Aggregate G-Buffer Anti-Aliasing in Unreal Engine 4

Louis Bavoil (Developer Technology)
Cyril Crassin (Research)

Advances in Real-Time Rendering in Games course

Render the Possibilities
SIGGRAPH 2016
THE 43RD INTERNATIONAL CONFERENCE AND EXHIBITION ON

4x TAA = 4x SSAA with TAA

Scene courtesy of Quixel and Epic Games

4xSSAA

Scene courtesy of Quixel and Epic Games


4xAGAA
Lighting: 4.7 -> 2.8 ms (1.7x faster)
GBuffer Roughness: Kaplanyan
GBuffer Normal: Toksvig
Smith: Correct
Level B4
7 PLEASE WAIT ELEVATOR
WELCOME TO
B4SECURITY
OPEN
WANTED
VIDEO
Scene courtesy of Quixel and Epic Games
GPU times measured on GTX 1080 @ 1080p

Temporal Anti-Aliasing

Great increase in AA quality [Karis 2014]

However:

- Ghosting
- Flickering
- Over-smoothing visual features (Like specular highlights)

Needs combining with SSAA for best quality

1x TAA

8xAGAA

Lighting: 9.4 -> 3.6 ms (2.6x faster)

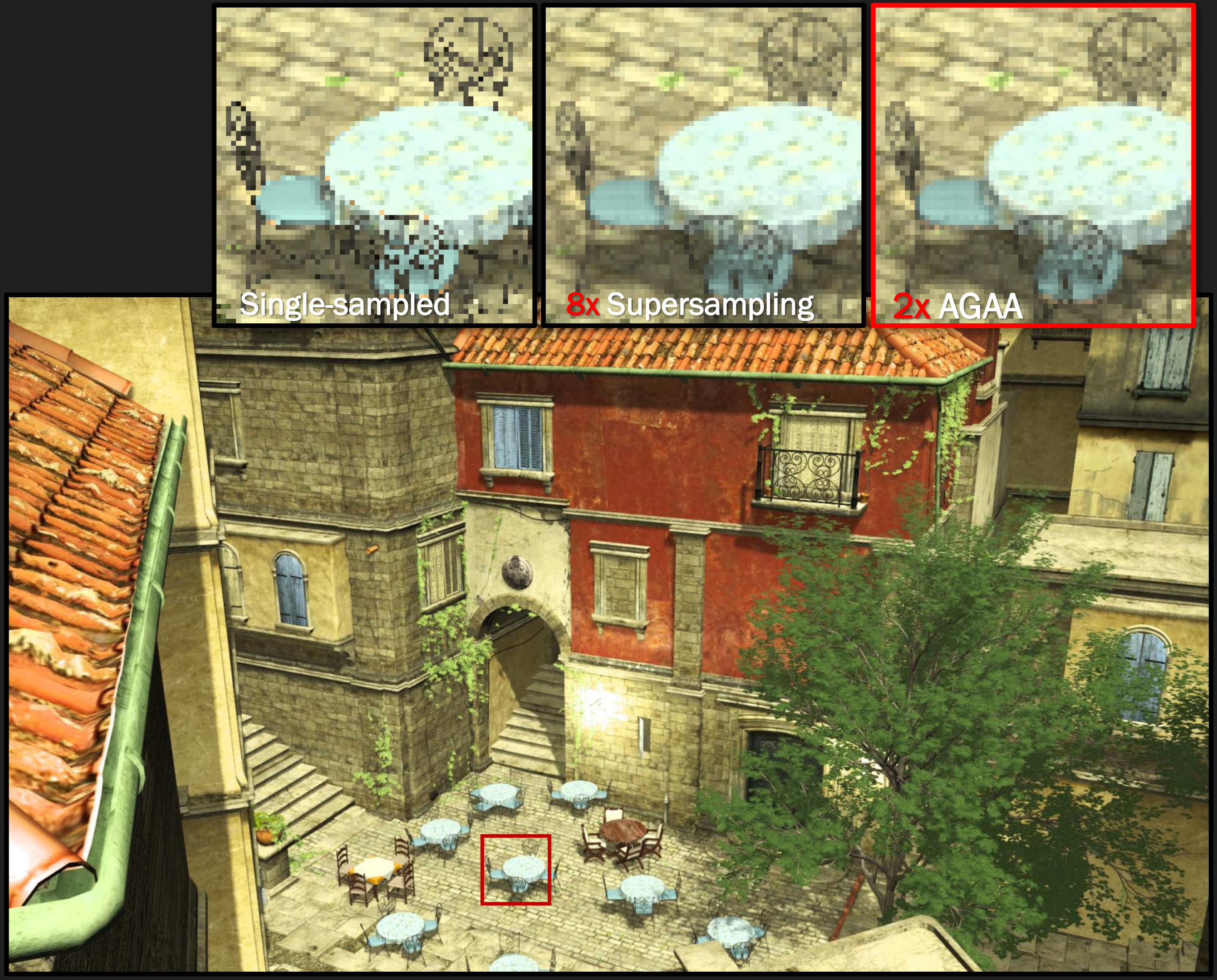
Scene courtesy of Quixel and Epic Games

GPU times measured on GTX 1080 @ 1080p

AGAA: Aggregate G-Buffer Anti-Aliasing

I3D 2015 + TVCG16

*Cyril Crassin, Morgan McGuire,
Kayvon Fatahalian, Aaron Lefohn*

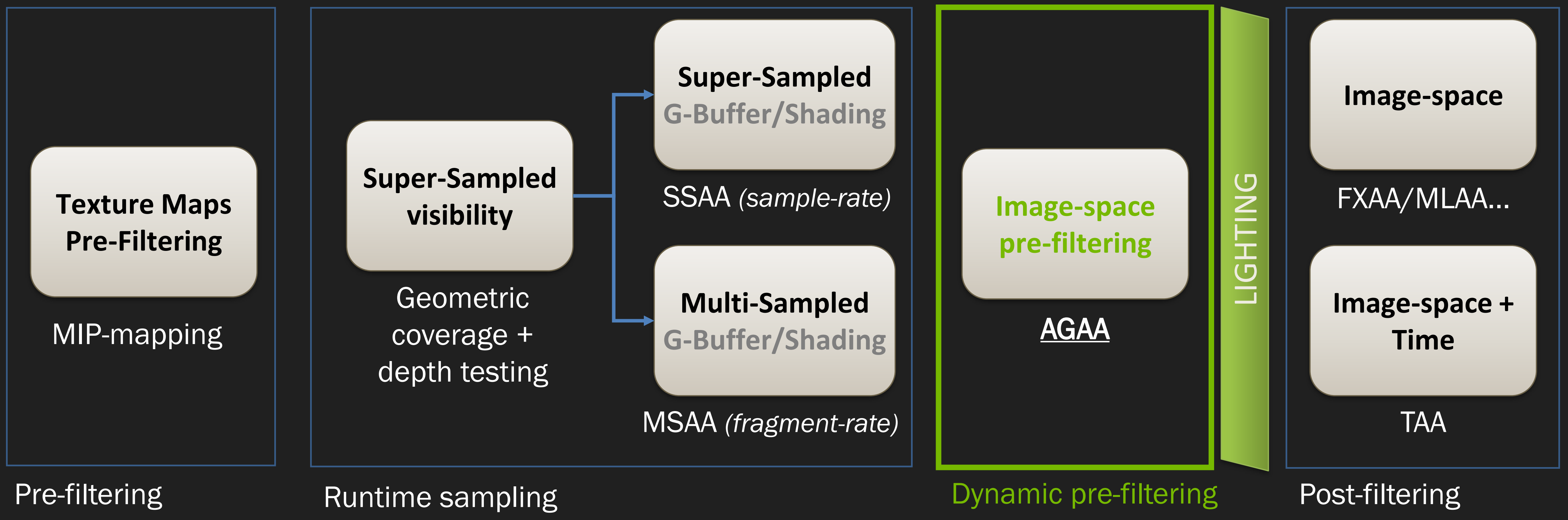


Decouples shading rate from the G-buffer sample count

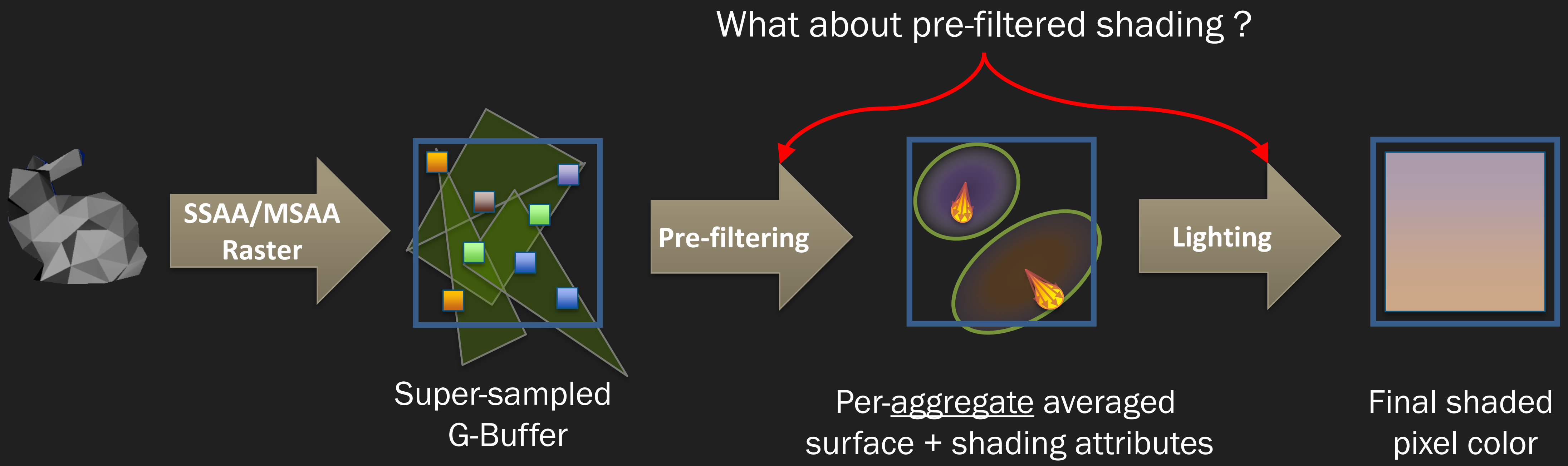
- Using dynamic pre-filtering
- Rasterize at 4x or 8x MSAA/SSAA
- Light at most 2x /pixel

Pre-filtering, Supersampling, Post-filtering

- Goal: Capturing or reproducing appearance of sub-pixel details
- Various tools for filtering various geometric scales



AGAA Pipeline: (Very) High-Level View



Lighting Pre-Filtered Aggregates

Goal: Approximate average reflectance over an aggregate's footprint

Independently pre-filtering the inputs of the shading function for each aggregate

- Inspired by texture-space and voxel-space pre-filtering schemes
- Attributes decorrelation assumption
- Far-field assumption

Per-aggregate statistical information:

- Average most shading parameters
- Build a Normal Distribution Function (NDF)
- Average attenuation from shadowing

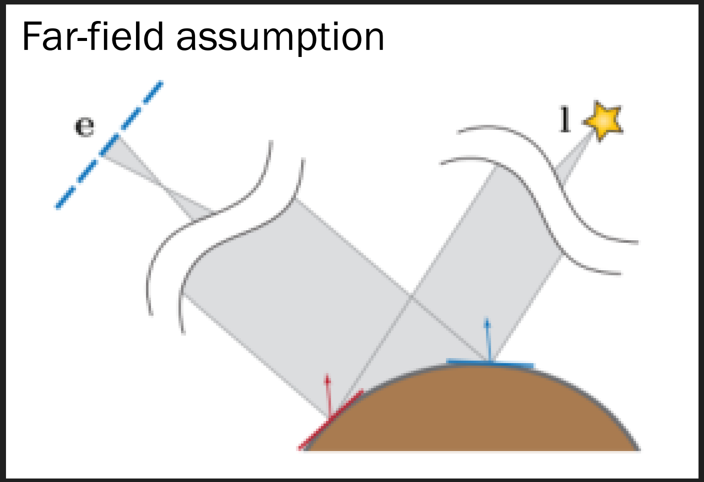


Image courtesy of Kaplanyan2016

Standard UE4 Shading Model

Restricted our investigation to the “Standard” shading model:

Diffuse BRDF:	Lambertian
Specular BRDF:	GGX/Trowbridge-Reitz

Pre-filtering schemes for GGX:

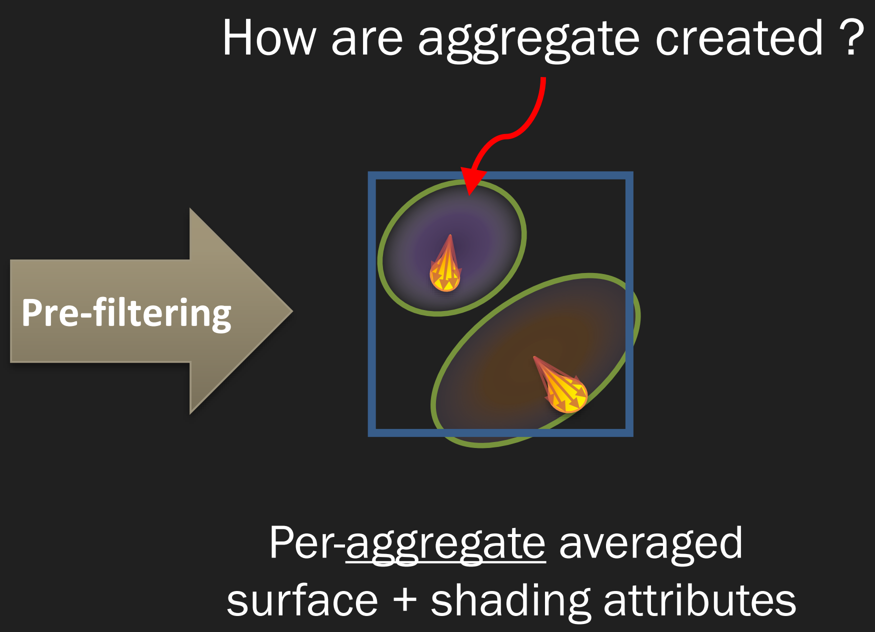
- [Toksvig 2005]: Isotropic NDF, but cheap
  Converting Phong specular exponent to Roughness

- SGGX [Heitz 2015] (Spherical GGX): Anisotropic NDF
  Represented as an ellipsoid, works on full spherical domain

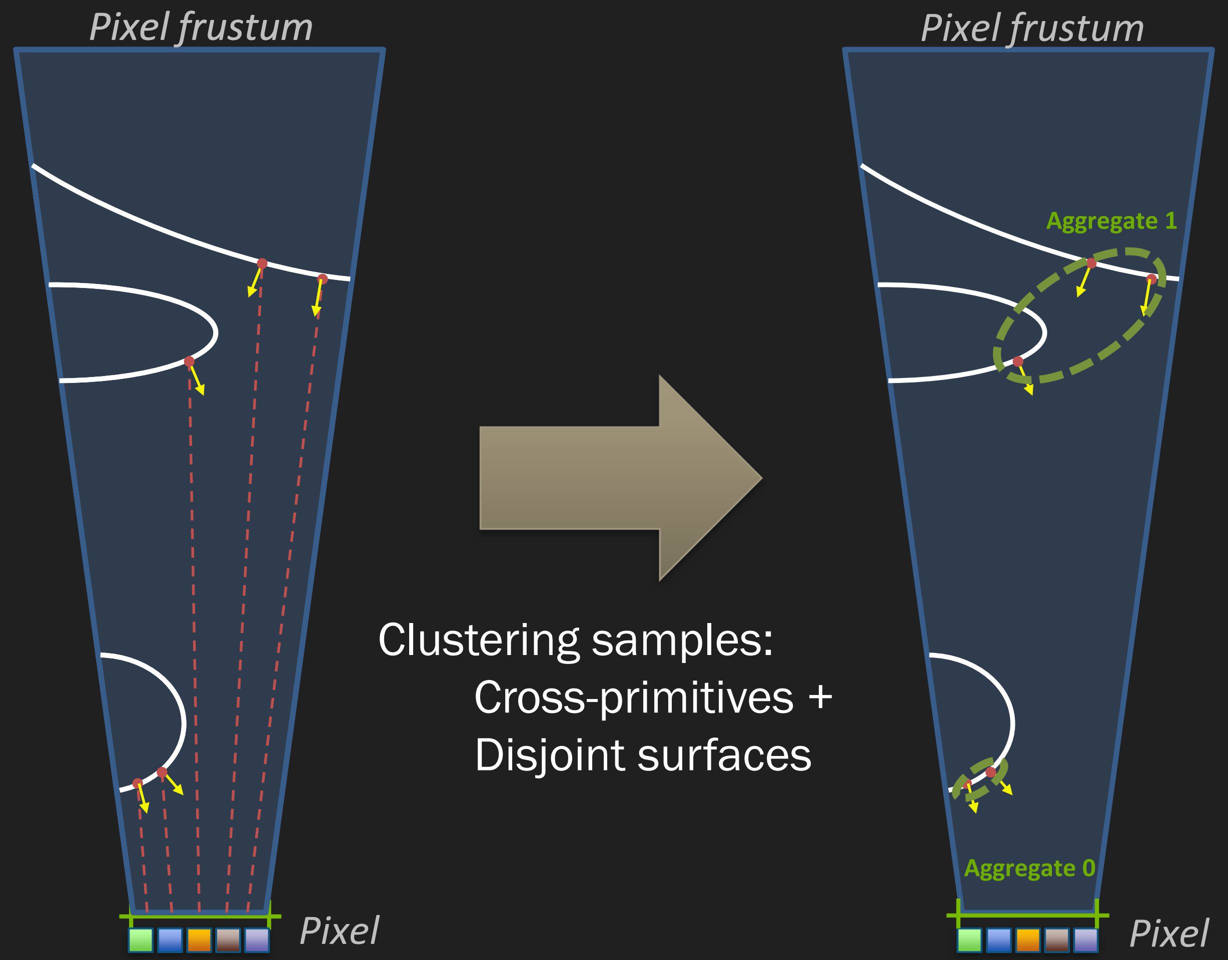
Lambertian: Analytic approx. using Toksvig

[Baker and Hill 2012]

AGAA Pipeline: (Very) High-Level View



Aggregate Creation: Clustering



1: MSAA/SSAA
G-Buffer Rasterization

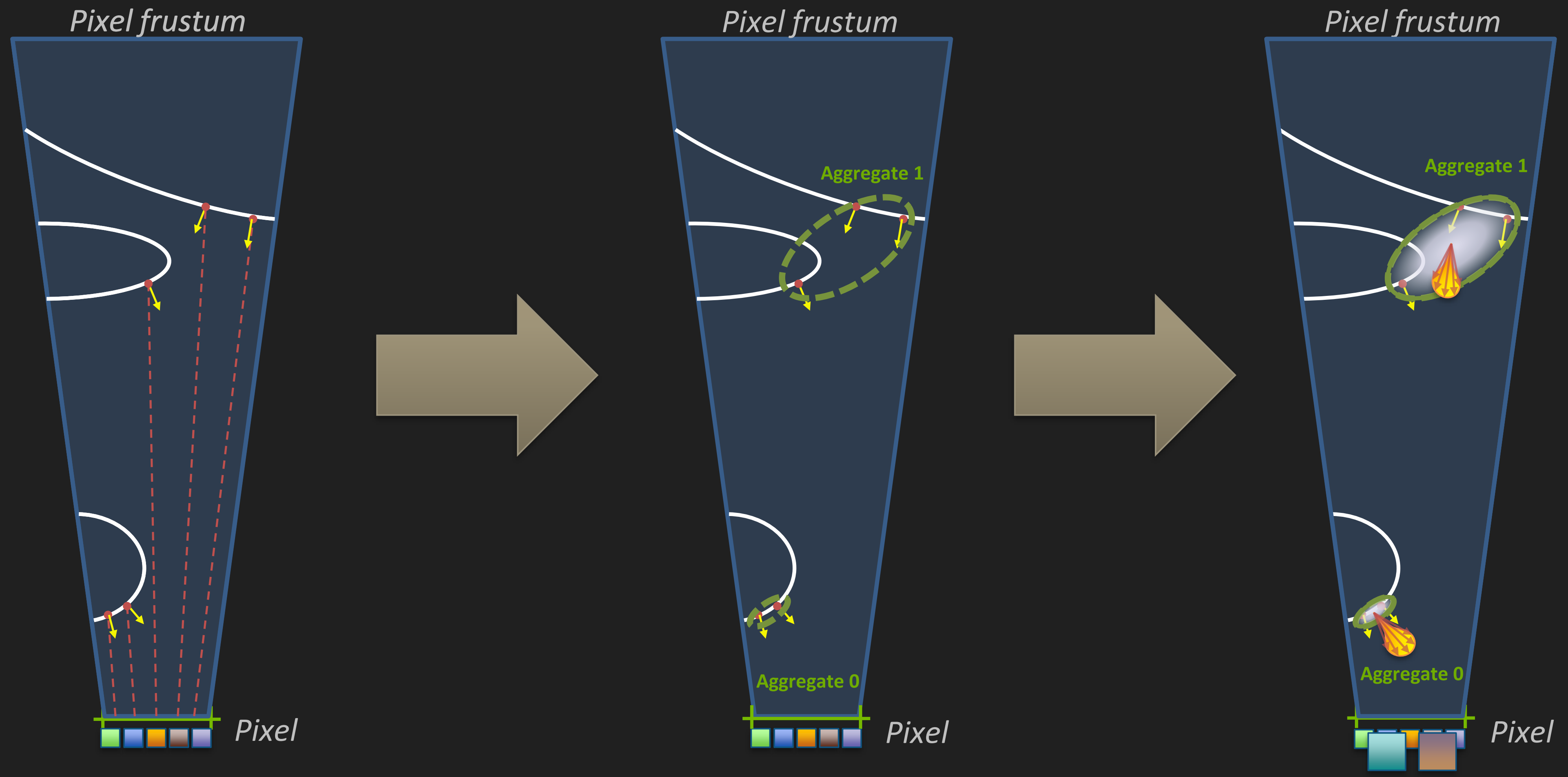
2: Clustering

Goal: Minimize shading errors due to correlated attributes [Bruneton and Neyret 2012]

Distance metric:

- Shading Model
- Normal
- Depth/Position

Aggregate Creation: Aggregation



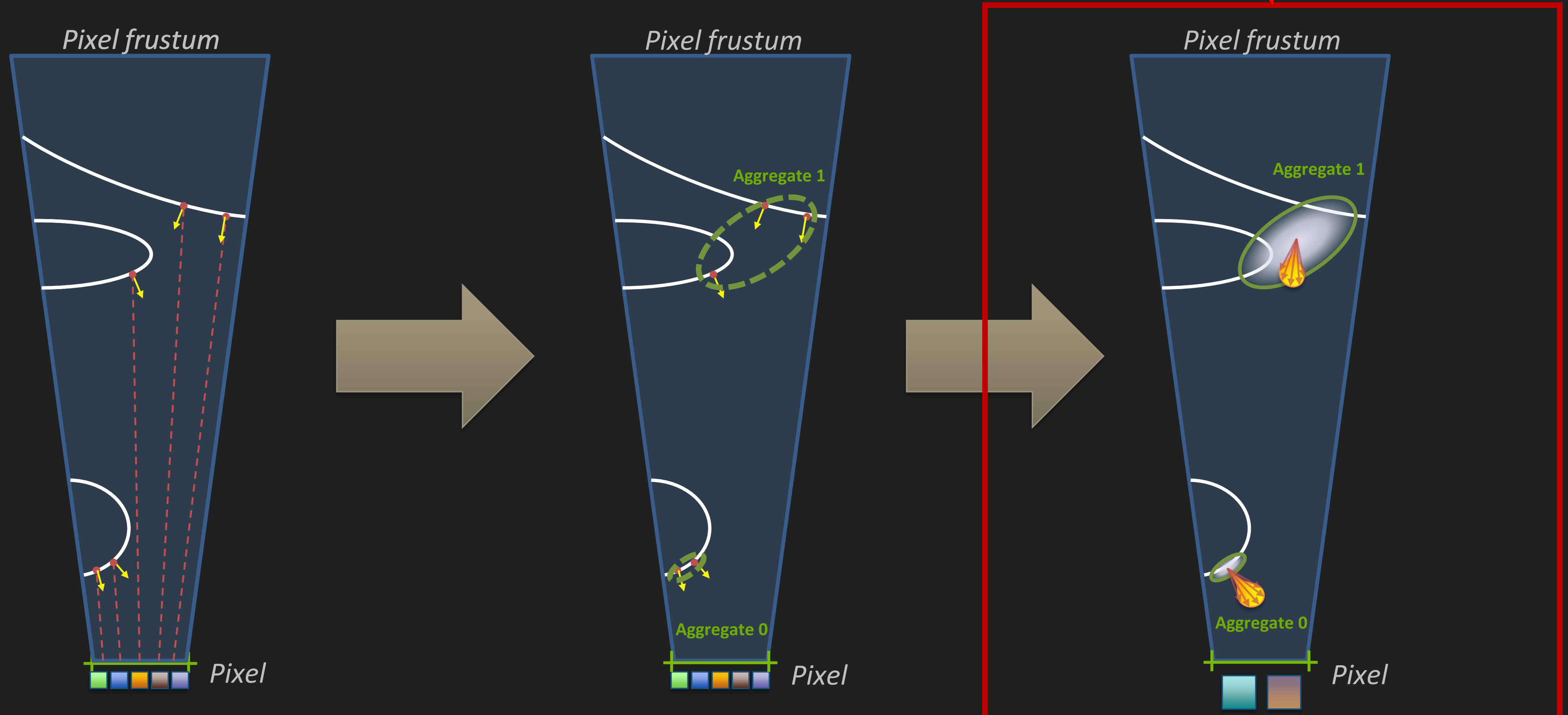
1: MSAA/SSAA
G-Buffer Rasterization

2: Clustering

3: Aggregation
Pre-filtering

Aggregate Creation

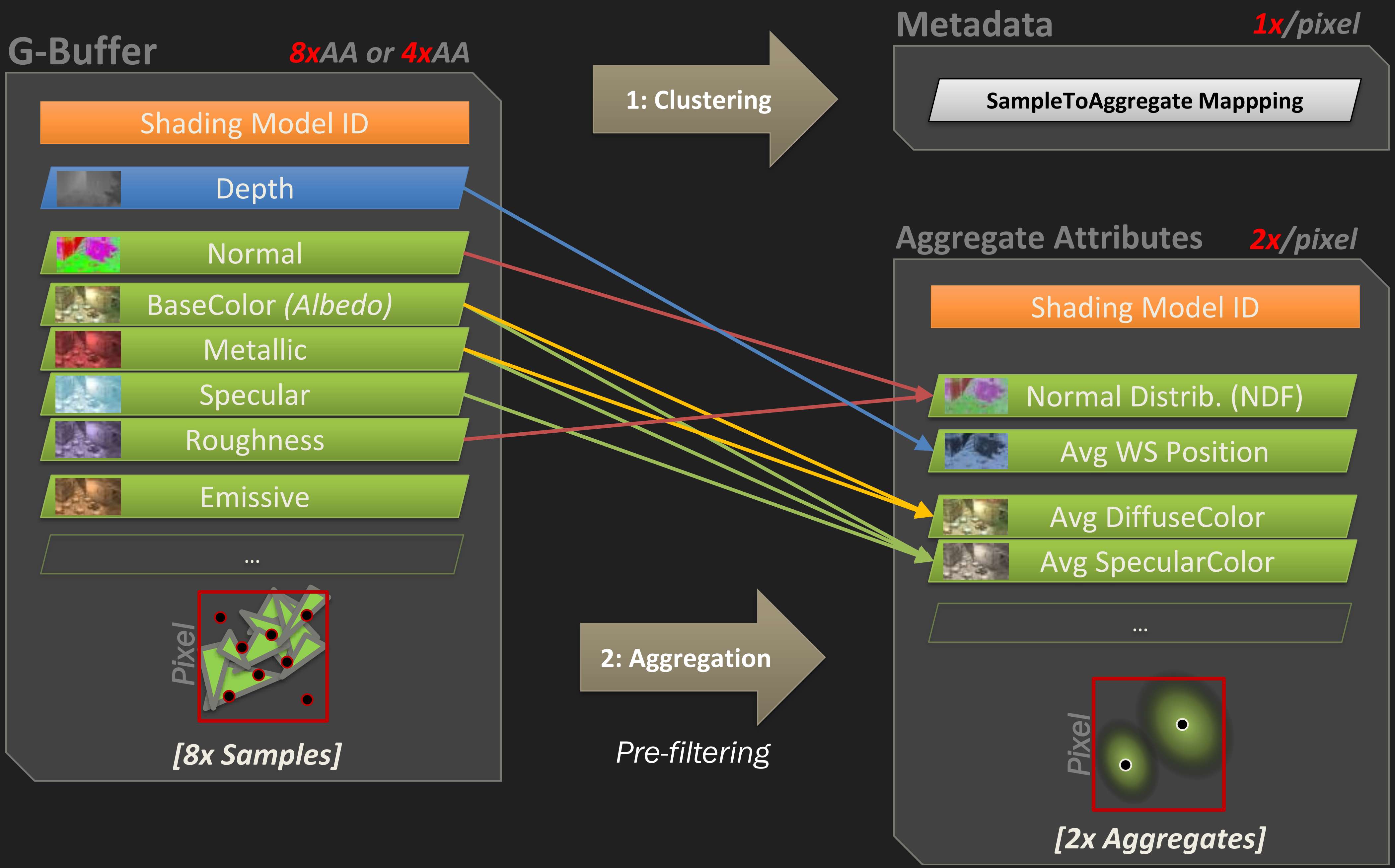
Implemented in the tiled-deferred shading pass



1: MSAA/SSAA G-Buffer Rasterization

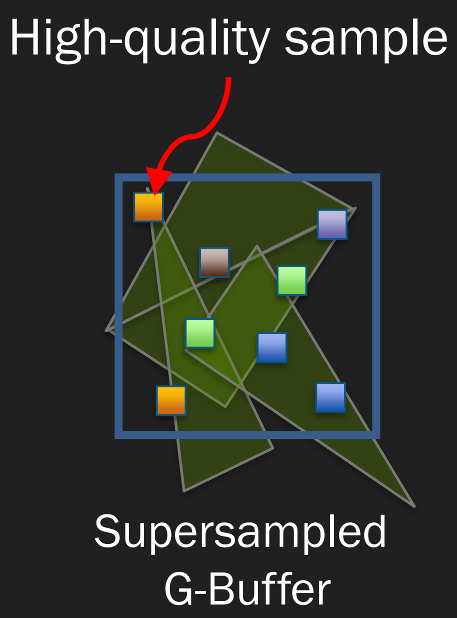
2: Clustering

3: Aggregation Pre-filtering


G-Buffer
8xAA or 4xAA
Shading Model ID
Depth
Normal
BaseColor (Albedo)
Metallic
Specular
Roughness
Emissive
...
Pixel
[8x Samples]
1: Clustering
2: Aggregation
Pre-filtering
Metadata
1x/pixel
SampleToAggregate Mappping
Aggregate Attributes
2x/pixel
Shading Model ID
Normal Distrib. (NDF)
Avg WS Position
Avg DiffuseColor
Avg SpecularColor
...
Pixel
[2x Aggregates]

4xAGAA
Artifacts with no G-Buffer sample NDF pre-filtering
Normal-map details
Non-normal mapped mesh
with high geometric curvature
Scene courtesy of Quixel and Epic Games

AGAA Pipeline: (Very) High-Level View



Pre-Filtering G-Buffer Samples

[Toksvig 2005] normal-map pre-filtering

[Kaplanyan 2016] filtering geometric curvature

Kaplanyan’s Curvature Filtering

```
float3 GetAgaaAverageQuadNormal(FMaterialPixelParameters MaterialParameters, float3 N)
{
    int2 PixelPos = MaterialParameters.SVPosition.xy;
    N -= ddx_fine(N) * (float(PixelPos.x & 1) - 0.5);
    N -= ddy_fine(N) * (float(PixelPos.y & 1) - 0.5);
    return N;
}
```

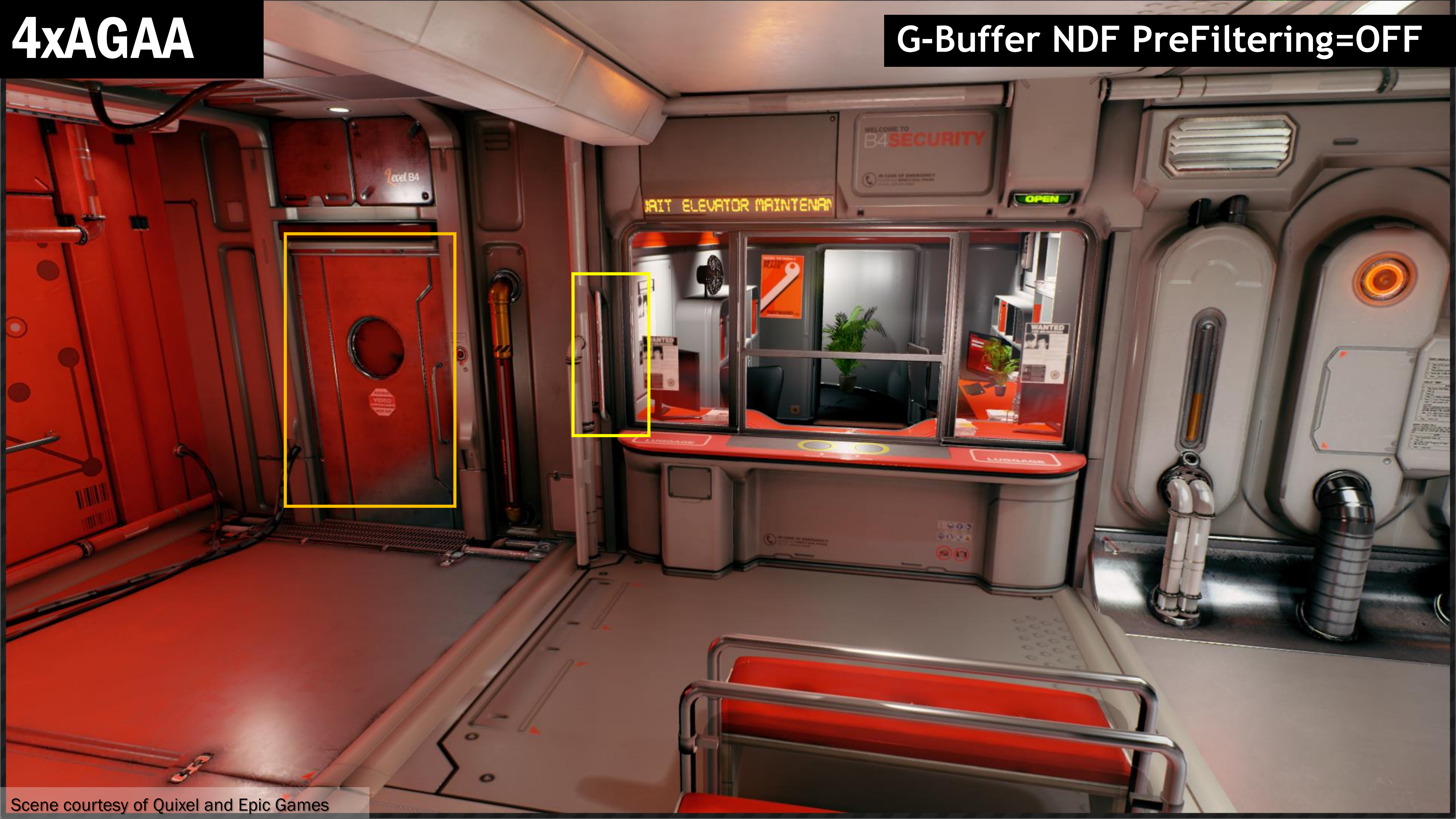
```
float2 GetAgaaKaplanyanFilteringRect(FMaterialPixelParameters MaterialParameters)
{
	// Shading frame
	float3 T = MaterialParameters.TangentToWorld[0];
	float3 ShFrameN = normalize(MaterialParameters.TangentToWorld[2]);
	float3 ShFrameS = normalize(T - ShFrameN * dot(ShFrameN, T));
	float3 ShFrameT = cross(ShFrameN, ShFrameS);

	// Use average quad normal as a half vector
	float3 hppW = GetAgaaAverageQuadNormal(MaterialParameters, ShFrameN);

	// Compute half vector in parallel plane
	hppW /= dot(ShFrameN, hppW);
	float2 hpp = float2(dot(hppW, ShFrameS), dot(hppW, ShFrameT));

	// Compute filtering region
	float2 rectFp = (abs(ddx_fine(hpp)) + abs(ddy_fine(hpp))) * 0.5f;

	// For grazing angles where the first-order footprint goes to very high values
	rectFp = min(View.AgaaKaplanyanRoughnessMaxFootprint, rectFp);
	return rectFp;
}
```

Kaplanyan's Curvature Filtering

```
float GetAgaaKaplanyanRoughness(FMaterialPixelParameters MaterialParameters, float InRoughness)
{
	float2 rectFp = GetAgaaKaplanyanFilteringRect(MaterialParameters);

	// Covariance matrix of pixel filter's Gaussian (remapped in roughness units)
	// Need to x2 because roughness = sqrt(2) * pixel_sigma_hpp
	float2 covMx = rectFp * rectFp * 2.f * View.AgaaKaplanyanRoughnessBoost;

	// Since we have an isotropic roughness to output, we conservatively take the largest edge of the filtering rectangle
	float maxIsoFp = max(covMx.x, covMx.y);

	return sqrt(InRoughness * InRoughness + maxIsoFp);		// Beckmann proxy convolution for GGX
}
```

4xAGAA

G-Buffer NDF PreFiltering=OFF



Scene courtesy of Quixel and Epic Games

4xAGAA

G-Buffer NDF PreFiltering=ON

Scene courtesy of Quixel and Epic Games

VRAM Overhead for 4xAGAA

AGAA Pass	Render Target	Video Memory Bytes
AGAA Clustering	AGAA MetaData	WxHx1 R16_UINT
AGAA Lighting & Reflections	Per-Aggregate Lit Colors	WxHx2 R11G11B10F
Merge Emissive	Per-Pixel Lit Color + Emissive	WxHx4 R11G11B10F
		Total VRAM overhead: 26 bytes / pixel

Resolve

- Emissive kept per-sample
- Always resolving tone-mapped colors [Karis2014]

Results

Image quality and performance

4xSSAA

Scene courtesy of Quixel and Epic Games

4xAGAA

Scene courtesy of Quixel and Epic Games


4xTAA
Scene courtesy of Epic Games

4xAGAA

Scene courtesy of Epic Games


4xSSAA
Scene courtesy of Epic Games

8xSSAA

Scene courtesy of Quixel and Epic Games

8xAGAA

Scene courtesy of Quixel and Epic Games

Disabling ReflectionEnvironment (this pass is not optimized yet)...

ReflectionEnvironment=ON

Scene courtesy of Quixel and Epic Games

ReflectionEnvironment=OFF

Scene courtesy of Quixel and Epic Games

4xAGAA Performance

GPU Time *(ms)*	4xSSAA	4xAGAA	4xAGAA/4xSSAA
Z PrePass	0.13	0.13	1.0x
GBuffer Fill	1.26	1.26	1.0x
Lighting	4.71	2.85	1.65x
PostProcessing	0.55	0.55	1.0x
Frame	6.65	4.79	1.39x

GPU times measured in 1080p on GTX 1080 (8GB) @ 1607 Mhz

Using the accurate Vis_Smith function (not Vis_SmithJointApprox)

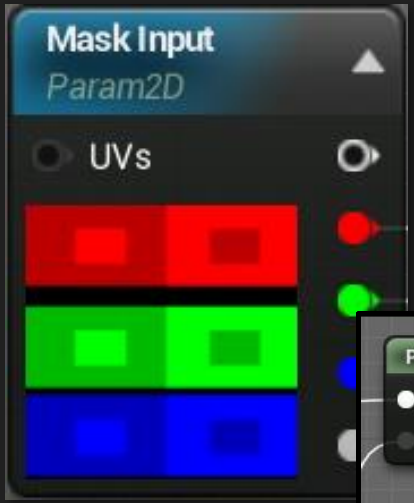
MSAA

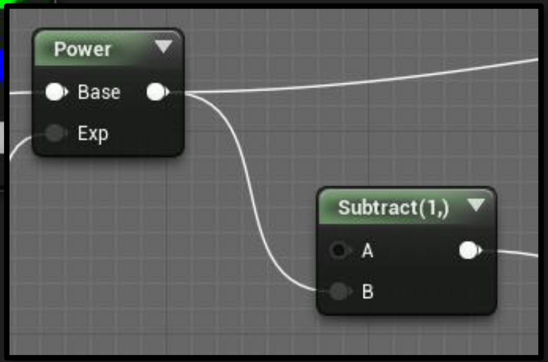
The cost of better performance

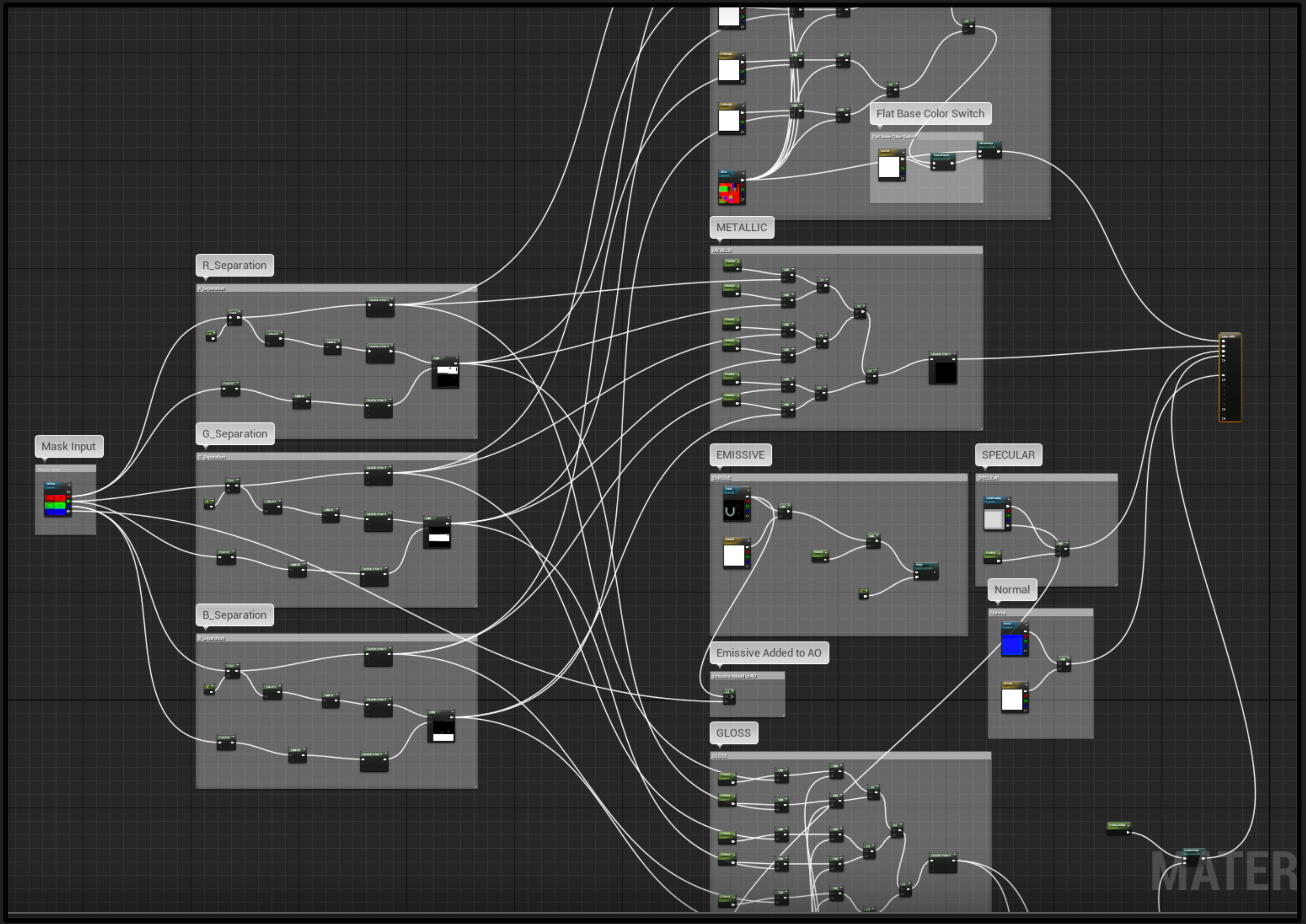
Why not MSAA ? Complex material graphs

Artist-controlled material definitions

- Non-linear operations
- Needs to be supersampled







Making MSAA More Feasible

Using MSAA in the GBuffer fill can produce great performance boosts (~2x) over super-sampling

However, per-fragment shading can introduce artifacts if the pixel shader is using discard or non-linear maths

Proposed solution:

1. Encourage artists to avoid non-linear material nodes (pow, clamp, ...)
2. Selectively super-sample the GBuffer attributes that have nonlinearities

Limitations

AGAA is speeding up only the lighting pass

Non-standard UE shading models not fully tested yet

More than 2 shading model IDs / pixel untested

Conclusion

AGAA speeds up super-sampled lighting

- 4x AGAA lighting is 1.7x faster than 4x SSAA
- 8x AGAA lighting is 2.6x faster than 8x SSAA

Can be combined with TAA or used alone

Still ongoing work

Thanks

Natalya Tatarchuk

Aaron Lefohn

Jon Jansen

Anton Kaplanyan

Questions?

References

[TVCG 2016] Cyril Crassin, Morgan McGuire, Kayvon Fatahalian, Aaron Lefohn, "Aggregate G-Buffer Anti-Aliasing - Extended Version", TVCG, 2016.
http://research.nvidia.com/sites/default/files/publications/AGAA_Extended_TVCG2016_AuthorsVersion.pdf

[Kaplanyan 2016] A. Kaplanyan, S. Hill, A. Patney & A. Lefohn, “Filtering Distributions of Normals for Shading”, HPG 2016.

[Heitz 2015] Eric Heitz, Jonathan Dupuy, Cyril Crassin and Carsten Dachsbacher, “The SGGX Microflake Distribution“, SIGGRAPH 2015.

[I3D 2015] Cyril Crassin, Morgan McGuire, Kayvon Fatahalian, Aaron Lefohn, "Aggregate G-Buffer Anti-Aliasing“, I3D, 2015.

[Karis 2014] Brian Karis (Epic), “High-Quality Temporal Super-Sampling”, SIGGRAPH 2014.

[Baker and Hill 2012] Stephen Hill (Ubisoft) & Dan Baker (Firaxis), “Rock-Solid Shading: Image Stability without Sacrificing Detail”, GDC 2012.

[Bruneton and Neyret 2012] Bruneton & Neyret, “A Survey of Non-linear Pre-filtering Methods for Efficient and Accurate Surface Shading”, TVCG 2011.

[Toksvig 2005] Toksvig, “Mipmapping normal maps”, Journal of Graphics Tools 10, 2005.