

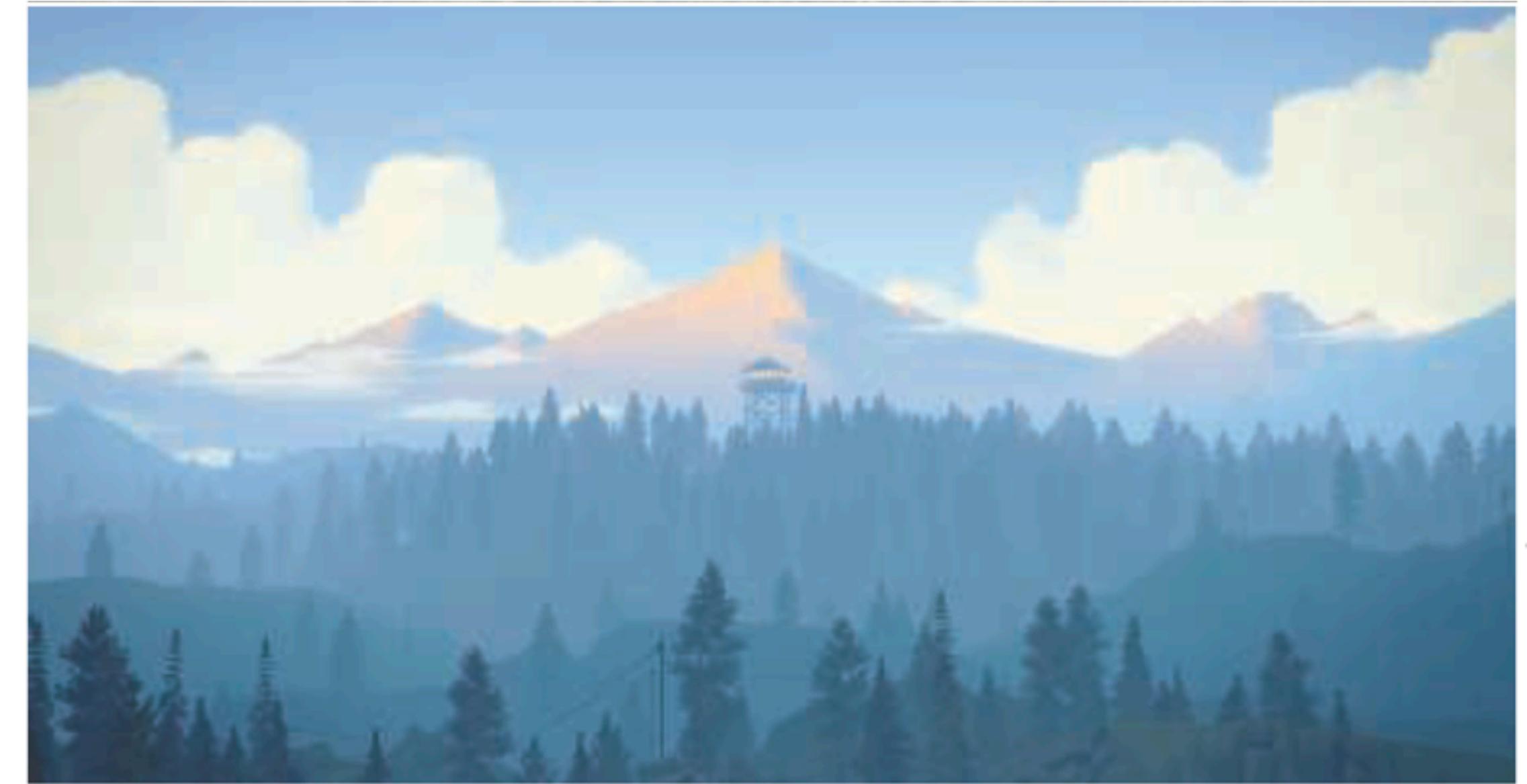
Shading Basics

230203 19101188 고은수

Contents

1. Shading Models
2. Light Sources
3. Implementing Shading Models
4. Aliasing and Antialiasing
5. Transparency, Alpha, and Compositing
6. Display Encoding

Shading Models



Shading Models

Gooch Shading - normal이 빛을 가리키면 따뜻한 톤, 반대 편을 가리키면 차가운 톤 사용

$$\mathbf{c}_{\text{shaded}} = s \mathbf{c}_{\text{highlight}} + (1 - s) (t \mathbf{c}_{\text{warm}} + (1 - t) \mathbf{c}_{\text{cool}}).$$

$$\mathbf{c}_{\text{cool}} = (0, 0, 0.55) + 0.25 \mathbf{c}_{\text{surface}},$$

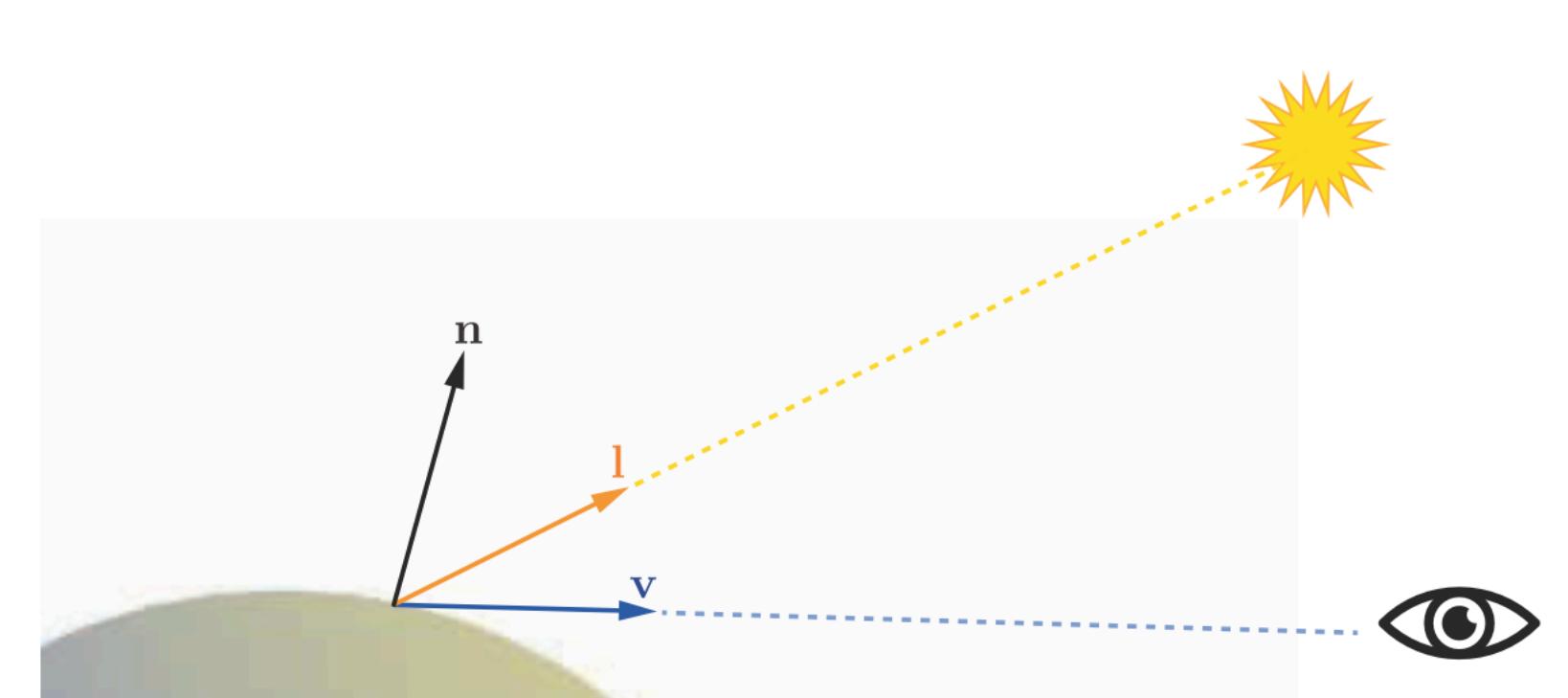
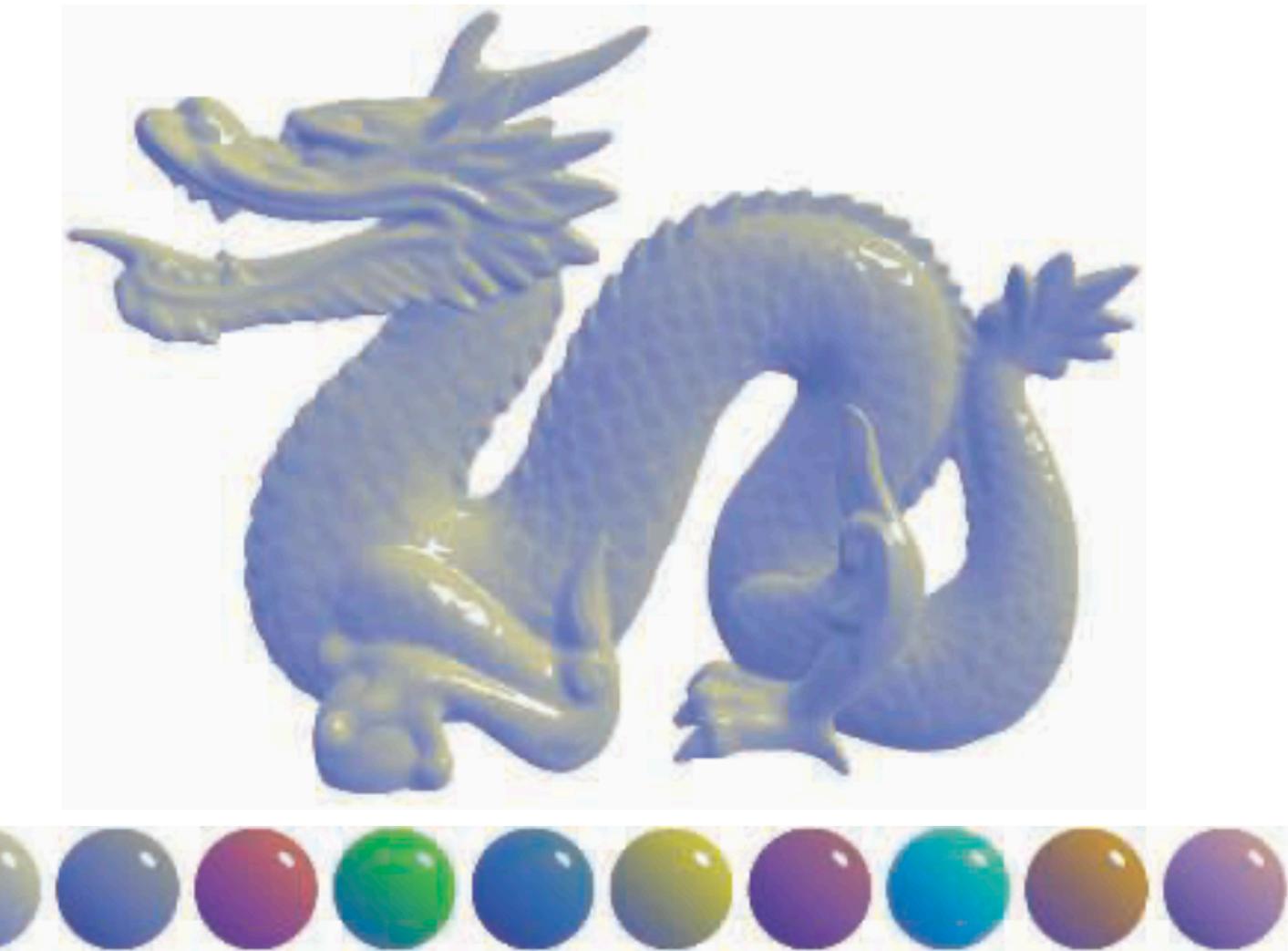
$$\mathbf{c}_{\text{warm}} = (0.3, 0.3, 0) + 0.25 \mathbf{c}_{\text{surface}},$$

$$\mathbf{c}_{\text{highlight}} = (1, 1, 1),$$

$$t = \frac{(\mathbf{n} \cdot \mathbf{l}) + 1}{2},$$

$$\mathbf{r} = 2 (\mathbf{n} \cdot \mathbf{l}) \mathbf{n} - \mathbf{l},$$

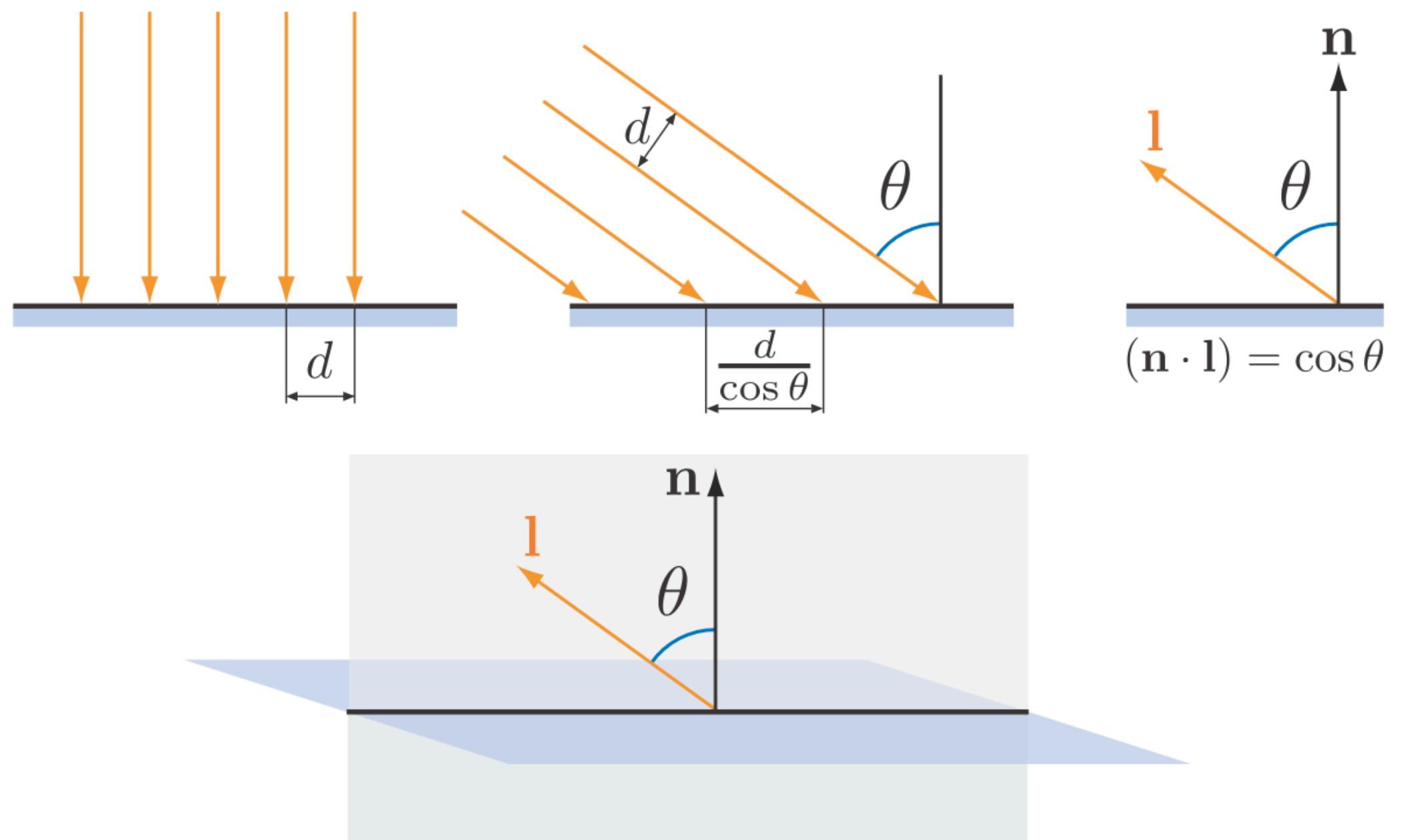
$$s = (100 (\mathbf{r} \cdot \mathbf{v}) - 97)^+.$$



Light Sources

$$\mathbf{c}_{\text{shaded}} = f_{\text{unlit}}(\mathbf{n}, \mathbf{v}) + \sum_{i=1}^n \mathbf{c}_{\text{light}_i} f_{\text{lit}}(\mathbf{l}_i, \mathbf{n}, \mathbf{v}).$$

$$\mathbf{c}_{\text{shaded}} = f_{\text{unlit}}(\mathbf{n}, \mathbf{v}) + \sum_{i=1}^n (\mathbf{l}_i \cdot \mathbf{n})^+ \mathbf{c}_{\text{light}_i} \mathbf{c}_{\text{surface}}.$$



Light Sources

Directional Lights - 감쇠 될 수 있다는 점을 제외하고는 $|$ 과 c_{light} 모두 일정한 광원

Light Sources

Punctual Lights - 위치가 있는 광원

$$\mathbf{l} = \frac{\mathbf{p}_{\text{light}} - \mathbf{p}_0}{\|\mathbf{p}_{\text{light}} - \mathbf{p}_0\|}.$$

$$\mathbf{d} = \mathbf{p}_{\text{light}} - \mathbf{p}_0,$$

$$r = \sqrt{\mathbf{d} \cdot \mathbf{d}},$$

$$\mathbf{l} = \frac{\mathbf{d}}{r}.$$

Light Sources

Point/Omni Lights - 점광원, 모든 방향으로 균일한 빛

$$\mathbf{c}_{\text{light}}(r) = \mathbf{c}_{\text{light}_0} \left(\frac{r_0}{r} \right)^2.$$

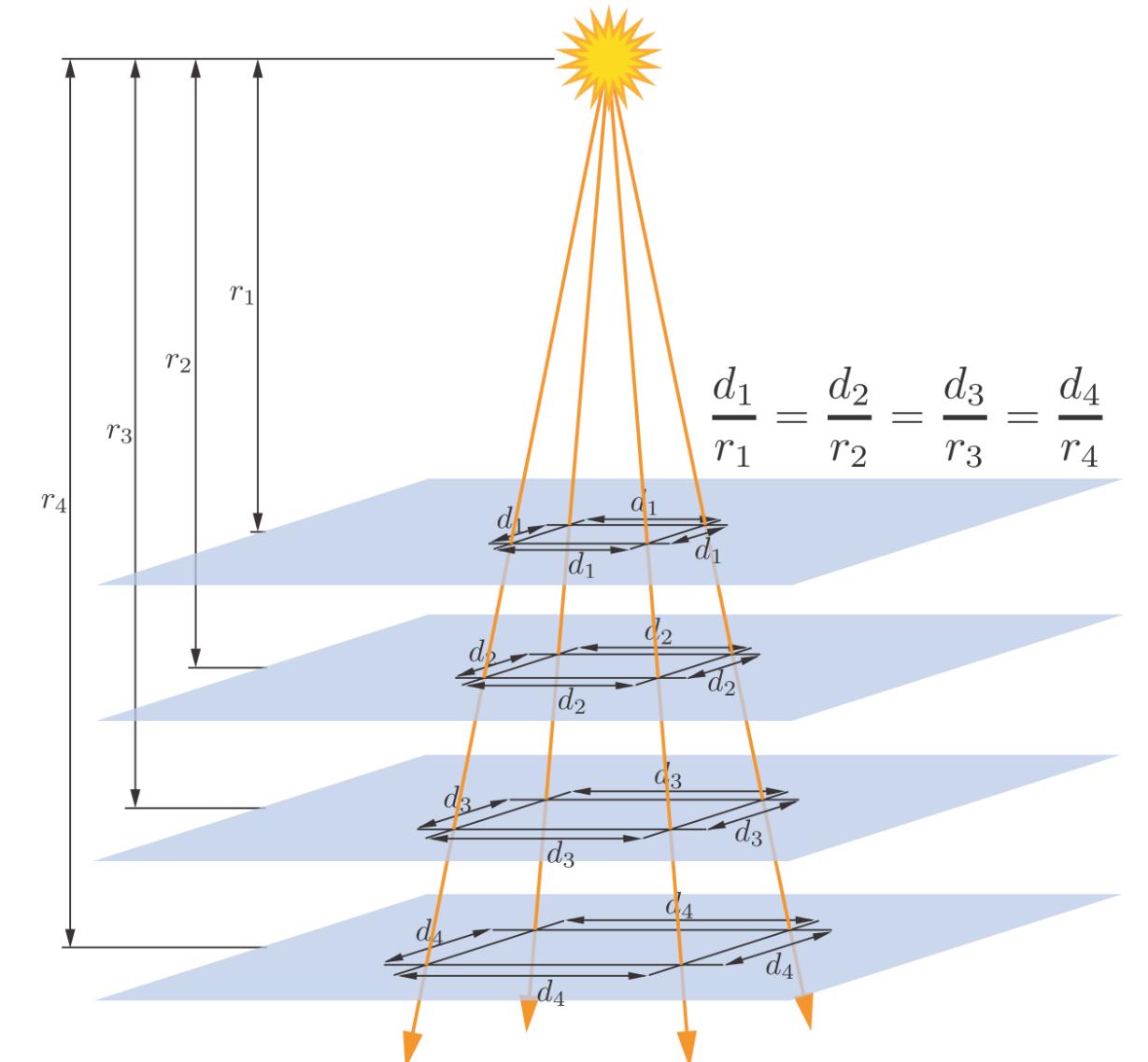
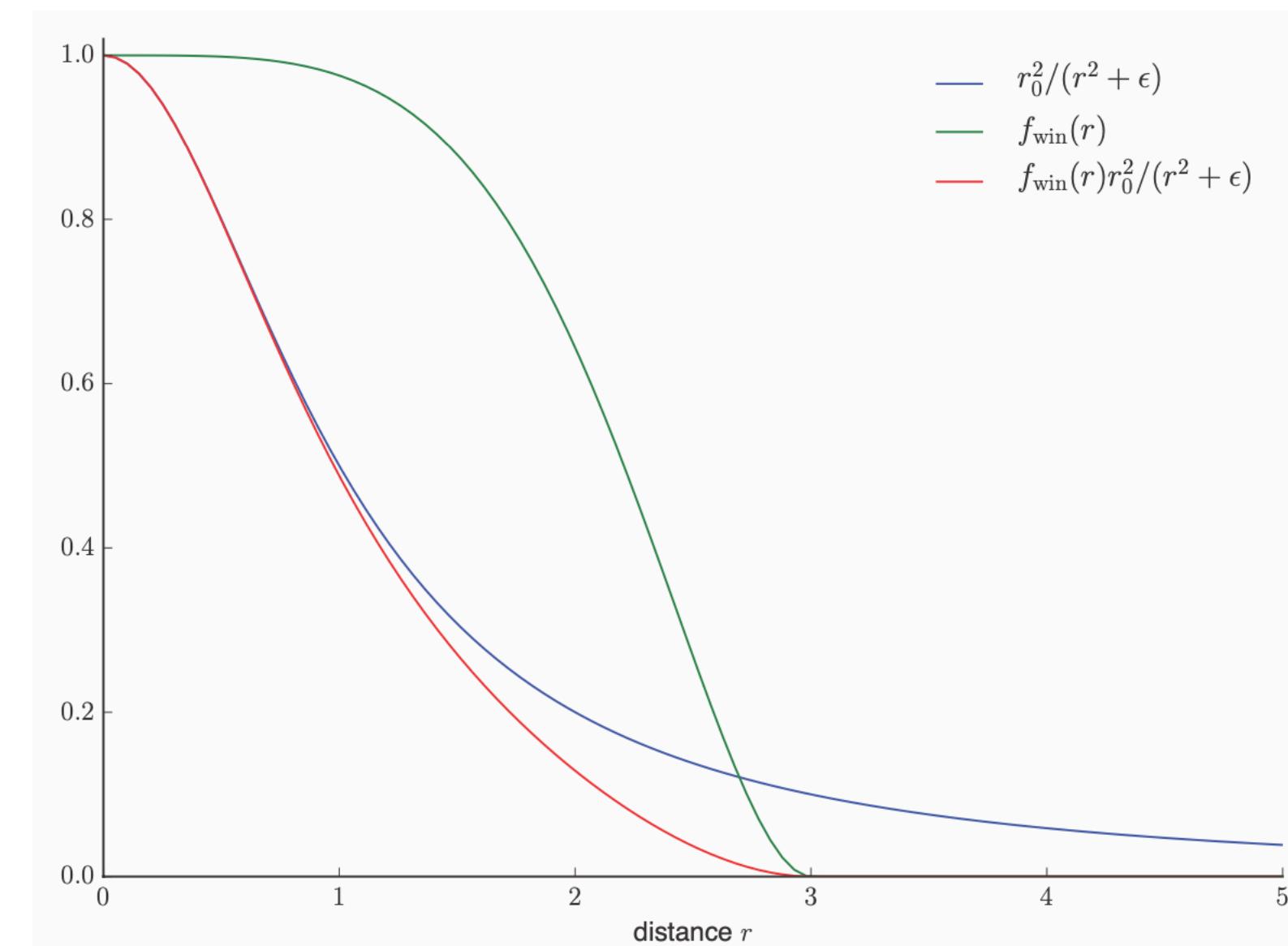
$$\mathbf{c}_{\text{light}}(r) = \mathbf{c}_{\text{light}_0} \frac{r_0^2}{r^2 + \epsilon}.$$

$$\mathbf{c}_{\text{light}}(r) = \mathbf{c}_{\text{light}_0} \left(\frac{r_0}{\max(r, r_{\min})} \right)^2.$$

$$f_{\text{win}}(r) = \left(1 - \left(\frac{r}{r_{\max}} \right)^4 \right)^{+2}.$$

$$\mathbf{c}_{\text{light}}(r) = \mathbf{c}_{\text{light}_0} f_{\text{dist}}(r),$$

$$f_{\text{dist}}(r) = \left(1 - \left(\frac{r}{r_{\max}} \right)^2 \right)^{+2}.$$



Light Sources

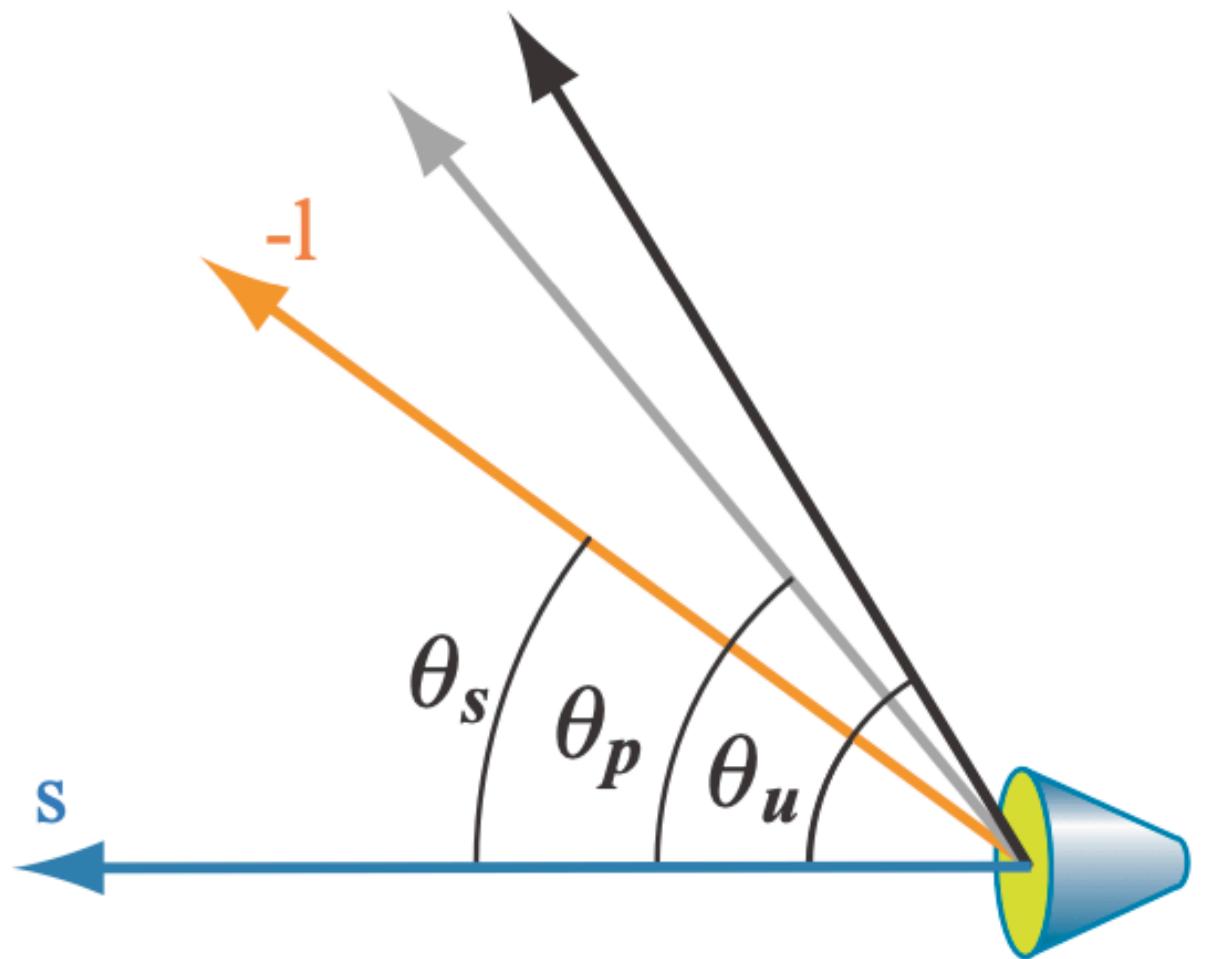
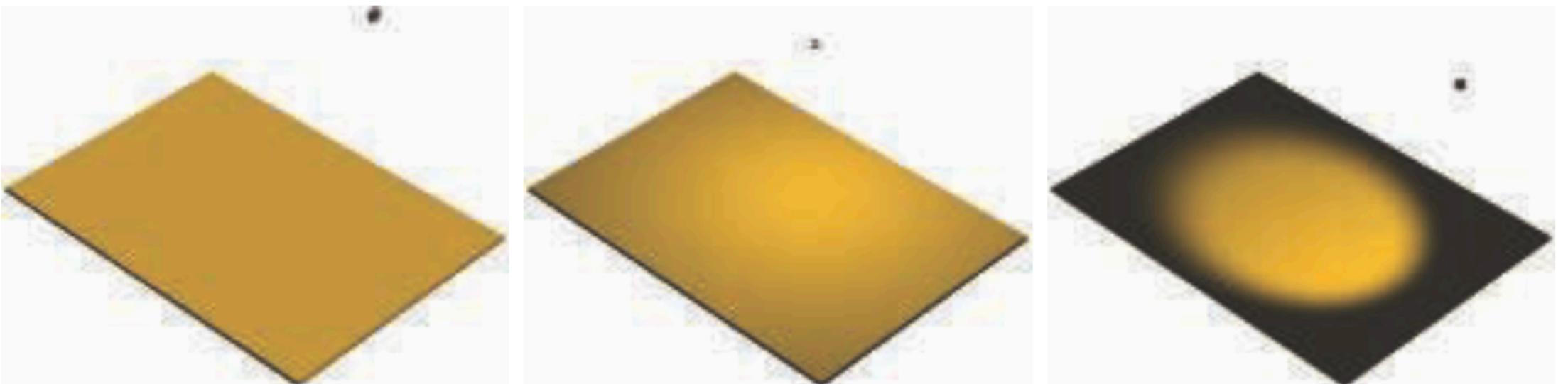
Spotlights

$$\mathbf{c}_{\text{light}} = \mathbf{c}_{\text{light}_0} f_{\text{dist}}(r) f_{\text{dir}}(\mathbf{l}).$$

$$t = \left(\frac{\cos \theta_s - \cos \theta_u}{\cos \theta_p - \cos \theta_u} \right)^+,$$

$$f_{\text{dir}_F}(\mathbf{l}) = t^2,$$

$$f_{\text{dir}_T}(\mathbf{l}) = \text{smoothstep}(t) = t^2(3 - 2t).$$



Other Light Types

Capsule Lights - 점 대신 선 세그먼트를 스스로 사용

영역 조명 렌더링 기법

- 부분적으로 가려진 영역 조명에서 발생하는 그림자 가장자리의 연화를 시뮬레이션
- 표면 음영에 대한 영역 조명의 효과를 시뮬레이션 하는 기법

Implementing Shading Models

Frequency of Evaluation : 쉐이딩 모델을 설계할 때 계산은 평가 빈도에 따라 나누어짐

1. 전체 호출에 대해 주어진 계산의 결과가 항상 일정한지 결정 - 일반적으로 CPU에서 계산 수행
2. 계산이 너무 느리게 변경되어 모든 프레임을 업데이트 할 필요가 없는 경우
3. 프레임, 모델, 추첨 호출당 한 번 계산되는 계산

쉐이딩 연산은 프로그래밍 가능한 모든 단계에서 실행 될 수 있음

- Vertex shader—Evaluation per pre-tessellation vertex
- Hull shader—Evaluation per surface patch
- Domain shader—Evaluation per post-tessellation vertex
- Geometry shader—Evaluation per primitive
- Pixel shader—Evaluation per pixel

Implementing Shading Models

대부분의 쉐이딩 연산은 픽셀당 수행되고 일반적으로 픽셀 쉐이더에서 구현된다.

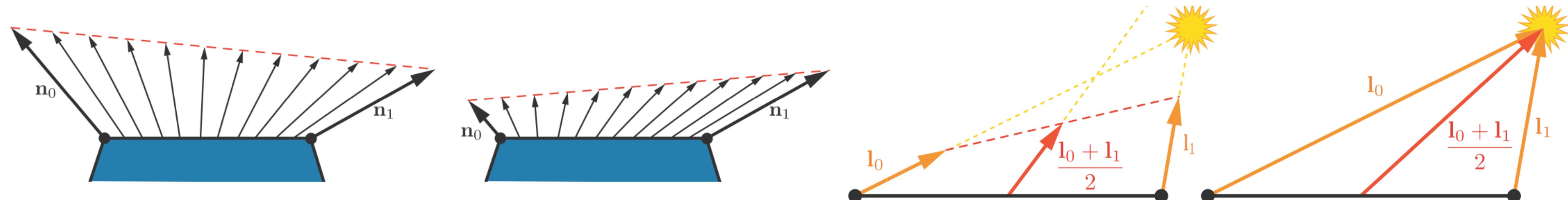


Implementing Shading Models

Pixel shader에서 쉐이딩 모델을 선택하고 vertex shader에서 나머지를 계산 - 시각적 아티팩트를 초래하지 않고 계산 절약 가능

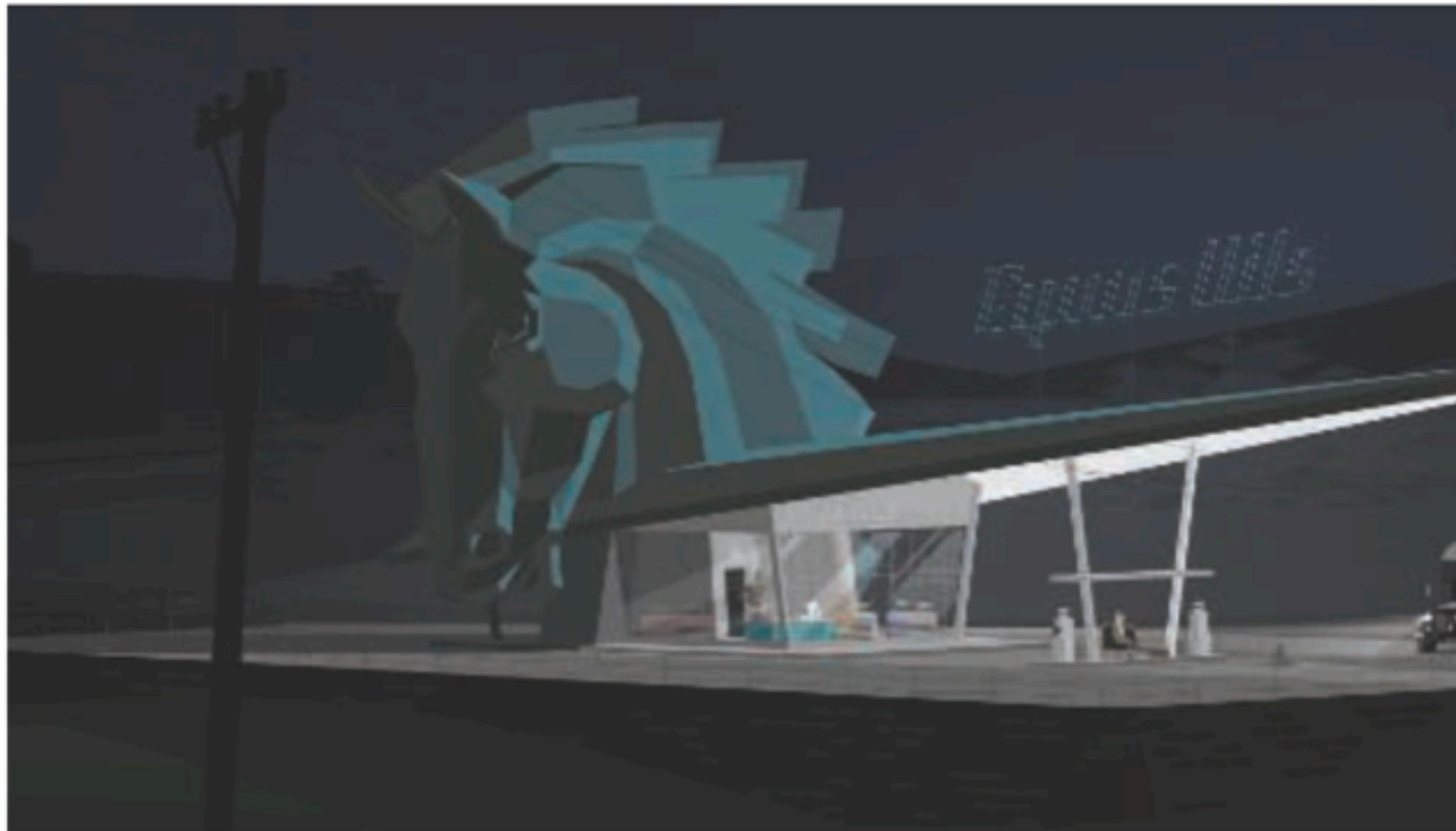
Vertex shader는 기하학적 변환 및 변형과 같은 작업 담당 - 지표면의 위치, 지표면 정규 분포, 지표면 normal 벡터

Vertex shader가 항상 단위 길이의 표면 normal을 생성하더라도 보간은 길이를 변경 할 수 있음
또한 정점 혼합의 부작용과 같이 정점 간에 정규 길이가 크게 달라지는 경우 보간을 왜곡시킬 수 있음. 따라서 보간 전후 vertex shader, pixel shader 모두 보간벡터 정규화



Implementing Shading Models

Flat shading - geometry shader에서 수행 가능하지만 최근 구현에서는 일반적으로 vertex shader 사용



Implementing Shading Models

Implementing Example

$$\mathbf{c}_{\text{shaded}} = \frac{1}{2}\mathbf{c}_{\text{cool}} + \sum_{i=1}^n (\mathbf{l}_i \cdot \mathbf{n})^+ \mathbf{c}_{\text{light}_i} (s_i \mathbf{c}_{\text{highlight}} + (1 - s_i) \mathbf{c}_{\text{warm}}),$$

$$\mathbf{c}_{\text{cool}} = (0, 0, 0.55) + 0.25 \mathbf{c}_{\text{surface}},$$

$$\mathbf{c}_{\text{warm}} = (0.3, 0.3, 0) + 0.25 \mathbf{c}_{\text{surface}},$$

$$\mathbf{c}_{\text{highlight}} = (2, 2, 2),$$

$$\mathbf{r}_i = 2(\mathbf{n} \cdot \mathbf{l}_i)\mathbf{n} - \mathbf{l}_i,$$

$$s_i = (100(\mathbf{r}_i \cdot \mathbf{v}) - 97)^+.$$

$$\mathbf{c}_{\text{shaded}} = f_{\text{unlit}}(\mathbf{n}, \mathbf{v}) + \sum_{i=1}^n (\mathbf{l}_i \cdot \mathbf{n})^+ \mathbf{c}_{\text{light}_i} f_{\text{lit}}(\mathbf{l}_i, \mathbf{n}, \mathbf{v}).$$

$$f_{\text{unlit}}(\mathbf{n}, \mathbf{v}) = \frac{1}{2}\mathbf{c}_{\text{cool}},$$

$$f_{\text{lit}}(\mathbf{l}_i, \mathbf{n}, \mathbf{v}) = s_i \mathbf{c}_{\text{highlight}} + (1 - s_i) \mathbf{c}_{\text{warm}},$$

Implementing Shading Models

```
in vec3 vPos;
in vec3 vNormal;
out vec4 outColor;
struct Light {
    vec4 position;
    vec4 color;
};
uniform LightUBlock {
    Light uLights[MAXLIGHTS];
};

uniform uint uLightCount;
vec3 lit(vec3 l, vec3 n, vec3 v) {
    vec3 r_l = reflect(-l, n);
    float s = clamp(100.0 * dot(r_l, v) - 97.0, 0.0, 1.0);
    vec3 highlightColor = vec3(2,2,2);
    return mix(uWarmColor, highlightColor, s);
}

void main() {
    vec3 n = normalize(vNormal);
    vec3 v = normalize(uEyePosition.xyz - vPos);
    outColor = vec4(uFUnlit, 1.0);

    for (uint i = 0u; i < uLightCount; i++) {
        vec3 l = normalize(uLights[i].position.xyz - vPos);
        float NdL = clamp(dot(n, l), 0.0, 1.0);
        outColor.rgb += NdL * uLights[i].color.rgb * lit(l,n,v);
    }
}
```

```
layout(location=0) in vec4 position;
layout(location=1) in vec4 normal;
out vec3 vPos;
out vec3 vNormal;
void main() {
    vec4 worldPosition = uModel * position;
    vPos = worldPosition.xyz;
    vNormal = (uModel * normal).xyz;
    gl_Position = viewProj * worldPosition;
}
var fSource = document.getElementById("fragment").text.trim();

var maxLights = 10;
fSource = fSource.replace(/MAXLIGHTS/g, maxLights.toString());

var fragmentShader = gl.createShader(gl.FRAGMENT_SHADER);
gl.shaderSource(fragmentShader, fSource);
gl.compileShader(fragmentShader);
```

Implementing Shading Models

Material Systems - 렌더링 프레임워크는 단일 쉐이더만 구현하는 경우는 거의 없고 다양한 materials, shading models, shader를 처리하기 위한 전용 시스템 필요

- 기하학적 처리를 사용하여 surface shading 구성. Surface shading은 material에 따라 다르고 geometry processing은 mesh에 따라 다름
- 픽셀 폐기 및 혼합과 같은 합성 작업을 사용하여 surface shading 합성
- 쉐이딩 모델 자체의 계산을 사용하여 쉐이딩 모델 매개변수를 계산하는 데 사용되는 연산 구성
- 개별적으로 선택 가능한 재료 형상, 선택 로직 및 나머지 쉐이더 구성
- 광원 평가를 사용한 shading model 구성 및 매개변수 계산

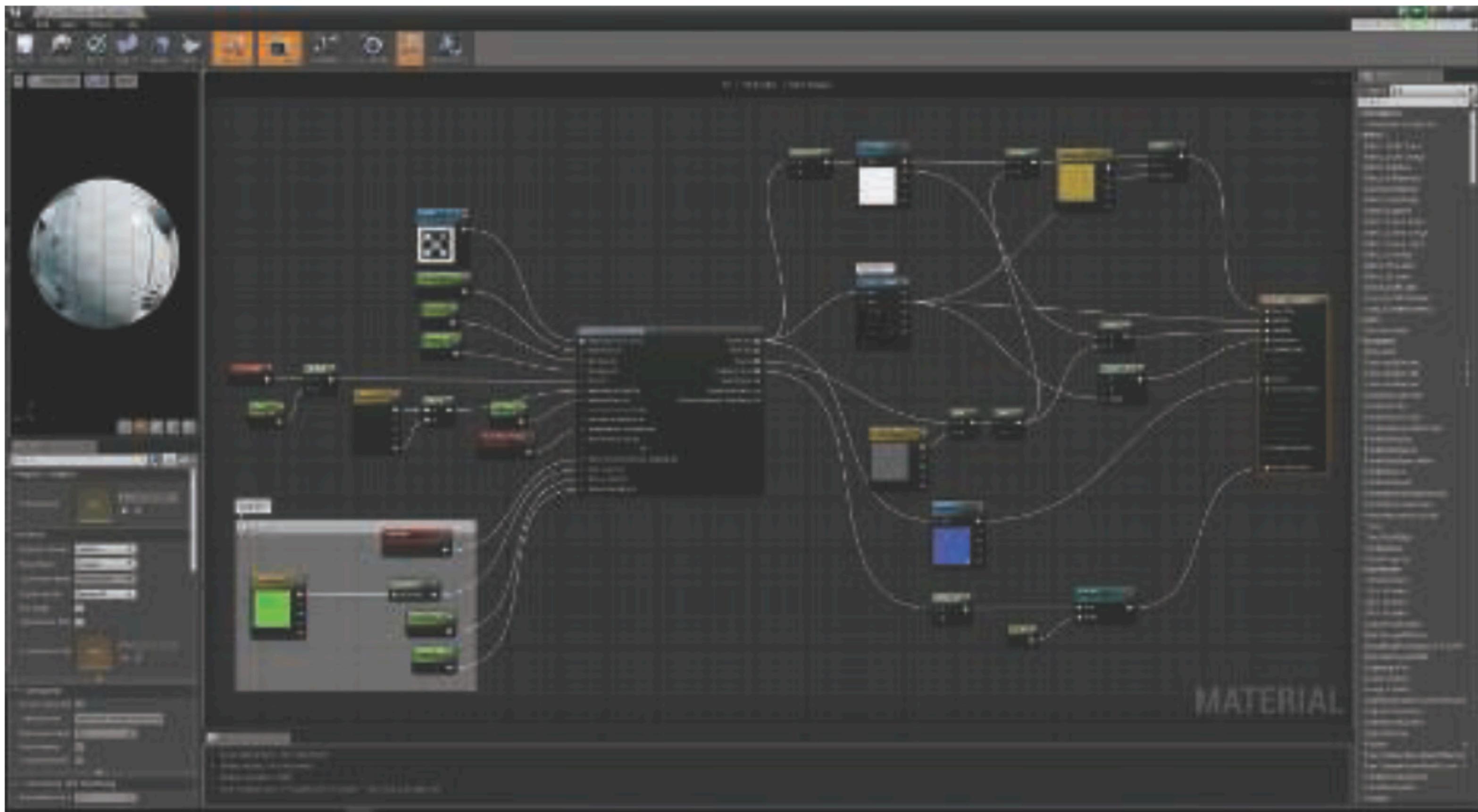
Implementing Shading Models

초기 렌더링 시스템은 상대적으로 적은 수의 쉐이더 변형을 가졌으며 각각의 쉐이더 변형이 수동으로 작성됨. 이로 인해 각 변형은 최종 쉐이더 프로그램에 대해 완전한 지식으로 최적화 될 수 있는 이점을 가지고 있으나 변형의 수가 증가함에 따라 실용적이지 않게 되었음.

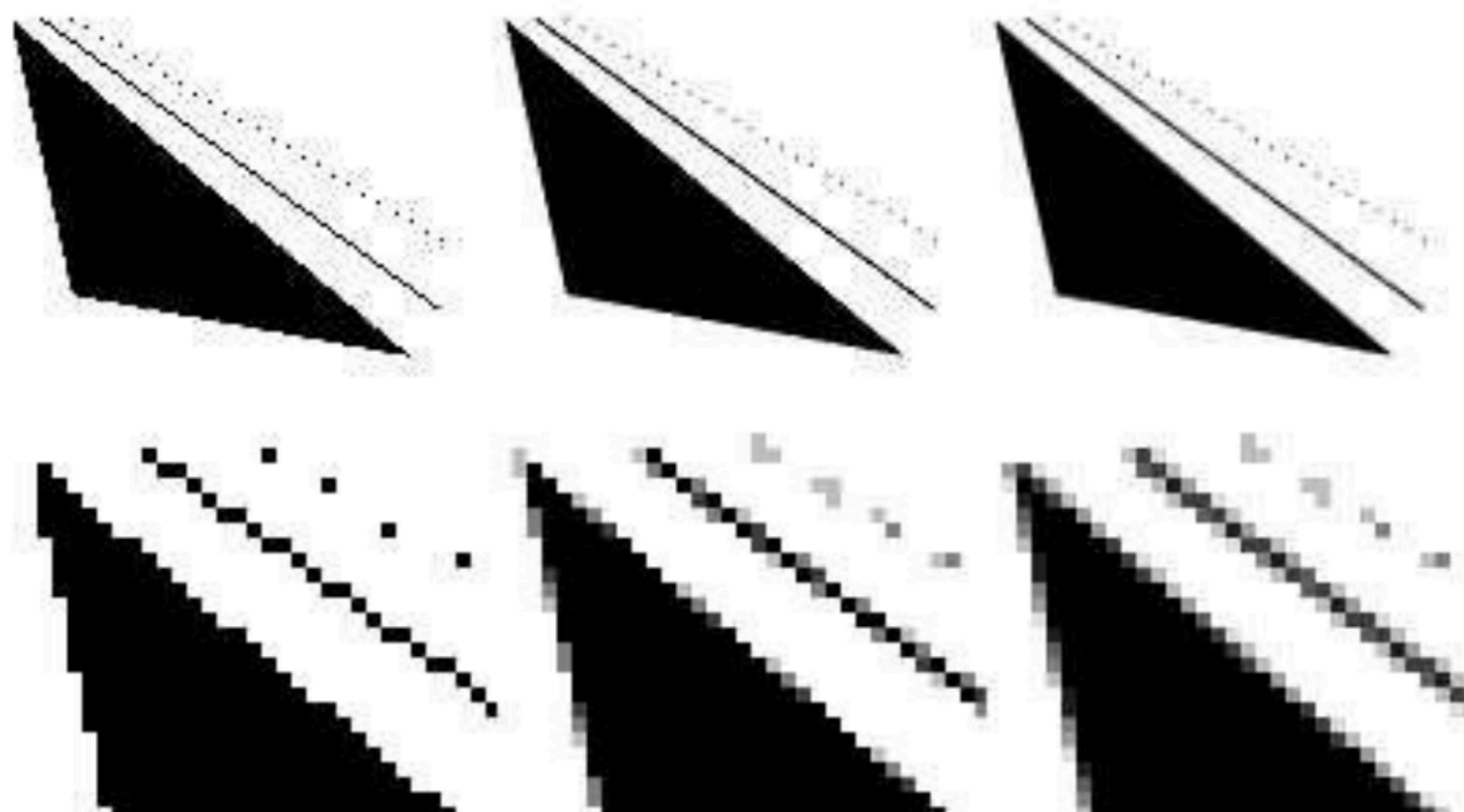
현대의 material system은 런타임과 컴파일 타임 쉐이더 변형을 모두 사용함. 모든 일이 컴파일 시간에만 처리되진 않지만 전체적인 복잡성과 변형 수는 계속 증가하기 때문에 많은 수의 쉐이더 변형을 컴파일 해야함.

- Code reuse
- Subtractive
- Additive - 기술 예술가와 같은 비기술자들이 새로운 재료 템플릿을 쉽게 작성 가능
- Template-based - 인터페이스가 정의되며, 인터페이스에 부합하는 한 다양한 구현 연결 가능

Implementing Shading Models



Aliasing and Antialiasing

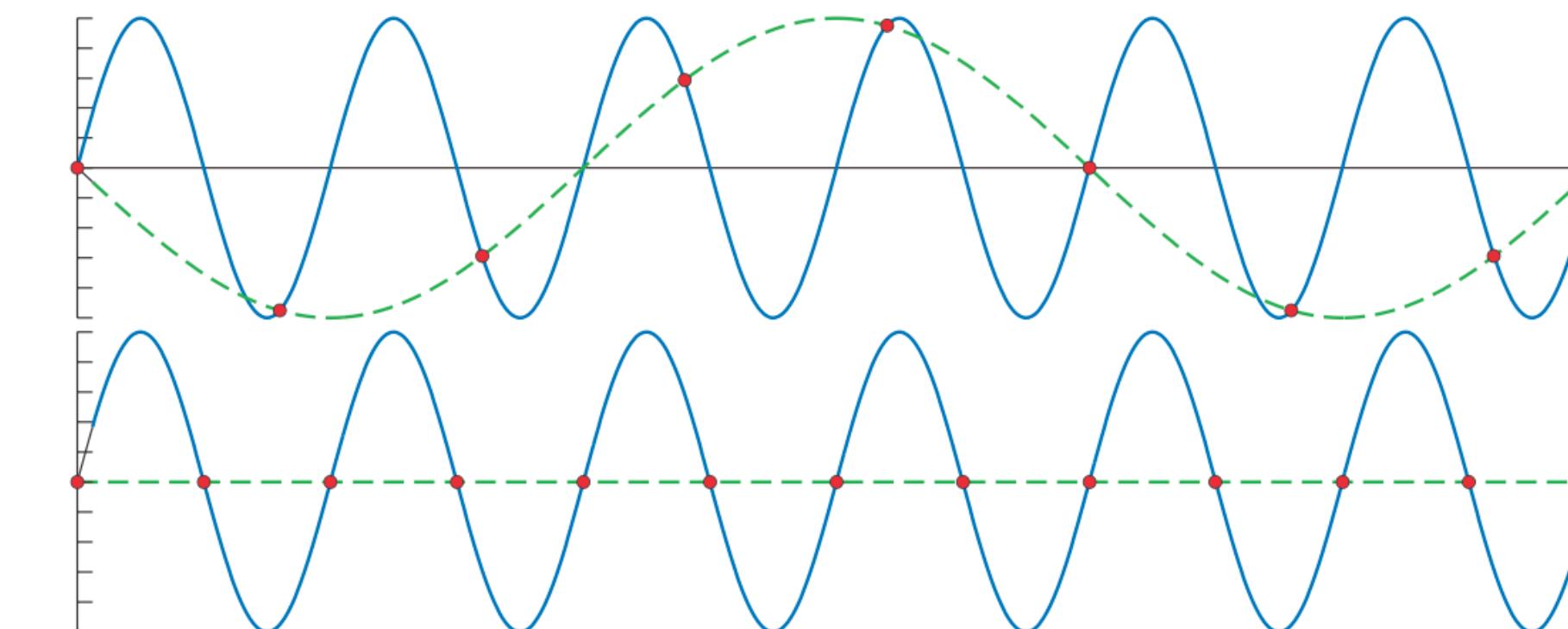
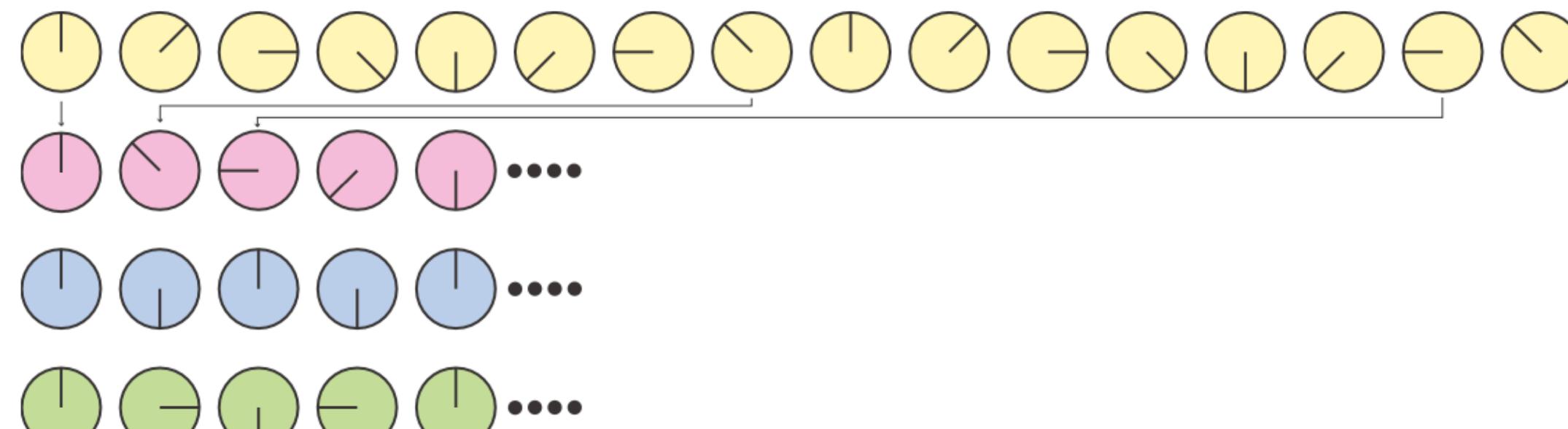
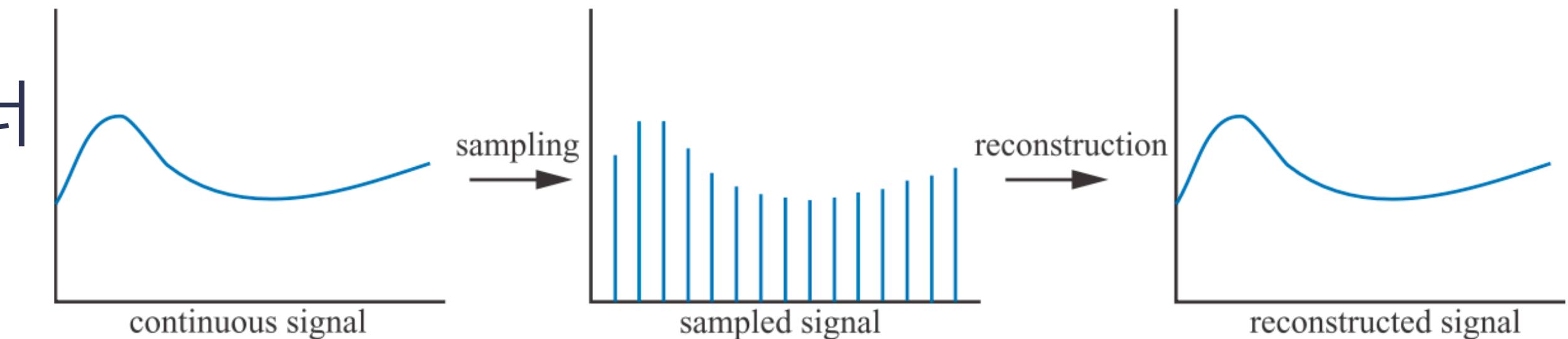


Aliasing and Antialiasing

Sampling and Filtering Theory

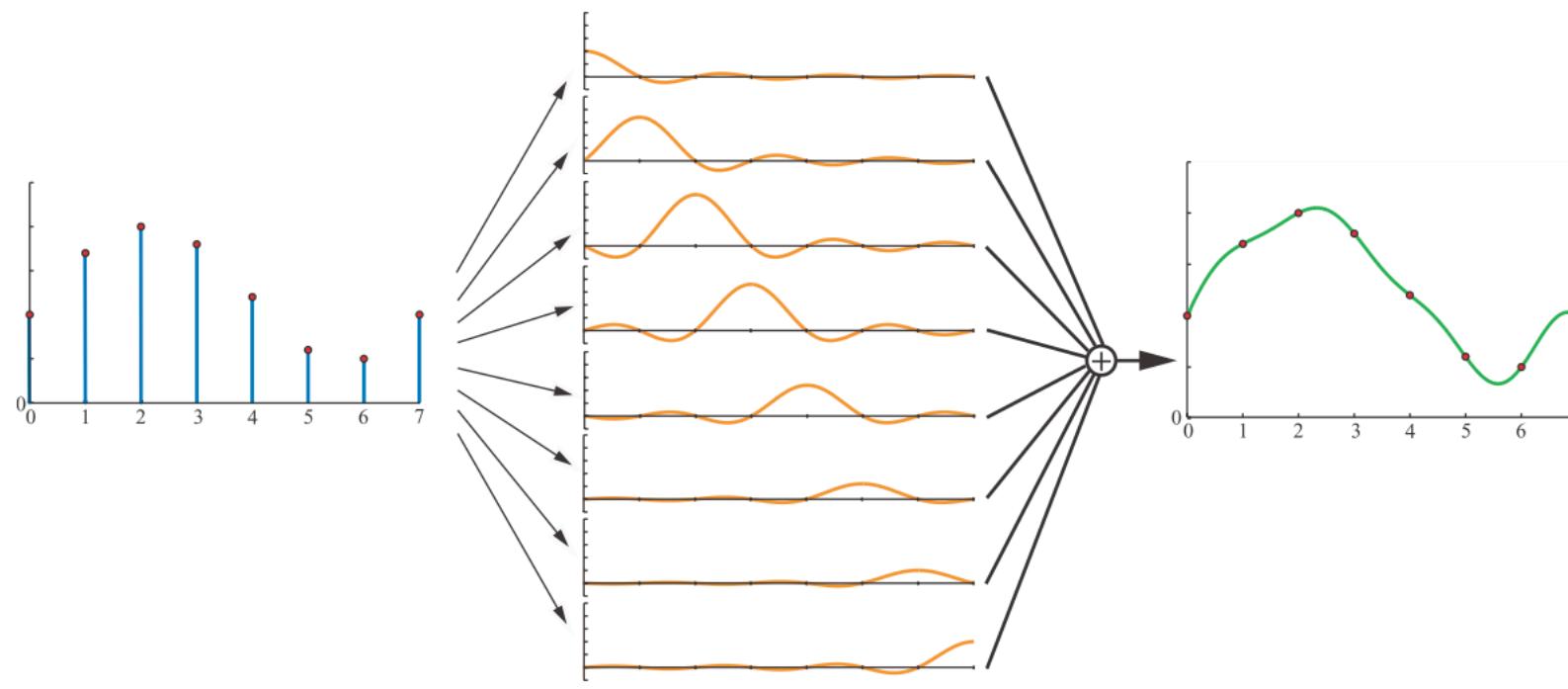
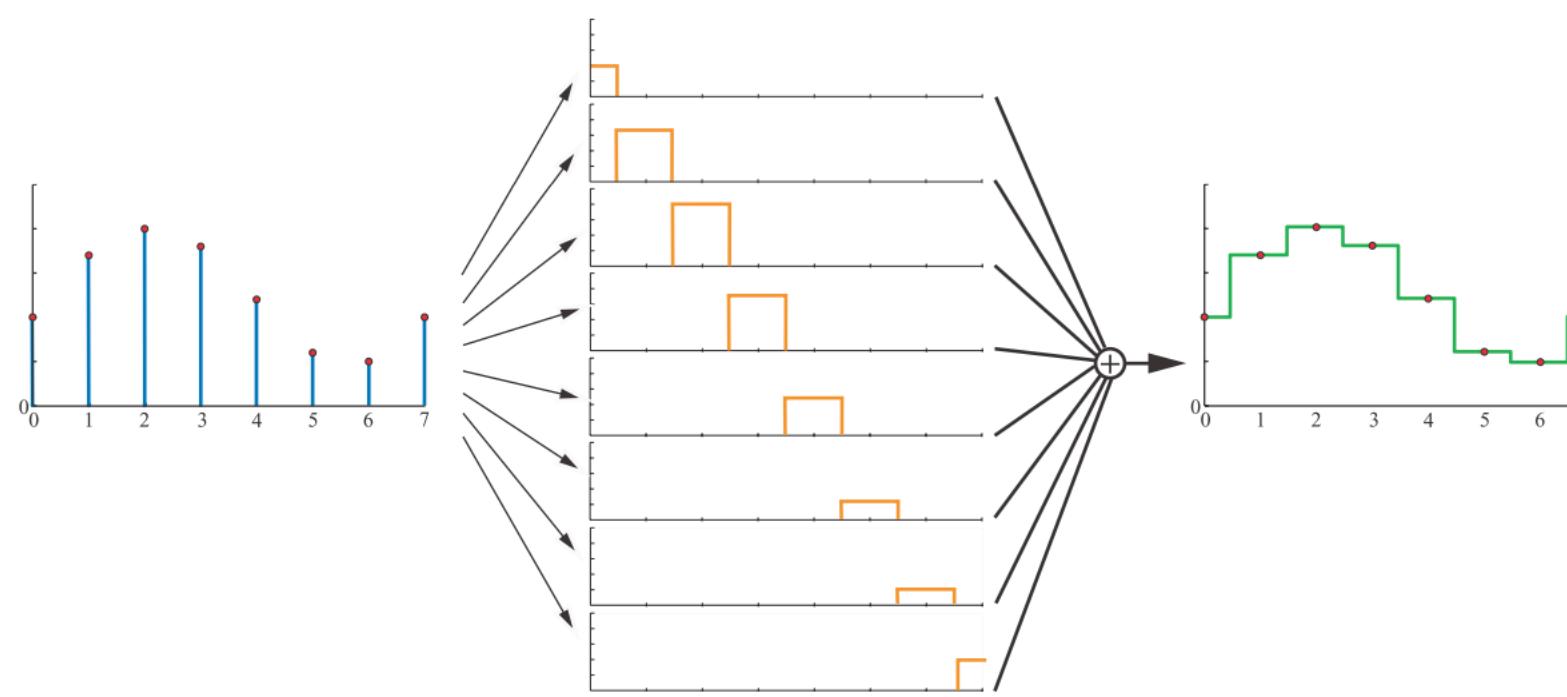
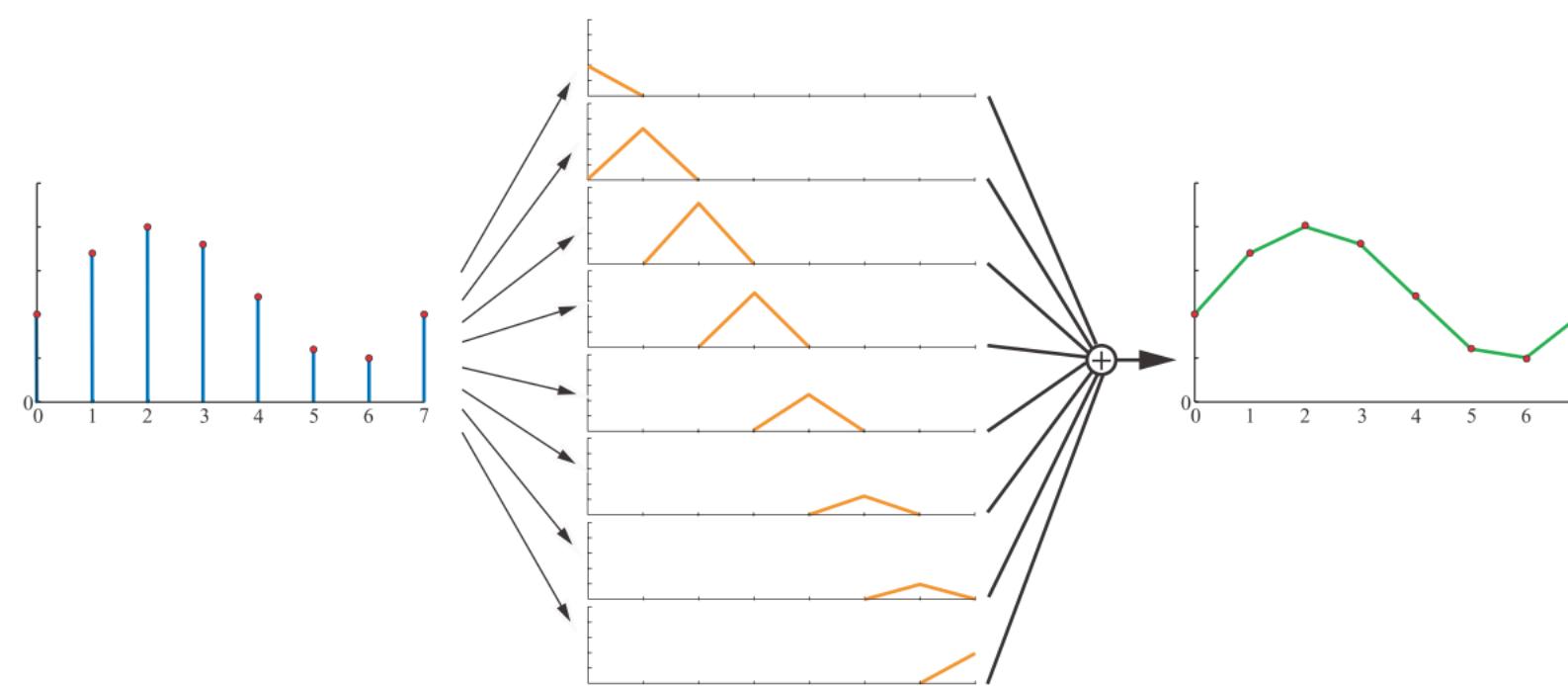
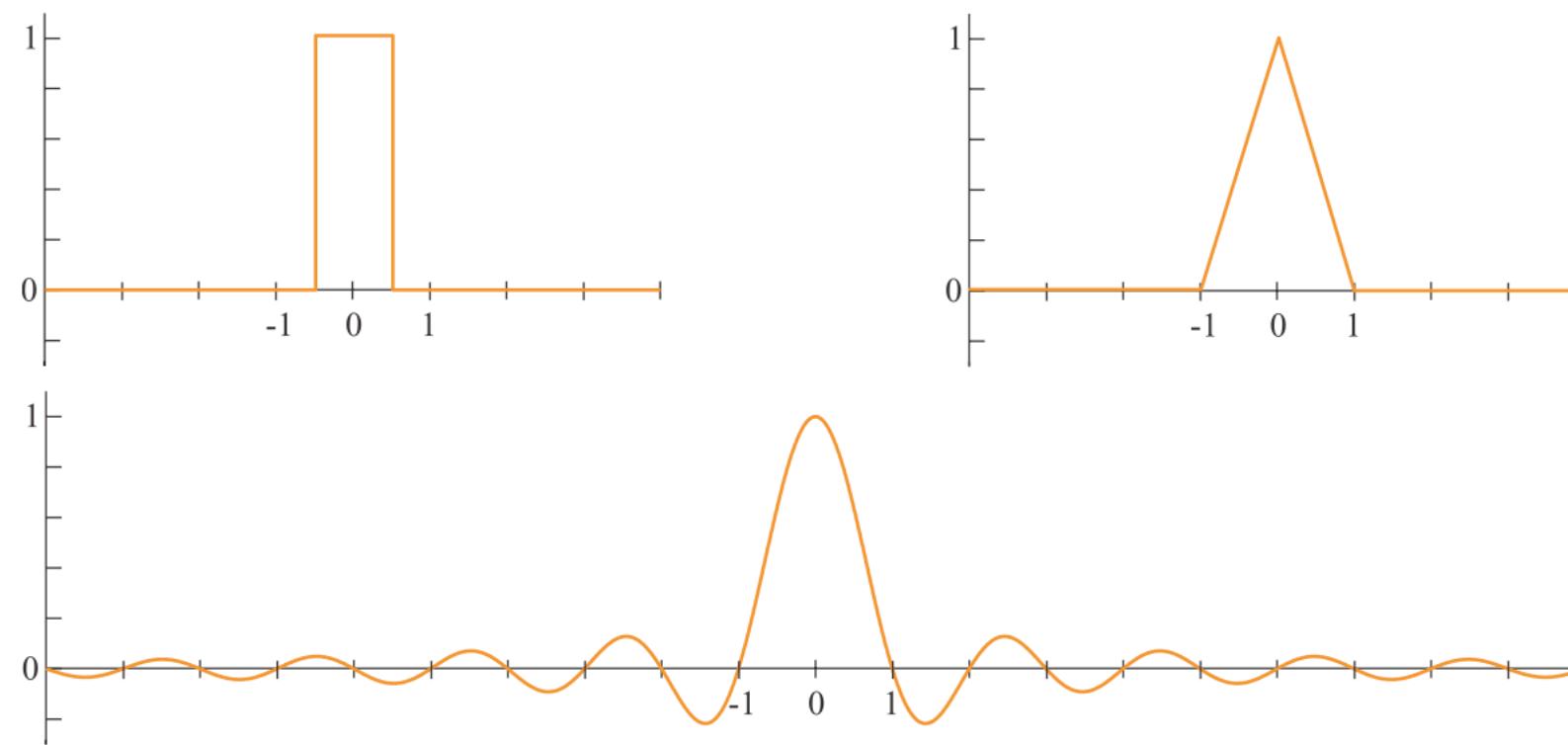
Nyquist주파수 - 최대 주파수의 2배 이상

텍스처 샘플의 주파수가 Nyquist 한계보다 너무 높으면 텍스처의 주파수 제한 필요



Aliasing and Antialiasing

Reconstruction



$$\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}.$$

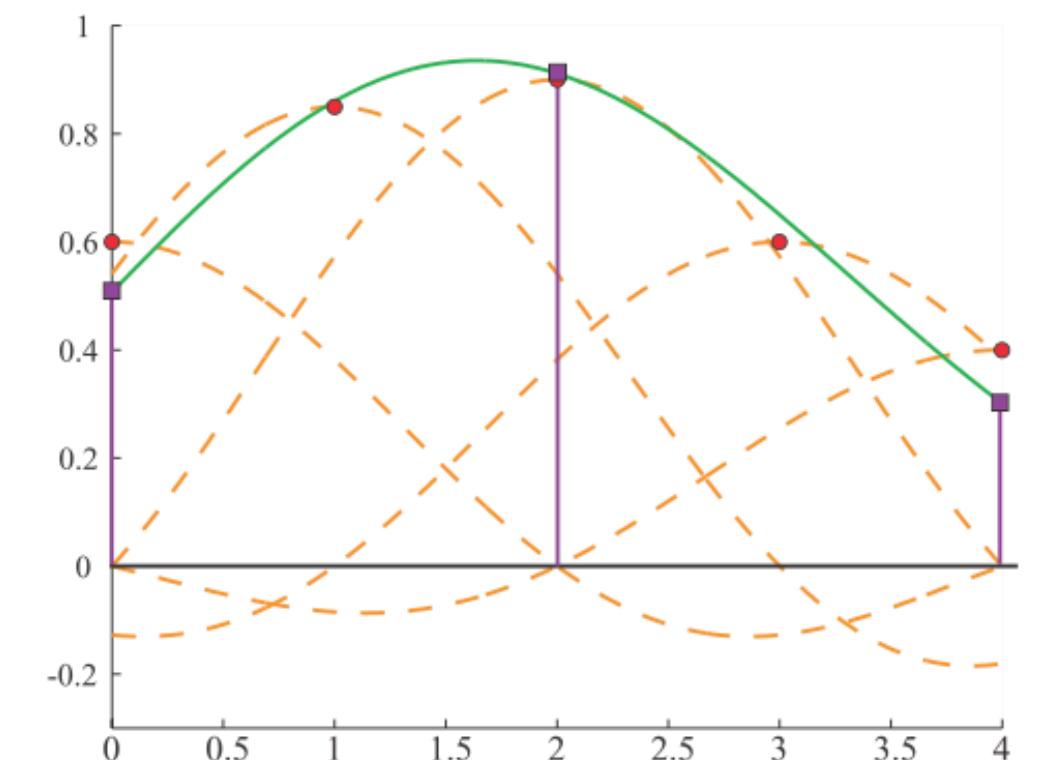
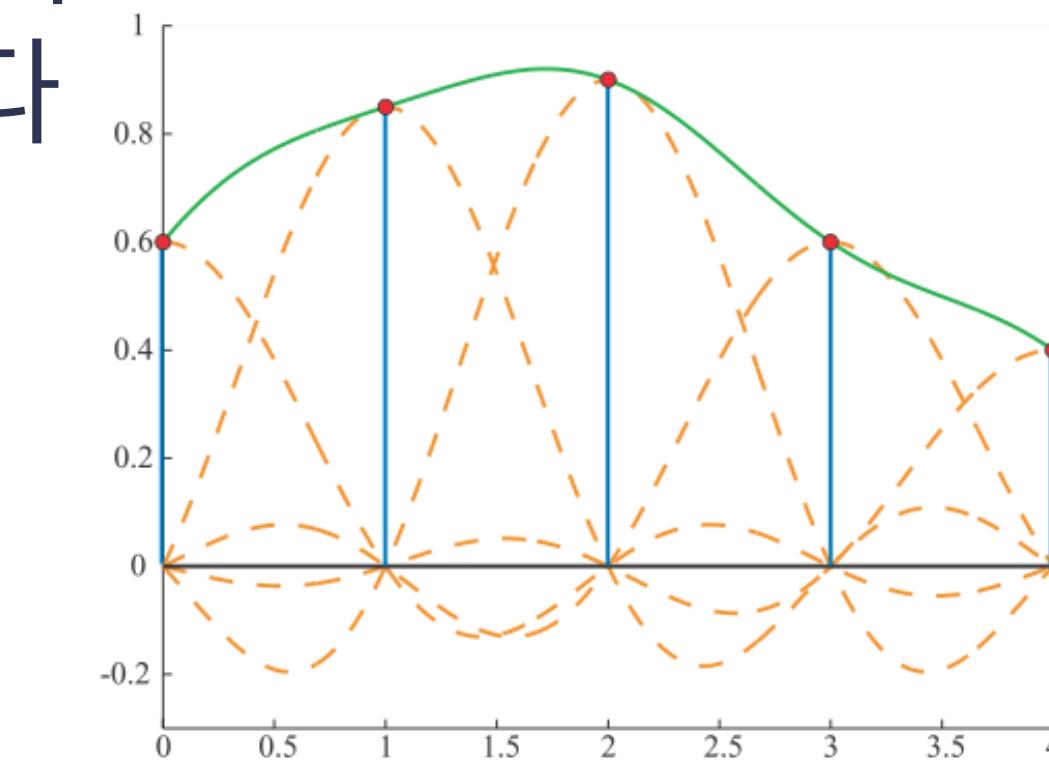
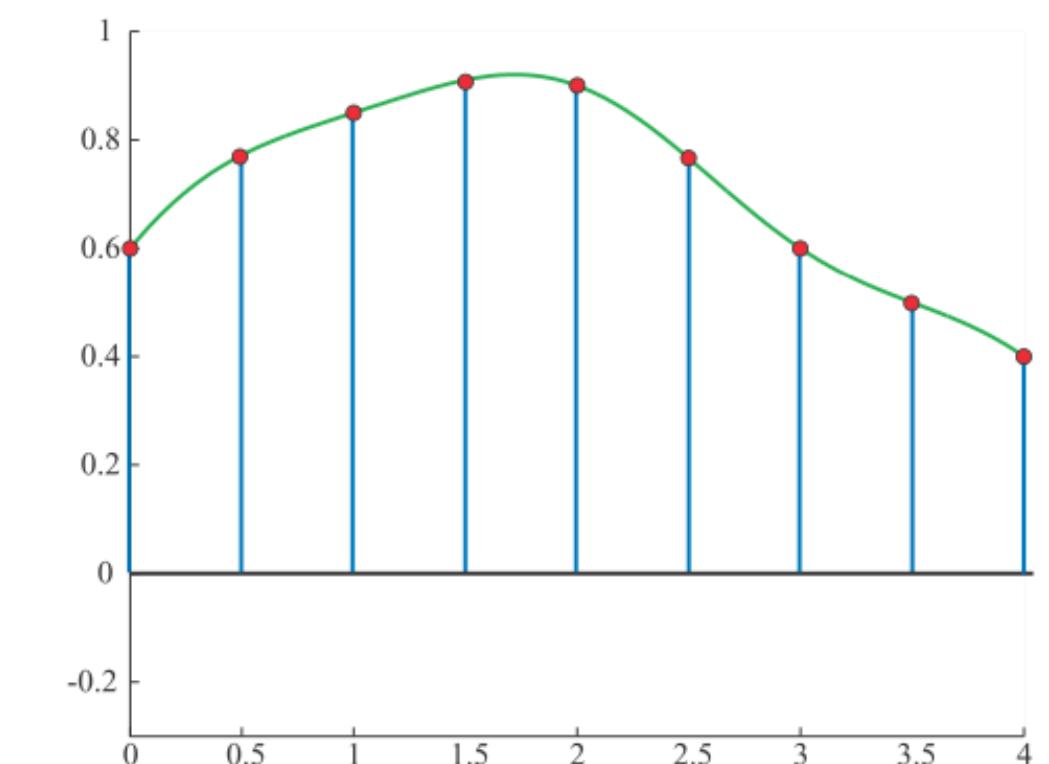
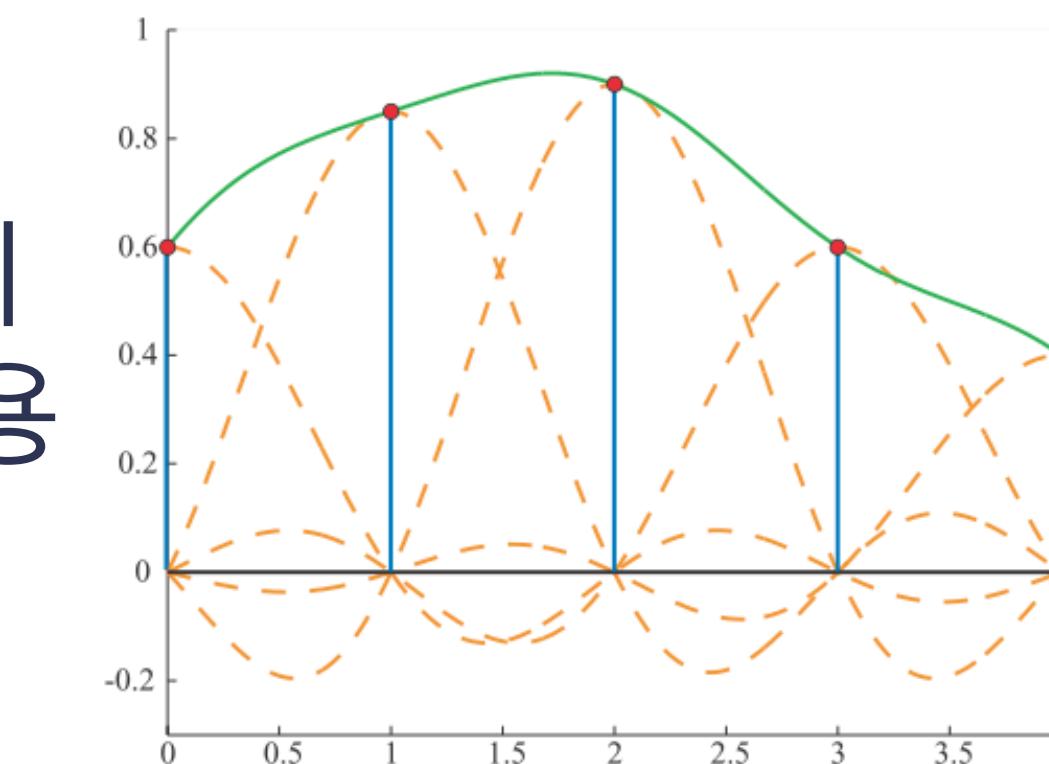
Aliasing and Antialiasing

Resampling

컴퓨터 그래픽스에서는 연속된 신호를 사용할 수 없기 때문에 연속된 신호를 다른 크기로 재샘플링하여 사용

확대는 복원된 신호를 원하는 간격으로 재샘플링

축소는 이 기법을 사용할 수 없음. 대신 $\sin c(ax)$ 를 사용하는 필터를 사용하여 원하는 간격으로 재샘플링 다른 방법으로는 고주파 차단 필터의 너비를 크게 하여 더 많은 고주파 성분들을 제거

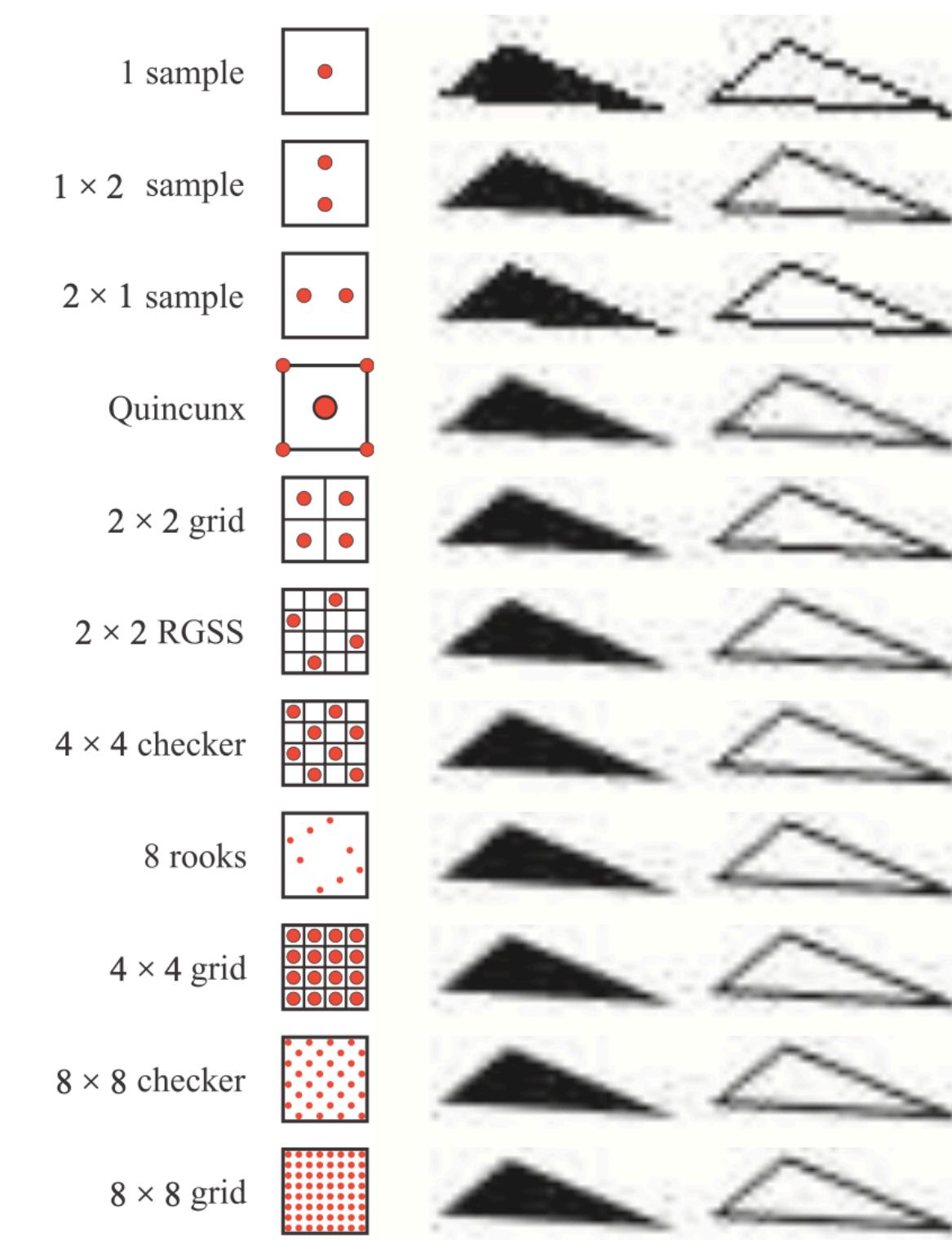


Aliasing and Antialiasing

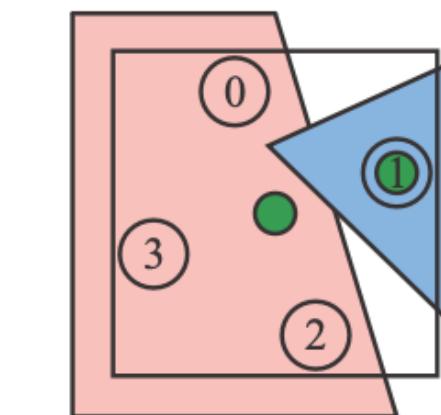
Screen-Based Antialiasing - 파이프라인의 출력 샘플들만을 가지고 동작

Supersampling, accumulation buffer, RGSS, MSAA, stochastic sampling, Quincunx, FLIPQUAD

$$\mathbf{p}(x, y) = \sum_{i=1}^n w_i \mathbf{c}(i, x, y)$$



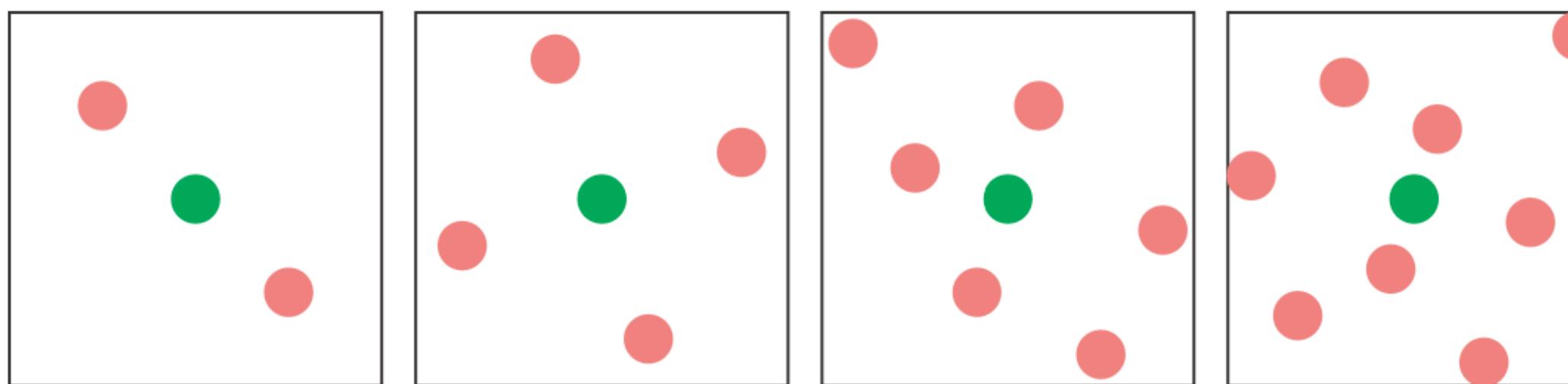
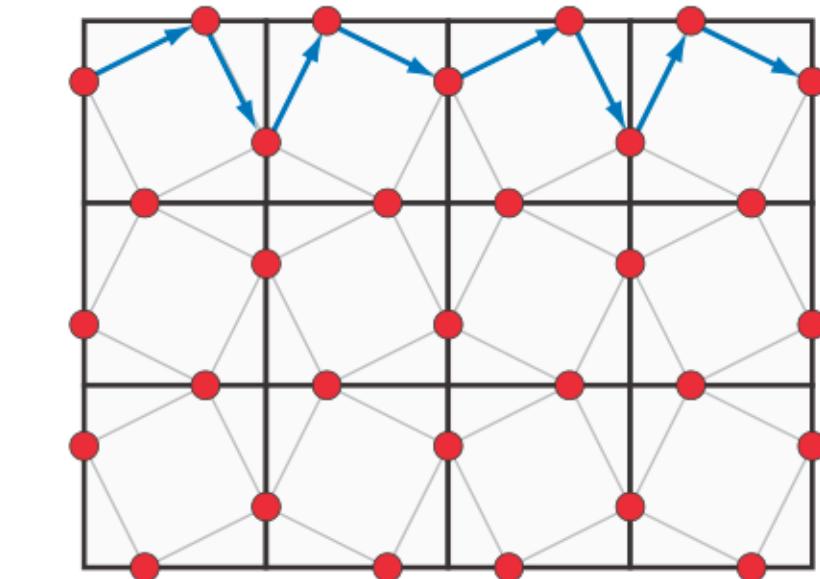
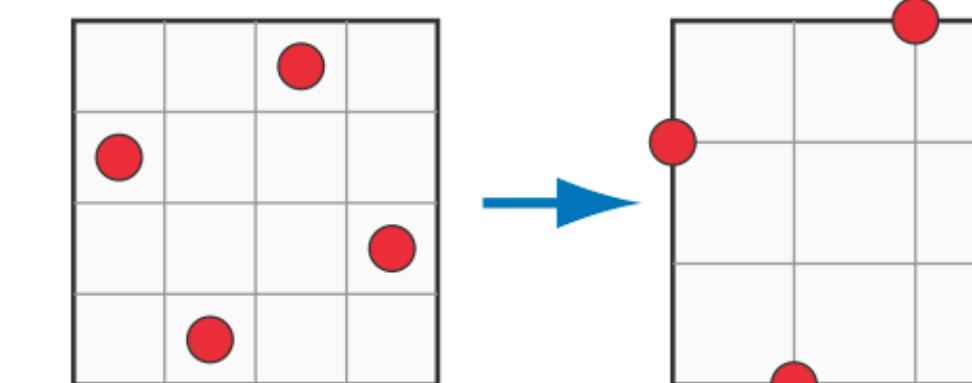
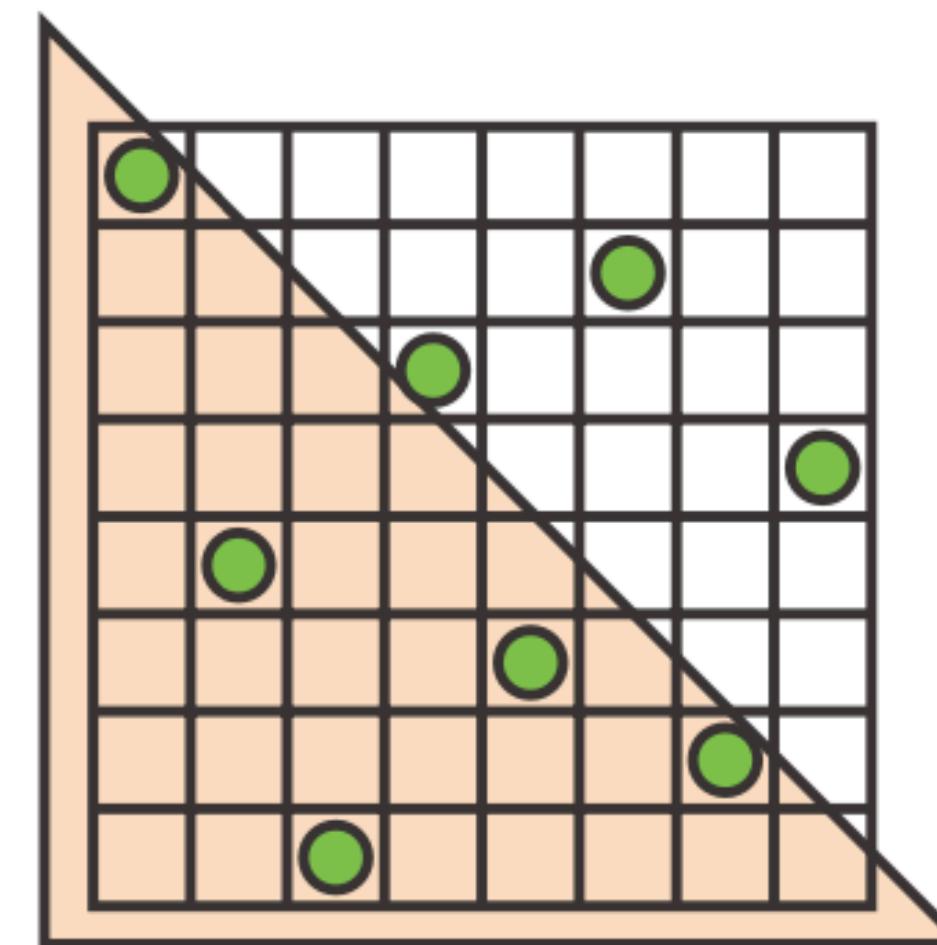
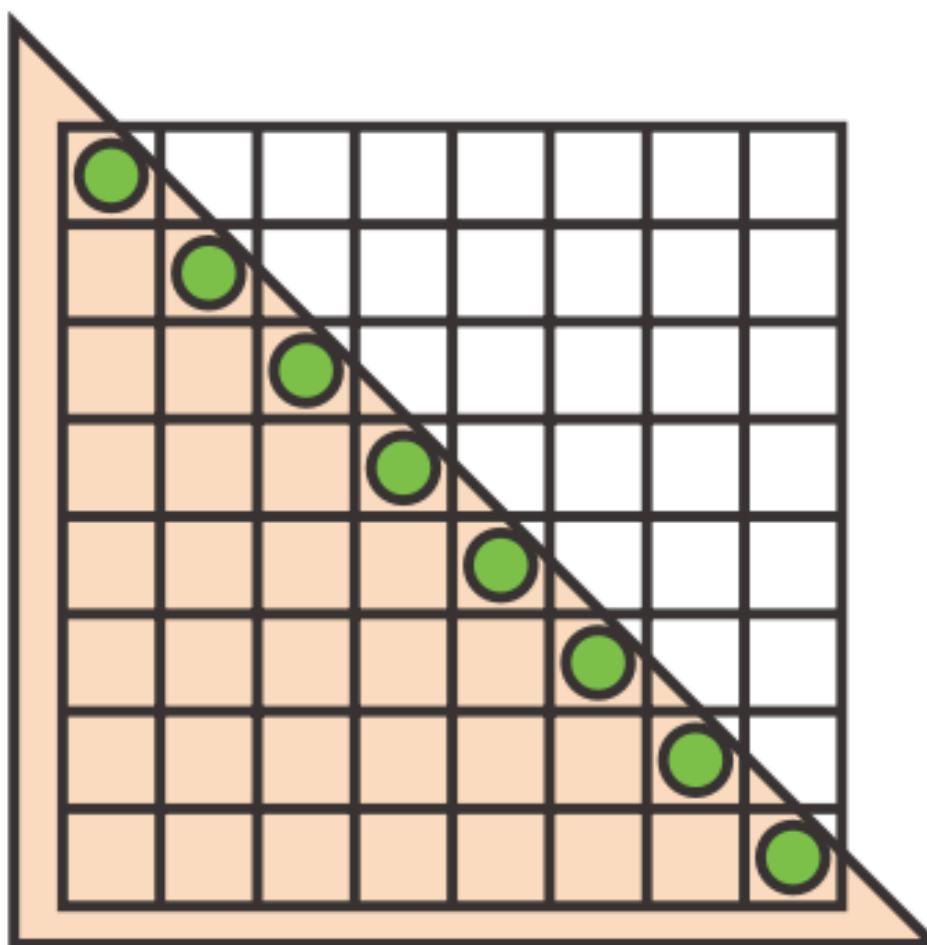
#	color & z
0	pink
1	blue
2	pink
3	pink



#	ID	color & z
0	B	pink
1	A	blue
2	B	pink
3	B	pink

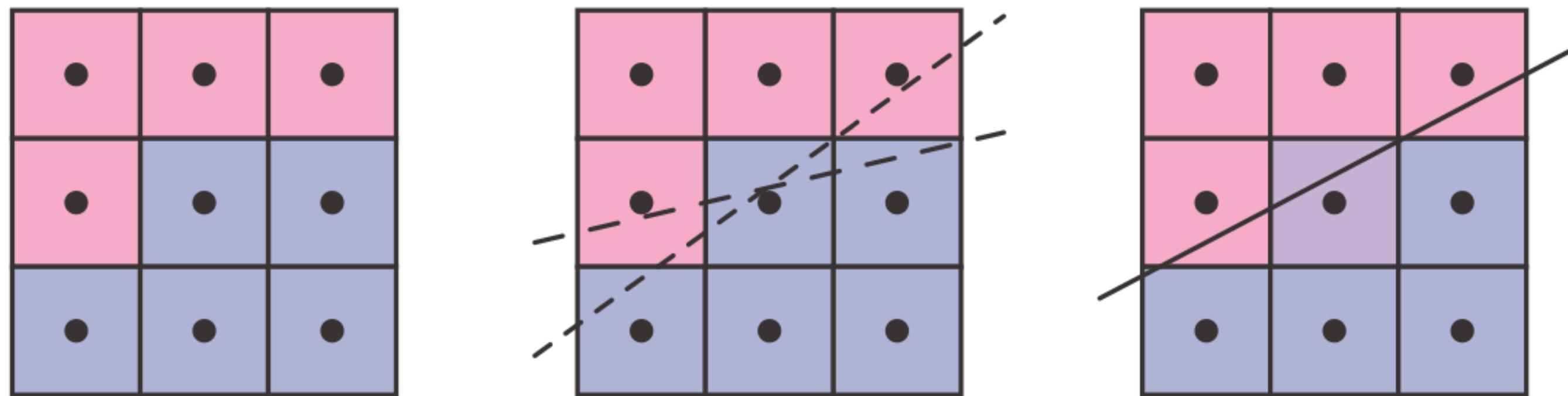
Aliasing and Antialiasing

Sampling Patterns



Aliasing and Antialiasing

Morphological method - post-process, 렌더링은 일반적인 방식으로 수행되고 결과는 안티앨리어싱 결과를 생성하는 프로세스에 전달

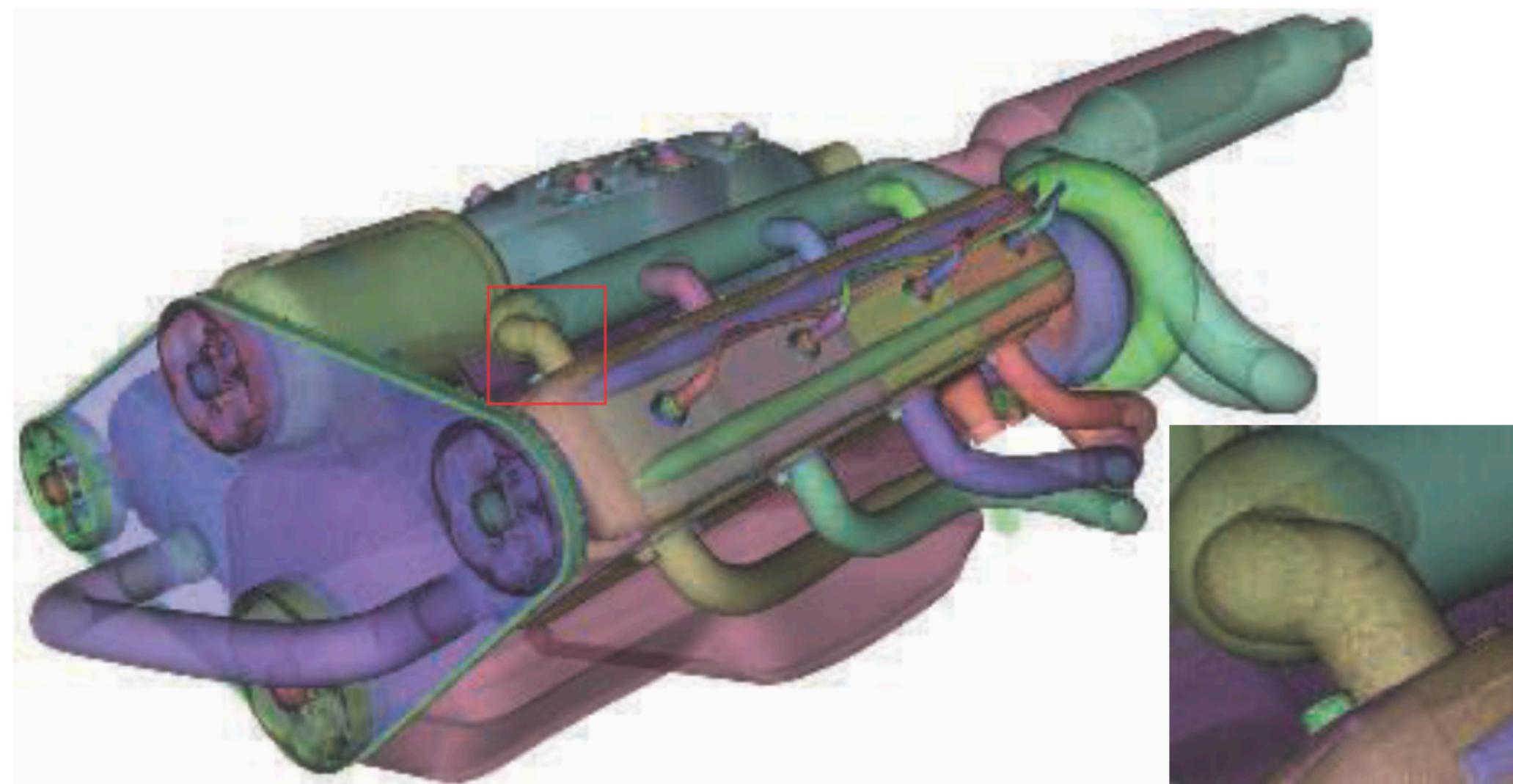


Transparency, Alpha, and Compositing

Screen-door transparency - 50%만 투명해질수있음, 하나의 투명한 물체만 렌더링 가능

Stochastic transparency - 확률적 샘플링과 결합된 서브 픽셀 스크린도어 마스킹

Alpha blending



Transparency, Alpha, and Compositing

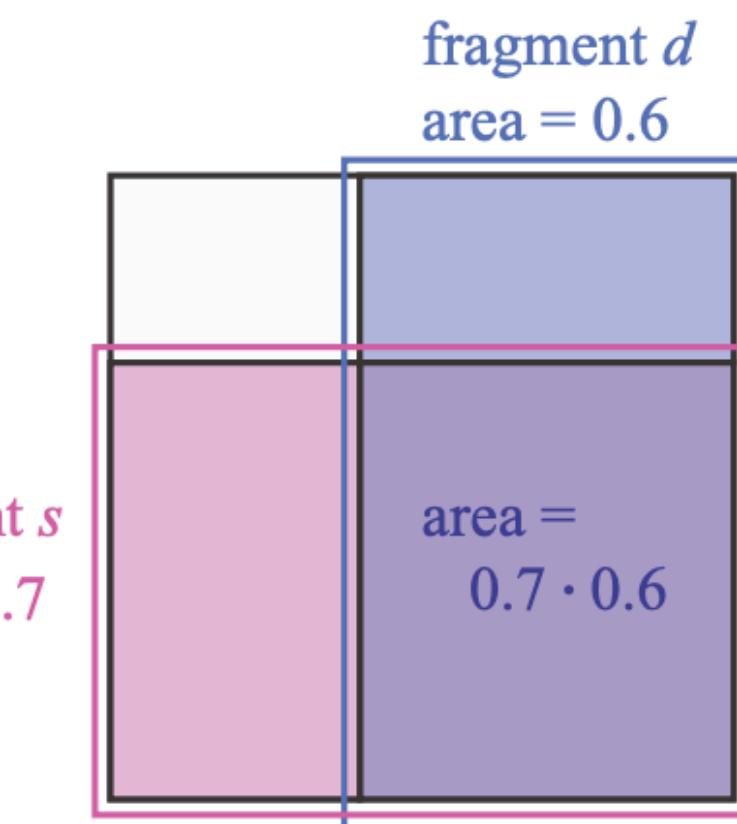
Blending Order

$$\mathbf{c}_o = \alpha_s \mathbf{c}_s + (1 - \alpha_s) \mathbf{c}_d \quad [\text{over operator}],$$

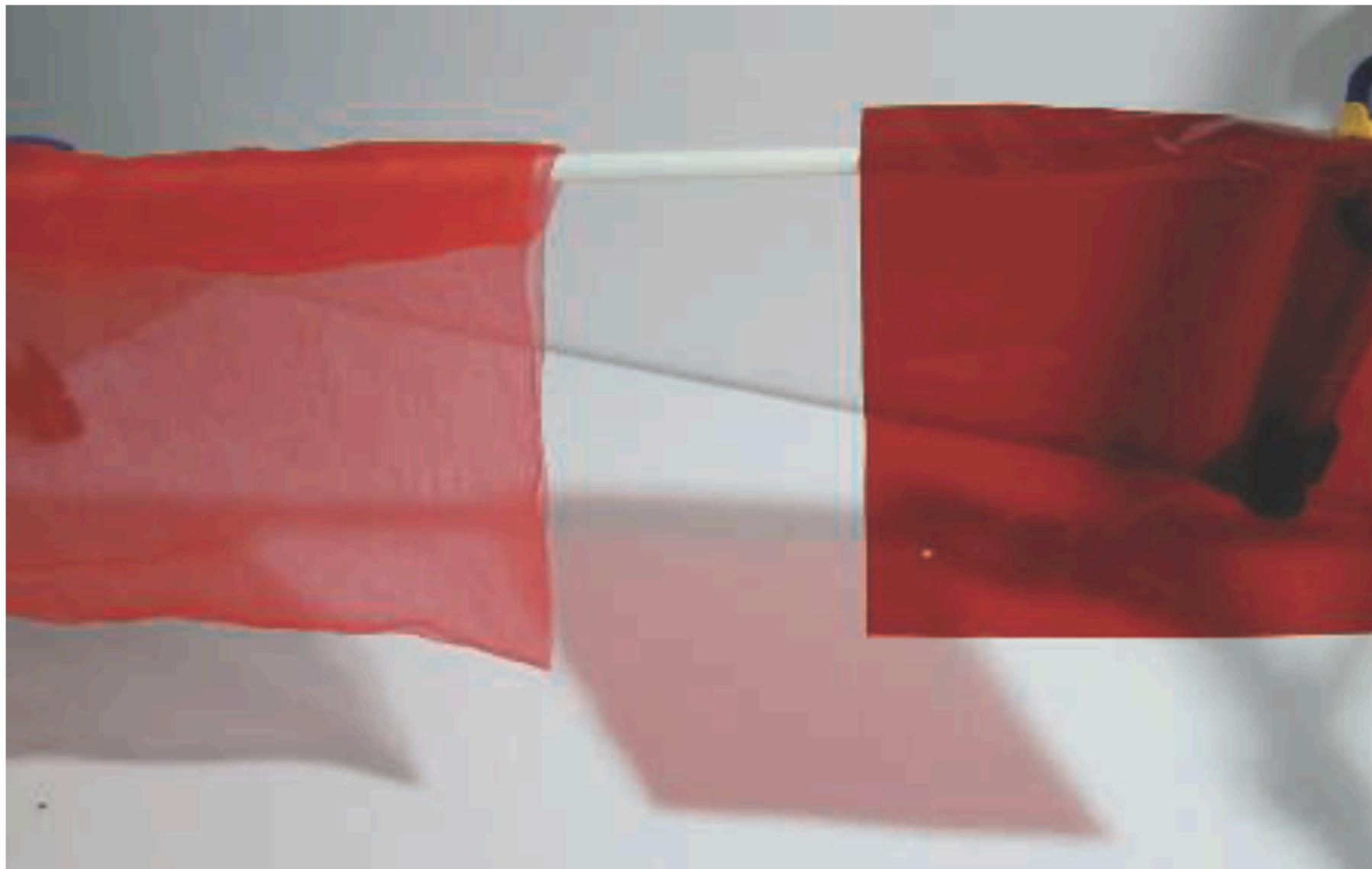
$$\mathbf{c}_o = \alpha_s \mathbf{c}_s + \mathbf{c}_d.$$

$$\mathbf{c}_o = \alpha_d \mathbf{c}_d + (1 - \alpha_d) \alpha_s \mathbf{c}_s \quad [\text{under operator}],$$

$$\mathbf{a}_o = \alpha_s(1 - \alpha_d) + \alpha_d = \alpha_s - \alpha_s \alpha_d + \alpha_d.$$



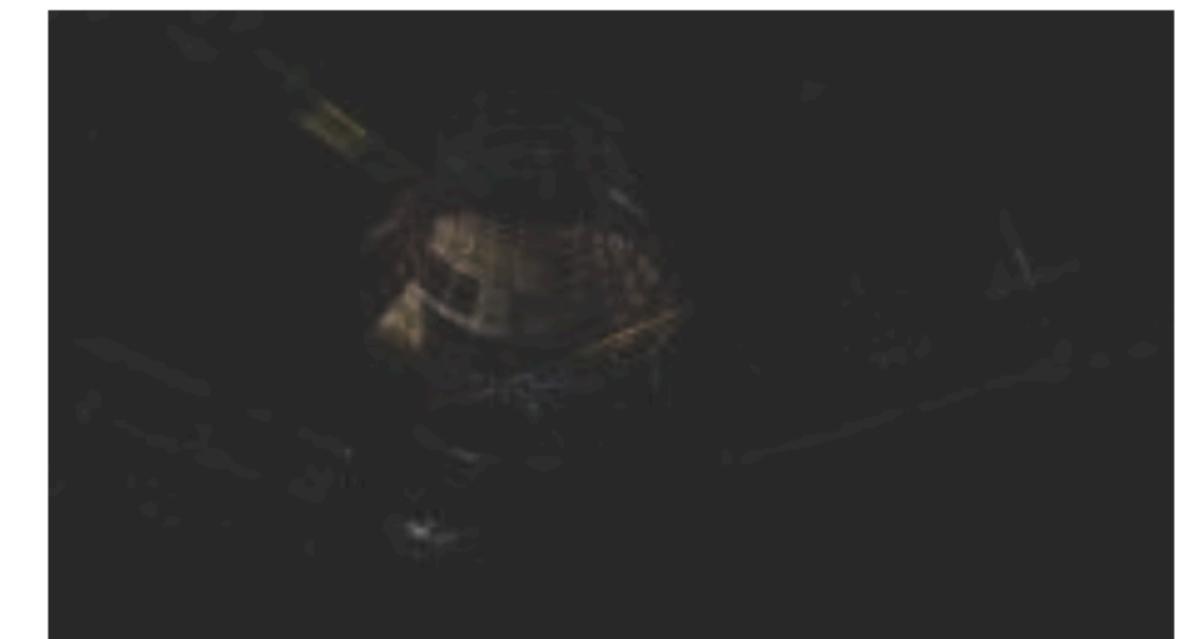
Given two fragments of areas (alphas) 0.7 and 0.6,
total area covered =
 $0.7 - 0.7 \cdot 0.6 + 0.6 = 0.88$



Transparency, Alpha, and Compositing

Order-Independent Transparency

Depth peeling - 두개의 z버퍼와 여러개의 패스 사용. 렌더링 패스가 생성되어 모든 표면의 z깊이가 첫번째 z버퍼에 있게 하고 두번째 패스에서는 모든 투명 객체 렌더링



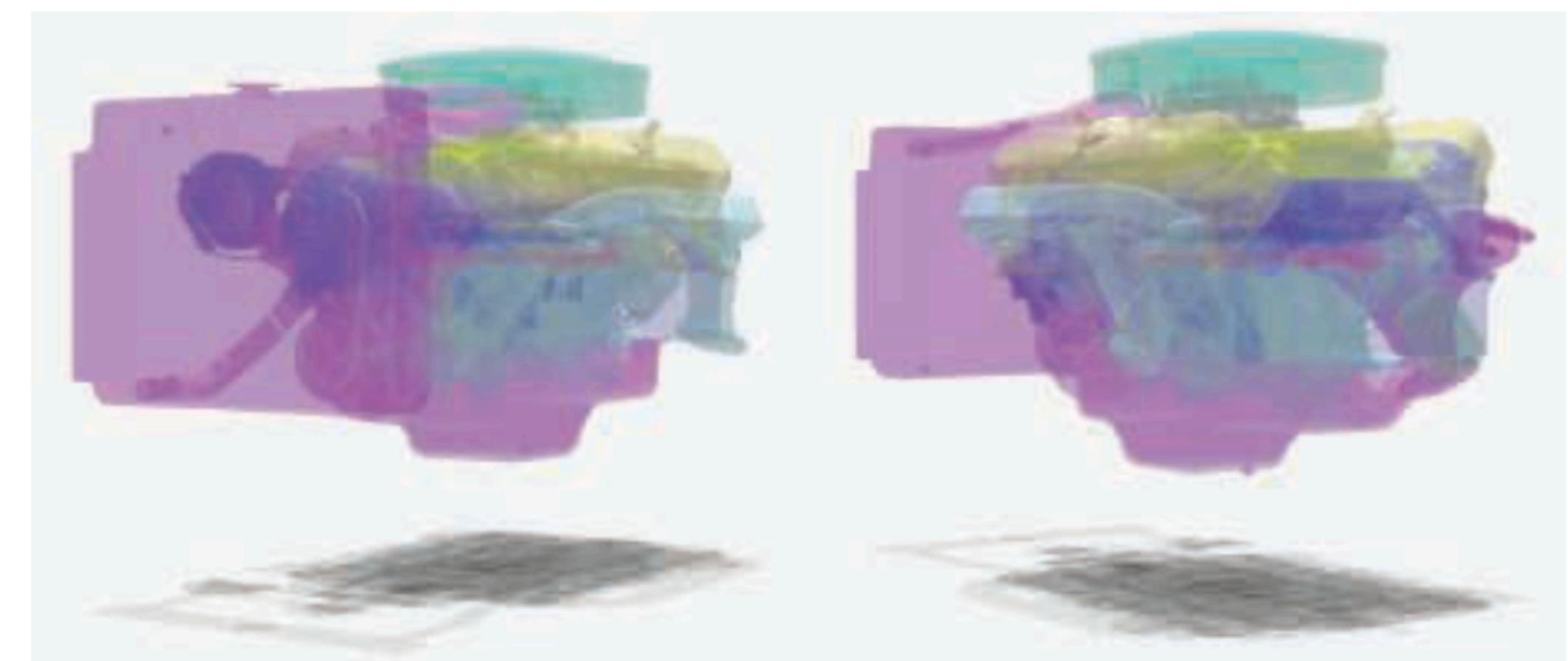
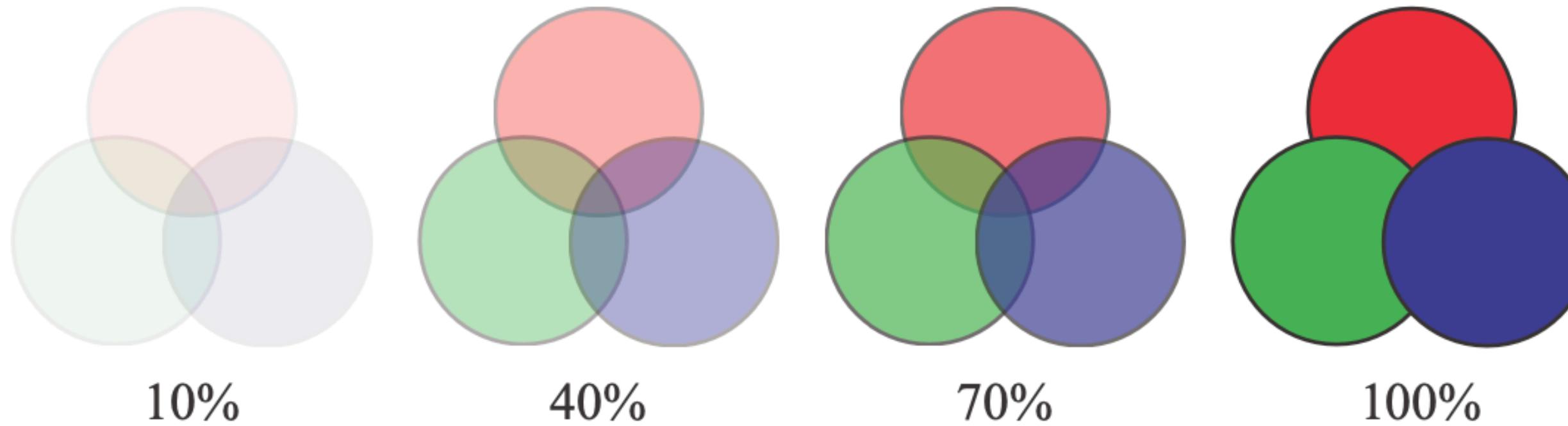
Transparency, Alpha, and Compositing

Multi-layer alpha blending - 아주 적은 오버헤드로 blending 가능하지만 높은 성능비용

K-buffer의 아이디어 기반으로 처음 몇개의 가시 레이어는 저장 및 정렬, 더 깊은 레이어는 폐기

동일한 알파의 경우 순서에 관계없이 모든 색상을 동일하게 혼합하는 문제를 해결하기 위해 가중 혼합 순서 독립적 투명성 도입

$$\mathbf{c}_o = \sum_{i=1}^n (\alpha_i \mathbf{c}_i) + \mathbf{c}_d (1 - \sum_{i=1}^n \alpha_i), \quad \mathbf{c}_{\text{sum}} = \sum_{i=1}^n (\alpha_i \mathbf{c}_i), \quad \alpha_{\text{sum}} = \sum_{i=1}^n \alpha_i,$$
$$\mathbf{c}_{\text{wavg}} = \frac{\mathbf{c}_{\text{sum}}}{\alpha_{\text{sum}}}, \quad \alpha_{\text{avg}} = \frac{\alpha_{\text{sum}}}{n},$$
$$u = (1 - \alpha_{\text{avg}})^n,$$
$$\mathbf{c}_o = (1 - u) \mathbf{c}_{\text{wavg}} + u \mathbf{c}_d.$$



Transparency, Alpha, and Compositing

Premultiplied Alphas and Compositing - RGB는 저장되기 전에 미리 a값과 곱해지고 합성 이미지를 렌더링하는 것은 premultiplied alpha와 일치함

$$\mathbf{c}_o = \mathbf{c}'_s + (1 - \alpha_s)\mathbf{c}_d,$$

Unmultiplied alpha - 한 픽셀에서 우리가 보는 최종 색상이 다각형의 음영값이 아니기 때문에 합성 이미지를 저장하는 데는 거의 사용되지 않음

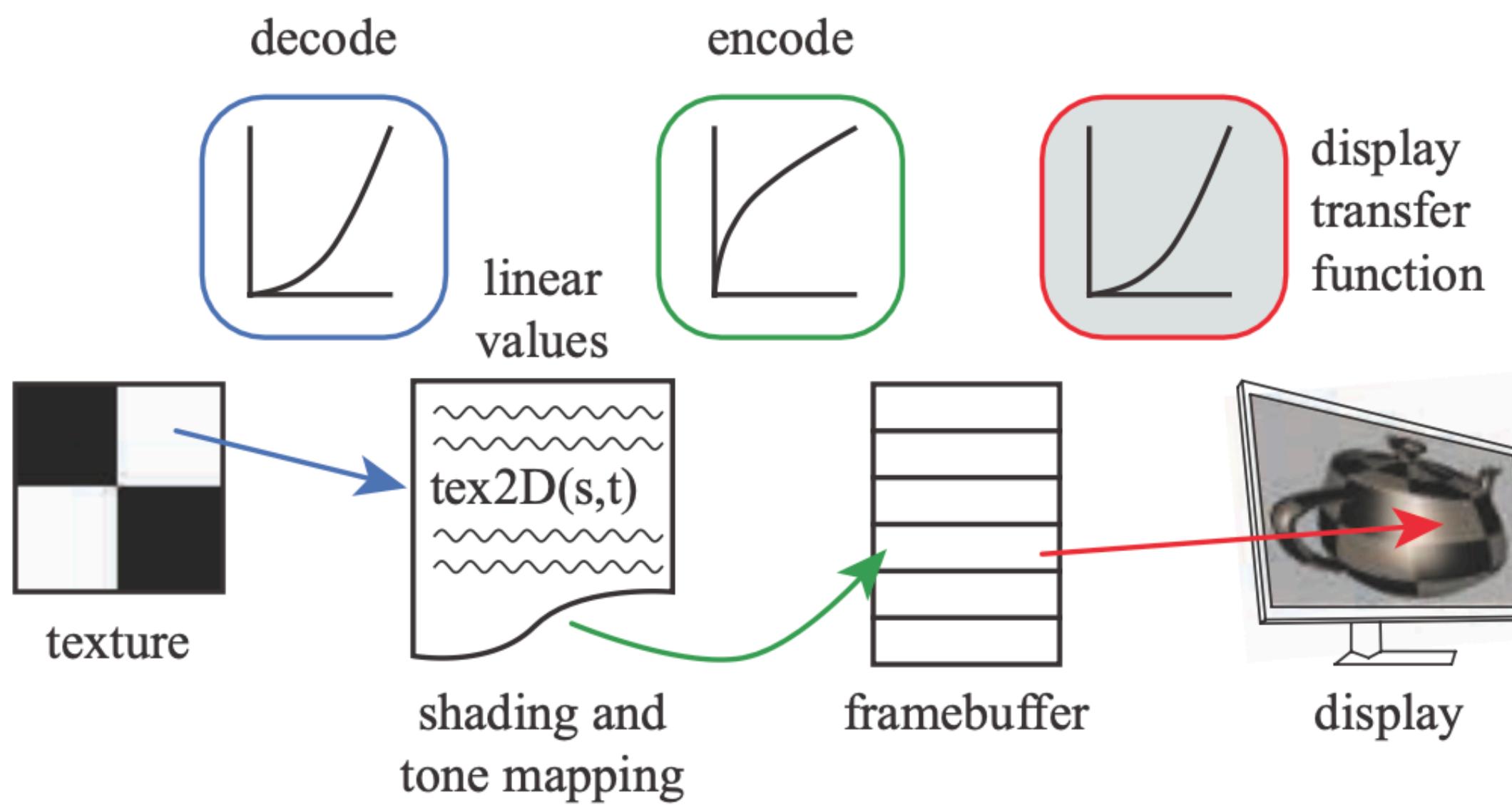
크로마키 - 특정한 색상을 투명한 것으로 간주하여 그 색상이 감지되면 그 위치에는 배경이 그려지도록 함

Display Encoding

시각적 아티팩트를 피하기 위해 디스플레이 버퍼와 텍스처는 비선형 인코딩 사용 - 감마 보정

전력 함수는 인간 시력의 밝기 민감도의 역과 거의 일치함 - 이를 기반으로 CRT가 화상을 표시하기 위한 최적의 휘도값을 결정 할 수 있게됨

비선형 효과를 상쇄하기 위해 디스플레이 전송 함수의 역을 적용 - 텍스처 값을 디코딩 할 때 쉐이딩에 사용할 선형 값을 생성하기 위해 디스플레이 전송 기술 적용



Display Encoding

$$y = f_{\text{sRGB}}^{-1}(x) = \begin{cases} 1.055x^{1/2.4} - 0.055, & \text{where } x > 0.0031308, \\ 12.92x, & \text{where } x \leq 0.0031308, \end{cases}$$

$$x = f_{\text{sRGB}}(y) = \begin{cases} \left(\frac{y + 0.055}{1.055}\right)^{2.4}, & \text{where } y > 0.04045, \\ \frac{y}{12.92}, & \text{where } y \leq 0.04045, \end{cases}$$

$$x = f_{\text{display}}(y) = y^\gamma.$$

$$y = f_{\text{simpl}}^{-1}(x) = \sqrt{x},$$

$$x = f_{\text{simpl}}(y) = y^2;$$

Display Encoding

감마 보정을 무시할 때 발생하는 문제

- 낮은 선형 값이 너무 어둡게 나타남
- 물리적으로 선형 방사 값에 맞는 음영계산이 비선형 값에 대해 수행
- 안티앨리어싱 품질에 영향
- 로핑

