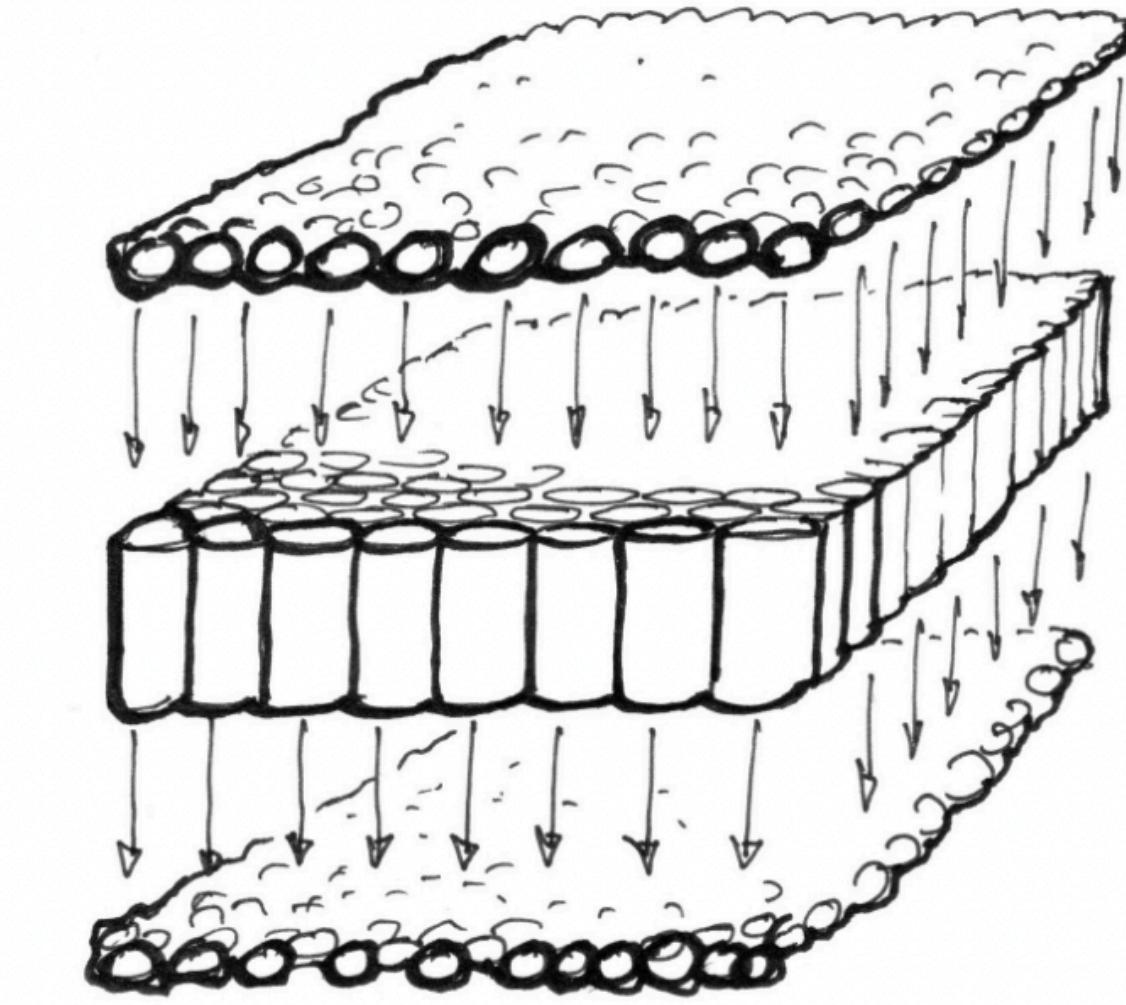
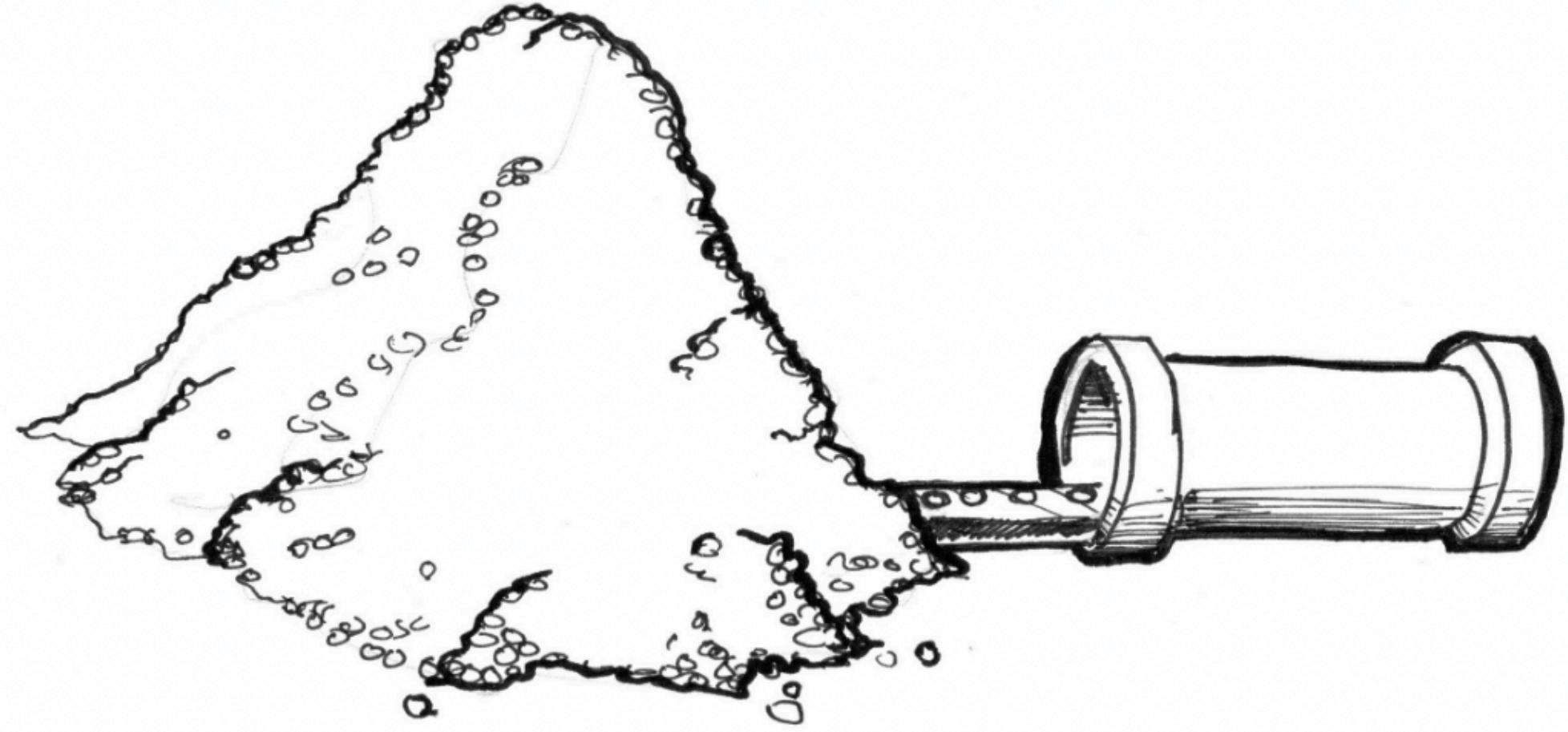
The stencil buffer is an offscreen buffer used to record the locations of the rendered primitive.

Stencil buffer

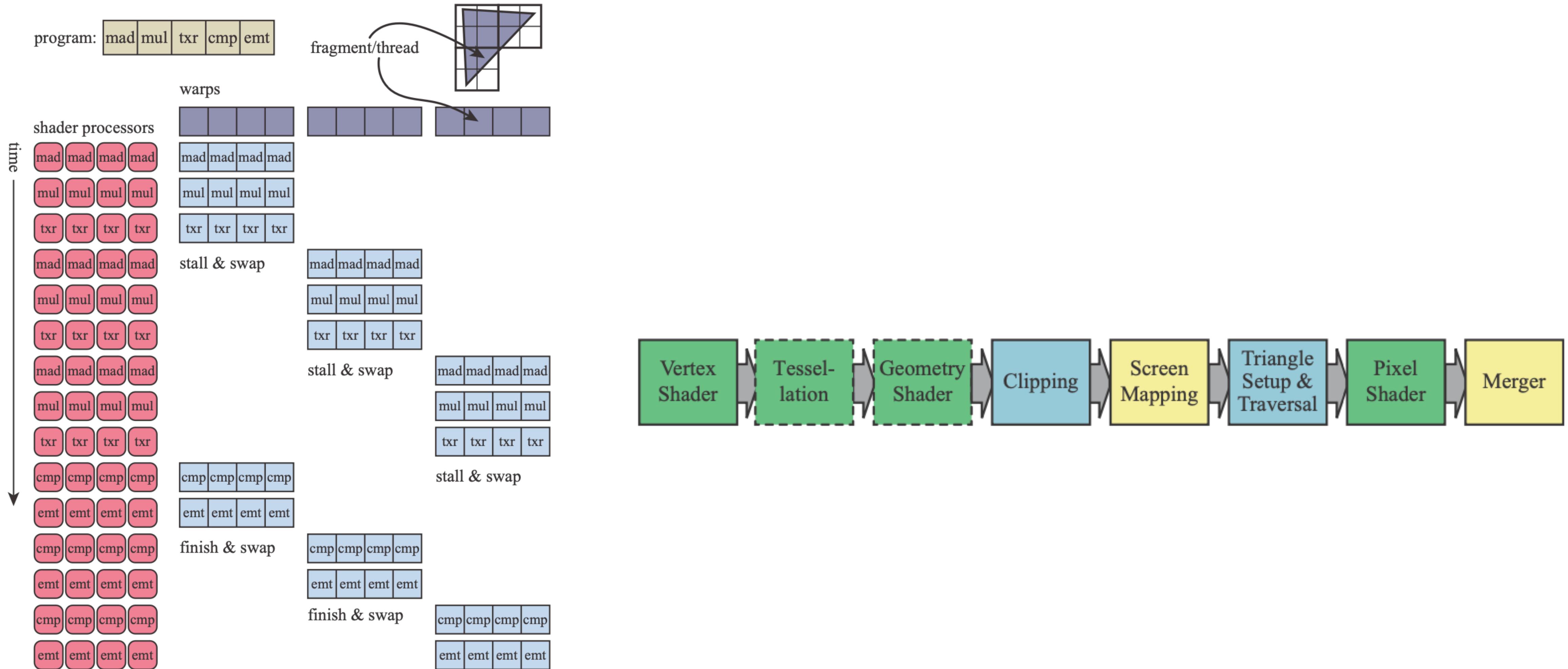


After stencil test

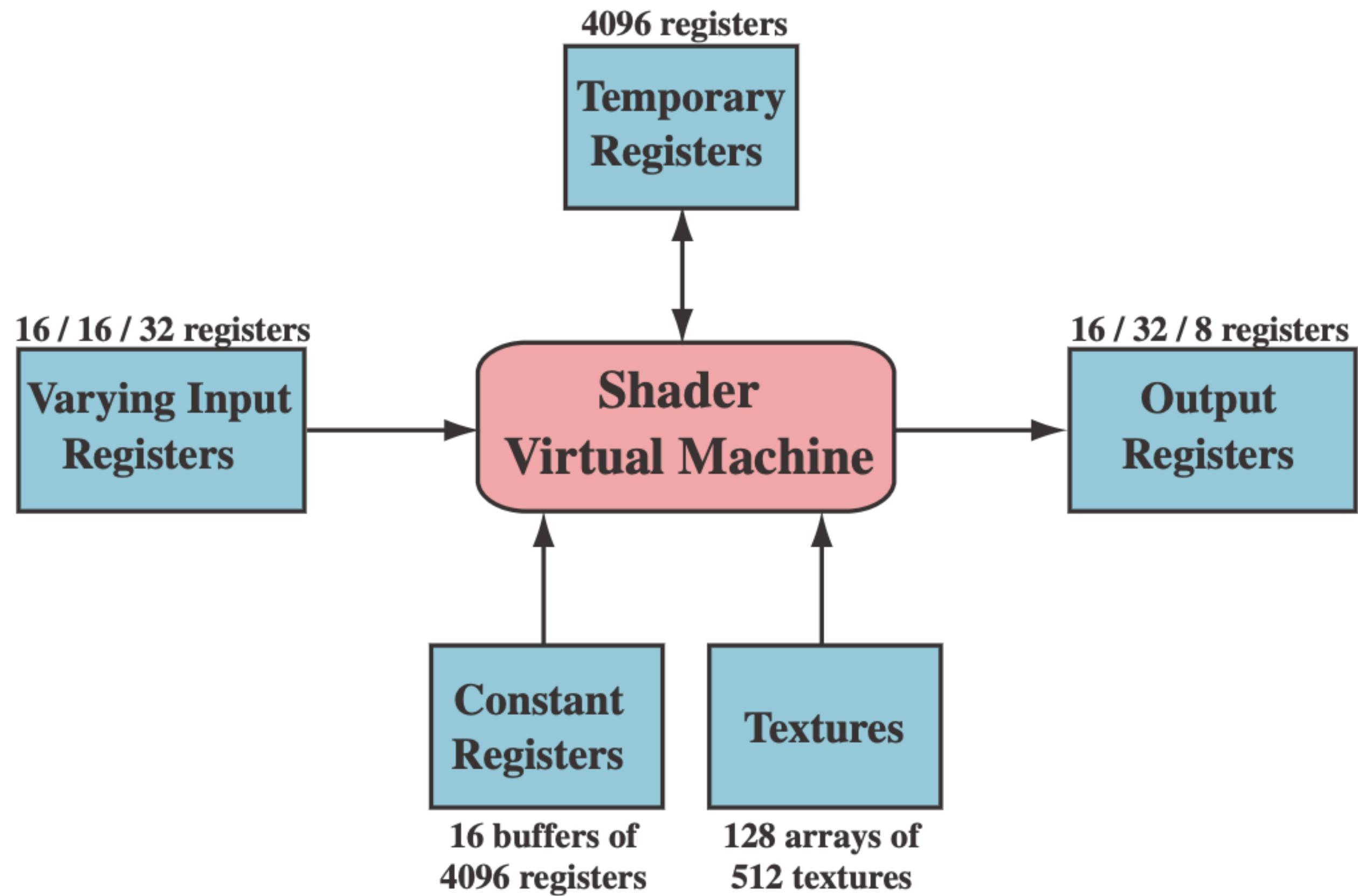
The Graphic Processing Unit



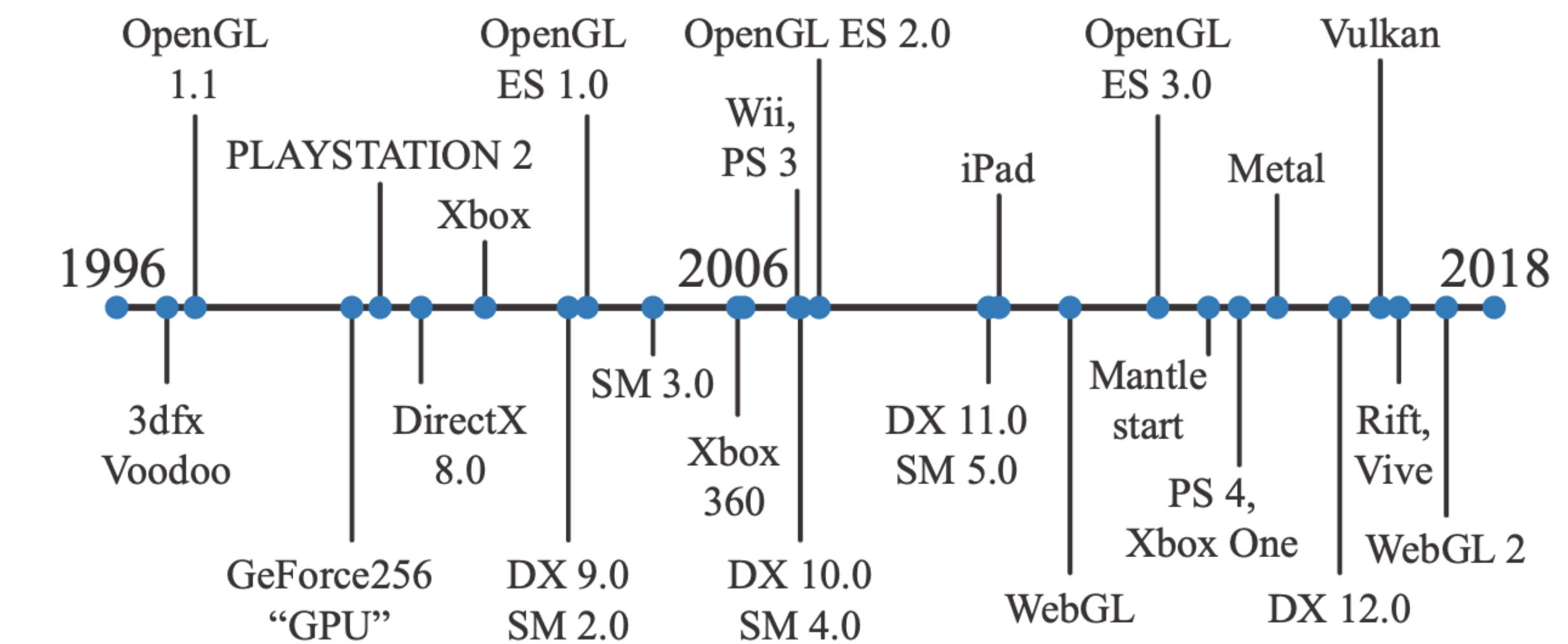
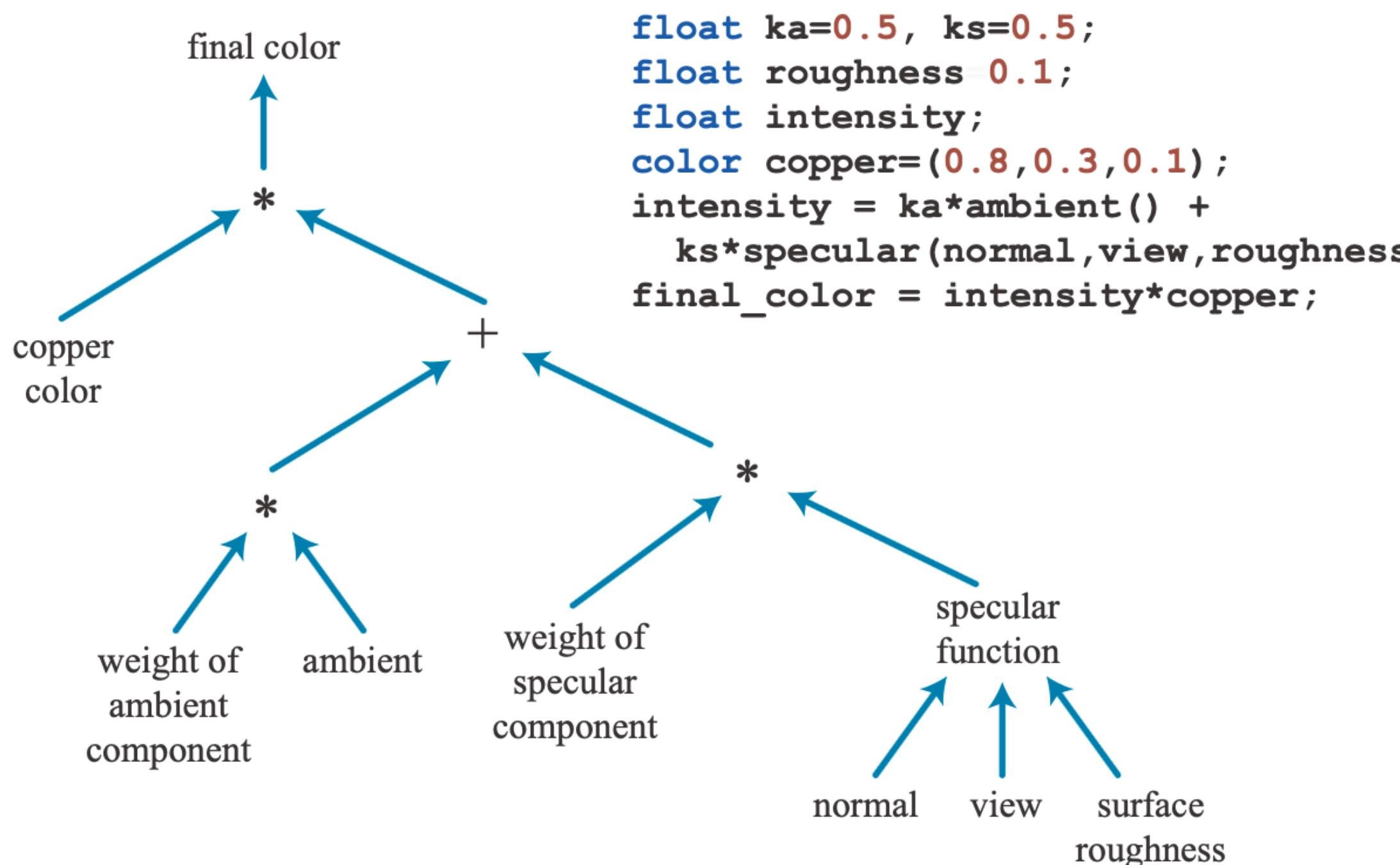
The Graphic Processing Unit



The Programmable Shader Stage



The Evolution of Programmable Shading and APIs



The Vertex Shader

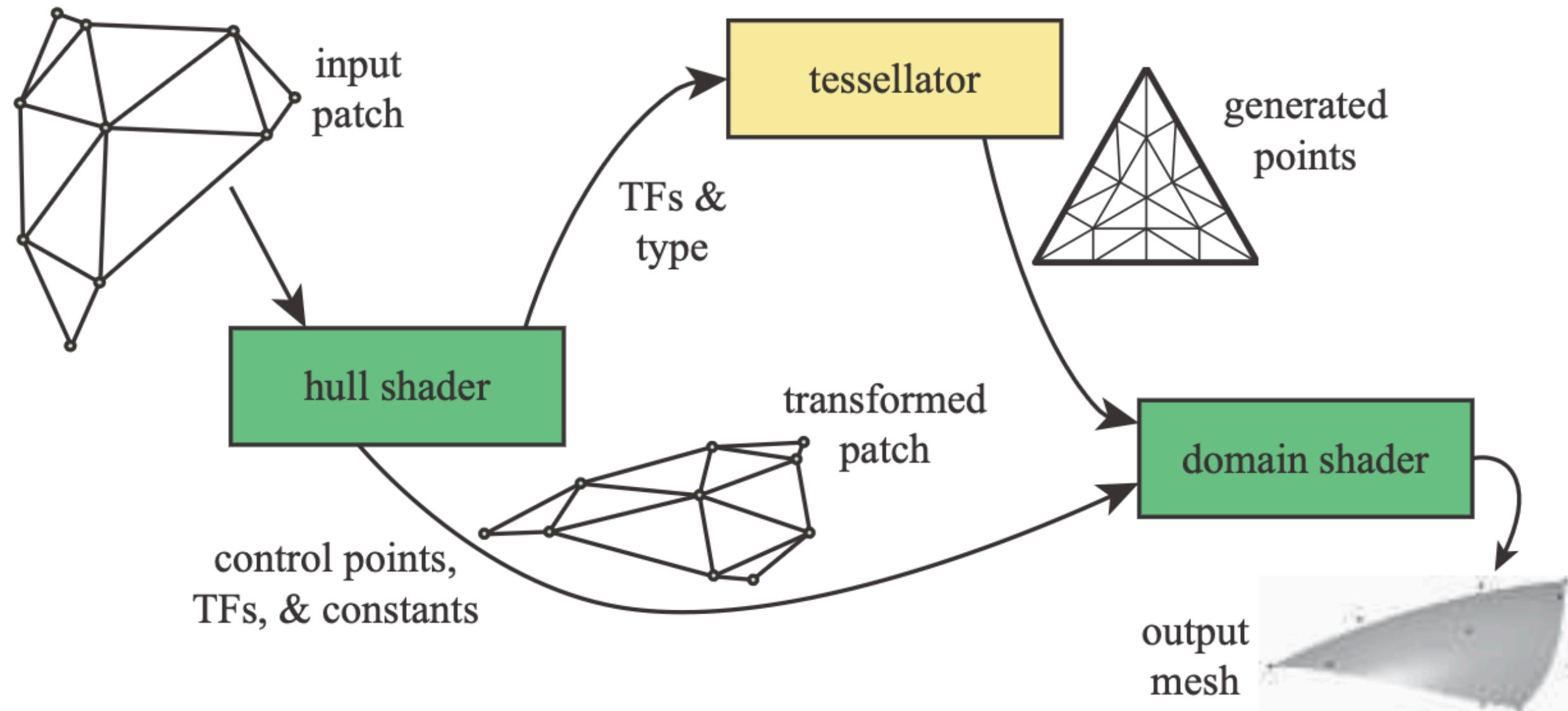
정점의 좌표, 색상, 텍스처 등 정보값 변환, 기존의 정점을 지우거나 새로운 정점 추가 불가



The Tessellation Stage

Tessellation control shader (hull shader)

Special patch primitive를 입력받음. Tessellation evaluation shader에 생성해야 할 삼각형 개수와 구성 알려주고 제어점에 대한 처리 수행, 선택적으로 제어점 추가, 제거. 제어점 세트를 primitive generator에 전달



The Tessellation Stage

Tessellation evaluation shader (tessellator)

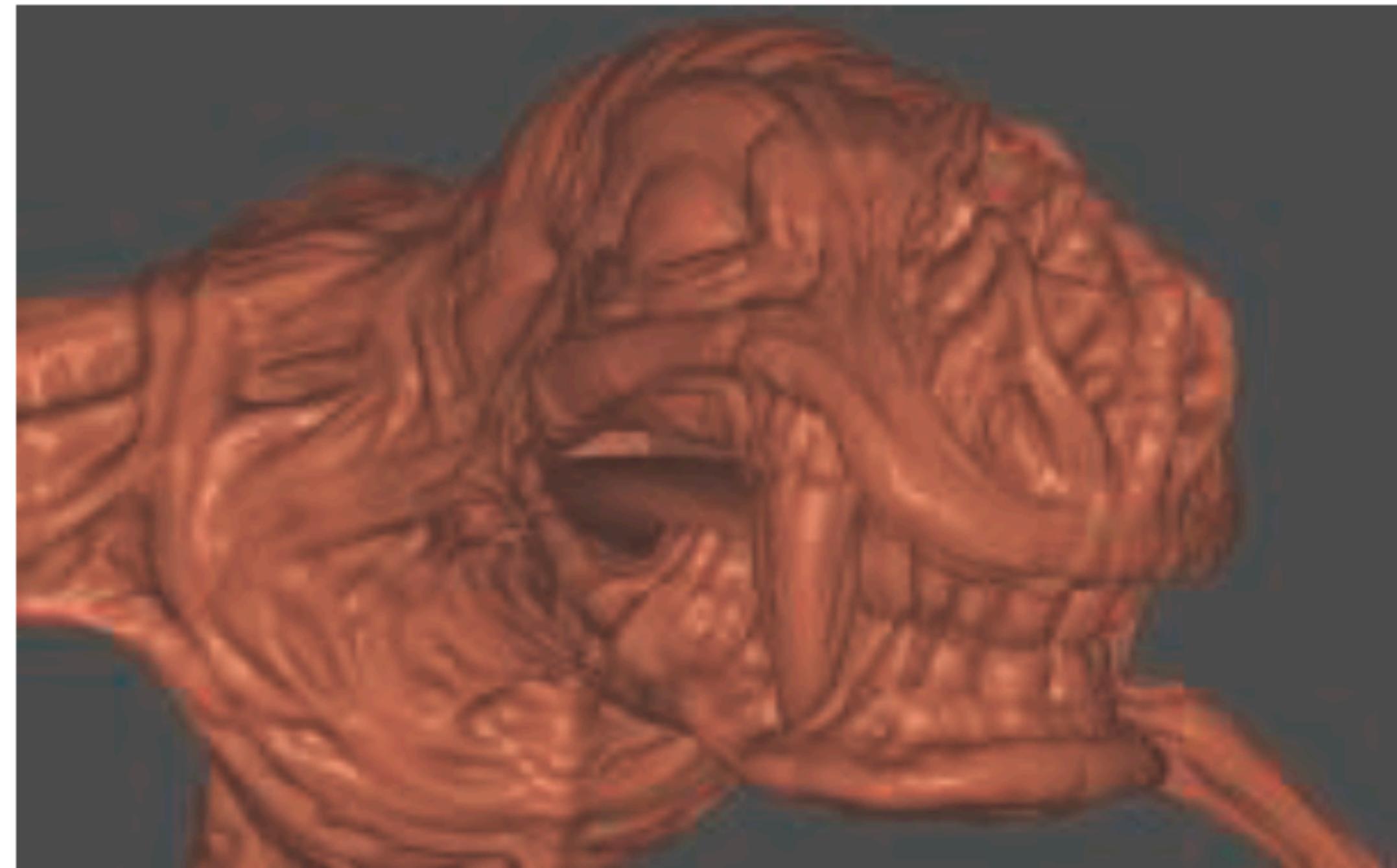
Tessellation control shader로부터 표면 유형에 대한 정보와 테셀레이션 계수를 입력받고
도메인 쉐이더에 몇 가지 정점 추가



The Tessellation Stage

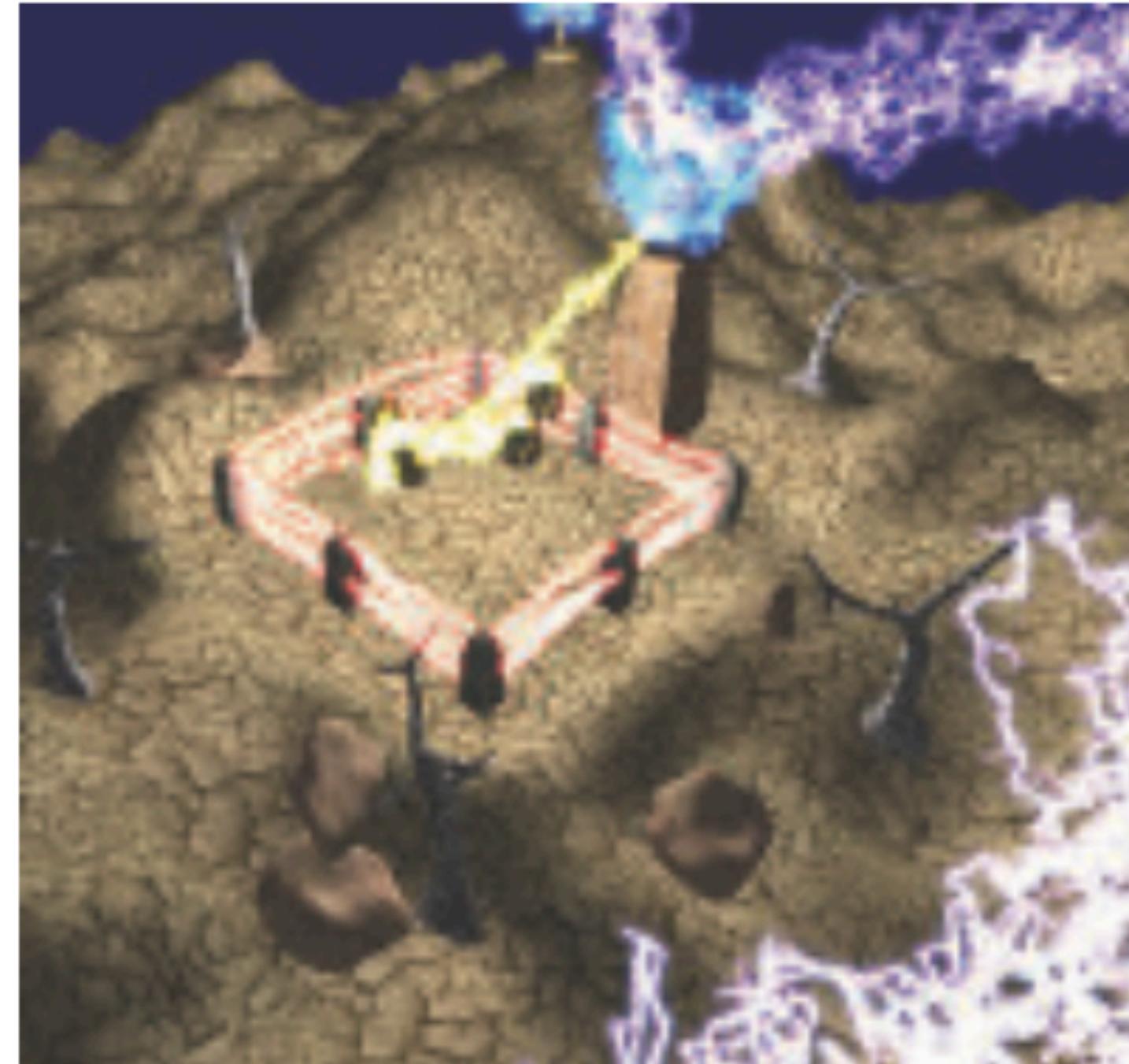
Primitive generator (domain shader)

위치, 노말, 텍스처 좌표 및 원하는 다른 정점 정보 생성



The Geometry Shader

Tessellation stage에서 처리 할 수 없는 primitive를 다른 primitive로 바꿀 수 있음.
vertex를 입력으로 받아 다음 쉐이더로 보내기 전에 적절한 형태로 변환, 추가 생성



Stream Output

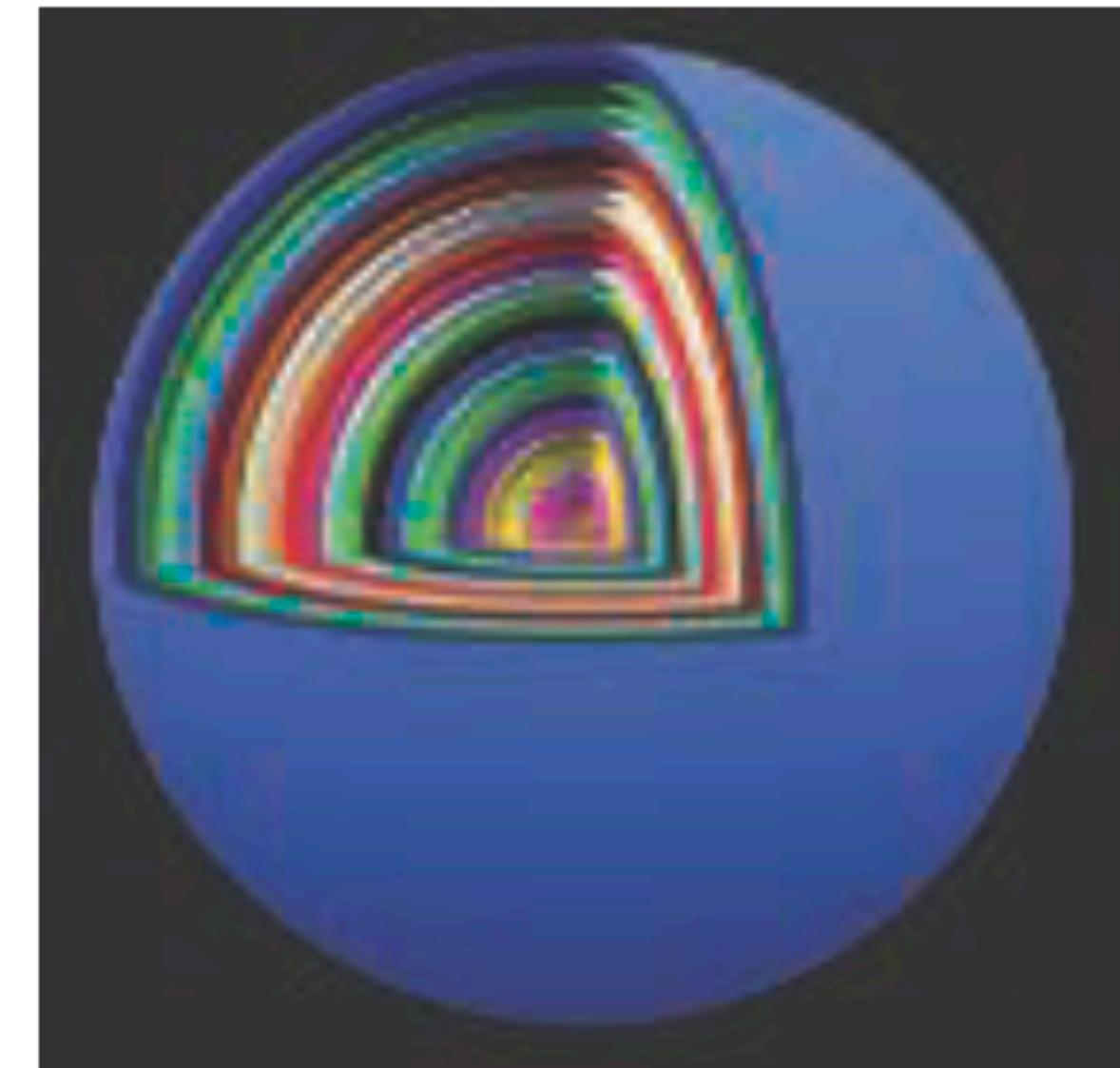
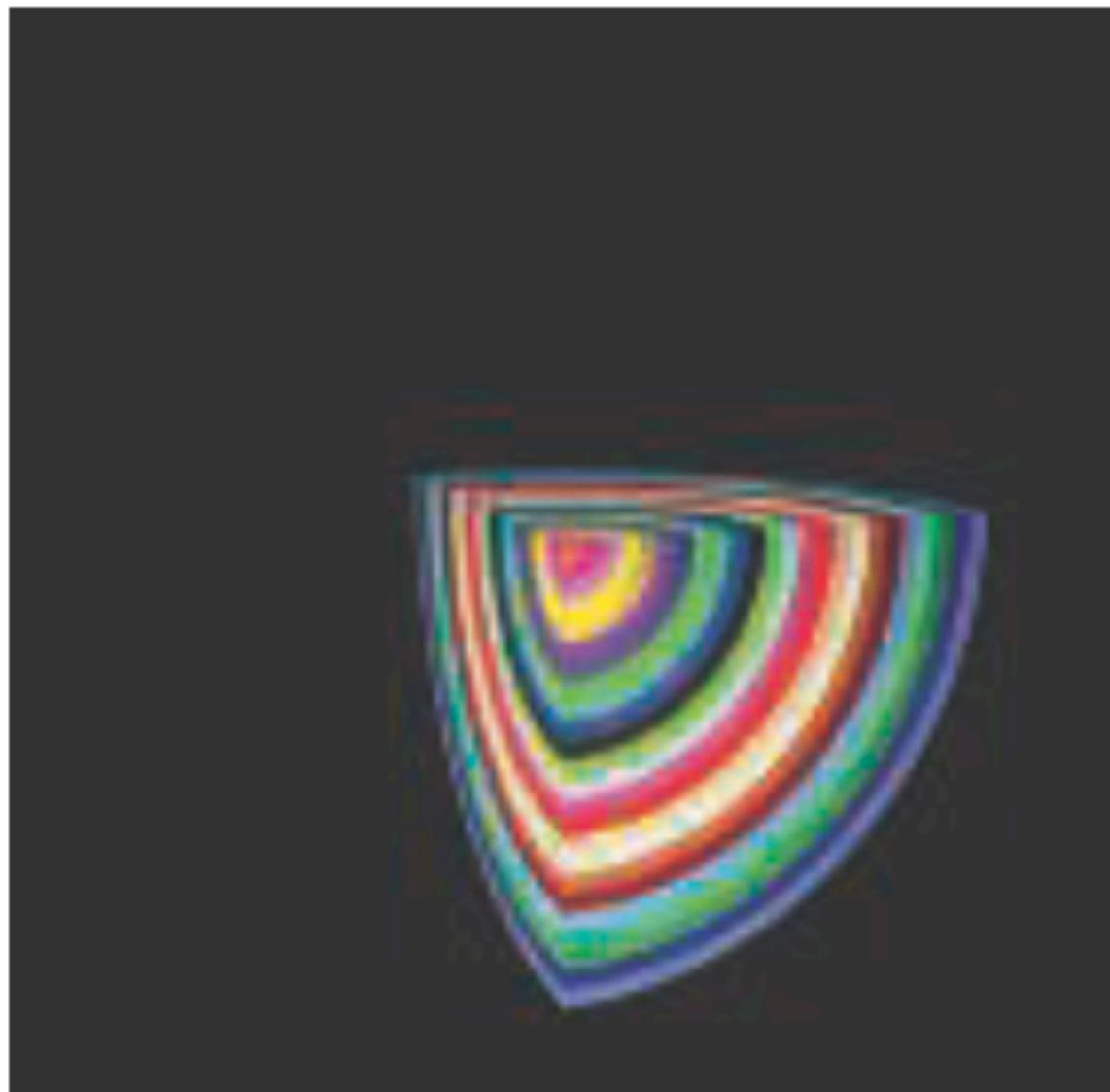
Vertex 가 vertex shader 에 의해 처리된 후 rasterization 단계로 전송되는 것 대신 순서 배열로 출력

파이프라인을 통해 다시 전송 될 수 있으므로 반복적인 처리가 가능. 물 또는 기타 입자를 시뮬레이션 하는데 사용 가능

The Pixel Shader

렌더링 될 픽셀 각각의 색, depth 등을 계산

Unique ability to discard an incoming fragment



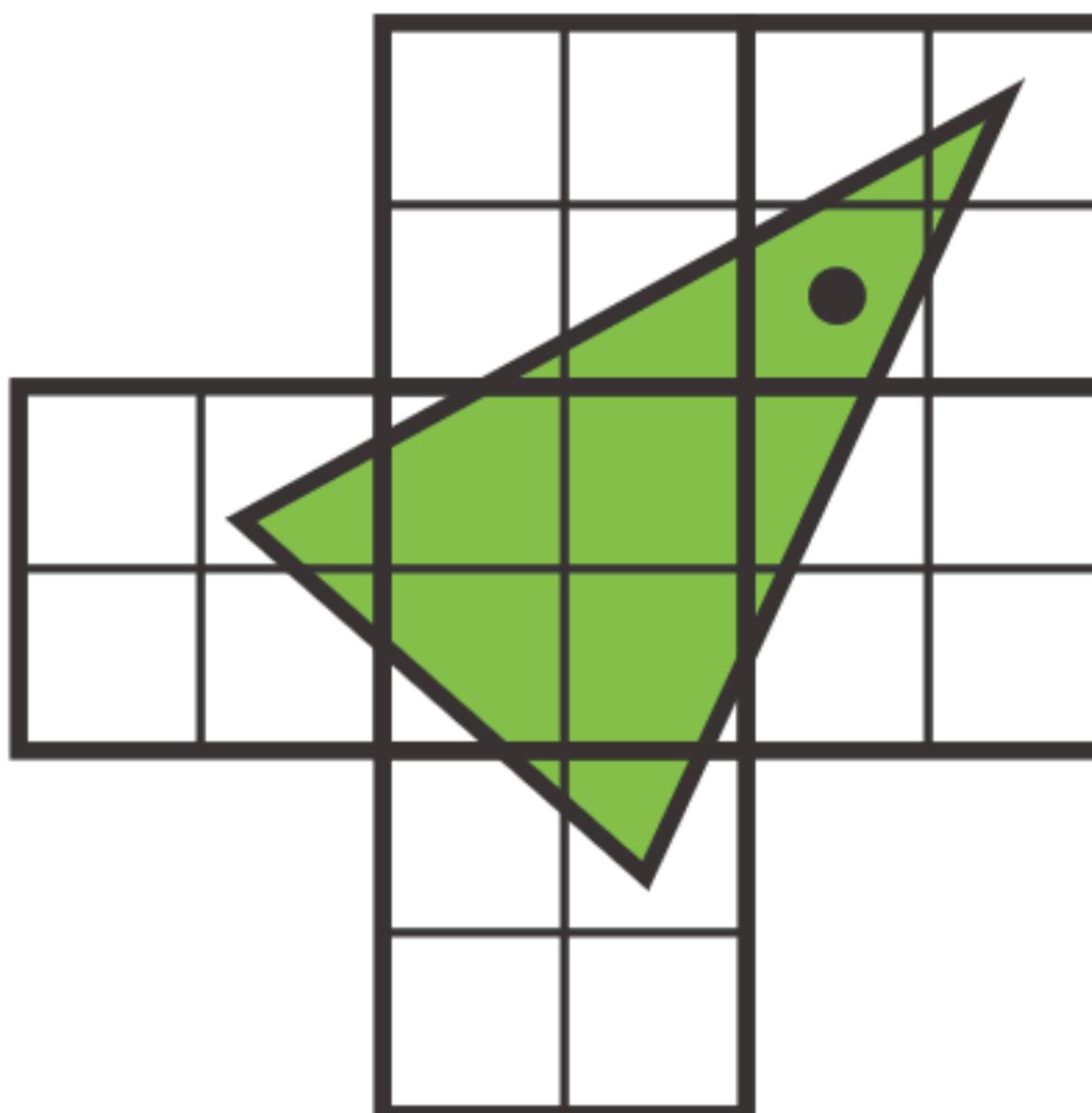
Multiple Render Targets (MRT)

한번의 fragment shader 연산을 수행하여 여러 개의 texture (render target textures)를 생성하는 방법

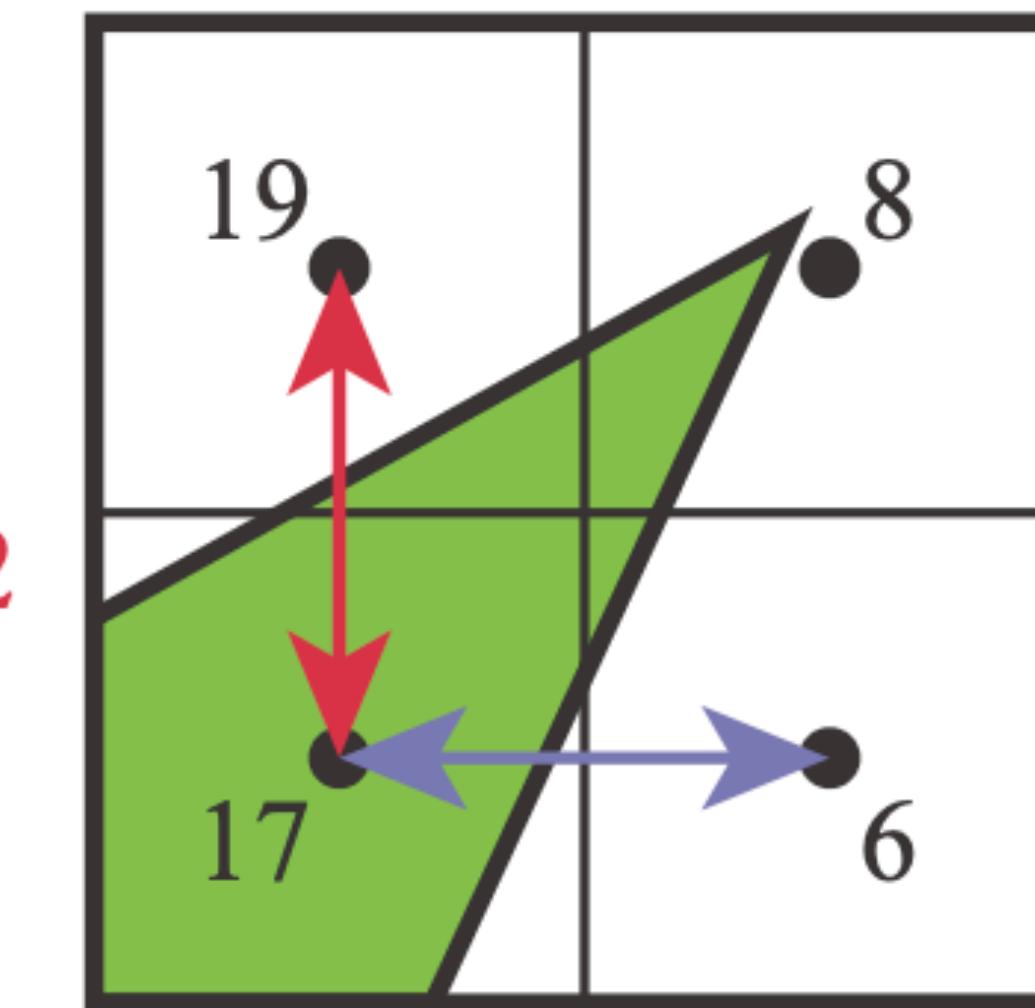
여러 개의 texture 생성을 위해 같은 vertex shader 연산을 여러 번 수행하는 대신 vertex shader 연산을 한 번만 수행하고 fragment shader에서 여러 개의 texture에 다른 결과값을 저장 할 수 있다.



Exception to the rule that a pixel shader cannot know or affect neighboring pixels' results



$$dv/dy = \\ 19 - 17 = 2$$



$$dv/dx = 6 - 17 = -11$$

The Merging Stage

Early-z - z버퍼가 적용될 때 숨겨진 fragment에 의해 생기는 낭비를 피하기 위해 병합 테스트를 수행하고 숨겨진 경우 fragment를 잘라냄

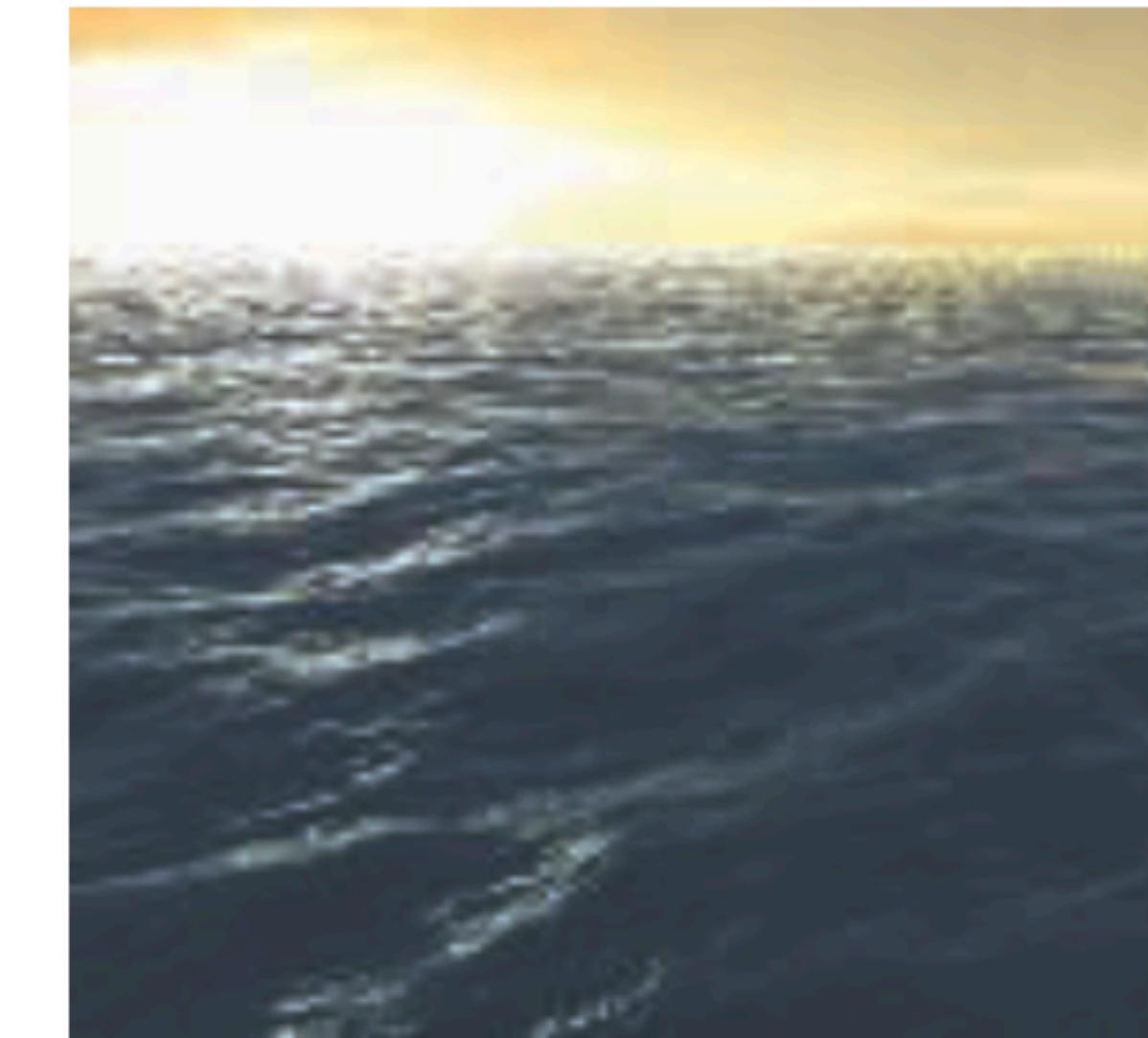
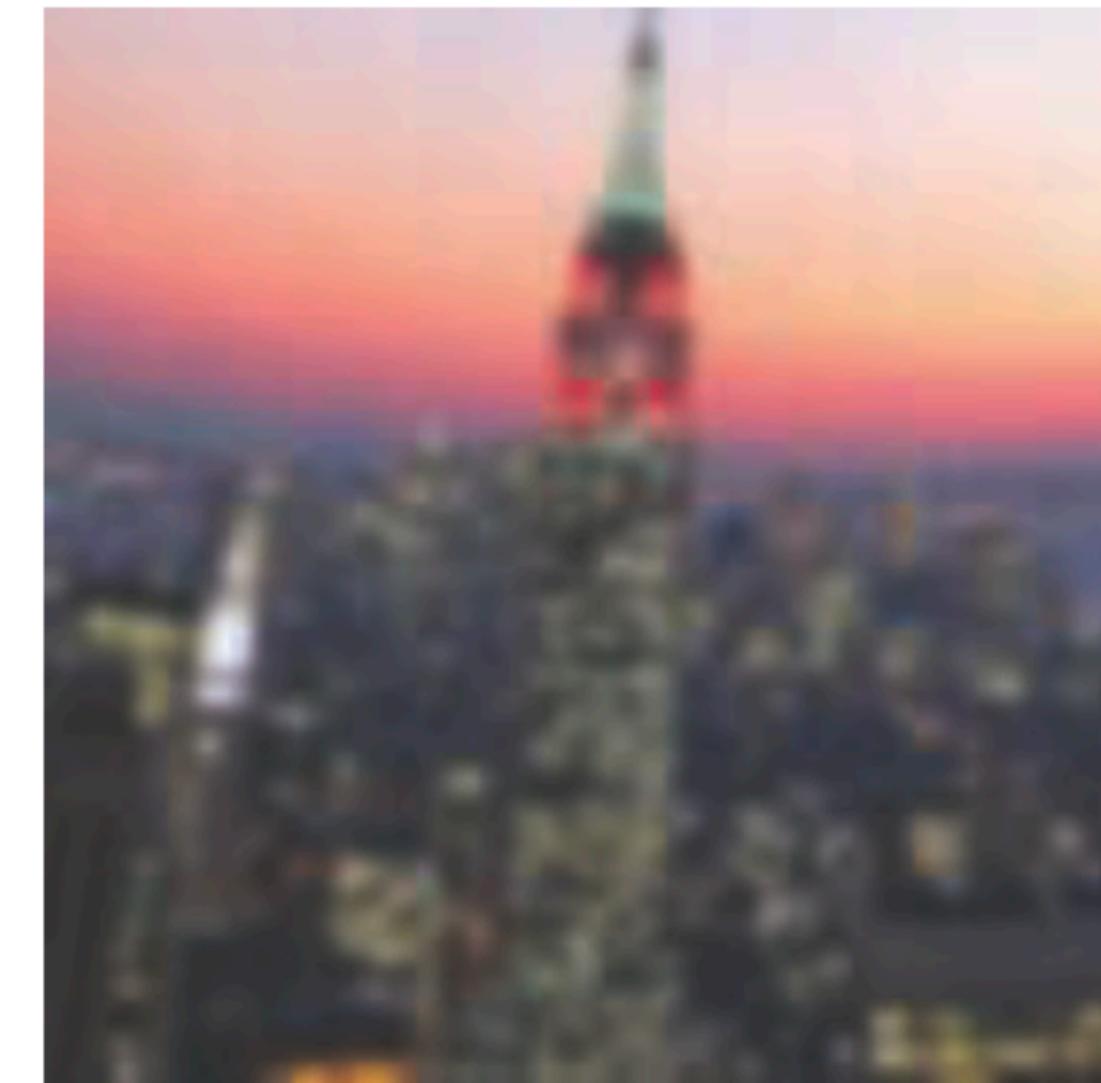
Color blending - color, alpha, other operations like minimum and maximum

The Compute Shader

GPU computing - 스톡옵션 가치 계산, 딥러닝을 위한 신경망 훈련 등으로도 사용

GPU에서 생성된 데이터에 액세스 가능.

Particle system, facial animation, culling, image filtering 등에 유용



Basic Transforms

Translation

Rotation

Scaling

Shearing

Concatenation of transform

The Rigid-Body Transform

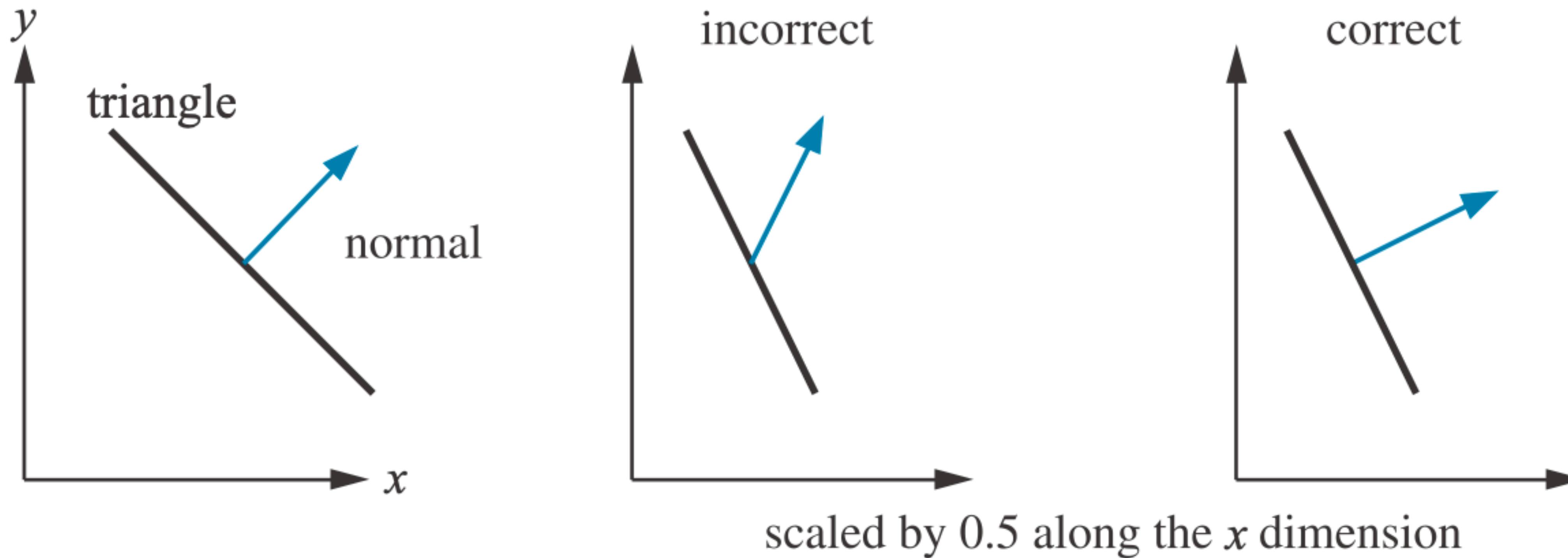
물체의 위치와 방향만 바뀌고 물체의 모양은 영향을 받지 않는 변환

$$\mathbf{X} = \mathbf{T}(t)\mathbf{R} = \begin{pmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Normal Transform

노말 벡터는 기하 구조 변환에 사용된 특정 행렬의 역행렬의 전치 행렬에 의해 변환되어야함

$$N = (M^{-1})^T$$



Computation of Inverses

- 행렬이 주어진 매개변수로 이루어진 단일 변환하거나 일련의 단순 변환들을 위한 것일 경우 $M = T(t)R(\phi)$, $M^{-1} = R(-\phi)T(-t)$
- 행렬이 직교한다는 사실을 알고 있을경우 $M^{-1} = M^T$
- Adjoint 방법, cramer의 법칙, LU분해, 가우스 소거법 등을 이용하여 계산 가능

The Special Transforms and Operations - The Euler Transform

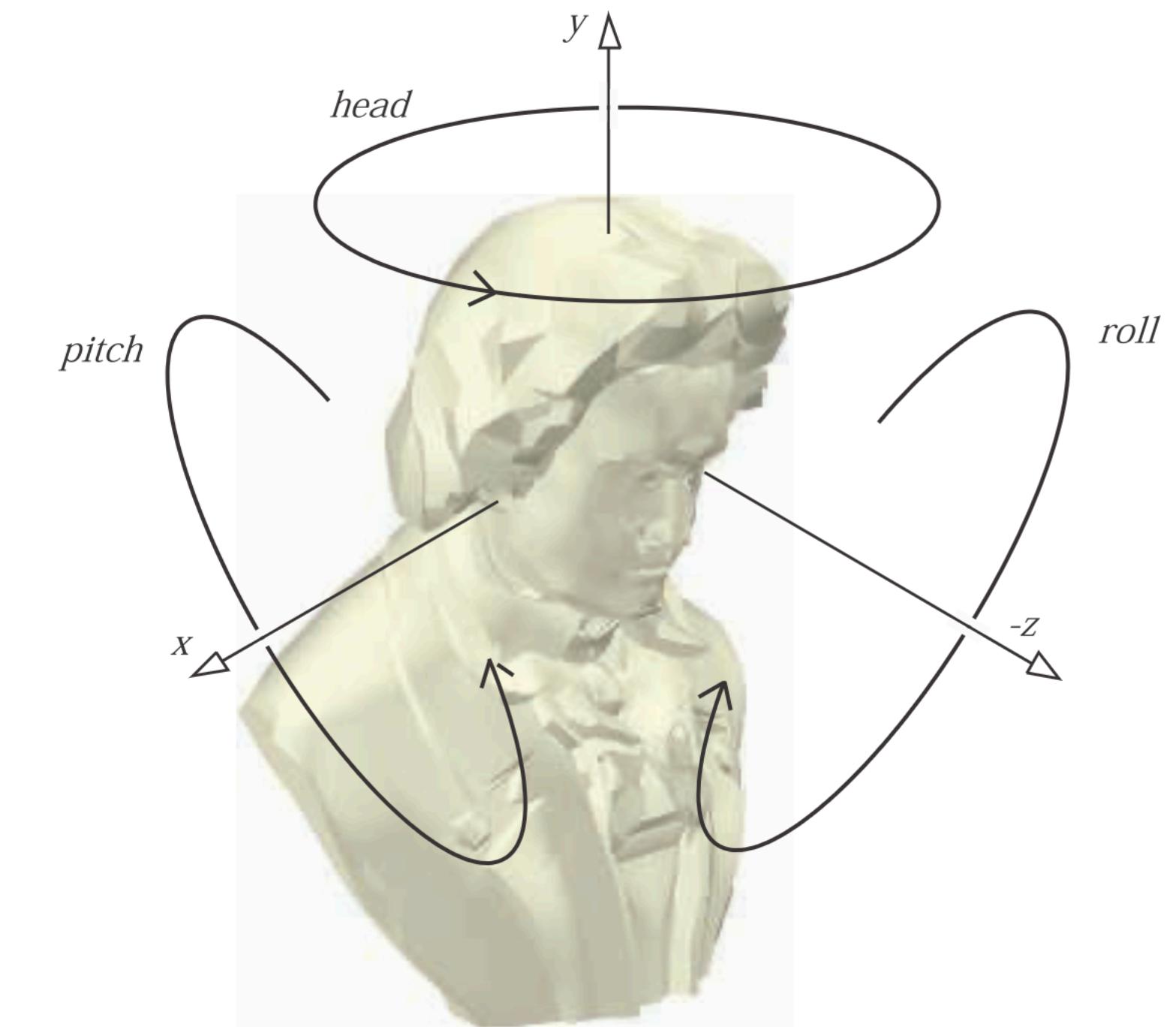
방향을 정하는 직관적인 방법 - 카메라나 임의의 다른 객체의 방향을 정하는 행렬을 만들어내기 위한 방법

$$\mathbf{E}(h, p, r) = \mathbf{R}_z(r)\mathbf{R}_x(p)\mathbf{R}_y(h).$$

\mathbf{E} 는 회전의 결합이므로 직교행렬이고 이의 역행렬은

$$\mathbf{E}^{-1} = \mathbf{E}^T = (\mathbf{R}_z \mathbf{R}_x \mathbf{R}_y)^T = \underline{\mathbf{R}_y^T} \underline{\mathbf{R}_x^T} \underline{\mathbf{R}_z^T}$$

Gimbal lock - 회전 할 때 하나의 회전 자유도를 상실하는 현상



Extracting Parameters from the Euler Transform

$$\mathbf{E}(h, p, r) = \begin{pmatrix} e_{00} & e_{01} & e_{02} \\ e_{10} & e_{11} & e_{12} \\ e_{20} & e_{21} & e_{22} \end{pmatrix} = \mathbf{R}_z(r)\mathbf{R}_x(p)\mathbf{R}_y(h).$$

$$\mathbf{E} = \begin{pmatrix} \cos r \cos h - \sin r \sin p \sin h & -\sin r \cos p & \cos r \sin h + \sin r \sin p \cos h \\ \sin r \cos h + \cos r \sin p \sin h & \cos r \cos p & \sin r \sin h - \cos r \sin p \cos h \\ -\cos p \sin h & \sin p & \cos p \cos h \end{pmatrix}$$

$$\frac{e_{01}}{e_{11}} = \frac{-\sin r}{\cos r} = -\tan r \quad \text{and} \quad \frac{e_{20}}{e_{22}} = \frac{-\sin h}{\cos h} = -\tan h.$$

$$h = \text{atan2}(-e_{20}, e_{22}), \quad p = \arcsin(e_{21}), \quad r = \text{atan2}(-e_{01}, e_{11}). \quad \mathbf{E} = \begin{pmatrix} \cos r & \sin r \cos p & \sin r \sin p \\ \sin r & \cos r \cos p & -\cos r \sin p \\ 0 & \sin p & \cos p \end{pmatrix}$$

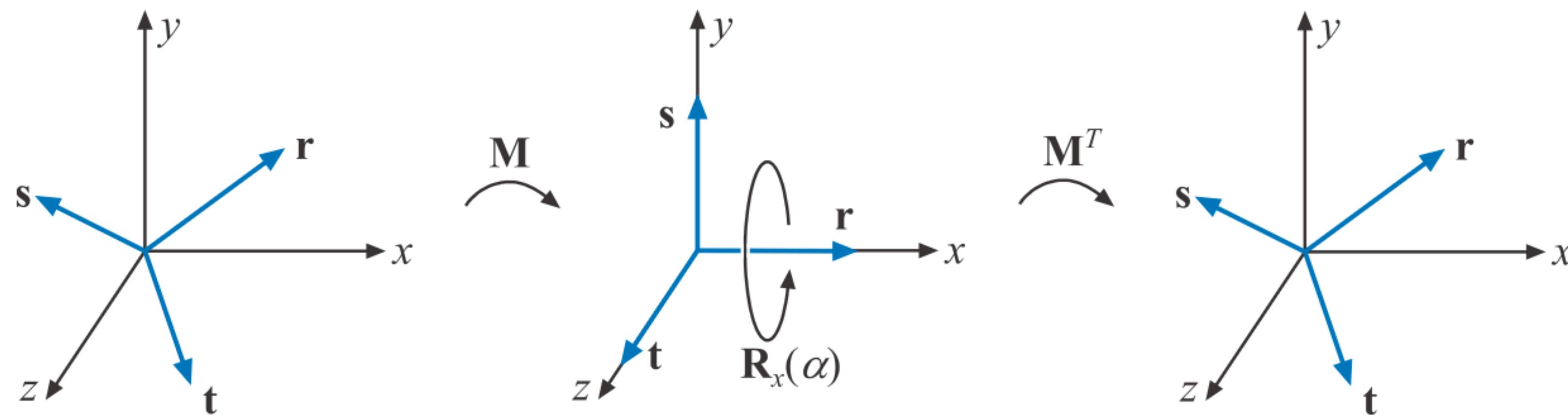
Matrix Decomposition

지금까지는 우리가 사용하게 될 변환 행렬이 무엇으로부터 시작되었고 어떤 변환 과정을 거쳐왔는지 있다고 가정하고 설명하였으나 실제로 이런 경우는 거의 없음.

행렬 분해 - 결합된 행렬로부터 다양한 변환들을 찾아내는 것

- 한 객체에 대한 크기 조정 비율만 추출하고자 할 경우
- 특정 시스템이 필요로 하는 변환을 찾을 경우
- 모델에 단지 강체 변환만 적용되었는지의 여부를 결정해야 할 경우
- 물체에 대한 행렬만 사용 가능한 애니메이션에서 키프레임들 사이를 보간할 경우
- 회전 행렬로부터 쉬어 변환을 떼어낼 경우

Rotation about an Arbitrary Axis



Quaternions

복소수를 확장한 것으로 회전이나 방향 지정시에 뛰어난 성능을 발휘한다.

오일러 각 체계를 사용시 발생하는 Gimbal lock 현상을 없앨 수 있음

Mathematical Background

$$\hat{\mathbf{q}} = (\mathbf{q}_v, q_w) = iq_x + jq_y + kq_z + q_w = \mathbf{q}_v + q_w,$$

$$\mathbf{q}_v = iq_x + jq_y + kq_z = (q_x, q_y, q_z),$$

$$i^2 = j^2 = k^2 = -1, \quad jk = -kj = i, \quad ki = -ik = j, \quad ij = -ji = k.$$

$$\hat{\mathbf{q}}\hat{\mathbf{r}} = (iq_x + jq_y + kq_z + q_w)(ir_x + jr_y + kr_z + r_w) \quad \text{Addition:} \quad \hat{\mathbf{q}} + \hat{\mathbf{r}} = (\mathbf{q}_v, q_w) + (\mathbf{r}_v, r_w) = (\mathbf{q}_v + \mathbf{r}_v, q_w + r_w).$$

$$= i(q_y r_z - q_z r_y + r_w q_x + q_w r_x) \quad \text{Conjugate:} \quad \hat{\mathbf{q}}^* = (\mathbf{q}_v, q_w)^* = (-\mathbf{q}_v, q_w).$$

$$+ j(q_z r_x - q_x r_z + r_w q_y + q_w r_y) \quad \text{Norm:} \quad n(\hat{\mathbf{q}}) = \sqrt{\hat{\mathbf{q}}\hat{\mathbf{q}}^*} = \sqrt{\hat{\mathbf{q}}^*\hat{\mathbf{q}}} = \sqrt{\mathbf{q}_v \cdot \mathbf{q}_v + q_w^2}$$

$$+ k(q_x r_y - q_y r_x + r_w q_z + q_w r_z) \quad = \sqrt{q_x^2 + q_y^2 + q_z^2 + q_w^2}.$$

$$+ q_w r_w - q_x r_x - q_y r_y - q_z r_z$$

$$= (\mathbf{q}_v \times \mathbf{r}_v + r_w \mathbf{q}_v + q_w \mathbf{r}_v, q_w r_w - \mathbf{q}_v \cdot \mathbf{r}_v). \quad \text{Identity:} \quad \hat{\mathbf{i}} = (\mathbf{0}, 1).$$

$$n(\hat{\mathbf{q}})^2 = \hat{\mathbf{q}}\hat{\mathbf{q}}^* \iff \frac{\hat{\mathbf{q}}\hat{\mathbf{q}}^*}{n(\hat{\mathbf{q}})^2} = 1. \quad \hat{\mathbf{q}}^{-1} = \frac{1}{n(\hat{\mathbf{q}})^2}\hat{\mathbf{q}}^*.$$

Mathematical Background

Conjugate rules:

$$\begin{aligned}(\hat{\mathbf{q}}^*)^* &= \hat{\mathbf{q}}, & \hat{\mathbf{q}} &= (\sin \phi \mathbf{u}_q, \cos \phi) = \sin \phi \mathbf{u}_q + \cos \phi, \\(\hat{\mathbf{q}} + \hat{\mathbf{r}})^* &= \hat{\mathbf{q}}^* + \hat{\mathbf{r}}^*, \\(\hat{\mathbf{q}}\hat{\mathbf{r}})^* &= \hat{\mathbf{r}}^*\hat{\mathbf{q}}^*. \end{aligned}$$

Norm rules:

$$\begin{aligned}n(\hat{\mathbf{q}}^*) &= n(\hat{\mathbf{q}}), \\n(\hat{\mathbf{q}}\hat{\mathbf{r}}) &= n(\hat{\mathbf{q}})n(\hat{\mathbf{r}}).\end{aligned}$$

Laws of Multiplication:

Linearity:

$$\begin{aligned}\hat{\mathbf{p}}(s\hat{\mathbf{q}} + t\hat{\mathbf{r}}) &= s\hat{\mathbf{p}}\hat{\mathbf{q}} + t\hat{\mathbf{p}}\hat{\mathbf{r}}, \\(s\hat{\mathbf{p}} + t\hat{\mathbf{q}})\hat{\mathbf{r}} &= s\hat{\mathbf{p}}\hat{\mathbf{r}} + t\hat{\mathbf{q}}\hat{\mathbf{r}}.\end{aligned}$$

Associativity:

$$\hat{\mathbf{p}}(\hat{\mathbf{q}}\hat{\mathbf{r}}) = (\hat{\mathbf{p}}\hat{\mathbf{q}})\hat{\mathbf{r}}.$$

$$\begin{aligned}n(\hat{\mathbf{q}}) &= n(\sin \phi \mathbf{u}_q, \cos \phi) = \sqrt{\sin^2 \phi (\mathbf{u}_q \cdot \mathbf{u}_q) + \cos^2 \phi} \\&= \sqrt{\sin^2 \phi + \cos^2 \phi} = 1\end{aligned}$$

$$\hat{\mathbf{q}} = \sin \phi \mathbf{u}_q + \cos \phi = e^{\phi \mathbf{u}_q}.$$

Logarithm: $\log(\hat{\mathbf{q}}) = \log(e^{\phi \mathbf{u}_q}) = \phi \mathbf{u}_q,$

Power: $\hat{\mathbf{q}}^t = (\sin \phi \mathbf{u}_q + \cos \phi)^t = e^{\phi t \mathbf{u}_q} = \sin(\phi t) \mathbf{u}_q + \cos(\phi t).$

Quaternion Transforms

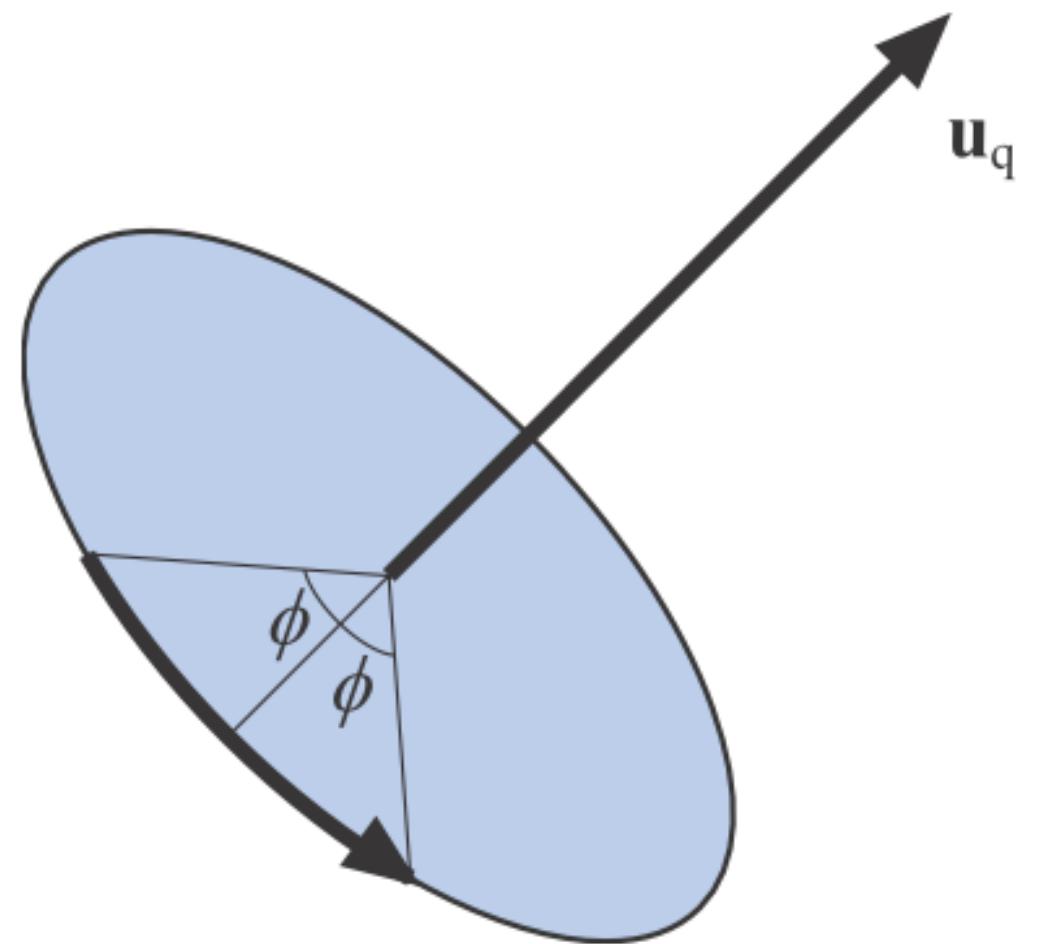
단위 사원수를 이용하여 어떤 형태의 3차원 회전이든 자유롭게 표현할 수 있음

$$\hat{\mathbf{q}}\hat{\mathbf{p}}\hat{\mathbf{q}}^{-1}$$

은 \mathbf{p} 을 \mathbf{u}_q 를 축으로 2ϕ 만큼 회전시키는 것이다

두 개의 단위 사원수 \mathbf{q} 과 \mathbf{r} 가 주어졌을 때 사원수 \mathbf{p} 에 \mathbf{q} 를 먼저 적용하고 다음에 \mathbf{r} 을 적용하는 결합 연산은 다음과 같다

$$\hat{\mathbf{r}}(\hat{\mathbf{q}}\hat{\mathbf{p}}\hat{\mathbf{q}}^*)\hat{\mathbf{r}}^* = (\hat{\mathbf{r}}\hat{\mathbf{q}})\hat{\mathbf{p}}(\hat{\mathbf{r}}\hat{\mathbf{q}})^* = \hat{\mathbf{c}}\hat{\mathbf{p}}\hat{\mathbf{c}}^*.$$



Matrix Conversion

일부 시스템에서는 행렬 곱셈이 하드웨어로 구현되어 있고 거기서는 행렬을 곱하는 것이

수식 $\hat{\mathbf{q}}\hat{\mathbf{p}}\hat{\mathbf{q}}^{-1}$ 보다 더 효율적이므로 사원수를 행렬로 변환하고, 행렬을 사원수로 변환한다

$$\mathbf{M}^q = \begin{pmatrix} 1 - 2(q_y^2 + q_z^2) & 2(q_x q_y - q_w q_z) & 2(q_x q_z + q_w q_y) & 0 \\ 2(q_x q_y + q_w q_z) & 1 - 2(q_x^2 + q_z^2) & 2(q_y q_z - q_w q_x) & 0 \\ 2(q_x q_z - q_w q_y) & 2(q_y q_z + q_w q_x) & 1 - 2(q_x^2 + q_y^2) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

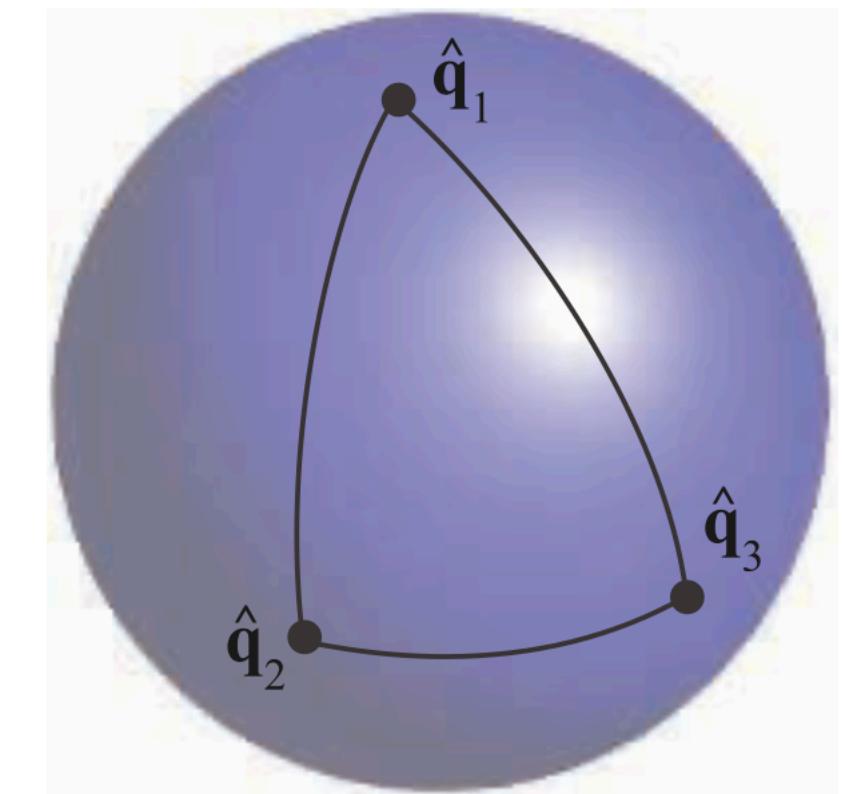
$$\begin{aligned} m_{21}^q - m_{12}^q &= 4q_w q_x, & \text{tr}(\mathbf{M}^q) &= 4 - 2s(q_x^2 + q_y^2 + q_z^2) = 4 \left(1 - \frac{q_x^2 + q_y^2 + q_z^2}{q_x^2 + q_y^2 + q_z^2 + q_w^2} \right) \\ m_{02}^q - m_{20}^q &= 4q_w q_y, & &= \frac{4q_w^2}{q_x^2 + q_y^2 + q_z^2 + q_w^2} = \frac{4q_w^2}{(n(\hat{\mathbf{q}}))^2}. \\ m_{10}^q - m_{01}^q &= 4q_w q_z. & & \end{aligned}$$

$$\begin{aligned} q_w &= \frac{1}{2}\sqrt{\text{tr}(\mathbf{M}^q)}, & q_x &= \frac{m_{21}^q - m_{12}^q}{4q_w}, & m_{00} &= t + 2q_x^2, & 4q_x^2 &= +m_{00} - m_{11} - m_{22} + m_{33}, \\ & & & & m_{11} &= t + 2q_y^2, & 4q_y^2 &= -m_{00} + m_{11} - m_{22} + m_{33}, \\ q_y &= \frac{m_{02}^q - m_{20}^q}{4q_w}, & q_z &= \frac{m_{10}^q - m_{01}^q}{4q_w}. & m_{22} &= t + 2q_z^2, & 4q_z^2 &= -m_{00} - m_{11} + m_{22} + m_{33}, \\ & & & & u &= m_{00} + m_{11} + m_{22} = t + 2q_w^2, & 4q_w^2 &= \text{tr}(\mathbf{M}^q). \end{aligned}$$

Spherical Linear Interpolation

구선형 보간은 두개의 단위 사원수 phat과 rhat, 매개변수 $t \in [0, 1]$ 이 정해졌을 때 이들 사이에 보간된 사원수를 계산한다. 이는 물체의 애니메이션에 적합하나 카메라 방향을 보간하는 데는 카메라의 up벡터가 보간 과정에서 기울어지는 disturbing effect가 발생 할 수 있기 때문에 적합하지 않다.

$$\hat{s}(\hat{\mathbf{q}}, \hat{\mathbf{r}}, t) = (\hat{\mathbf{r}}\hat{\mathbf{q}}^{-1})^t\hat{\mathbf{q}}. \quad \hat{s}(\hat{\mathbf{q}}, \hat{\mathbf{r}}, t) = \text{slerp}(\hat{\mathbf{q}}, \hat{\mathbf{r}}, t) = \frac{\sin(\phi(1-t))}{\sin \phi}\hat{\mathbf{q}} + \frac{\sin(\phi t)}{\sin \phi}\hat{\mathbf{r}}.$$



Rotation from One Vector to Another

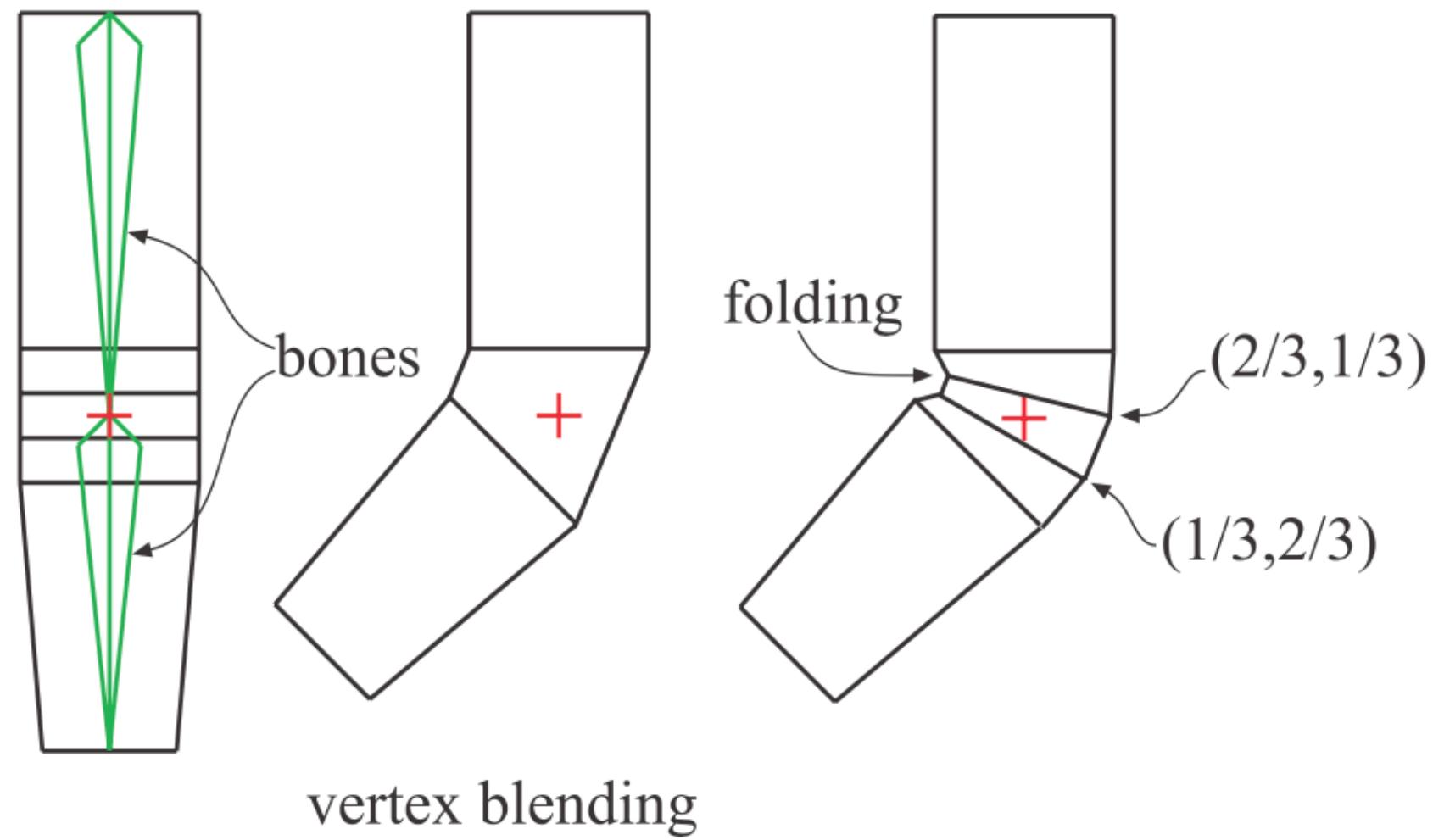
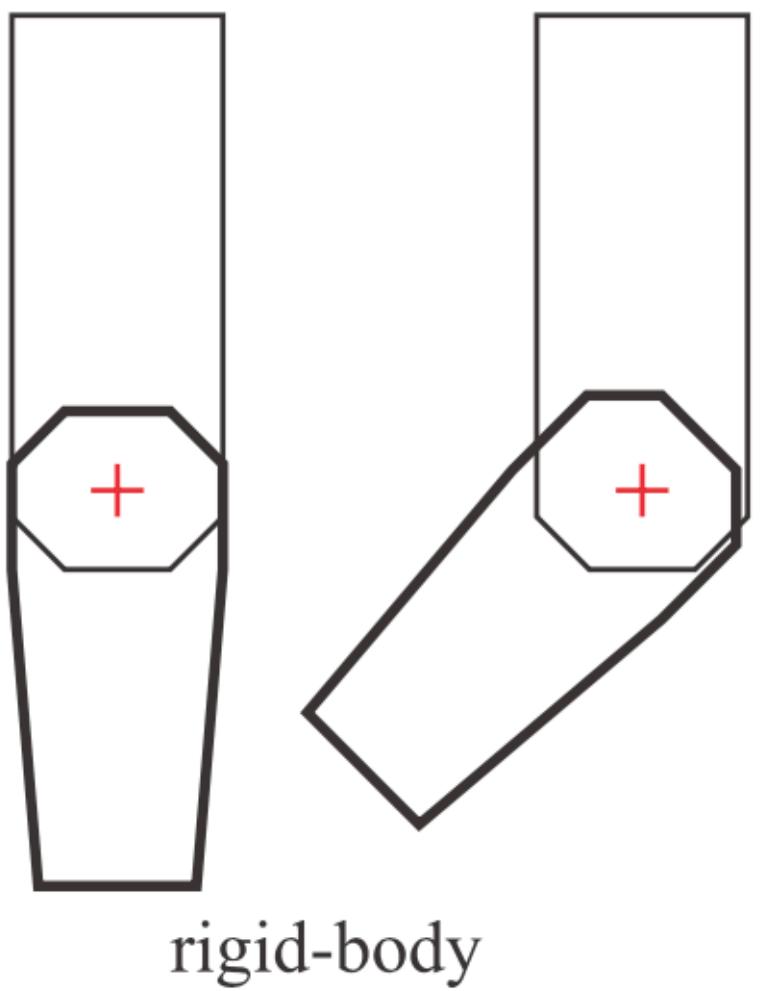
사원수 수학을 이용하여 한 방향 \mathbf{s} 를 다른 방향 \mathbf{t} 로 변환하는 연산을 단순하게 할 수 있다

$$\hat{\mathbf{q}} = (\mathbf{q}_v, q_w) = \left(\frac{1}{\sqrt{2(1+e)}}(\mathbf{s} \times \mathbf{t}), \frac{\sqrt{2(1+e)}}{2} \right).$$

$$\mathbf{R}(\mathbf{s}, \mathbf{t}) = \begin{pmatrix} e + hv_x^2 & hv_xv_y - v_z & hv_xv_z + v_y & 0 \\ hv_xv_y + v_z & e + hv_y^2 & hv_yv_z - v_x & 0 \\ hv_xv_z - v_y & hv_yv_z + v_x & e + hv_z^2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Vertex Blending

상박과 하박이 따로 애니메이션 되도록 하고 탄력있는 피부로 두 객체의 연결부를 이어준다. 탄력있는 부분은 상박 행렬과 하박 행렬 각각에 의해 변환되는 정점들로 구성된다. 그 결과로 삼각형당 하나의 행렬을 사용하는 일반적인 경우와 달리 정점들이 서로 다른 행렬에 의해 변환되는 삼각형들이 만들어진다.

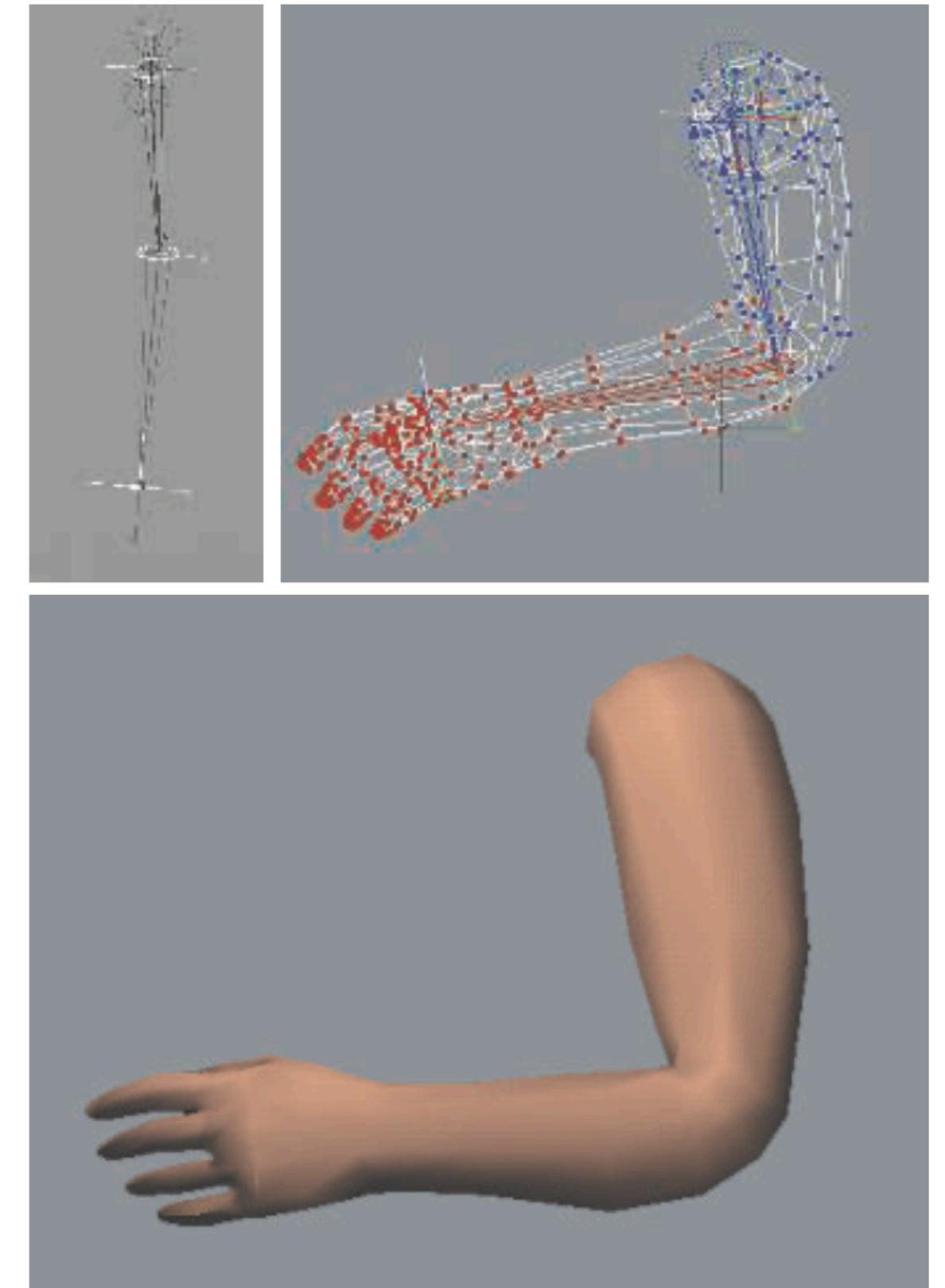
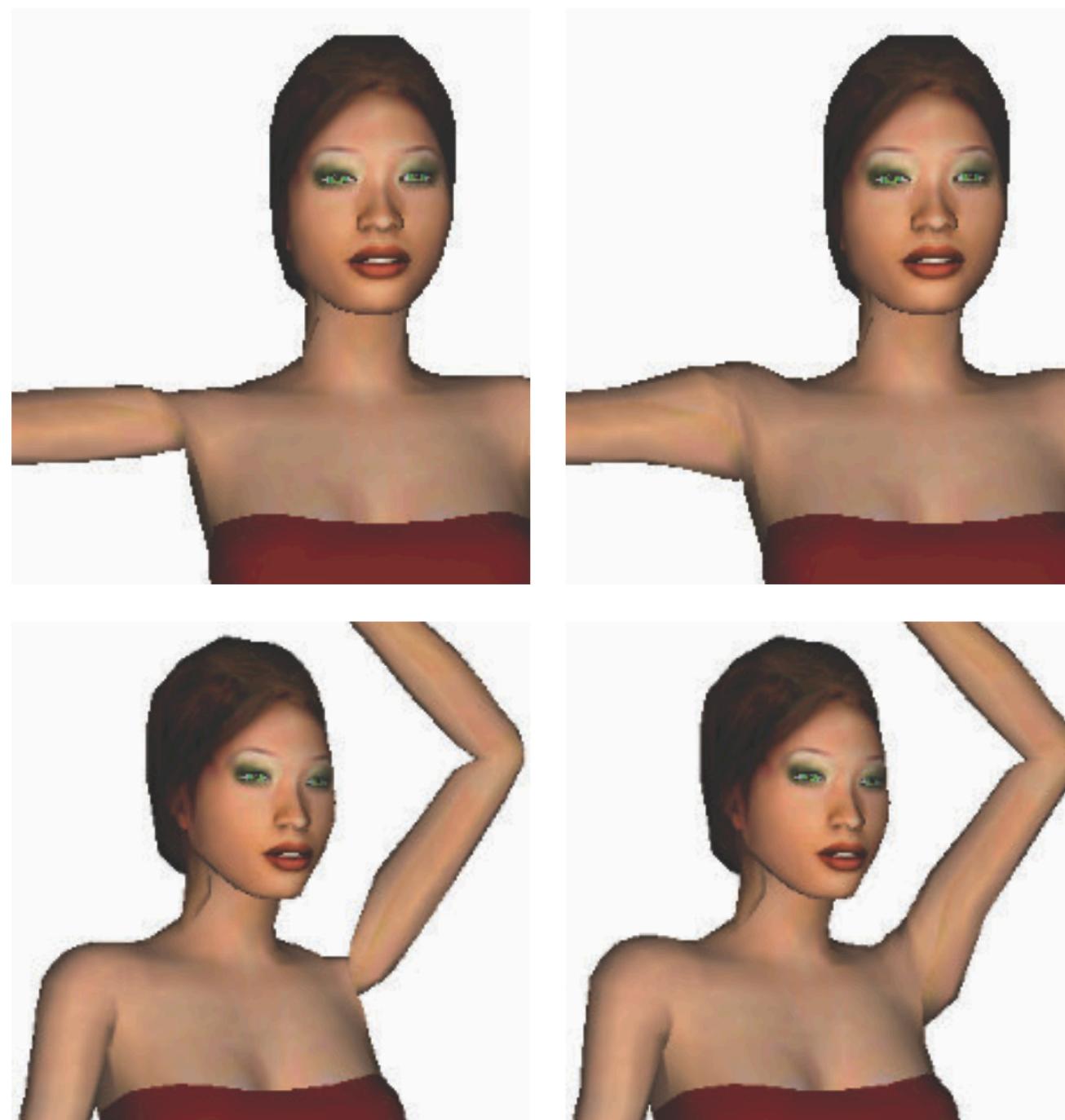


Vertex Blending

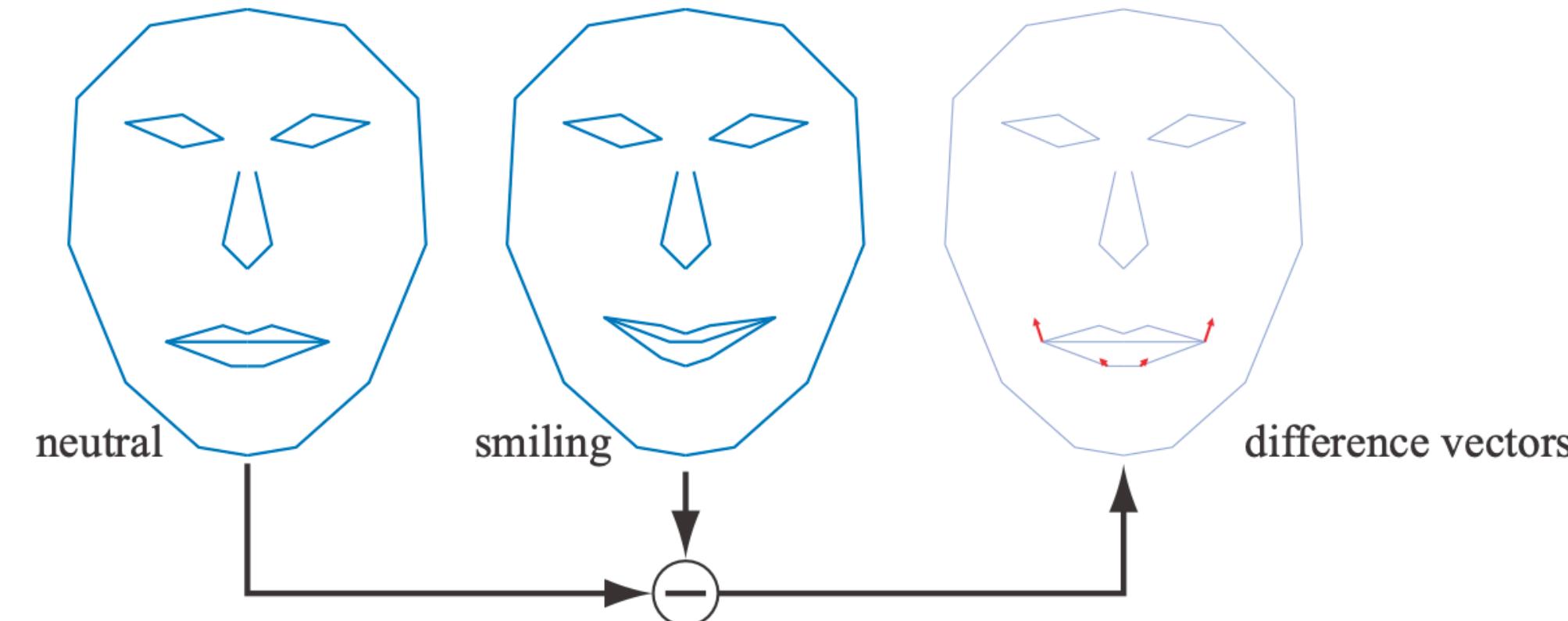
더 나아가서 하나의 정점이 여러 개의 서로 다른 행렬들에 의해 변환되도록 할 수 있으며 모든 정점들이 하나 이상의 행렬들로부터 영향을 받아 팔 전체가 탄력적일 수도 있다.

$$\mathbf{u}(t) = \sum_{i=0}^{n-1} w_i \mathbf{B}_i(t) \mathbf{M}_i^{-1} \mathbf{p}, \text{ where } \sum_{i=0}^{n-1} w_i = 1, \quad w_i \geq 0.$$

기본 정점 혼합의 단점은 원치 않는 접 힘, 비틀림 및 자체 교차가 발생할 수 있다는 것이다. 이는 이중 사원수를 사용하는 것 등으로 해결 할 수 있다.



Morphing



$$\mathbf{m} = (1 - s)\mathbf{p}_0 + s\mathbf{p}_1,$$

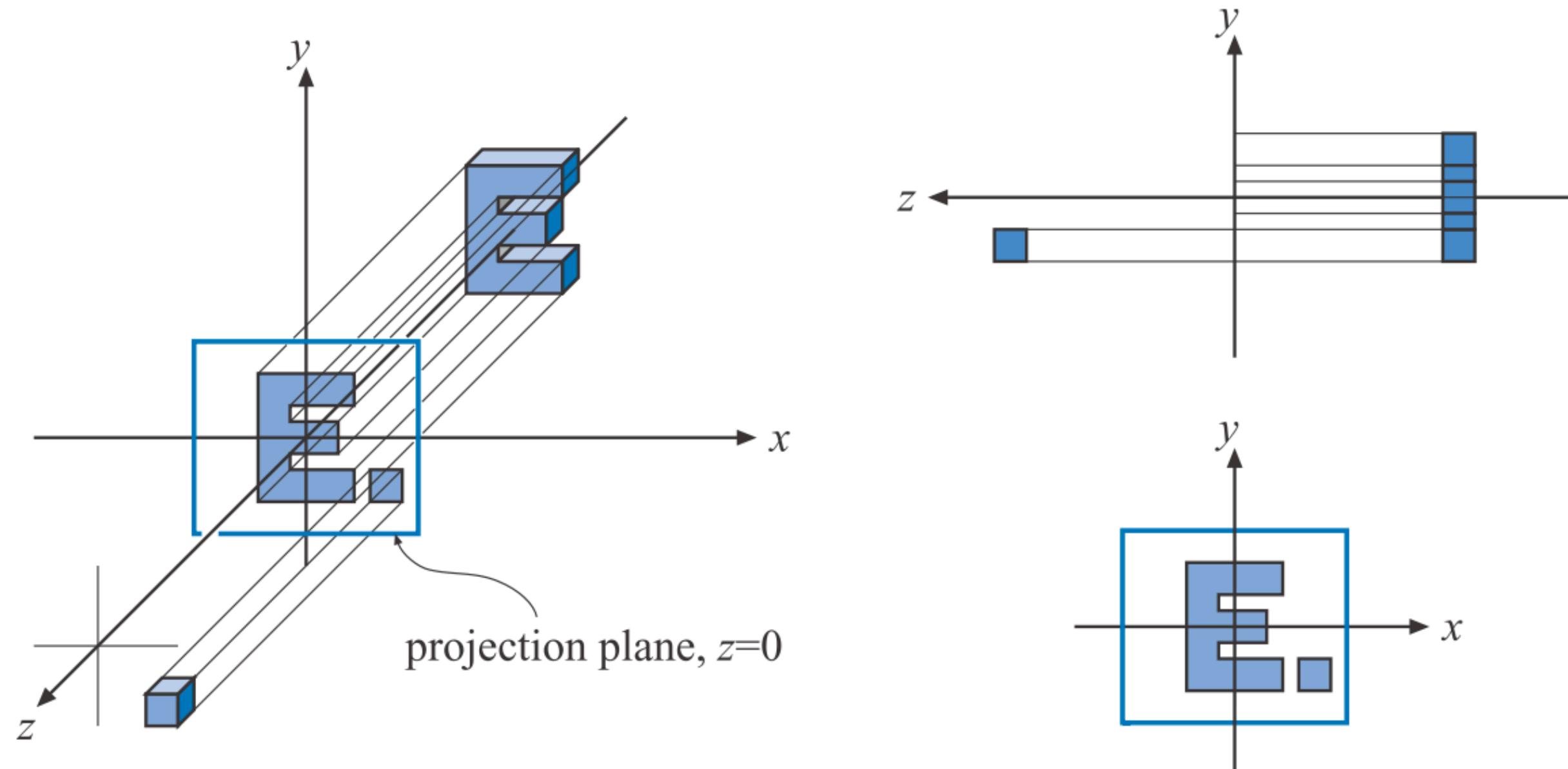
$$\mathcal{M} = \mathcal{N} + \sum_{i=1}^k w_i \mathcal{D}_i.$$



Geometry Cache Playback

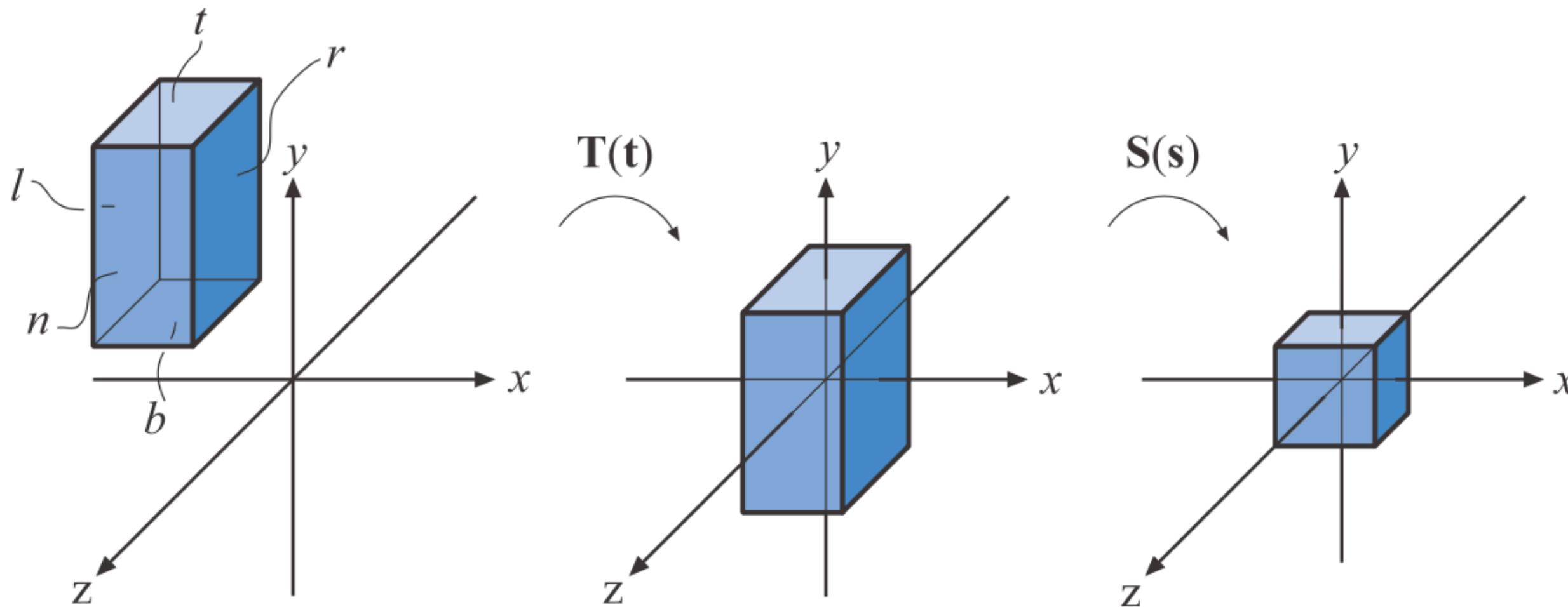
고품질의 애니메이션을 사용할 때 메모리 비용을 줄이기 위해 양자화, 공간 및 시간 예측 등
의 기술을 사용하여 스토리지를 줄여 이 시스템이 실시간으로 데이터를 스트리밍 하는 데 사
용 될 수 있도록 한다.

Orthographic Projection



$$\mathbf{P}_o = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Orthographic Projection



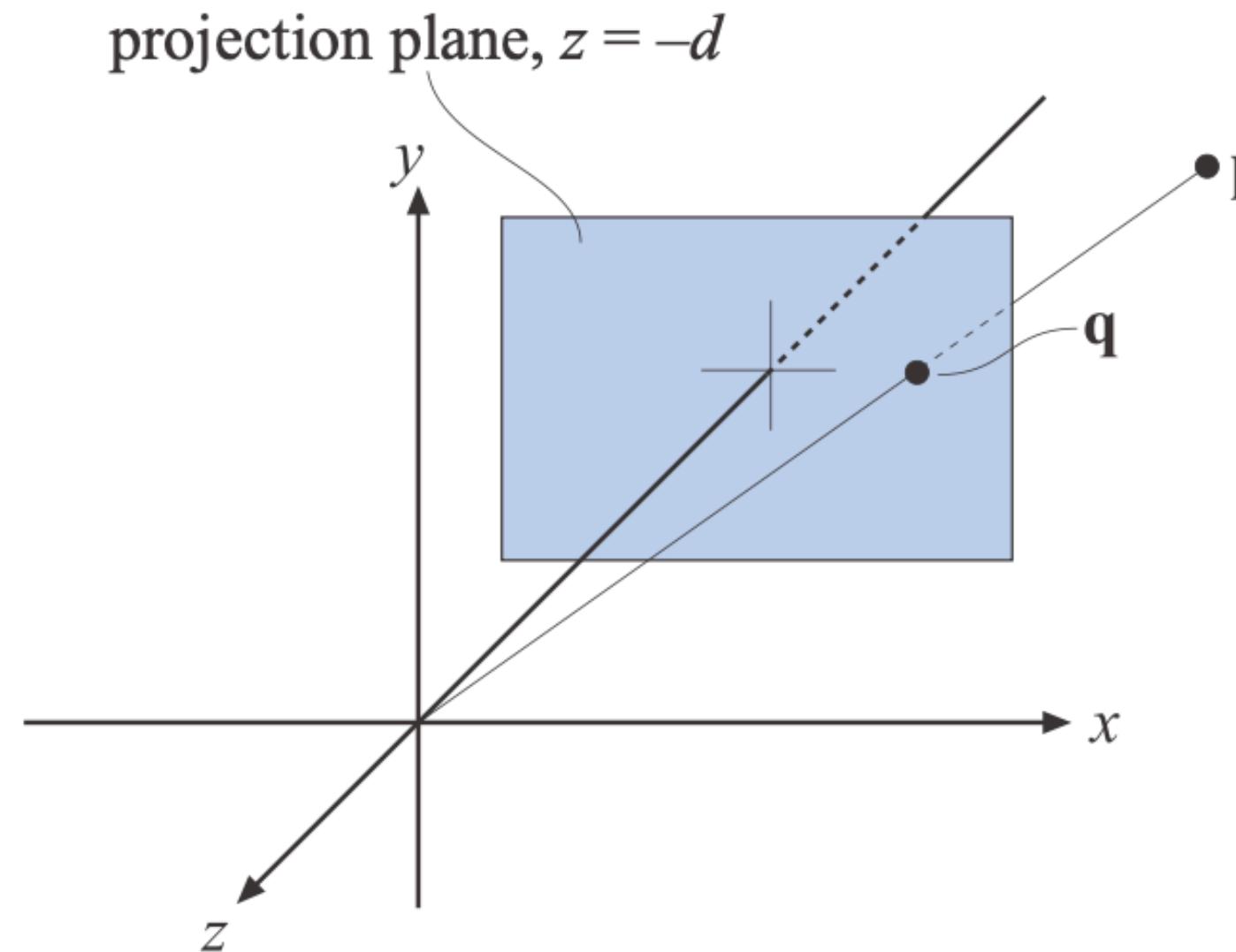
$$\mathbf{P}_o = \mathbf{S}(\mathbf{s})\mathbf{T}(\mathbf{t}) = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{2}{f-n} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -\frac{l+r}{2} \\ 0 & 1 & 0 & -\frac{t+b}{2} \\ 0 & 0 & 1 & -\frac{f+n}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

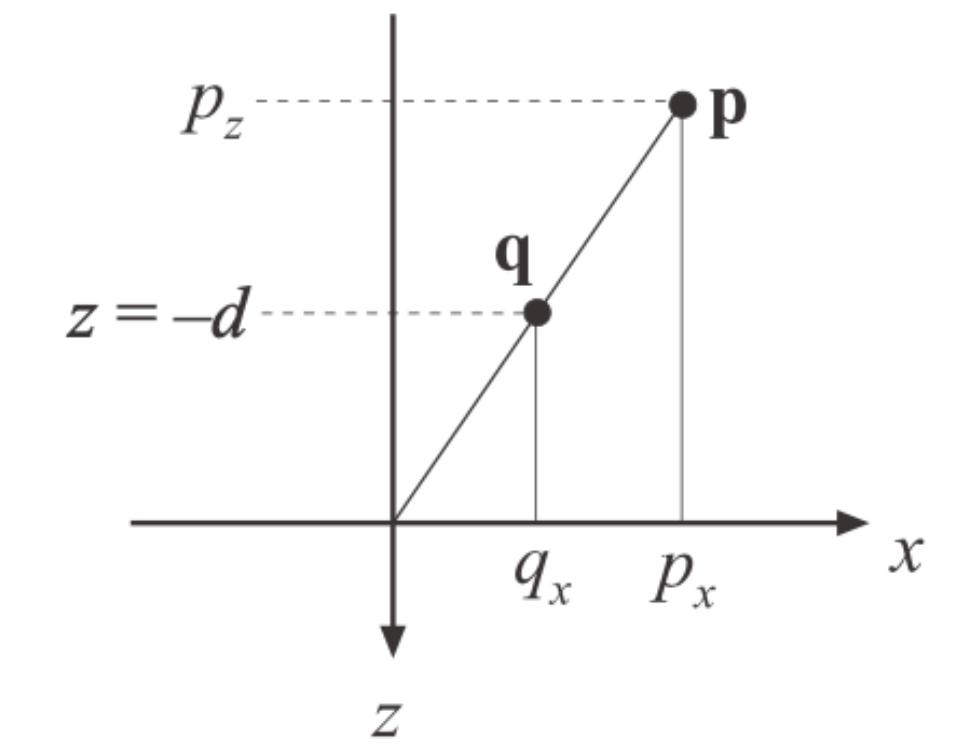
Perspective Projection

$$\mathbf{P}_p = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 0 \end{pmatrix}.$$

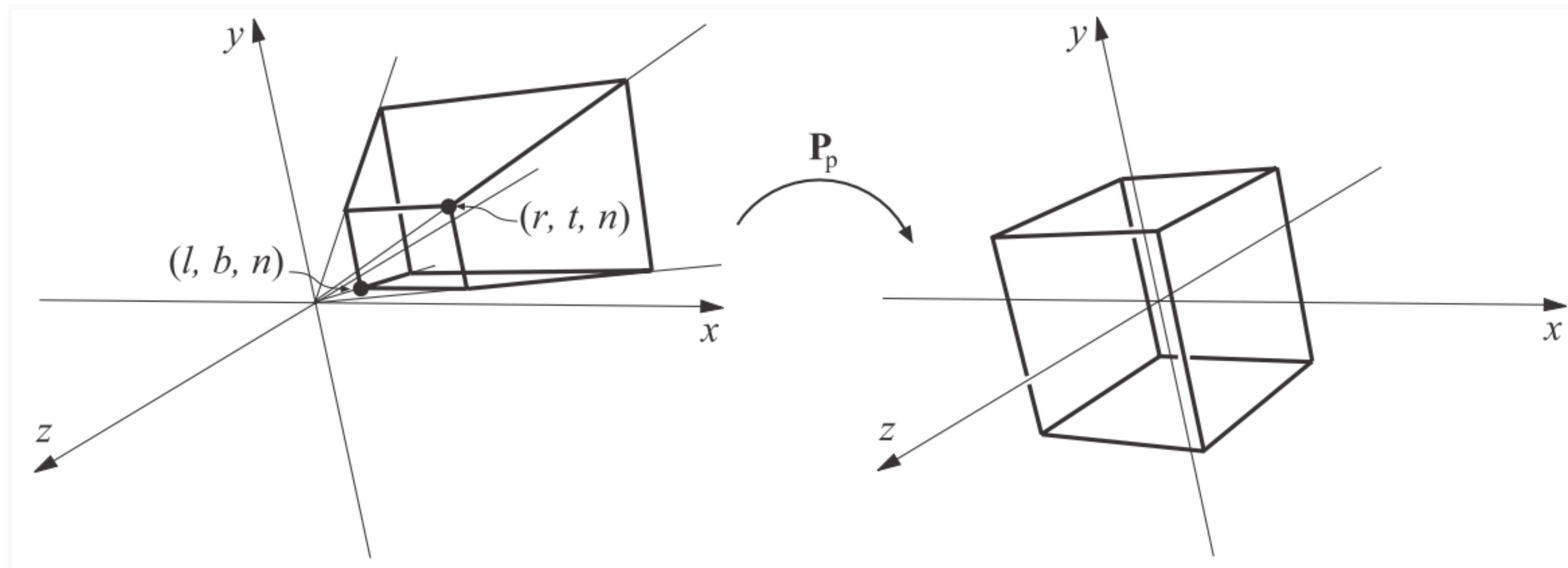
$$\frac{q_x}{p_x} = \frac{-d}{p_z} \iff q_x = -d \frac{p_x}{p_z}.$$



$$\mathbf{q} = \mathbf{P}_p \mathbf{p} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 0 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} p_x \\ p_y \\ p_z \\ -p_z/d \end{pmatrix} \Rightarrow \begin{pmatrix} -dp_x/p_z \\ -dp_y/p_z \\ -d \\ 1 \end{pmatrix}.$$



Perspective Projection



$$\phi = 2 \arctan(w/(2d)),$$

$$\mathbf{P}_p = \begin{pmatrix} \frac{2n}{r-l} & 0 & -\frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & -\frac{r+b}{t-b} & 0 \\ 0 & 0 & \frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

$$\mathbf{p} = (q_x/q_w, q_y/q_w, q_z/q_w, 1).$$

$$\mathbf{P}_p = \begin{pmatrix} \frac{2n}{r-l} & 0 & -\frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & -\frac{r+b}{t-b} & 0 \\ 0 & 0 & 1 & -2n \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

$$\mathbf{P}_{\text{OpenGL}} = \begin{pmatrix} \frac{2n'}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n'}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f'+n'}{f'-n'} & -\frac{2f'n'}{f'-n'} \\ 0 & 0 & -1 & 0 \end{pmatrix}.$$

$$\mathbf{P}_{\text{OpenGL}} = \begin{pmatrix} c/a & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & -\frac{f'+n'}{f'-n'} & -\frac{2f'n'}{f'-n'} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Perspective Projection

