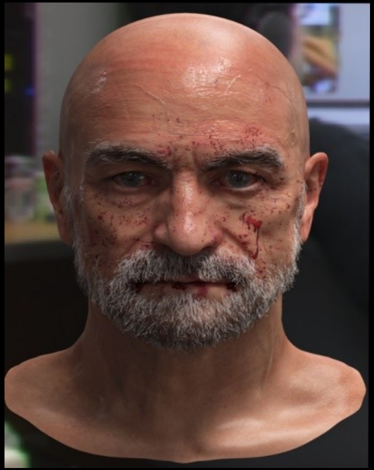
リアルタイムグラフィックス技術解説

株式会社スクウェア・エニックス
テクノロジー推進部
Driancourt Remi
Metaaphanon Napaporn

Agni’s Challenge

A Fantastic & Believable World

CG Movie-Quality in Real-Time



Plan of Presentation

- Lighting & Basic concepts
- Characters
  - Skin
  - Eyes
  - Hairs
- Effects
  - Volumetric light effect
  - Refraction
  - Particles

Remi (Lighting & Basic concepts through Volumetric light effect)

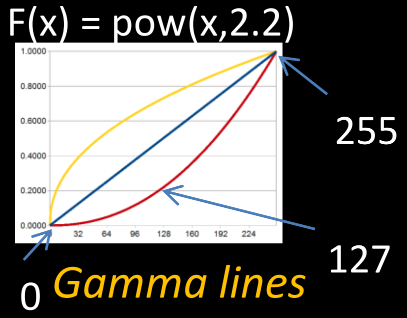
Napaporn (Noi) (Refraction, Particles)

Lighting & Basic Concepts

PhotoRealism

Linear Lighting

- Shading operations done in linear space
- Tonemapping HDR into LDR
- Color correction to change appearance & contrast



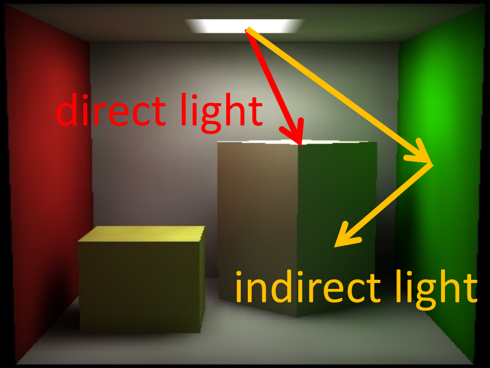
Gamma lines

Look-up table
For color correction

Recommended References:
- *"Uncharted 2: HDR Lighting" (J. Hable)*
- *"The Importance of Being Linear" (Eugene d'Eon) Gpu Gems 3*

Global Illumination

- To achieve photo-realism, we need to account for indirect lighting



Recommended References:
- *"Global Illumination Across Industries" SIGGRAPH 2010 Course*
- *"The State of the Art in Interactive Global Illumination" (T. Ritschel, C. Dachsbacher,T. Grosch, J. Kautz) Comput. Graph. Forum,2012*

Luminous GI Tech

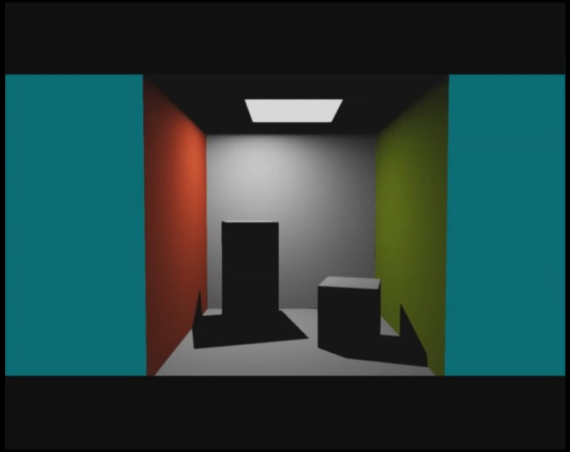
- Methods under investigation:
  - Many Lights (VPL)
  - Sparse Voxel Octree
  - Ray bundles
  - Etc..

Published papers

- *"Fast Global Illumination Baking via Ray-bundles" (Y. Tokuyoshi, T. Sekine and S. Ogaki) SIGGRAPH ASIA 2011*
- *"Real-Time Bidirectional Path Tracing via Rasterization" (Y. Tokuyoshi and S. Ogaki) SIGGRAPH Symposium on Interactive 3D Graphics and Games 2012*

Agni's GI choice

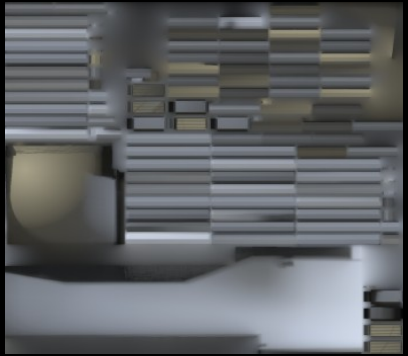
  - No interactivity
  - We want quality
  - We don't want to sacrifice all resources on GI
- Choice:
  - Internal baking tool（RayBundles）
  - Lightmaps: 2D & 3D

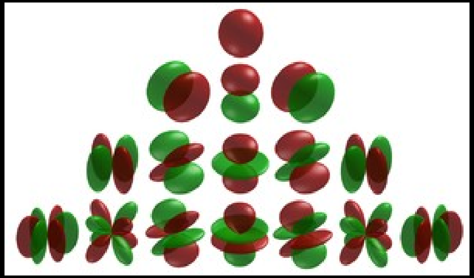


Reference: "GPGPUによる高速なグローバルイルミネーションベイクツールの作り方"(T. Sekine) Cedec 2012

2D: SH Lightmaps

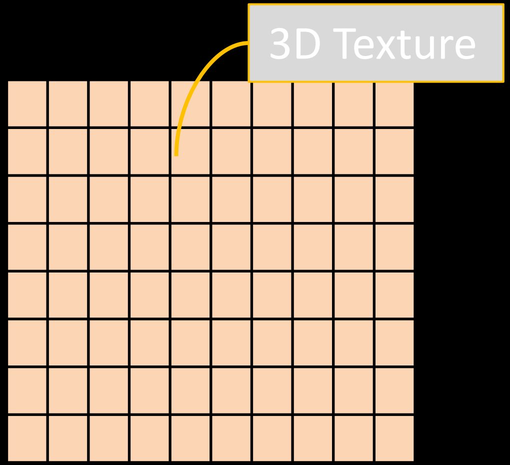
- Compute lighting over the surface of the model
- Store it inside a texture, in SH form
- Model & Light have to be static
- Light keeps a “sense of direction”





3D: SH Irradiance Volumes

- A grid of irradiance samples is taken & stored as SH
- At render time, GI estimated from near-by samples
- Light cannot change, but objects can move within this volume
- Need more memory than 2D but can be shared



Reference:
- Irradiance volumes (Greger et al.) 1997
- SH Irradiance Volumes(Tatarchuk) 2004

Impressions

- Merits
  - Good quality. Appropriate for Agni's philosophy
  - Cheap at runtime
- Demerits
  - Takes a long time to prepare the data
  - Does not allow changes in the scene

Baked GI Result

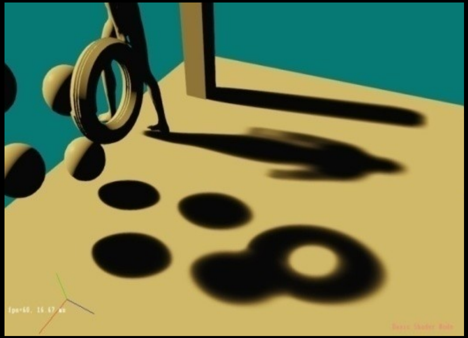
Baked GI Result

Agni's Lighting Policy

	Static Objects=BG	Dynamic Objects
Receive Direct Light ?	YES	YES
Indirect Light From..	SH Lightmaps	Irradiance Volume
Receive real-time shadow from static objects	NO	YES
Receive real-time shadow from dynamic objects	YES	YES

Shadows

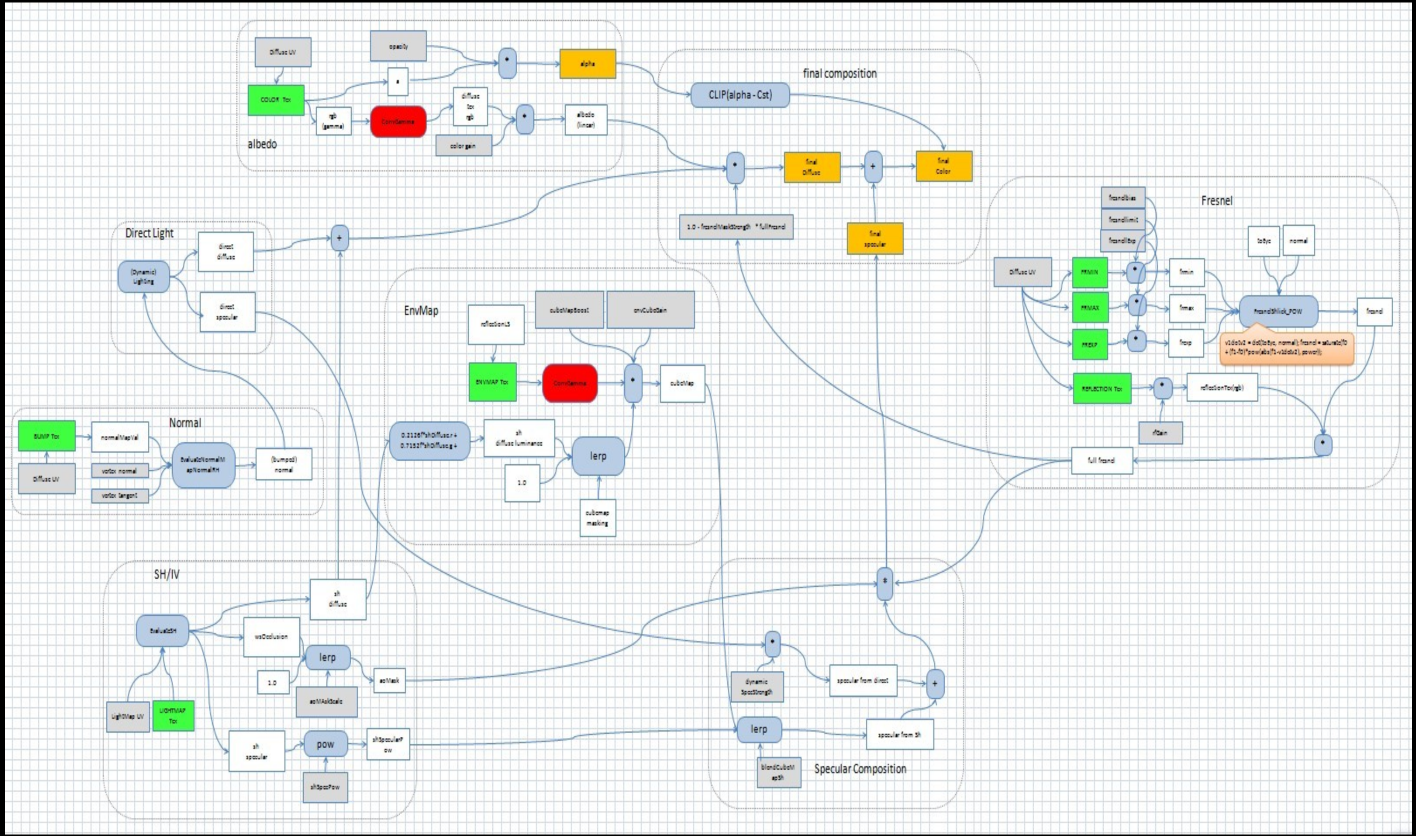
- A lot of interesting tech for an unsolved problem
  - Cascade Shadow Maps
  - Percentage Closer Soft Shadows
  - Variance Shadow Map
  - Screen Space Soft Shadow
  - Summed-Area Tables



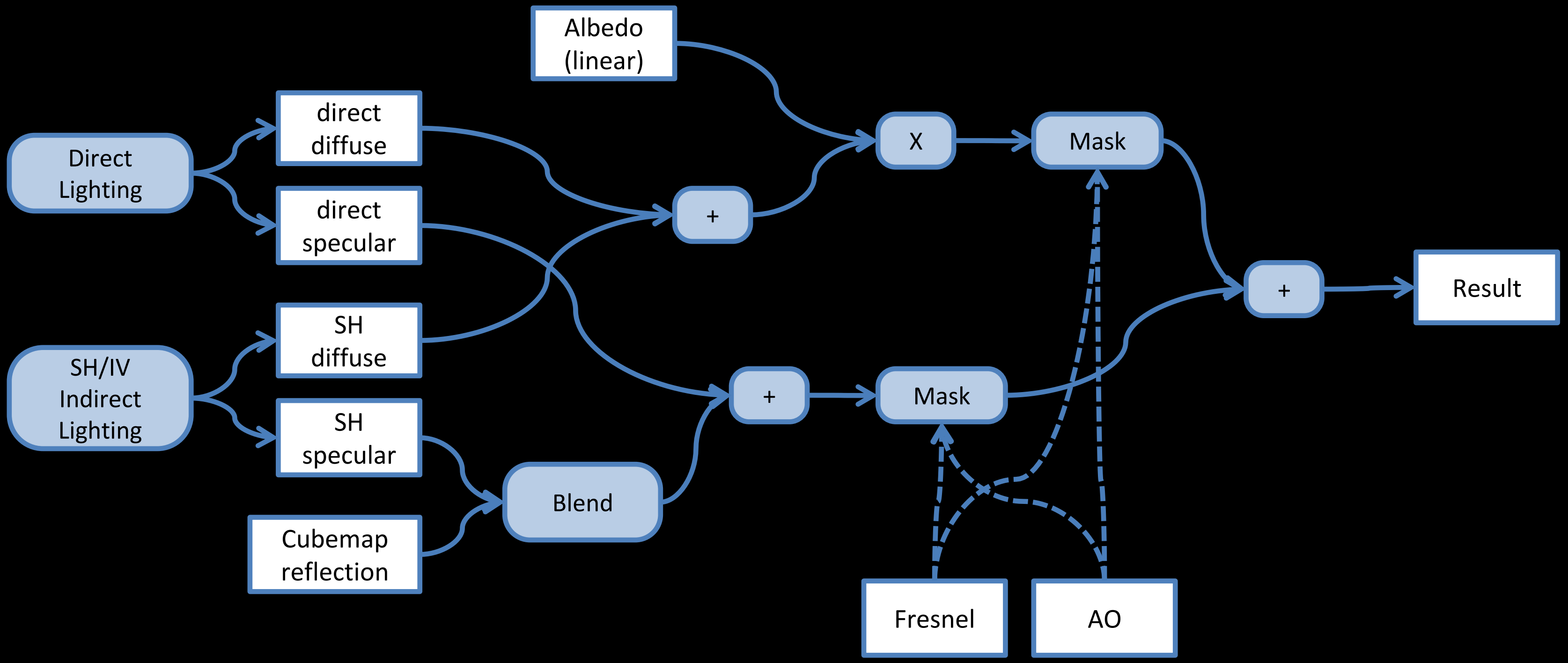
Recommended References

- *Real-Time Shadows (E. Eisemann, M. Schwarz, U. Assarsson, M. Wimmer)*
- *"Efficient real-time shadows" Siggraph 2012 course*
- *"Summed-Area Variance Shadow Maps" Gpu Gems 3*

Basic Shading Network



Shading Network (simplified)

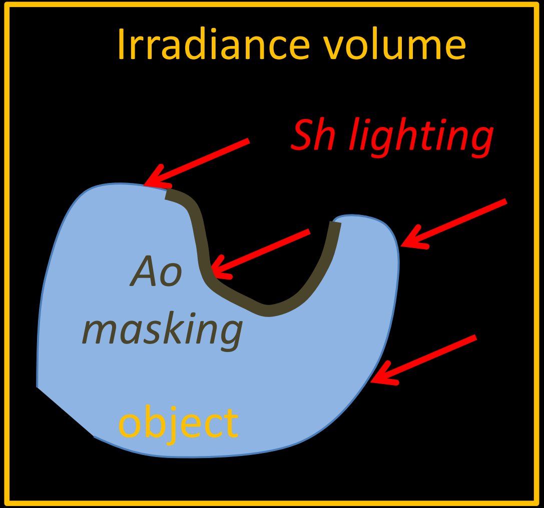


Fresnel

- The amount of reflectance you see on a surface depends on the viewing angle
- Everything has fresnel !

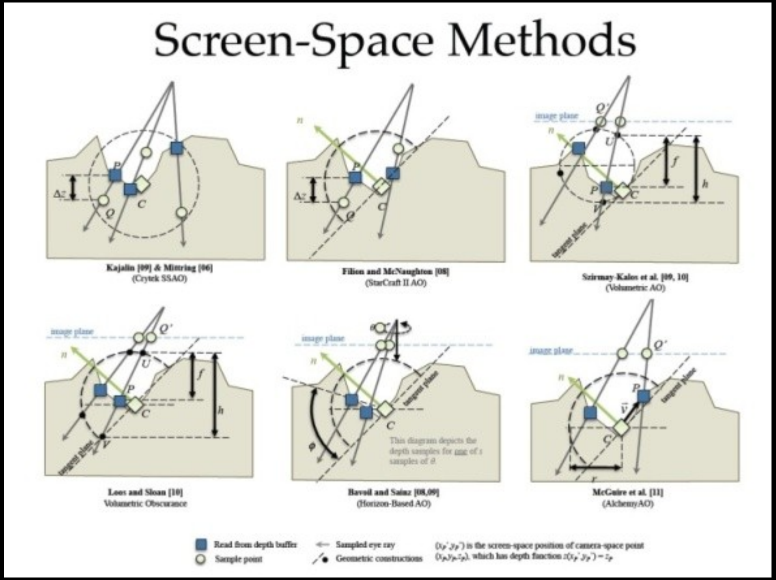
Ambient Occlusion

- Approximate effect of geometry on illumination
- Necessary for objects in Irradiance Volume



Ambient Occlusion

- Many methods: Image-based & Object-based



From "Scalable Ambient Obscurance" (Morgan McGuire)

References:
- *"State of the Art Report on Ambient Occlusion" (Martin Knecht)*
- *"Scalable Ambient Obscurance" (Morgan McGuire) Eurographics 2012*

NEED SLIDE TO SHOW
SH vs IVOL + AO

Agni's AO

- Baked AO
  - Using our Internal baking tool
  - High details
- Screen-Space AO
  - Inspired from "Horizon-Based AO"
  - Lower details, but more appropriate to dynamic objects
- Object-Based AO
  - Scarcely used
  - Analytical: can give stable & huge range AO on simple shapes

Baked AO

SSAO

Object-based Analytical AO : OFF

Object-based Analytical AO : ON

Character Rendering Tech:
Skin

Goal

Match VW Skin

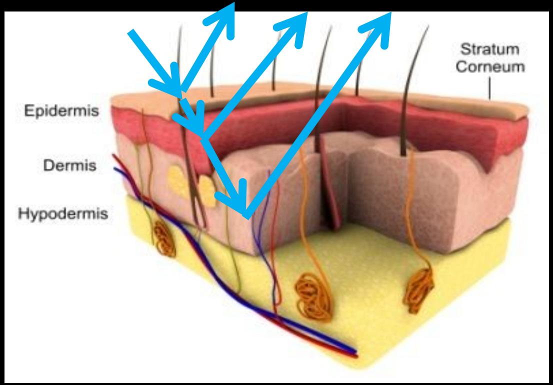
Challenges

- Usual shading doesn’t work well: Skin looks dry !

Challenges

- Real Skin looks softer, has a translucent appearance

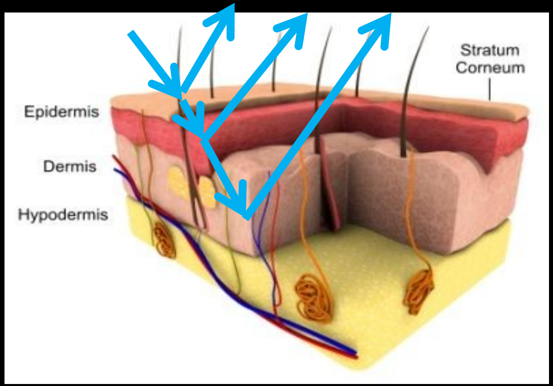
Skin & Light



Diffuse = 94% (Subsurface Scattering)

- Skin has different layers & light scatters under it !
  - incident angle does not matter after 1/10th first layer!
- Lights get colored depending on where it went
  - Epidermis scattering is narrow
  - Dermis scattering is wider, mostly red

Skin & light



Reflection = 6%

- Light not colored by the skin
- Rough surface
- Fresnel

Experiments on Skin:
Diffuse

SSS Tech

- Popular techniques:
  - Red wrapped lighting
  - Blended Normals
  - Pre-integrated SSS
  - Texture-space diffusion
  - 12-tap combined
  - Screen-space diffusion
  - Etc….

- Need to find a balance between quality and speed

Recommended References

- "Cheap Realistic Skin Shading" (Stephen Clement)
- "Real-Time Approximations to Subsurface Scattering" (Simon Green) GPU Gems 2004
- “Pre-Integrated Skin Shading” (Eric Penner) Siggraph 2011
- "GPU Gems 3: Advanced Skin Rendering" (Eugene D'Eon) Siggraph 2007
- "Uncharted 2: Character Lighting and Shading" (John Hable) Siggraph 2010
- "Separable Subsurface Scattering" (Jorge Jimenez, Adrian Jarabo & Diego Gutierrez) Siggraph 2012
- "Screen-Space Perceptual Rendering of Human Skin" (Jorge Jimenez and Veronica Sundstedt and Diego Gutierrez) ACM Trans. on Applied Perception 2009

Old Tricks

- Bent normals
  - Pretend that R/G/B come from different normals
  - R close to geometry -> softer change in lighting
  - GB closer to bump map -> sharper changes
- LightWrapping
  - diffuse ~= *dot*(L, N);
  - wrap_diffuse ~= (*dot*(L, N) + wrap) / (1 + wrap);

Experiment 1

- "Simulate" multiple layers of skin with different levels of lightwrapping, normal bending, shadow PCF radius..

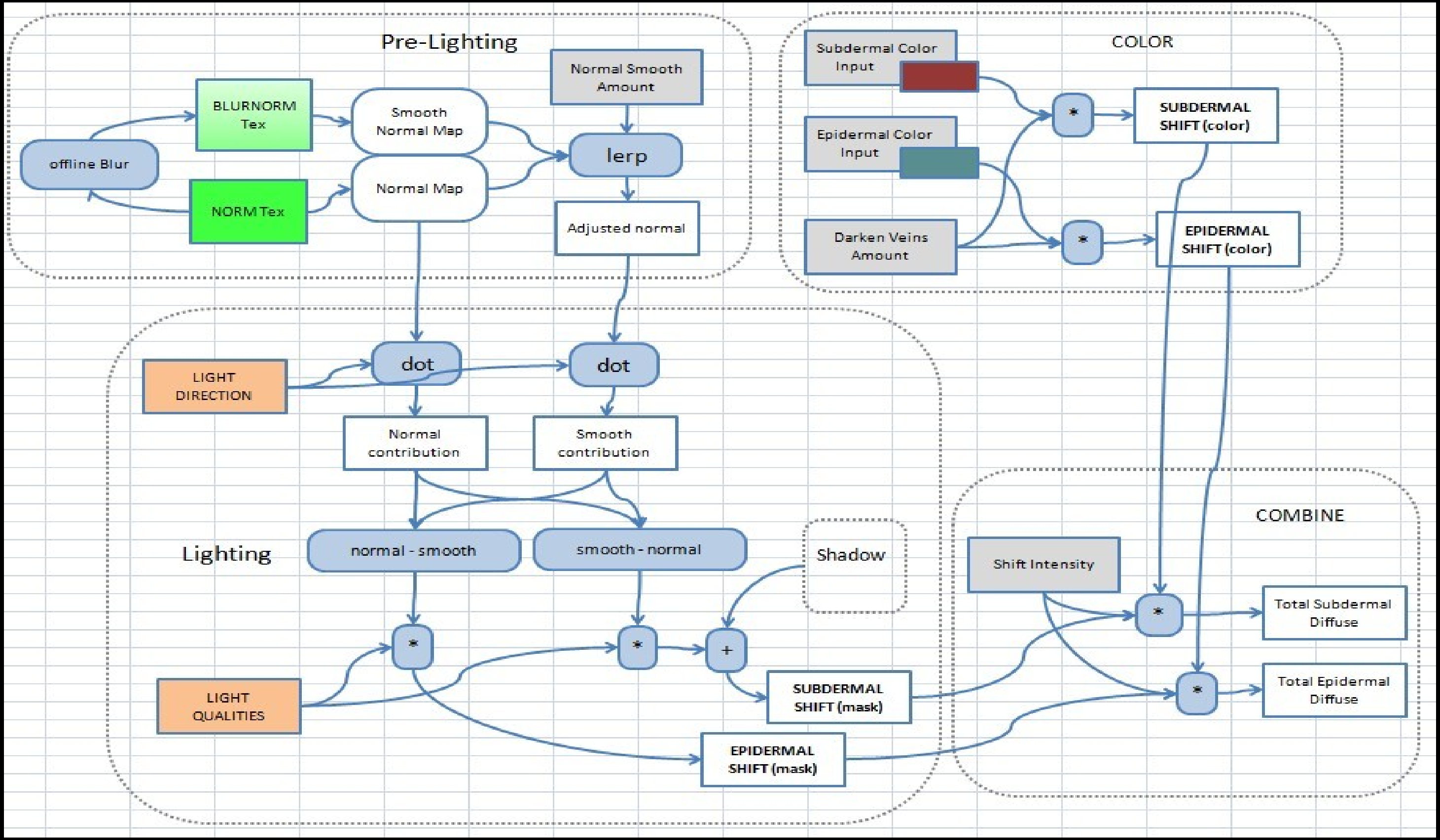
- Impressions:
  - Speed: cheap… but not so cheap
  - Quality: Not bad … but really not good enough

Experiment 2

- Use bump map & blurred bump map
- From the difference, generate a fake "color shift" at shading boundaries

  - Speed: Less expensive than previous tech
  - Quality: quite good
  - Problems: UV Seams & Animation

Experiment 2



Fake Color shift OFF

Fake Color shift ON

Fake Color Shift result

Texture Space Diffusion

- Offline: “the Matrix”

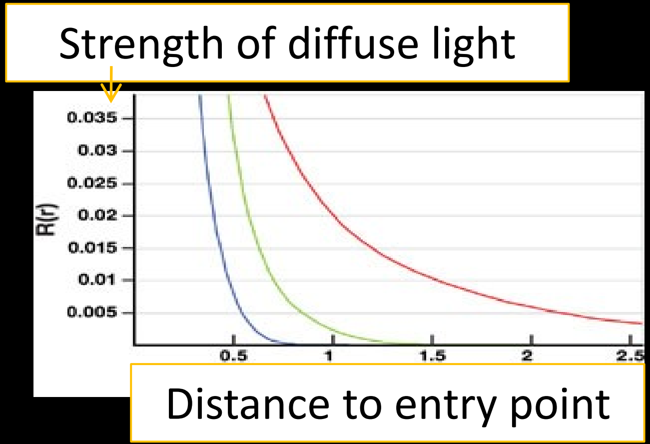
- RealTime: Nvidia’s “Human Head Demo”

Texture Space Diffusion

It is possible to measure how light scatters under the surface of a translucent material

=Diffusion Profile

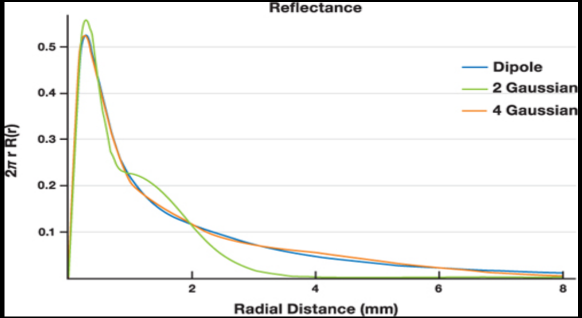
レーザーを透過させた結果



From http://http.developer.nvidia.com/GPUGems3/gpugems3_ch14.html

Texture Space Diffusion

- Diffusion profile approximated with 6 gaussian blurs & different R/G/B weights



For human skin we recommend

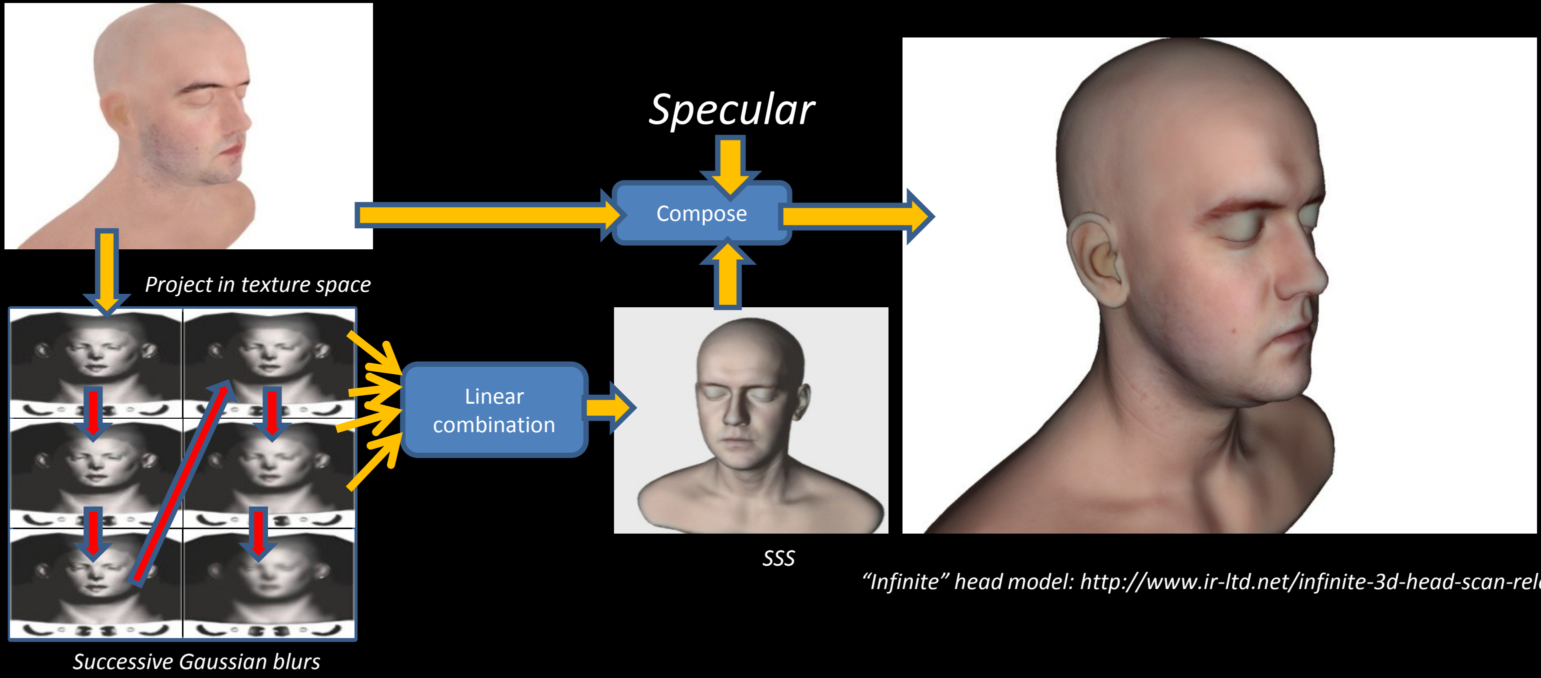
Blur Width(mm)	Blur Weights red	green	blue
0.042	0.233	0.455	0.649
0.220	0.100	0.336	0.344
0.433	0.118	0.198	0
0.753	0.113	0.007	0.007
1.412	0.358	0.004	0
2.722	0.078	0	0

Final linear combination

From http://http.developer.nvidia.com/GPUGems3/gpugems3_ch14.html

SSS = weighted average of blurred Diffuse

Texture-Space SSS pipeline



"Infinite" head model: http://www.ir-ltd.net/infinite-3d-head-scan-released

Texture-Space SSS: impressions

- Looks good !
- Problems:
  - Lots of passes
  - Lots of memory
  - Problem at UV seams
  - No early Z-rejection
  - Redundant calculations
  - Irradiance map size needs to be managed
  - Does not scale well

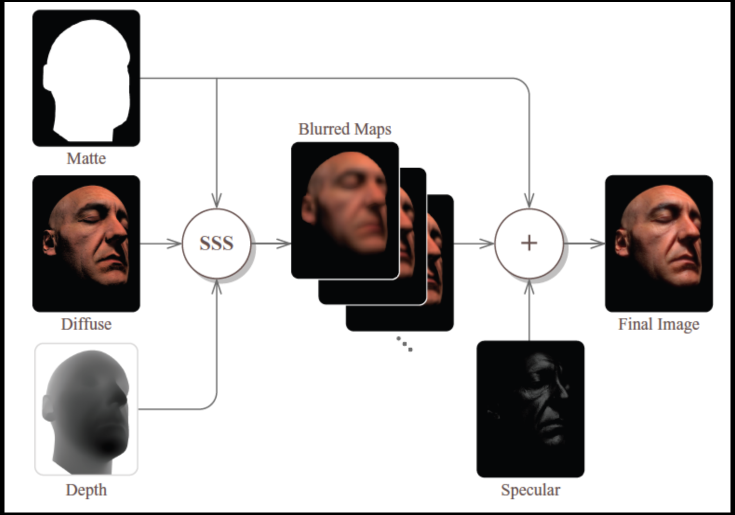
Screen-Space SSS

“Screen-Space Perceptual Rendering of Human Skin” [2009]
(Jorge Jimenez and Veronica Sundstedt and Diego Gutierrez)

- SSS = weighted average of blurred diffuse..

… in Screen Space !

  – Fixed cost for any number of objects

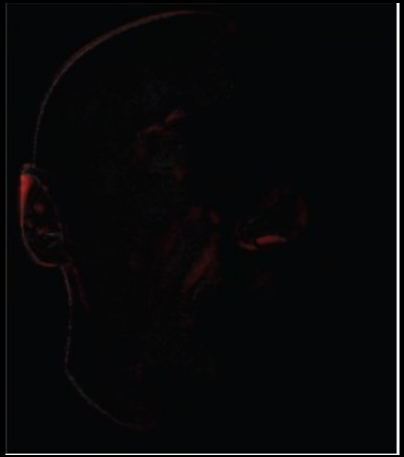


From “Screen-Space Perceptual Rendering of Human Skin” [2009]
(Jorge Jimenez and Veronica Sundstedt and Diego Gutierrez)

Screen Space SSS: Quality

- Not very different from texture space
- Some problems with highly curved surfaces (e.g. ears)

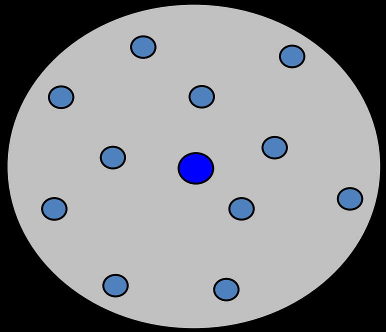
Difference in result:
Texture-Space
vs. Screen-Space



From "Screen-Space Perceptual Rendering of Human Skin" [2009]

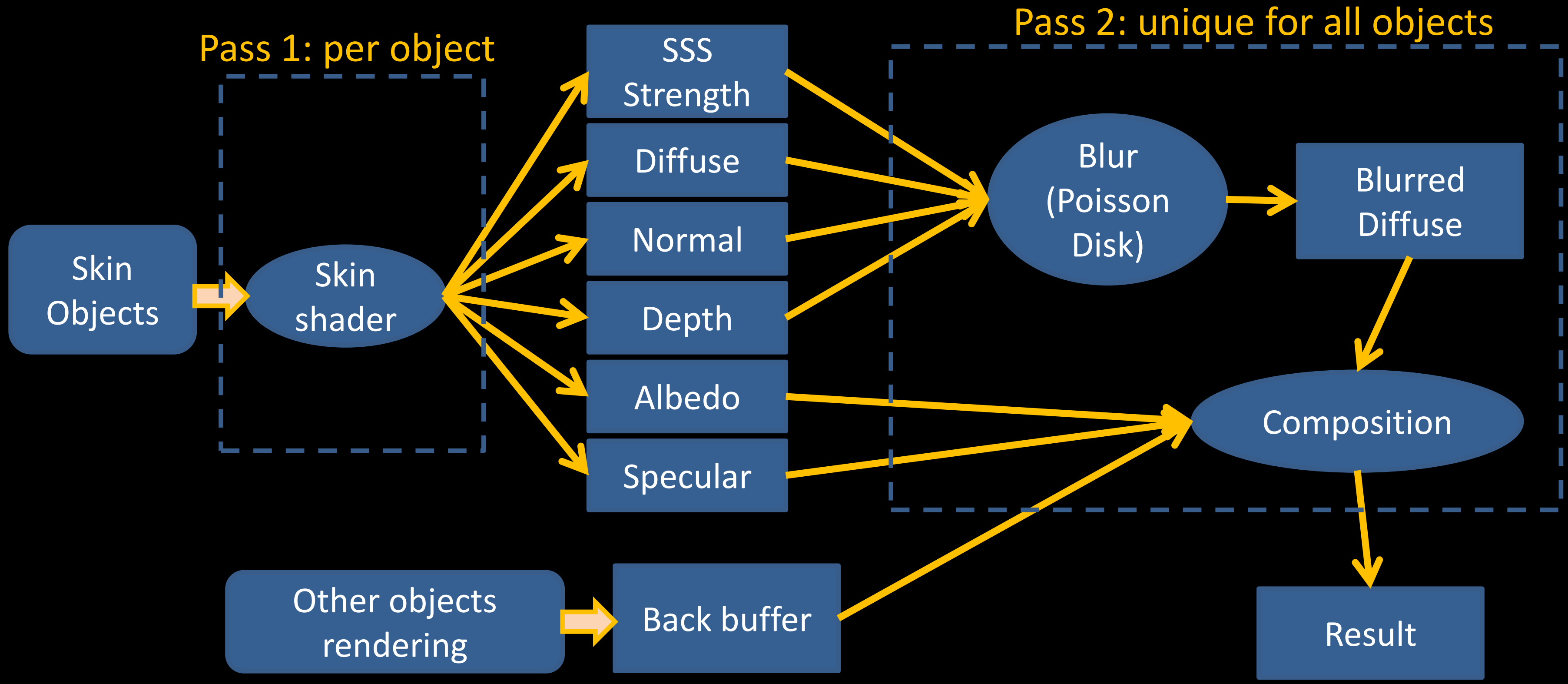
Our SS-SSS

- Poisson disk sampling
- Each sample receive a different weight depending on:
  - Distance to center
  - Difference in depth
  - Difference in normal



Disk can be jittered to trade banding artifact with noise

Our SS-SSS



Pass 1: per object
Pass 2: unique for all objects
Skin Objects
Skin shader
SSS Strength
Diffuse
Normal
Depth
Albedo
Specular
Blur (Poisson Disk)
Blurred Diffuse
Composition
Other objects rendering
Back buffer
Result

Pseudo Code

```
float4 Color        = Get(AlbedoMap, UV).rgb;
float3 Diffuse      = Get(DiffuseMap, UV).rgb;
float Depth         = Get(DepthMap, UV);
float3 Nrm          = Get(NrmMap, UV).rgb;
float3 Specular = Get(SpecularMap, UV).rgb;
float SSSStrength = Get(NrmMap, UV).w;
clip(Depth - ALPHA_CLIP);
float BlurRadius = SSSStrength * GlobalRadiusIntoUV;
for(int i=0; i<32; i++){
        UVOffset = poissonDisk32[i] * BlurRadius;
        float2 UVSample = UV + UVOffset;
        float SampleDepth = Get(DepthMap,  UVSample, MipMapLevel).x;
        float4 SampleColor = Get(DiffuseMap, UVSample, MipMapLevel).rgba;
        float3 SampleNrm  = Get(NrmMap, UVSample, MipMapLevel).rgb;
        float DepthDif = SampleDepth - Depth;
        float Weight =  SampleColor.a; //=1 if SSS surface =0 else
        Weight *=  exp( - DepthSensibility * DepthDif * DepthDif); //take depth difference into account
        Weight *=  exp( - NrmSensibility * (1.0 - saturate(dot(SampleNrm, Nrm.xyz)))); //take normal difference into account
        Weight *= exp(- Radius_RGB * poissonW32[i]);//take distance to kernel center into account
        Diffuse.rgb += SampleColor.rgb * Weight;
        TotalRGB += Weight;
}
Diffuse.rgb /= TotalRGB;
float3 Composition = Color.rgb * Diffuse.rgb + Specular.rgb;
```

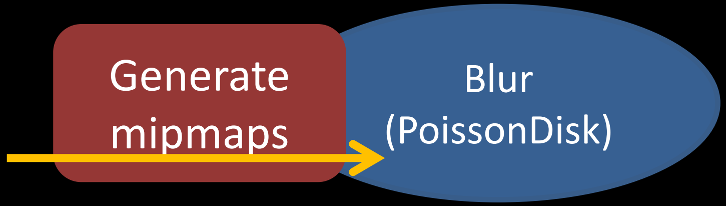
Our SS-SSS

- Merit: Allow varying size of blur
  - According to distance object-camera
- Merit: Allow varying sample number
  - LOD: use less samples for far objects

Use different SSS for different parts

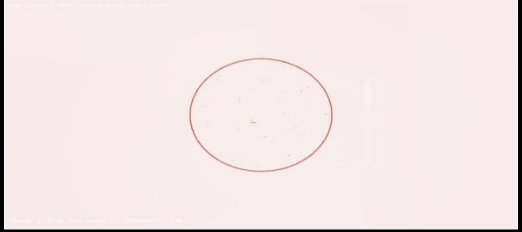
Our SS-SSS

- Performance:
  - Lot of texture fetches = performance is strongly driven by cache efficiency.
  - Cache coherency could be damaged by big radius, but the use of mipmap limits the problem

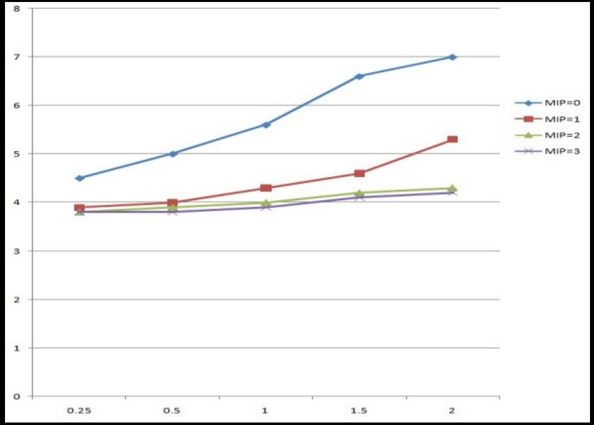


Our SS-SSS

- Looked at the worst case:
  – Full screen SSS, 32 samples, huge radius
  – For each sample: albedo, depth & normal fetch



	Radius:	0.25	0.5	1	1.5	2
Time(ms)	MIP=0	4.5	5	5.6	6.6	7
Time(ms)	MIP=1	3.9	4	4.3	4.6	5.3
Time(ms)	MIP=2	3.8	3.9	4	4.2	4.3
Time(ms)	MIP=3	3.8	3.8	3.9	4.1	4.2



Our SS-SSS

- Usual case:


~0.1ms


~0.25ms


~0.7ms

Diffuse Only, SSS=Off

Diffuse Only, SSS=Off

Samples' Normal ignored

Samples' Normal taken into account

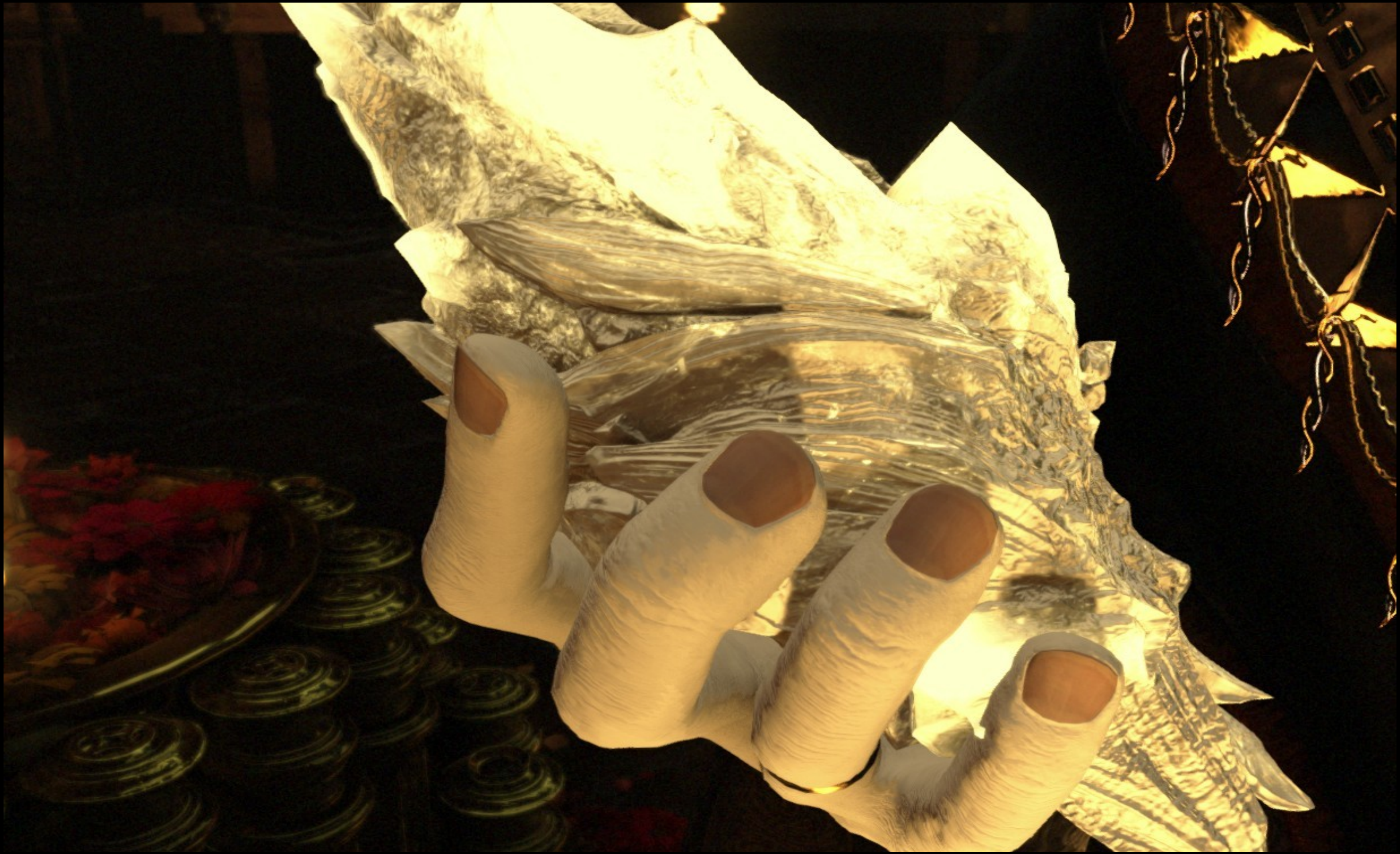
Results: SSS Only, Specular=Off

Results: SSS Only, Specular=Off

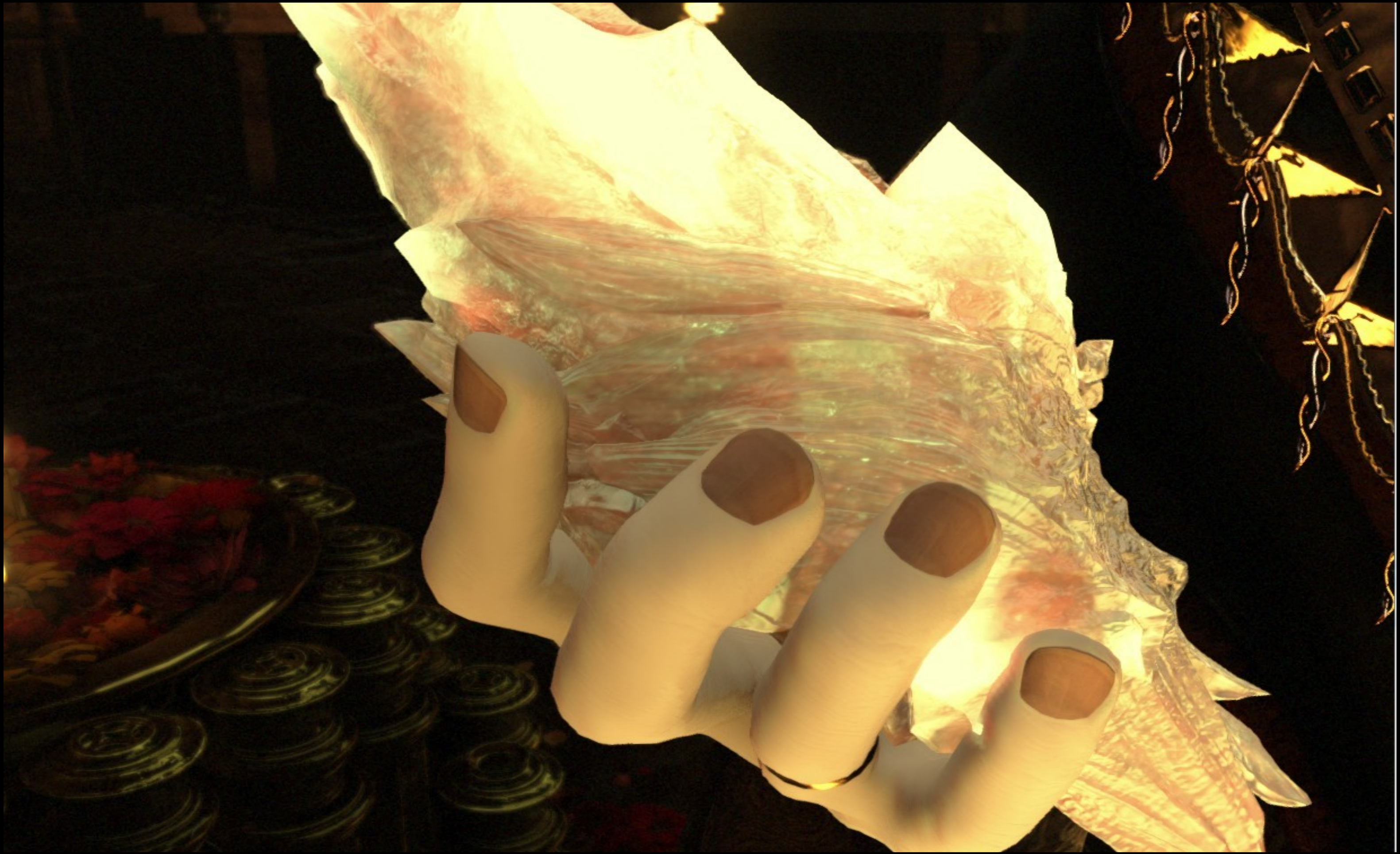
Results: SSS Only, Specular=Off

Results: SSS Only, Specular=Off

Results: SSS Only, Specular=Off



Results: SSS Only, Specular=Off



Experiments on Skin:
Specular

Specular BRDF

- Blinn / Phong not enough
- Kelemen Szirmay-Kalos 2001.
  - Faster than Torrance-Sparrow
  - Schlick approximation to Fresnel
  - Specular color : same as light

Using textbook Fresnel Formula   Using Schlick's Fresnel

Kelemen Szirmay Specular

```
float fFresnel = GetFresnelKS(DOT_HV , 0.028f );
float4 fBeckmann = GetBeckmannDistribution(DOT_NH,  0.6f);
float4 fSpec = max( (fBeckmann * fFresnel) / dot(H,H ), 0 );
float KelemenSzirmay = saturate(DOT_NL) * dot(fSpec, half4(1.0f, 0.625f, 0.075f, 0.005f))
Specular = lightSpecularXshadow * KelemenSzirmay;

float4 GetBeckmannDistribution( float NdotH, float Exp ){
	float4 m = half4(1.0f, 0.12f, 0.023f, 0.012f) * (Exp * Exp);
	float alpha = acos( NdotH );
	float ta = tan( alpha );
	float4 val = 1.0f / (m * pow(NdotH, 4.0f)) * exp(-(ta * ta) / m);
	return val;
}

float GetFresnelKS( float HdotV, float F0 ){
	float base = 1.0f - HdotV;
	float exponential = pow( base, 5.0f);
	exponential += F0 * ( 1.0f - exponential );
	return exponential;
}
```

In practice: 2 speculars

"Dry" Specular
-Blend between SH specular and Cubemap
-Cubemap reflection can be dulled

"Wet" Specular
-Pure Cubemap
-Can use a "Sweat texture" mask

Combination
Appreciated by artists !

Pseudo code

```
// Specular Wet Mask: White=Wet / Black=Dry
Color3 SpecularTexture = sampleTex( reflection_texture ).rgb;
Float  WetMaskTexture = sampleTex( reflection_texture ).alpha;

// Calculate Dry
Color3 drySpecular    = EvaluateLighting( params, dryBRDFSpecPower_param );
Color3 dryAmbSpecular = EvaluateGlobalIllum( params );
Color3 dryCubeMap     = sampleEnv( cubemap, dryEnvcubeDullBias_param);
Color3 dryReflection  = lerp( dryCubeMap, dryAmbientSpecular, CubeSH_BlendControl );
Color3 dryTotalSpec = drySpecular + dryReflection;

// Calculate Wet
Color3 wetSpecular    = EvaluateLighting( params, wetBRDFSpecPower_param );
Color3 wetCubemap     = sampleEnv( cubemap, noBias=0 );
Color3 wetReflection  = wetCubemap; // no lerp, no ambient specular. Pure, sharp cubemap
Color3 wetTotalSpec   = (wetSpecular + wetReflection) * wetIntensityControl;

// Combine
Color3 combinedSpecular  = lerp( dryTotalSpec , wetTotalSpec , SpecularWetMask) ;
```

Usual Phong Specular

Our speculars

Usual Phong Specular

Our speculars

Specular without Cubemap reflection

Adding Cubemap reflection

Dry Specular only.

Using Wet specular with a mask

“Dry” Sidoro

Adding Sweat

Final composition

speculars Only

Ambient Occlusion

- Very important "masking" role
- Baked AO or Screen-Space AO
- "Saturated AO"
  - Usual AO: Color = Color * AO
  - Saturated AO: Color = pow(Color, 1 – AO);

AO: OFF

AO: ON

AO: OFF

AO: ON

AO: OFF

With SSAO

With Baked AO

SSAO result

Baked AO result

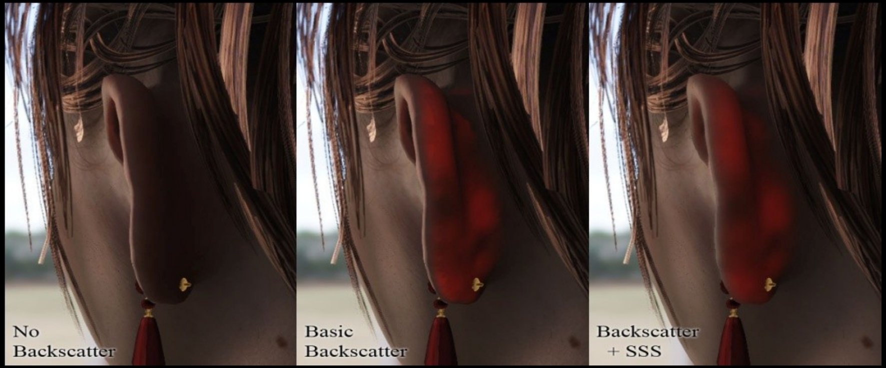
Usual “black” AO

Use “saturated” AO

Use “saturated” AO (change saturation)

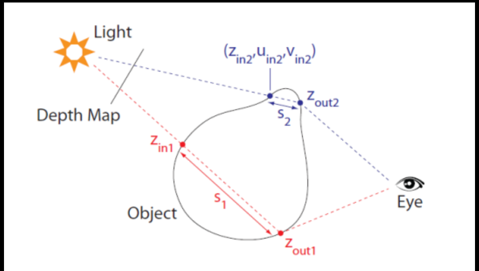
Back Scattering

- SS-SSS deals with the reflectance of the skin
- Transmittance also need to be considered
  - E.g. light coming through the ears



BackScattering

- Extremely simple implementation.
  - Lightwrapped Lambert diffuse: dot(Nback,L)
  - Consider Nback~= -NFront
- Attenuation of light depends on depth of skin
  - use shadow map to get depth
  - Or use a “thickness” texture



No Backscatter

Backscatter: ON

No Backscatter

Backscatter: ON

Backscatter. SSS=OFF

Conclusion

Quality

– Quite good
– SSS could create artifacts (aura)

Speed

– Quite cheap. Good Scalability

Implementation

– Easy

Skin: DEMO

Character Rendering Tech:
Eyes

Challenges

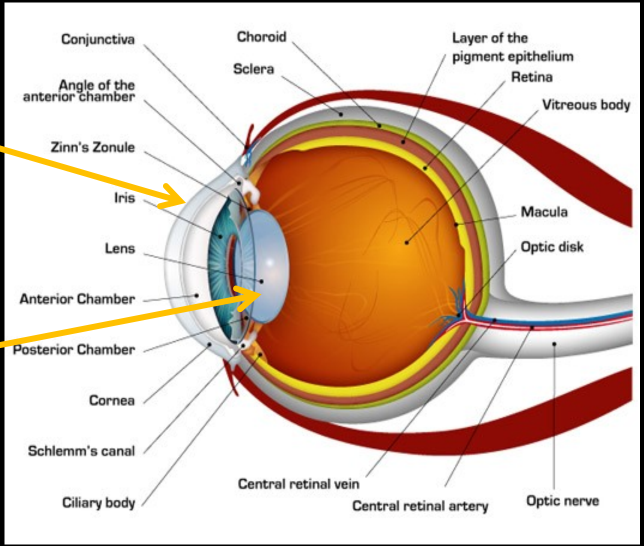
- Small part of the body…But really important !

Challenges

- More complex than just a sphere !

Lens
(outside)

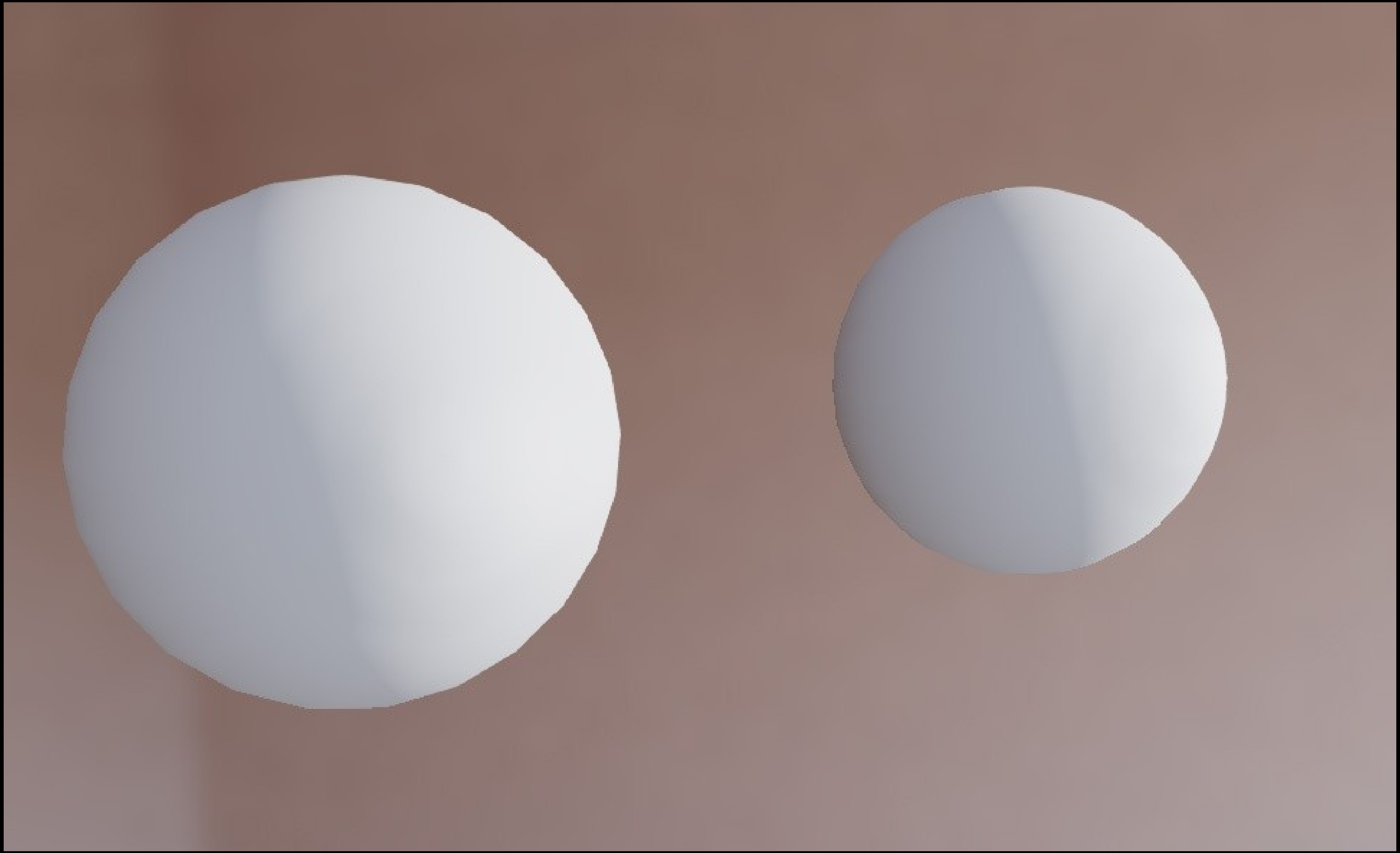
Iris
(inside)



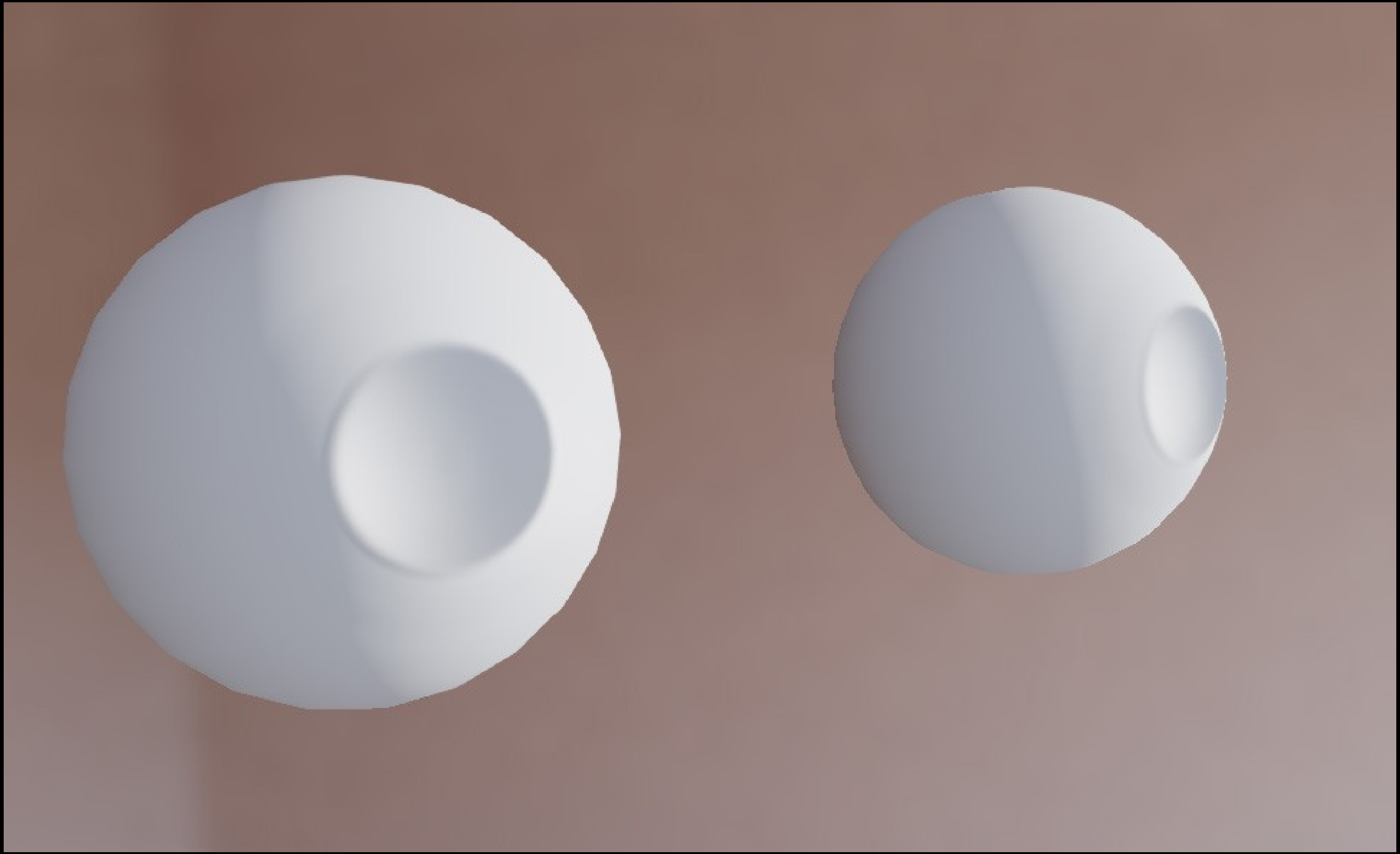
Eye Lighting

- To shade:
  - Use bump map for inside's diffuse
  - Use inverse bump map for outside's cubemap
  - Use both maps for specular (inside & outside)

Eye geometry

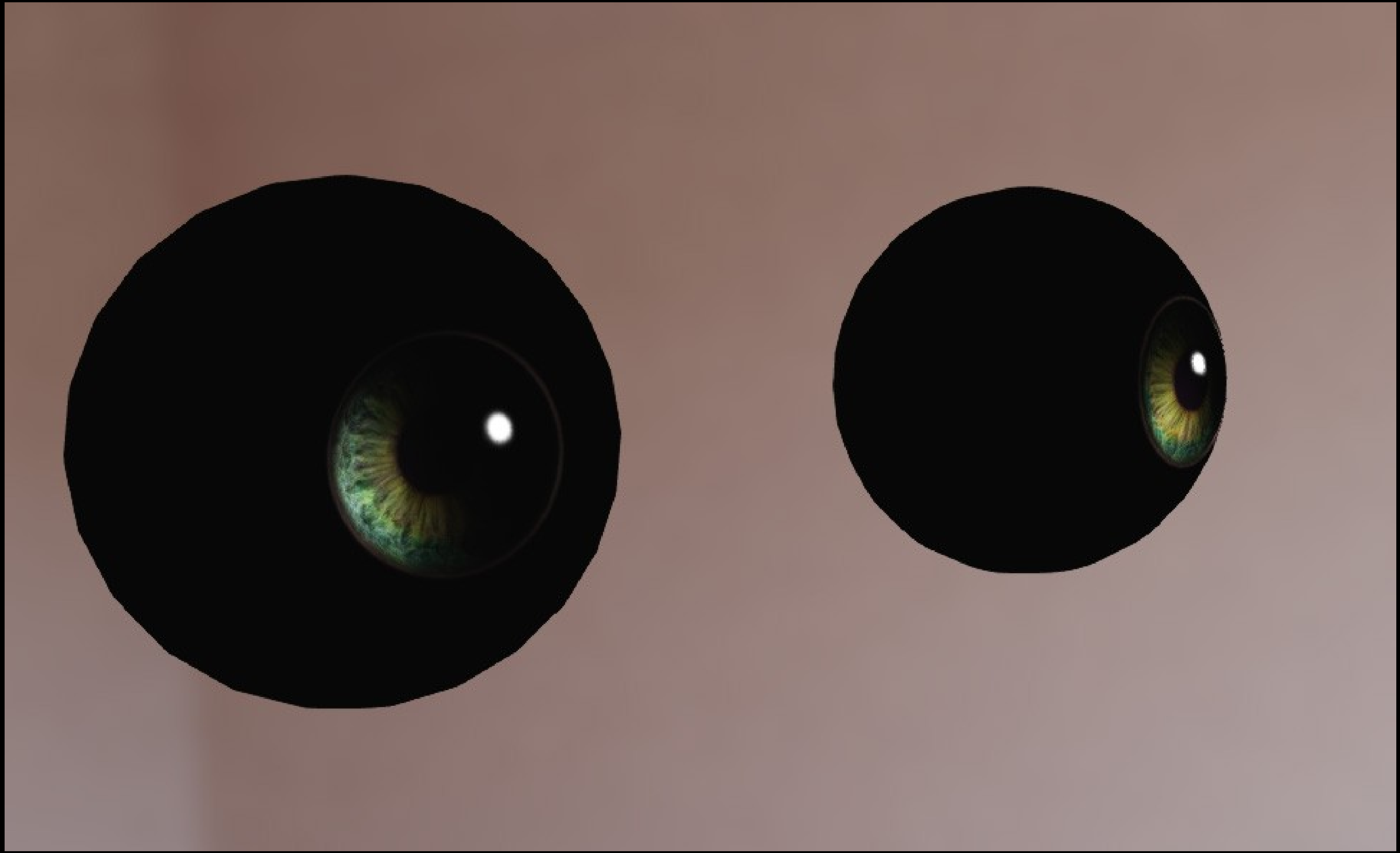


Diffuse lighting: use bump map for the inside

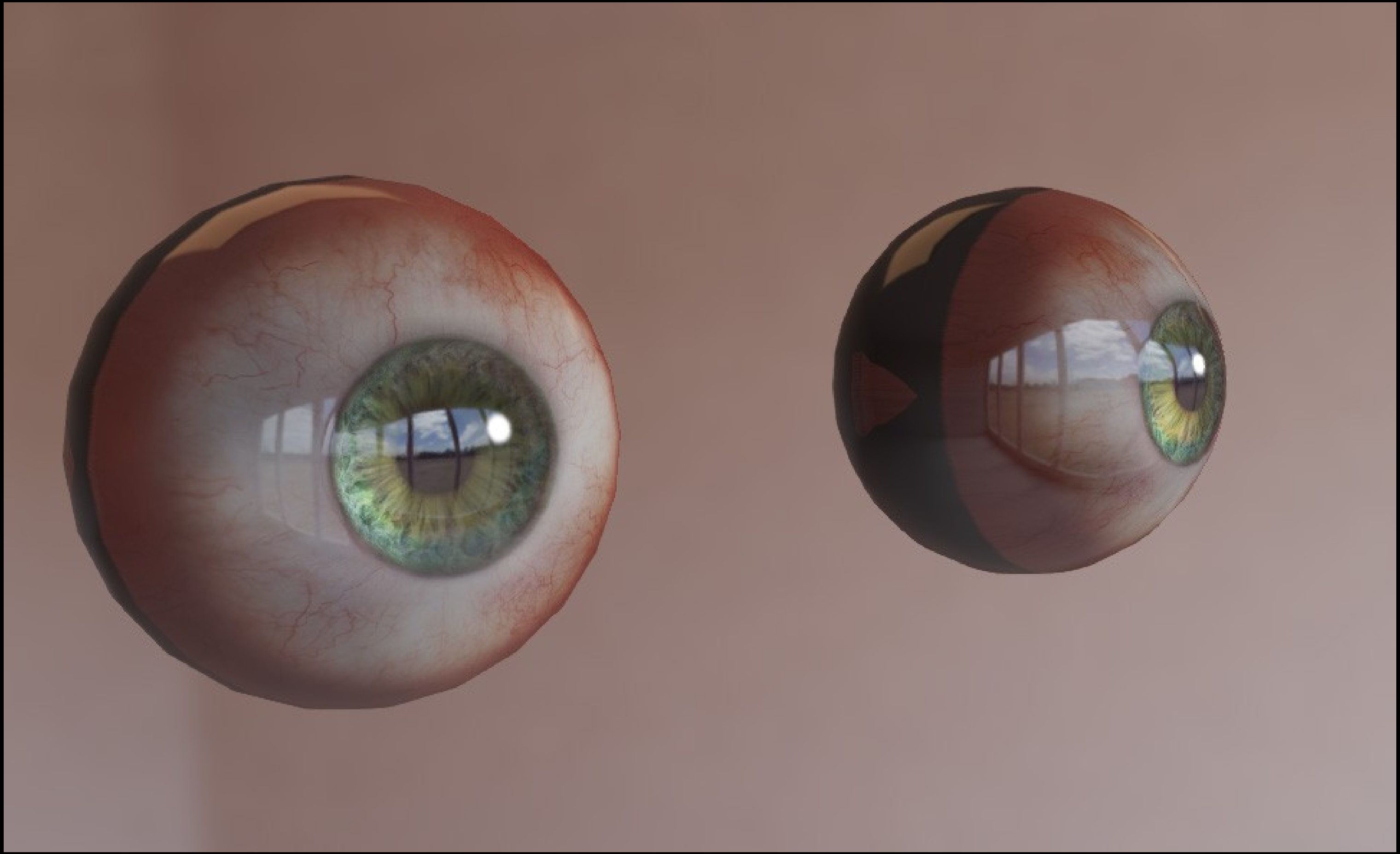


Eye cubemap reflection: use inverted bump map

Eye specular: use bump & inverted bump for the inside & outide



result

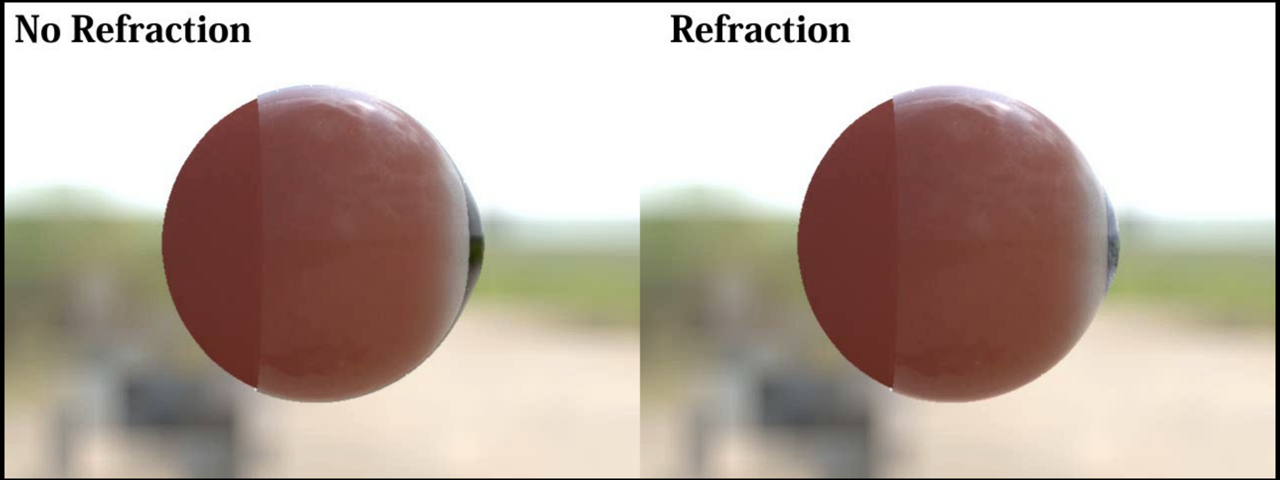


Pseudo Code

```
// Evaluate normals...
Color3 normalMap = sampleTex( normalmap_texture );
Vector3 innerCurve_Normal = EvaluateNormalMap( normalMap );
Vector3 outerLens_Normal  = EvaluateNormalMap( inverse(normalMap) );

// Ambient...
Color3 innerCurve_Ambient = EvaluateGlobalIllum( params, innerCurve_Normal );

// Diffuse & Specular, Loop per light...
Color3 outerLens_Diffuse,  outerLens_Specular  += EvaluateLighting( params, outerLens_Normal );
Color3 innerCurve_Diffuse, innerCurve_Specular += EvaluateLighting( param, innerCurve_Normal );

// Reflection
Vector3 outerLens_ReflectionVector = CalculateReflectionVector( params, outerLens_Normal );
Color3  outerLens_Reflection       = sampleEnv( cubemap, outerLens_ReflectionVector );

// Final Values
Color3 finalAmbient = innerCurve_Ambient;
Color3 finalDiffuse   = innerCurve_Diffuse;
Color3 finalSpecular = innerCurve_Specular + outerLens_Specular + outerLens_Reflection;
```

Eye Refraction

The eyes would look un-natural without it !



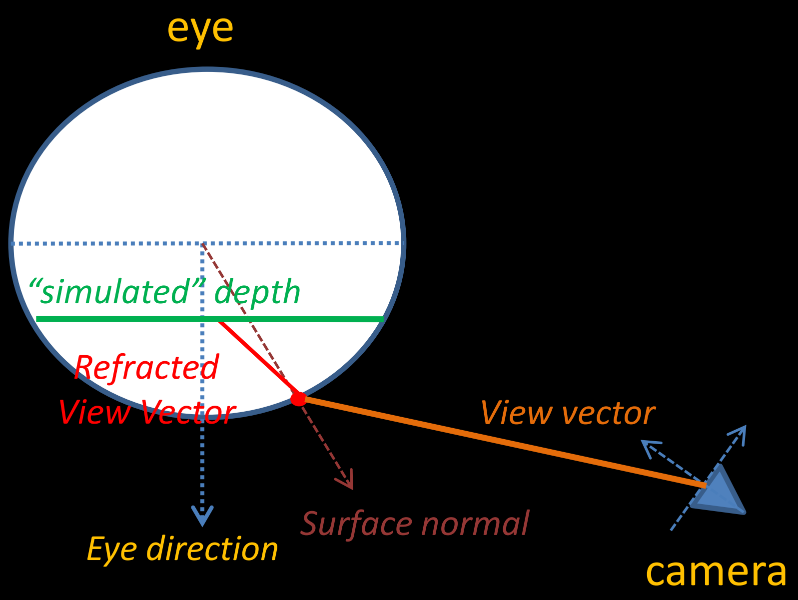
Eye Refraction

Basic Idea:

Parallax mapping

+

refraction



Pseudo Code

```
//local space View & Normal
Float2 TexOffset = lsView.xy;

//get a target depth for parallax
//Could be a constant  or the result of a function
// e.g.  dot(lsNrm, float3(0,0,-1))
SimDepth = DepthFunction(lsNrm);

//get the refraction vector
float3 lsRefracted = Refraction( lsView, lsNrm, refractionIndex );

//offset the UV
TexOffset += (lsView.xy - lsRefracted.xy);
diffuseUV += mask * SimDepth * TexOffset .xy;
```

Agni's Eyes

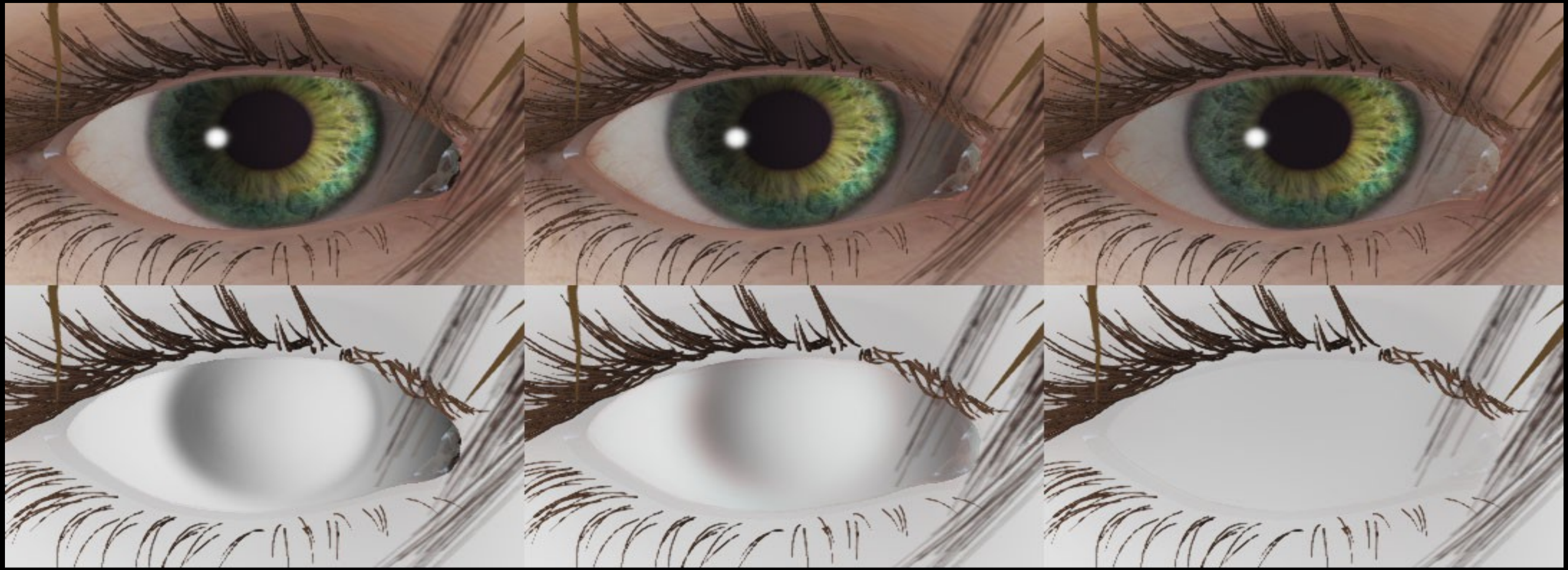
Speculars & reflection : OFF

Refraction OFF

Agni Eyes

Sidoro Eyes

Eye & SS-SSS



SSSBlur = 0.0 SSSBlur = 0.5 SSSBlur = 1.0

Eye: DEMO

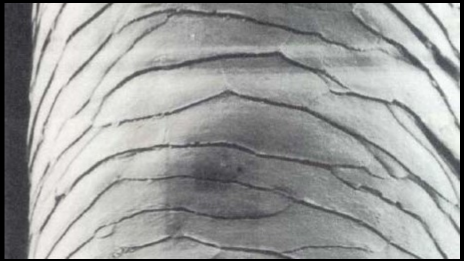
Character Rendering Tech:
Hairs

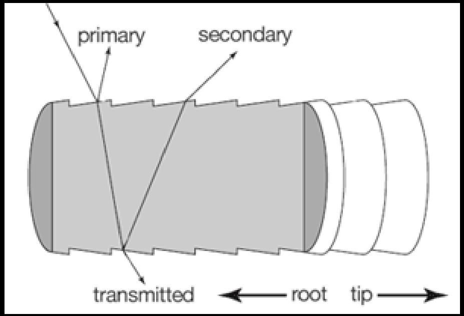
Goal

Match VW Hairs

Specificities of Hairs

- React in a special way to light
  - Transparent & refractive
  - Light scattered in different directions
- Hair are volumetric
  - self shadow & occlusion

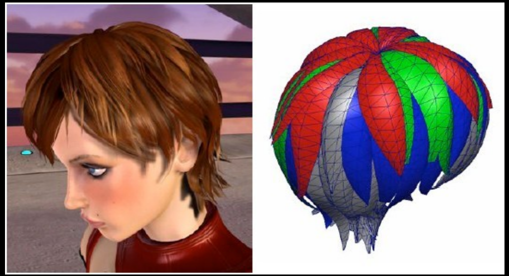




Strategy ?

- The Safe road:
  - Render hairs with a few textured polygons with transparency
  - Used in all present Games
  - Artist experience = good
  - But it would be hard to match VW !

Deus Ex: human revolution



“Practical Real-Time Hair Rendering and Shading”
(Thorsten Scheuermann, ATI Research, Inc.)

Strategy ?

- The new tech ?
  - Generate lot of thin geometry (~.5M lines)
  - Seen in research & stand-alone demos
  - Artist Experience = Zero

"Hair Animation and Rendering in the Nalu Demo" (*Hubert Nguyen & William Donnelly*. *NVIDIA) Gpu Gems 2*

"Real-Time Hair Rendering on the GPU" *(Sarah Tariq, NVidia) Siggraph 2008*

What we did: Use both !

- **Textured polygons**

  Give volume cheaply

  Codename: “Banana-Leaf”

- **Individual strands**

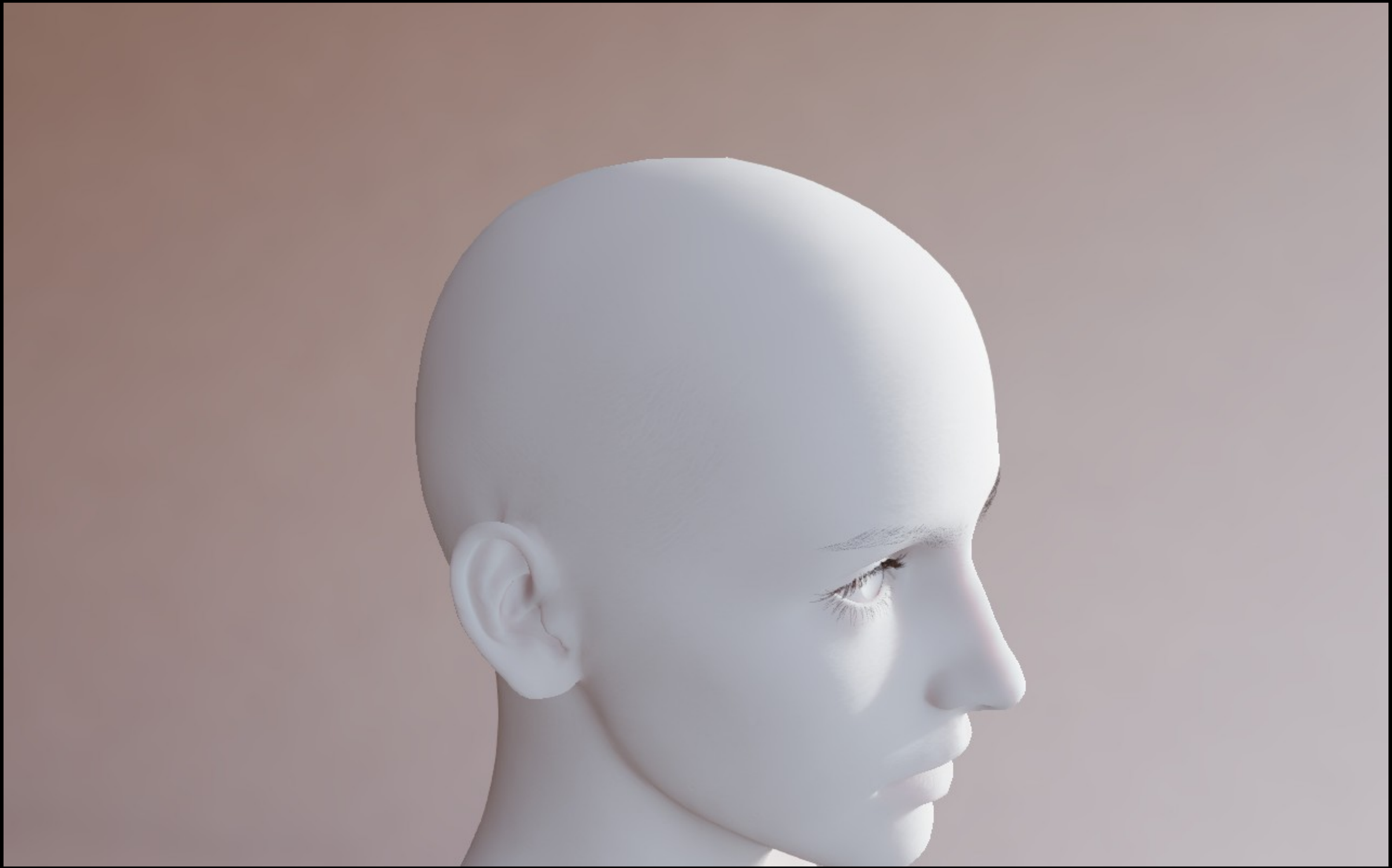
  Created on the fly

  Give variation and quality

  Codename: “Tariq Hair”

possible to mix & match on a case by case basis (e.g. LOD)

No Hair



Adding B-Hairs

Adding T-Hairs

No Hair

Adding B-Hairs

Adding T-Hairs

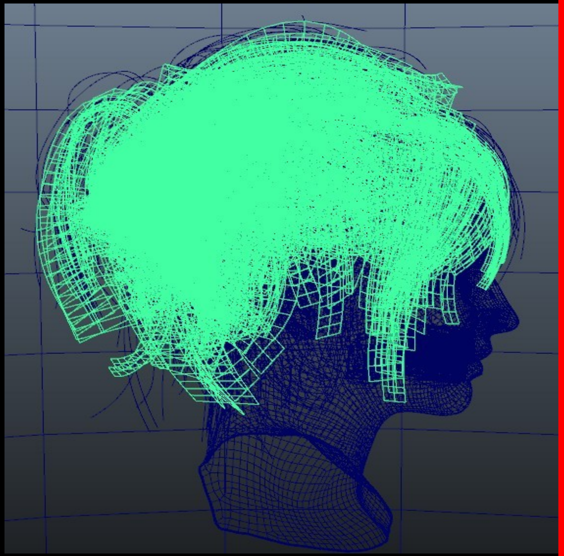
Tech References

- *"Hair Animation and Rendering in the Nalu Demo" (Hubert Nguyen & William Donnelly. NVIDIA) Gpu Gems 2*
- *"Practical Real-Time Hair Rendering and Shading" (Thorsten Scheuermann, ATI Research, Inc.)*
- *"Real-Time Hair Rendering on the GPU" (Sarah Tariq, NVidia) Siggraph 2008*
- *"Realtime Hair Rendering"(Erik Sintorn) www.cse.chalmers.se/edu/year/2011/course/TDA361/Advanced%20Computer%20Graphics/*
- *"Real-Time Hair Simulation and Visualization for Games", M.Sc. Thesis (Henrik Halén & Martin Wester)*

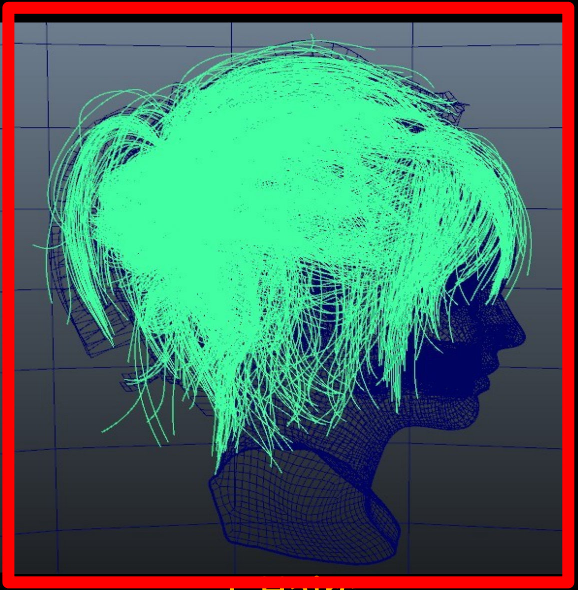
Hair Geometry

Model

- In Maya® software: mesh & nurbcurves



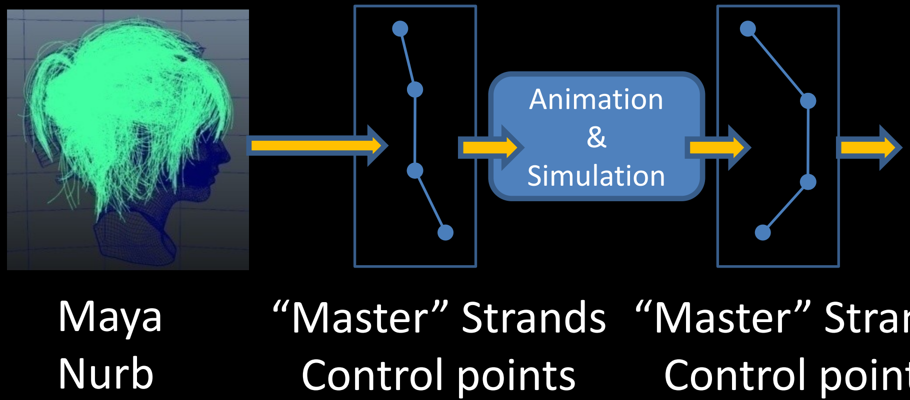
B-Hairs



T-Hairs

“Master Strands”

- “Master Strands” defined by a few control points
- These control points can be animated or simulated

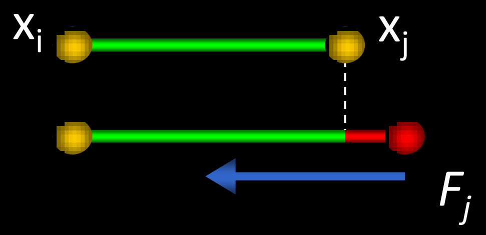


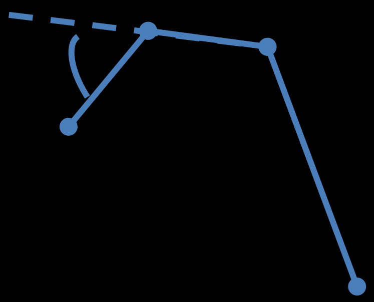
Hair Simulation techniques

- Mass-spring systems
- One dimensional projective equations
- Rigid multi-body serial chain
- Dynamic super-helices

Hair simulation

- Control Points connected by stiff springs
- Bending rigidity ensured at each joint by angular springs
- Simple and easy to implement
- Runs on the GPU



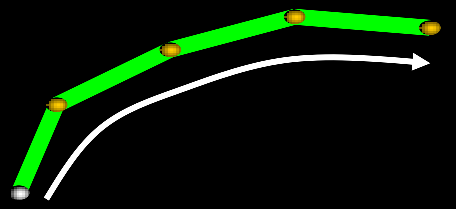


Hair simulation

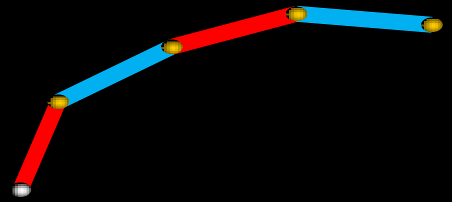
- A step of the simulation:
  - Add external forces
  - Verlet integration
  - Repeat
    - Apply distance constraints
    - Apply angular constraints
    - Apply collision constraints

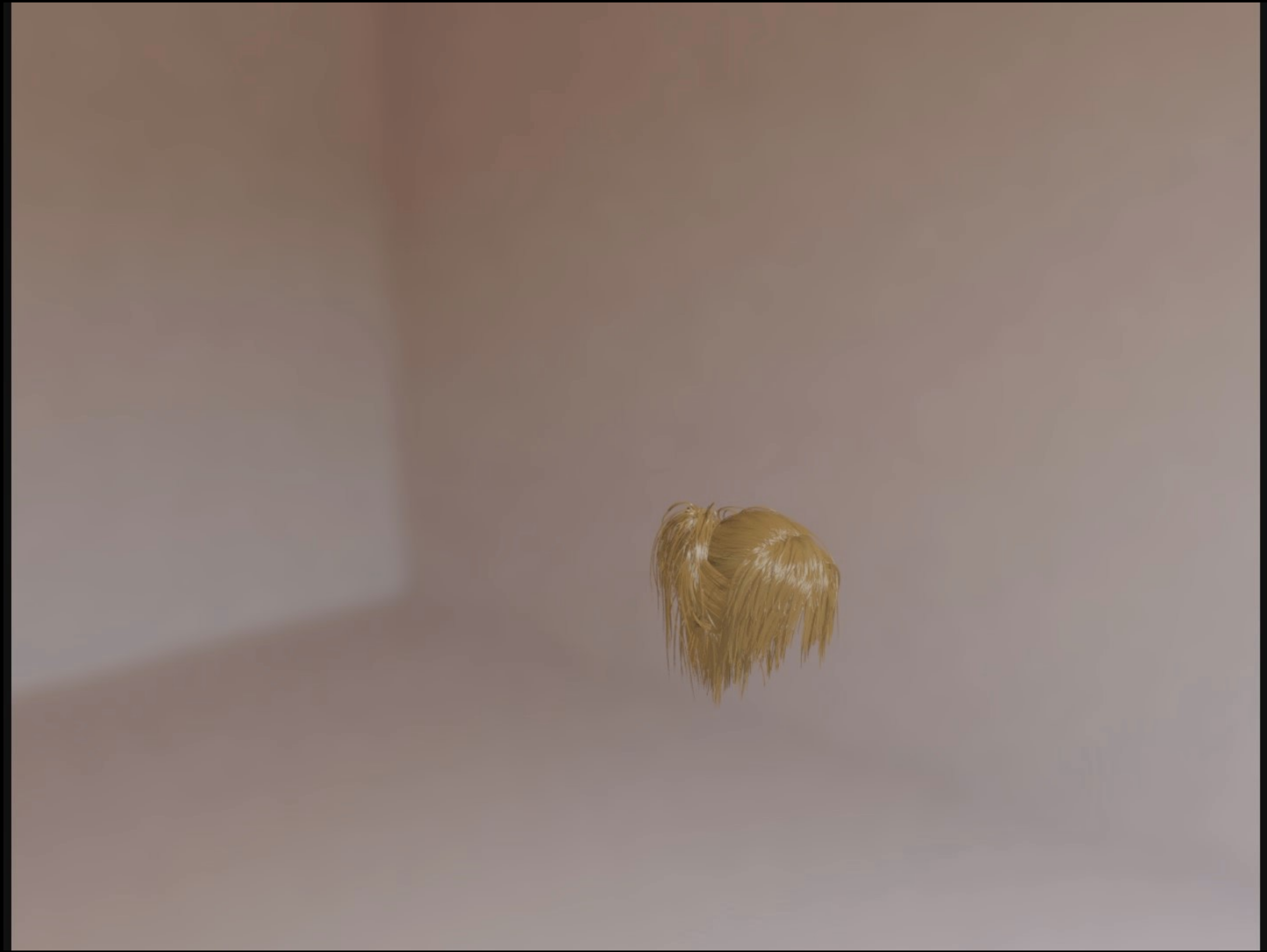
Methods for Constraint Relaxation

- Sequential
  - From Root to Tip
  - Less stretch but less stable



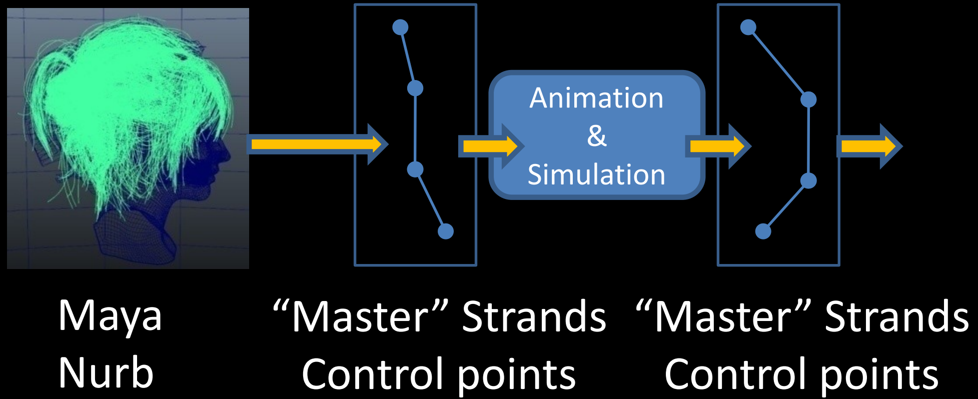
- Parallel
  - Split in independent batches
  - More stable but stretch more





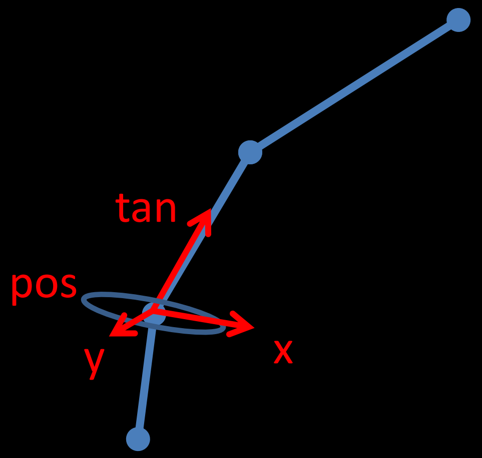
"Master Strands"

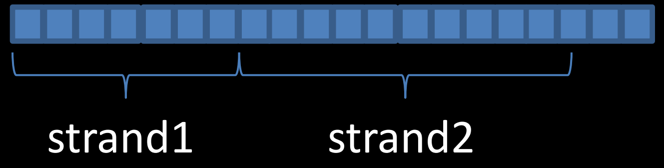
- "Master Strands" defined by a few control points
- These control points can be animated or simulated



Basic Data Encoding

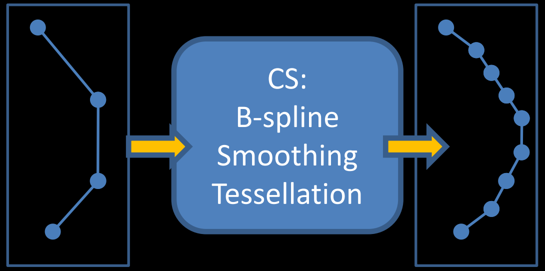
- For each control point:
  - 3D Position, UV, ID, Tangent
  - Longitudinal position [0-1]
  - “Circular” coordinate axis X,Y
- All hair vertices packed into 1 buffer





“Master” Strands smoothing

- Done with a Compute Shader
  - Could certainly use tessellation
- We use B-Splines
  - Do not interpolate end points



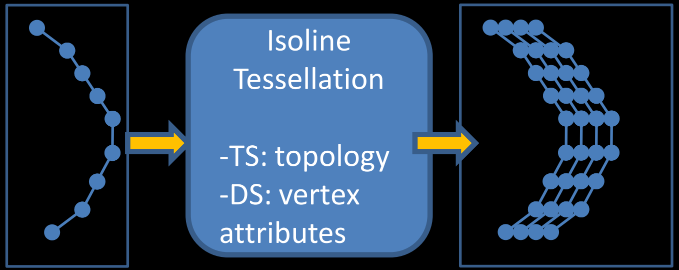
“Master” Strands Control points

tessellated “Master” Strands

$$x(t) = \frac{1}{6}\begin{bmatrix} P_0 & P_1 & P_2 & P_3 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 0 & 4 \\ -3 & 3 & 3 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

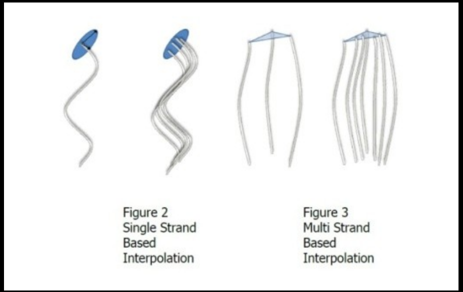
Hair & tessellation

- Use DX11 isoline tessellation to create directly on the GPU new strands from each master strand



Hair & tessellation

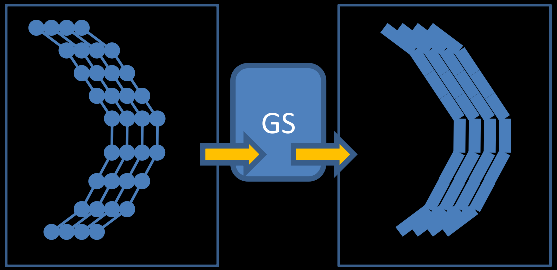
- Several complementary ways to create the strands *(see Sarah Tariq Hair Demo)*
  - – Single strand interpolation
  - – Multi-strand interpolation
- Possible to jitter/twist/ etc… using DS

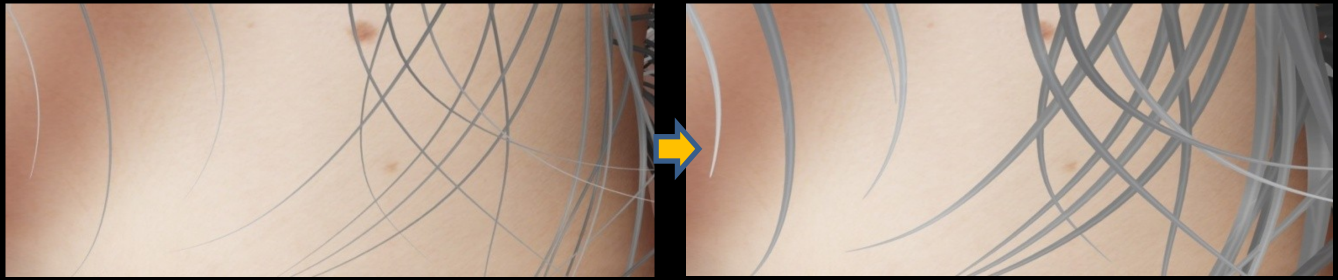


From Sarah Tariq Hair Demo Whitepaper

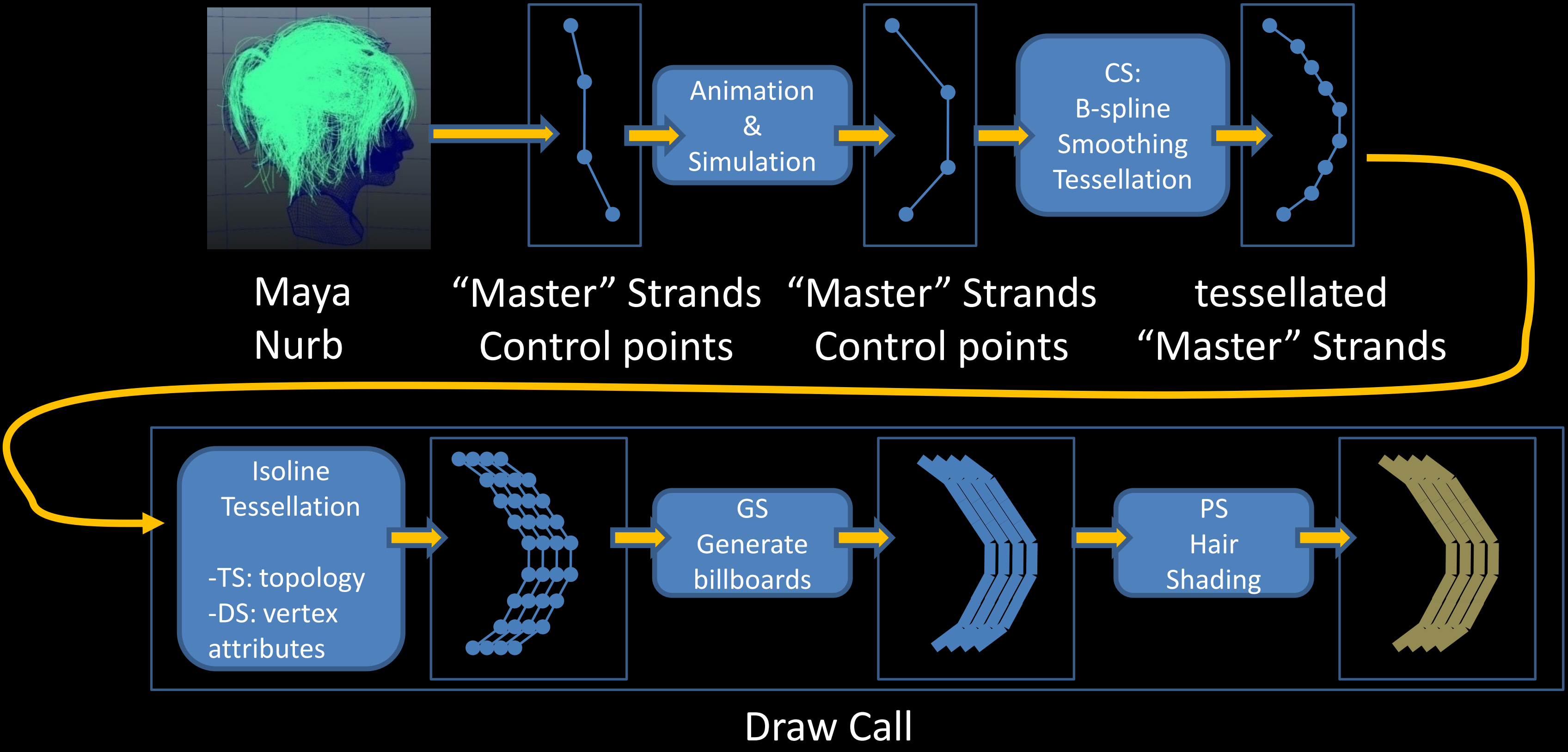
Billboard creation

- Render the hairs as billboards
  - Possible to texture them
  - Possible to change their size
  - Etc..





Pipeline: summary



Impressions

- Extremely flexible !
  - Possible to generate all type of geometry from the same data

Hairs = OFF

Master Strands. No smoothing. No Tessellation

Smoothed master strands

Using tessellation to generate more hairs

More hairs…

Clumping the tips together

Adding some twist

Use bigger clumps

Some other propositions that were refused….

Some other propositions that were refused….

Some other propositions that were refused….

Performance

- Guide Beard: vary tessellation, width & smoothing
- Example: 295 MasterStrands, 1656 control points
  - 3x CS: 295 MasterStrands , 7100 control points
  - 100x tessellation: 29500 strands 710,000 points

CSx3	STRANDS	NB=1	NB=5	NB=10	NB=20	NB=50	NB=100
size=0.1	Hair shadow	0.031	0.084	0.13	0.24	0.52	0.72
	Hair Rendering	0.34	1.3	1.9	3.1	5.6	7
size=1	Hair shadow	0.031	0.085	0.13	0.24	0.57	0.72
	Hair Rendering	0.64	1.9	2	3.1	5.3	6
size=2	Hair shadow	0.031	0.08	0.14	0.28	0.7	0.9
	Hair Rendering	0.82	2	2.3	3.1	5.1	5.7

LOD

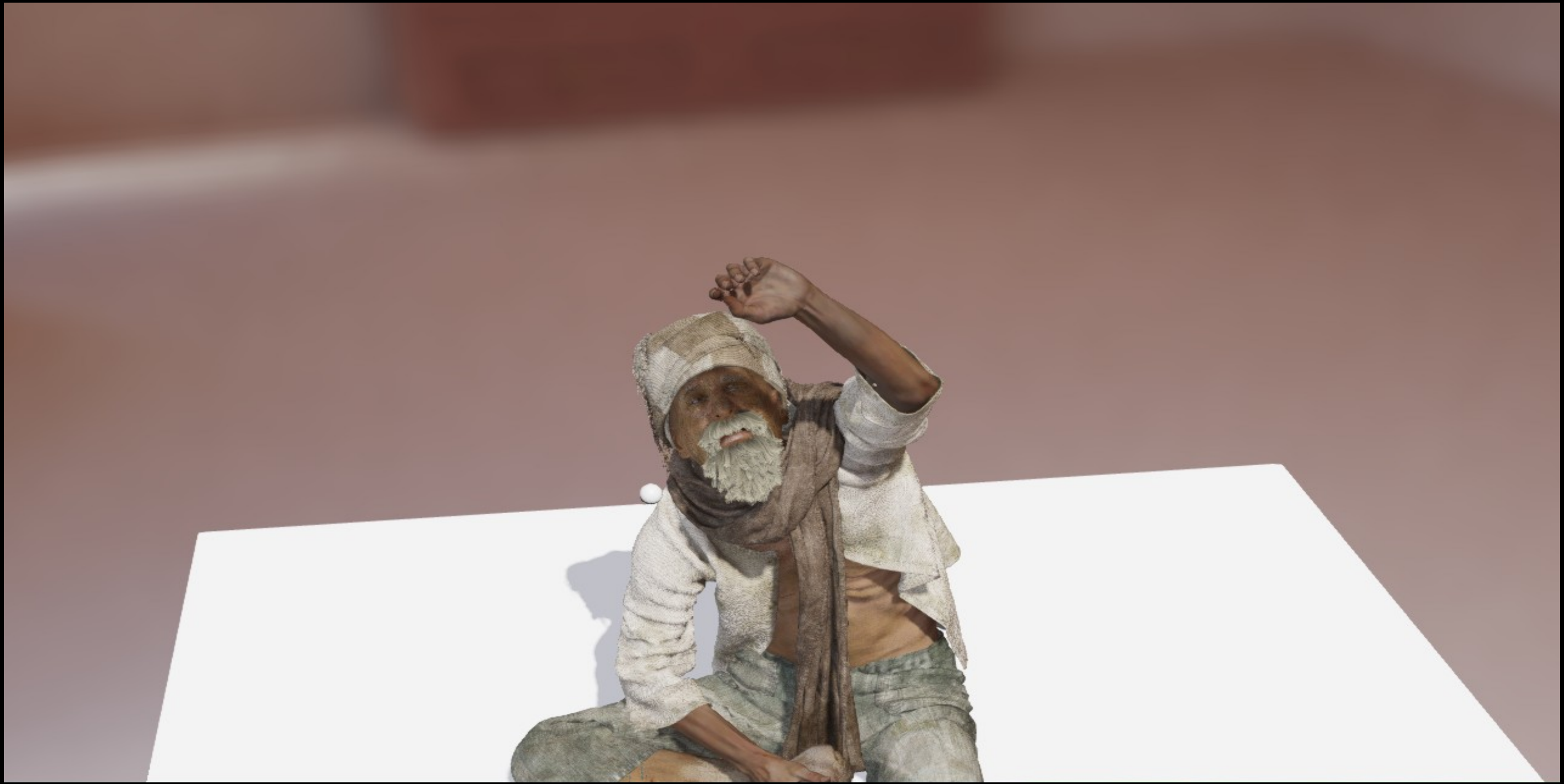
- If hair volume ~= Strand Nb * Strand Size

	Size=0.1 NB=100	Size=1 NB=10
Hair shadow	0.72	0.13
Hair Rendering	7	2

- Then, it may be interesting to use some LOD when models are far enough there is no visual difference

LOD=OFF [4.2 ms]

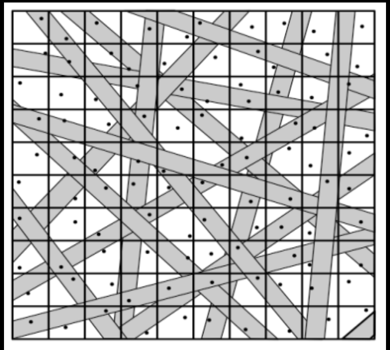
LOD=ON [1.7 ms]



Hair Shading

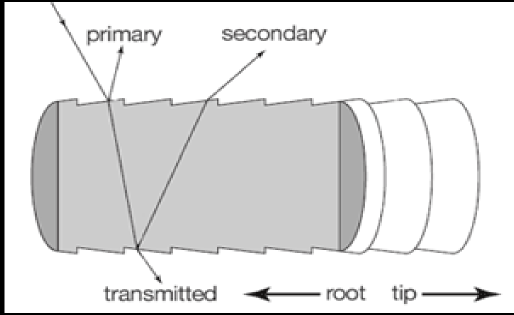
Challenges

- Anisotropic Hair Shading
- B-Hair & T-Hair shading have to match
  – B-Hairs have a surface & normal, T-Hairs don't
- Transparency & Aliasing



Hair scattering models

- Hair & light: light scattered in different directions



- Hair scattering function/models


[Kajiya & Kay 89]


[Marschner 03]

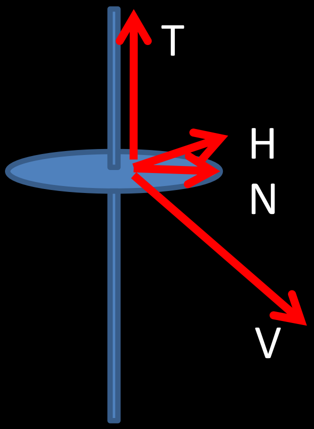

[Zinke & Webber 07]


[Sadeghi 10]


[d'Eon 11]

Kajiya & Kay

- Anisotropic strand lighting model
  - Pretends hair is infinitesimally thin specular cylinder
  - Captures the most obvious specular highlight from hair
  - Diffuse: ~~dot(N.H)~~ sin(T,H) = sqrt(1-dot(T.H)^2)
  - Specular = [T . L * T . E + sin(T,L) sin(T,E)] p



Early experiments

Last Year's OpenConference:

Comments on Kajiya-Kay

- Kajiya term looked too bright without proper self-shadowing
  - Use biased N.L : diffuse = A*N.L + (1-A)
- Artists wanted secondary highlights

Marschner shading

Kajiya

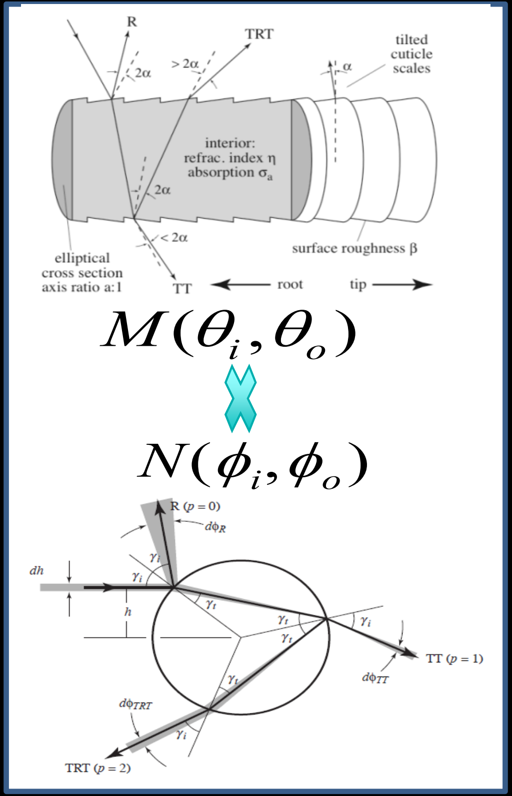
Marschner

Real Hairs

Marshner shading

- Based on measurements of hair properties
- $S = S_R + S_{TT} + S_{TRT}$

- R = Primary specular highlight
  - Shifted towards hair tip
- TRT = Secondary specular highlight
  - Colored
  - Shifted towards hair root
  - Sparkling appearance
- TT = Transmittance



Transmittance

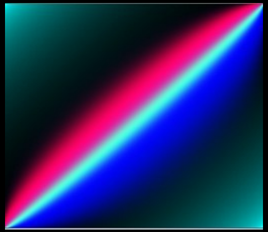
- The amount of light that goes through the hairs

Tech References

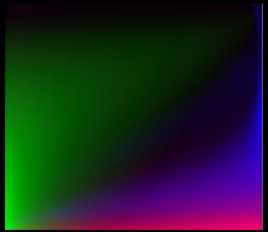
- “Hair Animation and Rendering in the Nalu Demo” (*Hubert Nguyen & William Donnelly*. *NVIDIA) Gpu Gems 2*
- “Practical Real-Time Hair Rendering and Shading” (Thorsten Scheuermann, ATI Research, Inc.)
- *“*Real-Time Hair Rendering on the GPU*” (Sarah Tariq, NVidia) Siggraph 2008*
- “Realtime Hair Rendering”(Erik Sintorn) www.cse.chalmers.se/edu/year/2011/course/TDA361/Advanced%20Computer%20Graphics/
- “Real-Time Hair Simulation and Visualization for Games“, M.Sc. Thesis (Henrik Halén & Martin Wester)

Pseudo Code & Lookup Textures

```
/////// diffuse///////
float  LDotT                      = dot(L, T);
diffuse                           = sqrt(1.0 - LDotT*LDotT);

///////specular///////
//compute the longitudinal angles
float SinThetaI1                  = dot(L, T1);                   float SinThetaI2   = dot(L, T2);
float SinThetaO1                  = dot(V, T1);                   float SinThetaO2  = dot(V, T2);
//Compute the azimuthal angle
float3 lightPerp1                 = L - SinThetaI1 * T1;          float3 lightPerp2  = L - SinThetaI2 * T2;
float3 eyePerp1                   = V - SinThetaO1 * T1;          float3 eyePerp2    = V - SinThetaO2 * T2;
float CosPhiD1 = dot(eyePerp1, lightPerp1) / sqrt( dot(eyePerp1, eyePerp1) * dot(lightPerp1, lightPerp1) );
float CosPhiD2 = dot(eyePerp2, lightPerp2) / sqrt( dot(eyePerp2, eyePerp2) * dot(lightPerp2, lightPerp2) );

//R, TT & TRT speculars
float4 m1 = SAMPLE( MARSCHNER_M, float2((SinThetaI1*0.5 + 0.5), (SinThetaO1*0.5 +0.5)));
float4 m2 = SAMPLE( MARSCHNER_M, float2((SinThetaI2*0.5 + 0.5), (SinThetaO2*0.5 +0.5)));
float4 n1 = SAMPLE( MARSCHNER_N, float2((CosPhiD1*0.5 +0.5), m1.a) ); // CosThetaD1 = m1.a
float4 n2 = SAMPLE( MARSCHNER_N, float2((CosPhiD2*0.5 +0.5), m2.a) ); // CosThetaD2 = m2.a
float spec_r                      =  KMarschnerR   *  pow( abs(m1.r * n1.r), KMarschnerSpecPowR);
float spec_tt                     =  KMarschnerTT  *  pow( abs(m1.g * n1.g), KMarschnerSpecPowTT);
float spec_trt                    =  KMarschnerTRT *  pow( abs(m2.b * n2.b), KMarschnerSpecPowTRT);
specular = spec_r +  spec_tt  * AbsorptionCol + spec_trt * AbsorptionCol;
```



MarschnerM



MarschnerN

Highlights & Tangents

- All Specular calculations depend on Tangent
  - Calculated on the fly from point positions
  - Possible to add jitter to add variation to shading
  - Possible to move tangents to shift the specular
    - Positive shift moves highlight towards tip
    - Negative shift moves highlight towards root

No specular

First Hightlight

Second Hightlight

Specular shift

Adding noise in shift

Adding cubemap, colored light

Changing absorption of light on 2nd hightlight

CubeMap & SH

- To keep a match between B-Hairs & T-Hairs, we decided to use “fake” normals
- Nrm1 = cross(T, cross(V, T))
  - Gives a lot of variation per hair
- Nrm2 = normalize(Pos - BBoxCenter)
  - Gives a sense of volume to the global hair system
- In practice: Blend both !

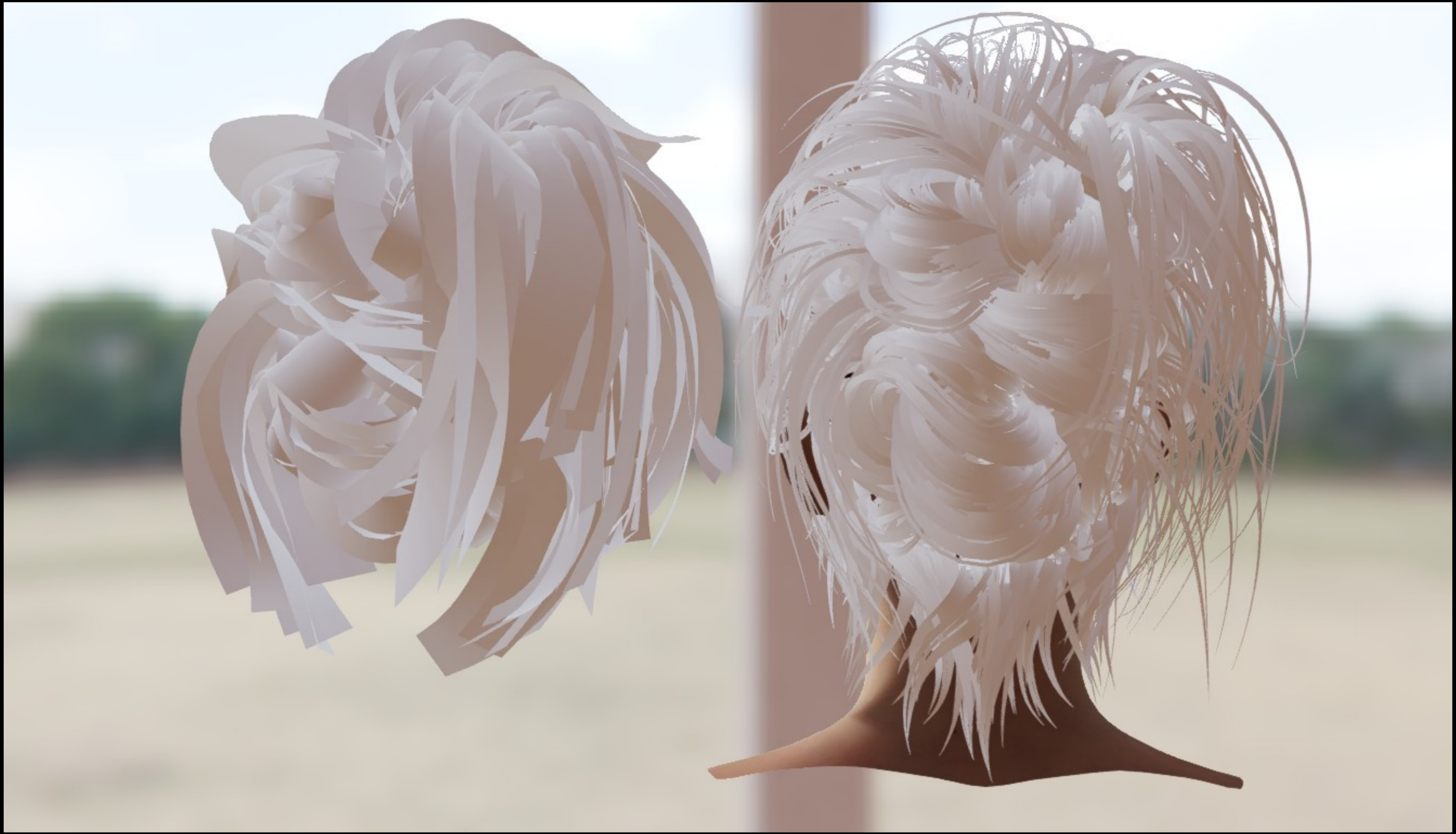
cross(T, cross(V, T))

normalize(Pos - BBoxCenter)

Blended fake normals

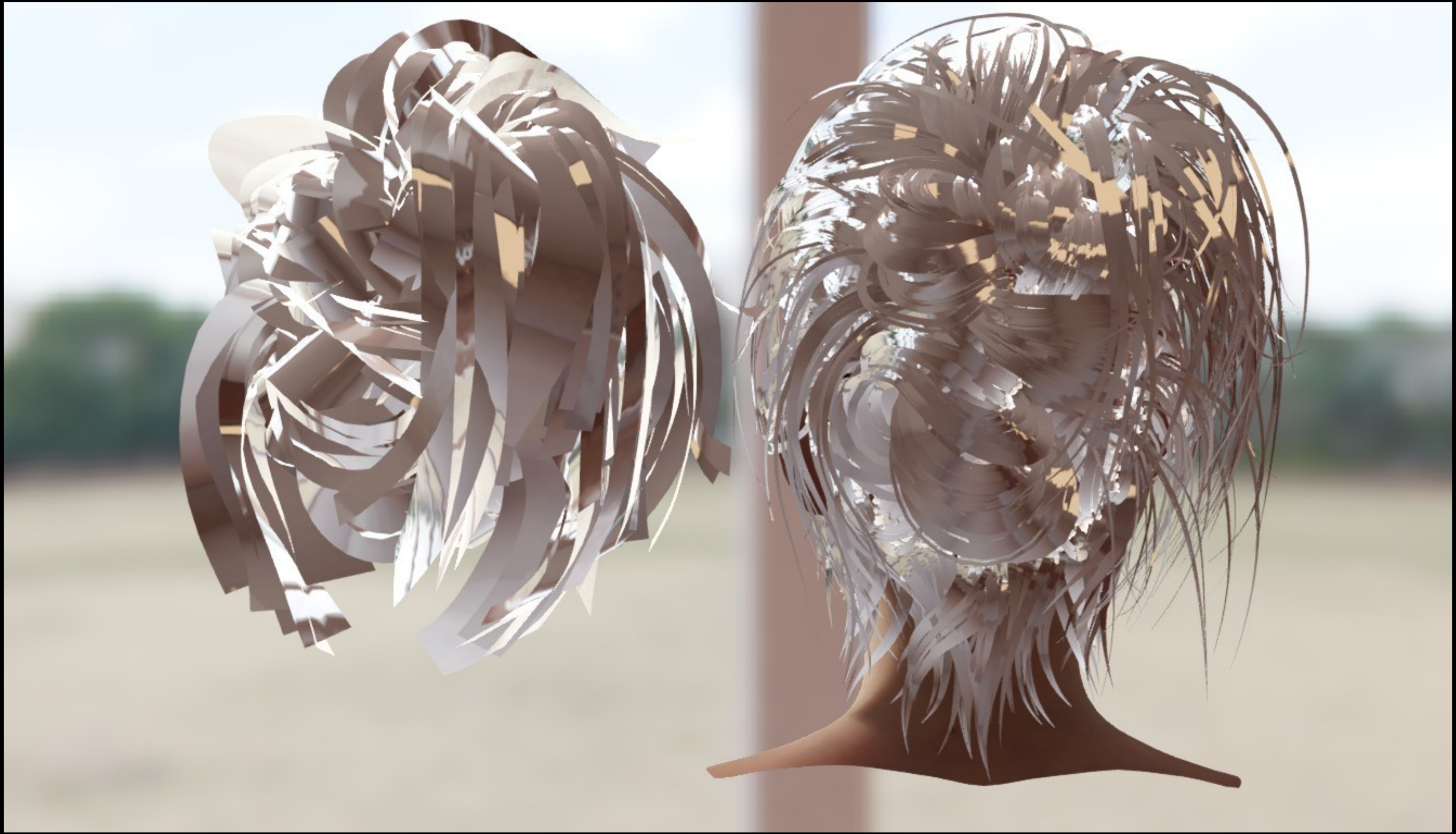
Light from Irradiance Volume

Light from Irradiance Volume



Reflection from Cubemap

Reflection from Cubemap



Hair shadowing

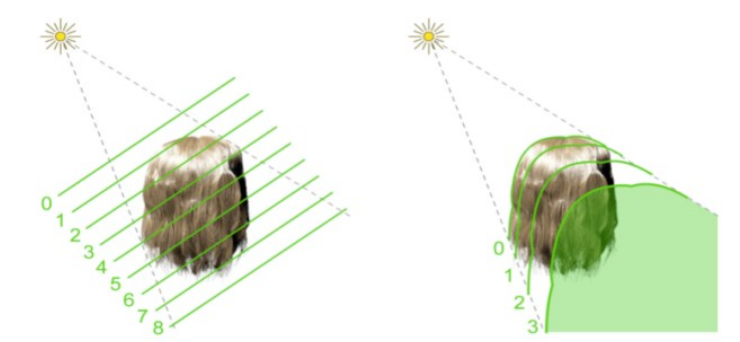
- Particular Challenges:
  - Volume with high frequency data
  - Hair are transparent

Many methods

- deep shadow maps [Lokovic & Veach 2000]
- opacity shadow maps [Kim & Neumann 2001]
- density clustering [Mertens et al. 2004]
- deep opacity maps [Yuksel & Keyser 2008]
- occupancy maps [Sintorn & Assarson 2009]

Shadow Tech

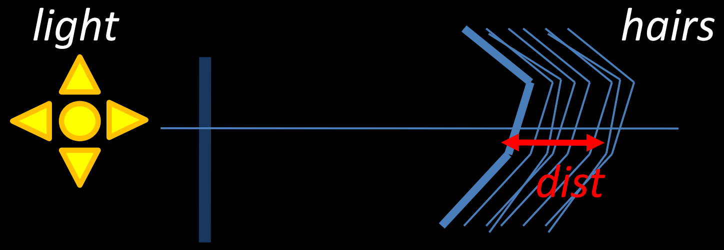
- Opacity Map–based techs:
  - Sample opacity at regular intervals
  - Render hair once for each slice
  - Requires a lot of slices !
  - Expensive !



(a) Opacity Shadow Maps (b) Deep Opacity Maps

Self-Shadow: Our choice

- Use simple PCF shadow
- But, weigh the sample depending on the distance with the occluder
- e.g.: exp(-a * dist(occluder, receiver))



Hair Shadow = OFF

Some Absorption

more Absorption

Geometry used for shadow mapping

Occluder-Strands can be of different number and size than rendered strands !

Fourier Opacity Map

- "Fourier Opacity Mapping" [Jansen and Bavoil 2010]
- We use it for smoke shadow
- Concept:
  - Formulate the absorption function as a Fourier series
  - Render the Fourier coefficients to textures

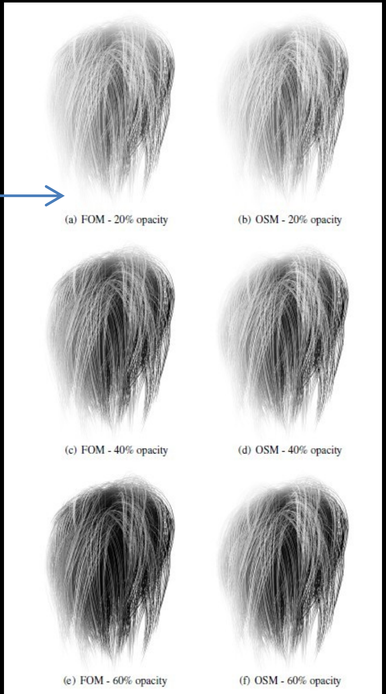


Figure 9: *Comparison of Fourier Opacity Mapping (FOM) and Opacity Shadow Maps (OSM) on hair. For both FOM and OSM, the opacity map had a 512x512 resolution and was pre-filtered with a 33x33 box filter to reduce aliasing and soften the shadows. The OSM images are using 32 slices and the FOM images are using 5 coefficients per texel. This hair model is a built-in example hair system from Maya®Hair 8.5.)*

No shadow

Fourier Opacity Map - Shadows

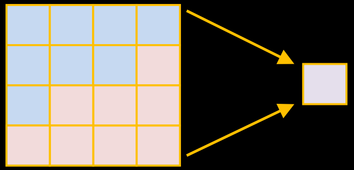
Our PCF-based Shadow

Fourier Opacity Map

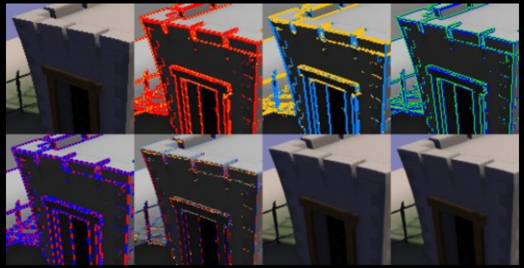
- Impressions:
  - Interesting concept
  - Worked well in some cases
  - Looks dirty in more cases
  - Much more expensive than our fake method

Anti-Aliasing

- Multisample Anti-Aliasing (MSAA)
  - saves pixel shader processing, but all the depth operations are still performed at sample granularity



- Fast Approximate Anti-Aliasing (FXAA)
  - Quick
  - Work with deferred renderers
  - Source Code on Nvidia site



- Depth Of Field

No AA

MSAA x2

MSAA x4

MSSAAx8

FXAA: On

DOF: ON

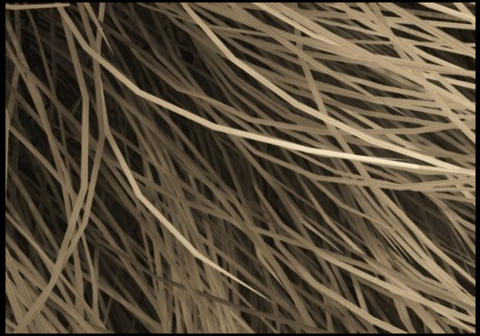
Transparency

- Has to be dealt with for B-Hairs
- Alpha blending? .. requires sorting or depth peeling
- Order Independent Transparency ? … too costly

- Our choice: alpha to coverage
  (=converts the alpha component output from the pixel shader to a coverage mask)

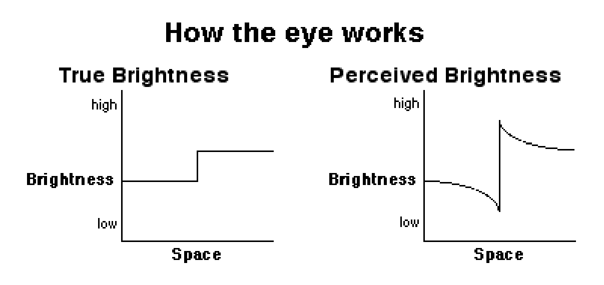
Add Variation & Contrast

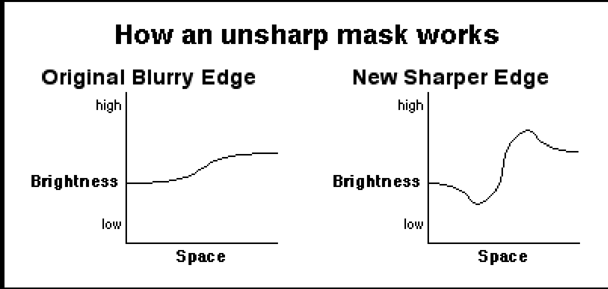
- Random color change based on ID
- Darken according to curvature
- Darken according to distance to root
- Texture the inside of each strand
- Texture the strands
- Etc..



Add Variation & Contrast

- "Unsharp Masking"
  - Kind of local, simple SSAO
  - Only use difference in depth
  - Works well on high frequency structures like hairs





No post-process

With Unsharp masking

Hair System : Conclusion

Expensive

but it has

Flexibility & Quality

it should certainly be seen on PC
or next-gen games

Hairs were used on animals too..

Hairs were used on animals too..

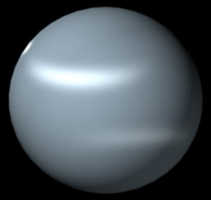
Hair: DEMO

Others:
Clothes

Anisotropic shading

The specular highlight of an anisotropic surface material is different depending on your viewing angle.

isotropic specular



anisotropic specular

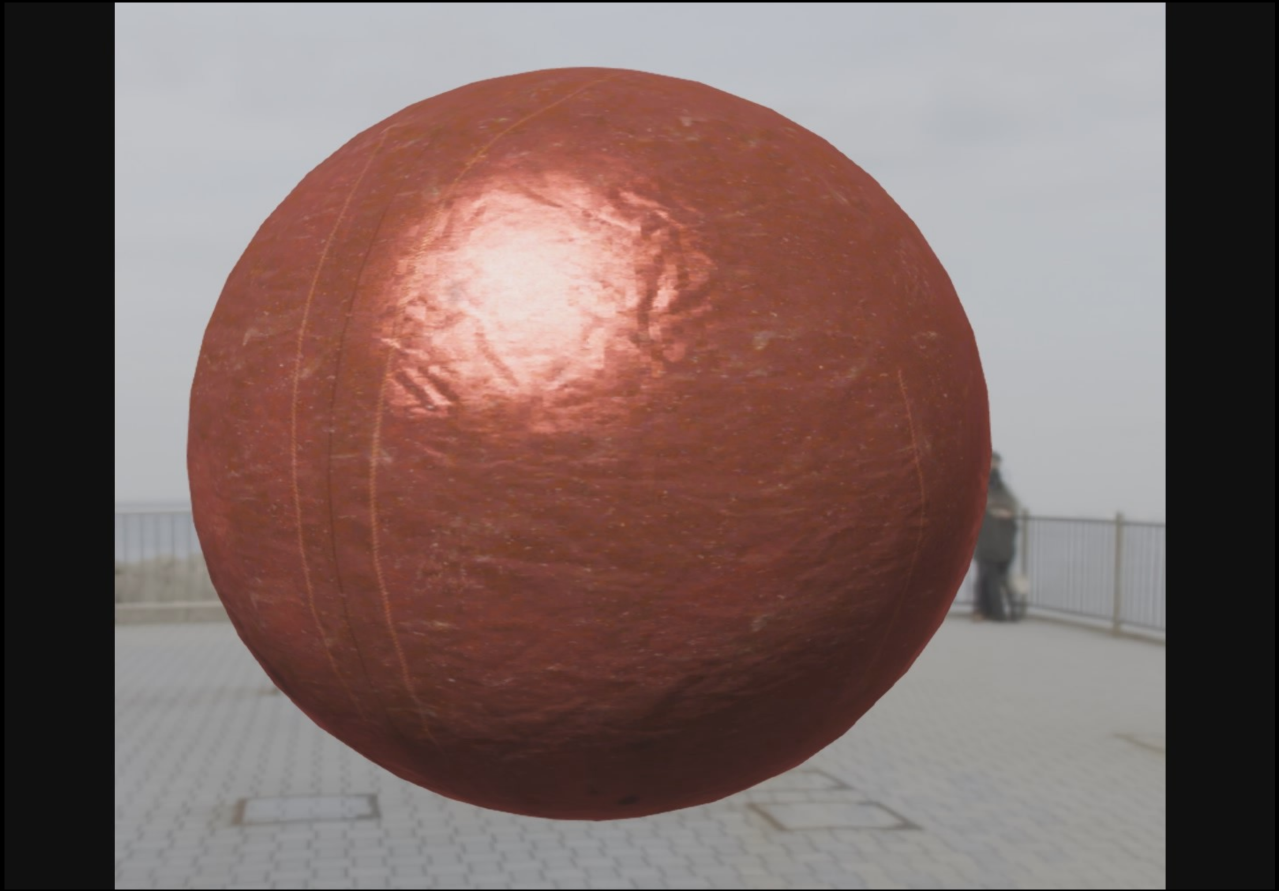
Pseudo Code

```
//isotropic specular
float3 H   = normalize(V + L);
float isotropicSpecular = dot(N, H);
```

```
//anisotropic specular
float3 H   = normalize(V + L);
float HdotA = dot(H, i_anisoAngle);
float anisotropicSpecular = 1.0-HdotA*HdotA;
```

Changing anisotropy on Agni's dress

Isotropic / Anisotropic Specular

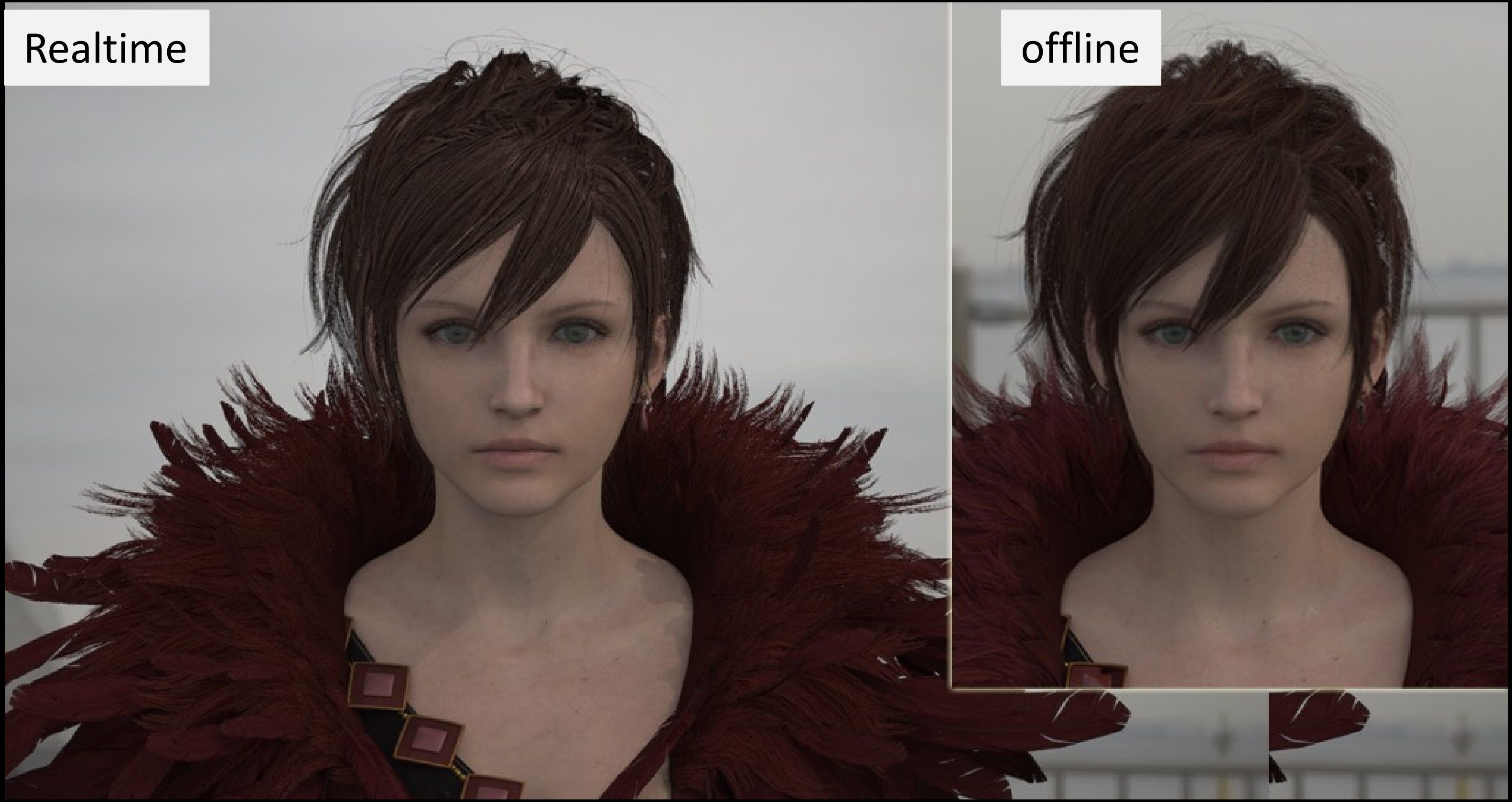


Isotropic specular

Anisotropic specular

Comparison
Offline / RealTime

Real time vs Offline



RealTime

Character Rendering: DEMO

Volumetric Light Effect:

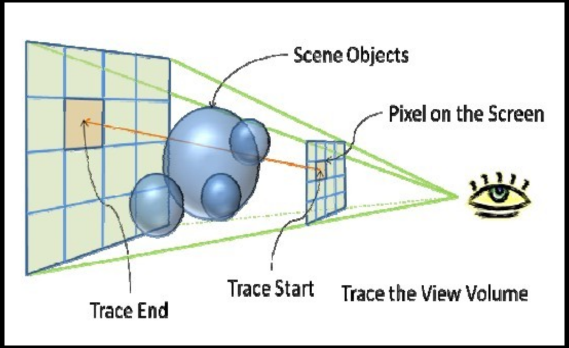
Torchlight

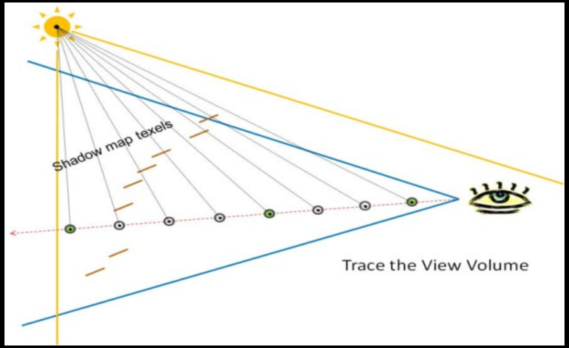
Torchlight

- Goal:
  - produce the volumetric look of light scattering
- Other names:
  - God ray
  - Volumetric light
  - Light shaft..

Basic Idea

- Similar to Nvidia "Volume Light" whitepaper
  - Use a light space shadow map
  - For each pixel, cast a ray and do ray marching
  - For each sample check if it is lit by the light or not





From Nvidia whitepaper
http://developer.download.nvidia.com/SDK/10.5/direct3d/Source/VolumeLight/doc/VolumeLight.pdf

Problems

- Expensive to do ray marching for each pixel
- Use coarser resolution !
- Then Aliasing appears on edges!

Pipeline

- Use MSAAx4 buffer
  - Size: 1/4x1/4 = 1/16 of Backbuffer
- 1. Edge detection
  - Flag pixels that are close to edges (stencil)
- 2. Non-Edge pixels
  - Ray marching. PS executed per fragment
- 3. Edge Pixels
  - Ray marching. PS executed per sample
- 4. Finalize Result (Blur)

Performance

- No fixed cost
  - Depends on sample density
  - Depends on edge ratio
- In practice
  - Sample NB = 32~64 / ray
  - Edge Ratio 10 ~ 20%
  - Cost 4~6 ms
  - Could be further improved (mipmaps)

Volumetric Light Effect:

Fog

Fog

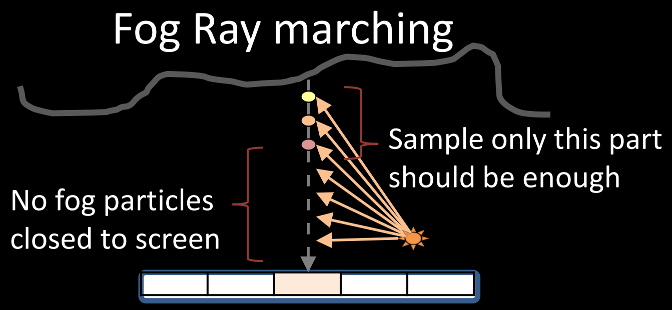
- Light scattering & absorption
- Give sense of depth

Basic Fog Equation

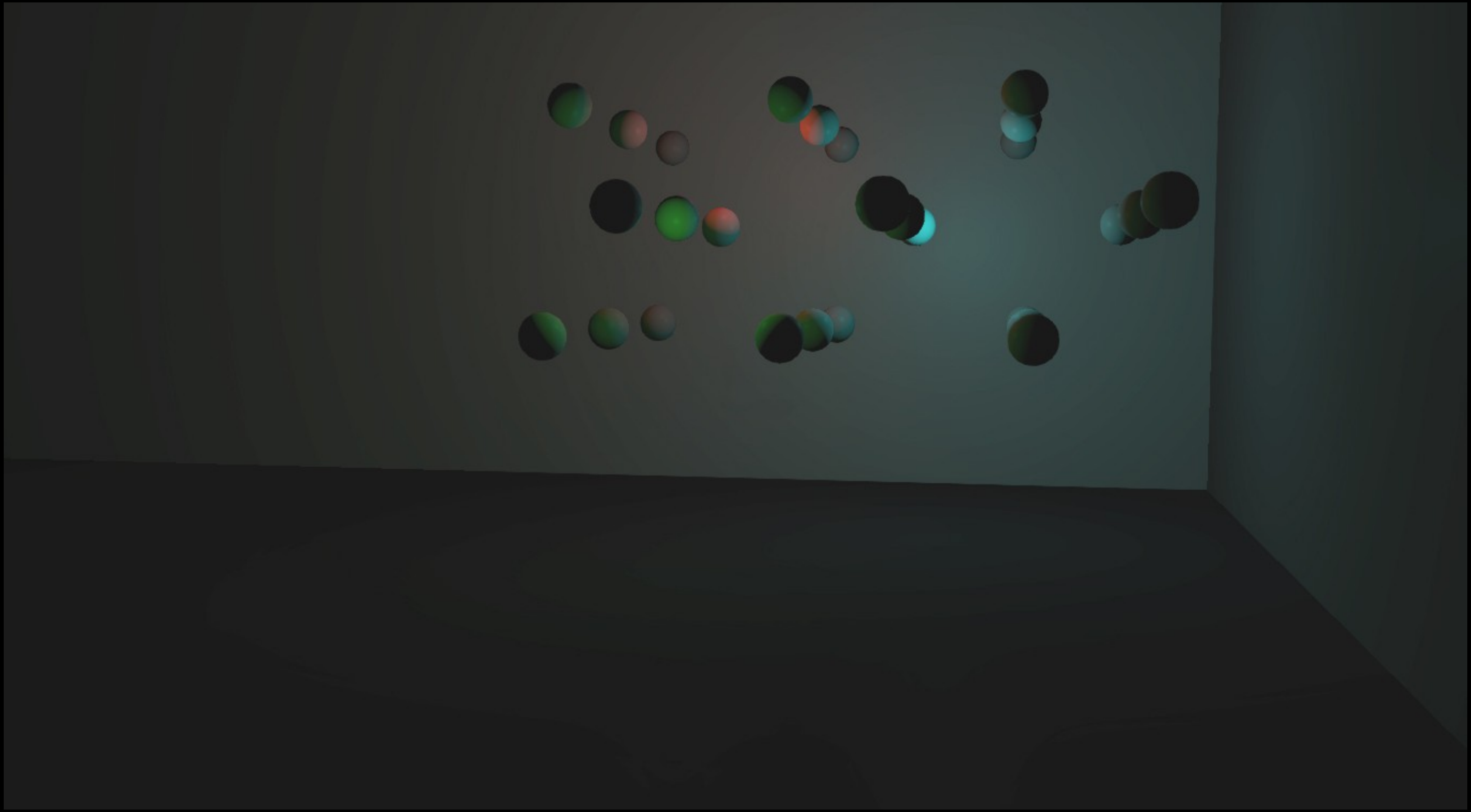
$$C_{final} = f \times C_{current} + (1 - f) \times C_{fog}$$
$$f = e^{-(density \times depthFade \times heightFade \times noise)^2}$$

Fog & Light

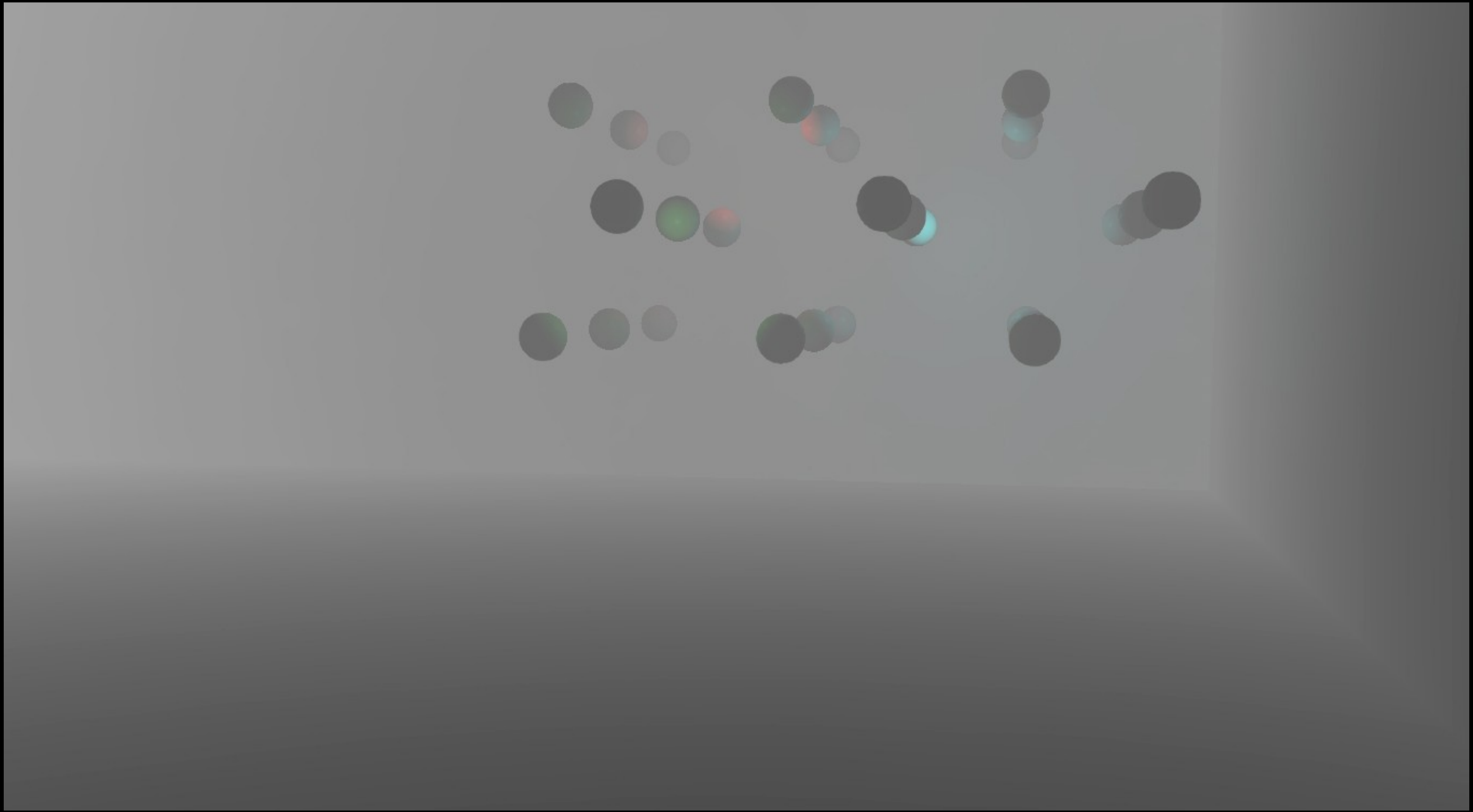
- If we want to add interaction with light
  - Use ray marching
  - Quality = lot of samples = Slow !
  - Possible to reduce sampling area



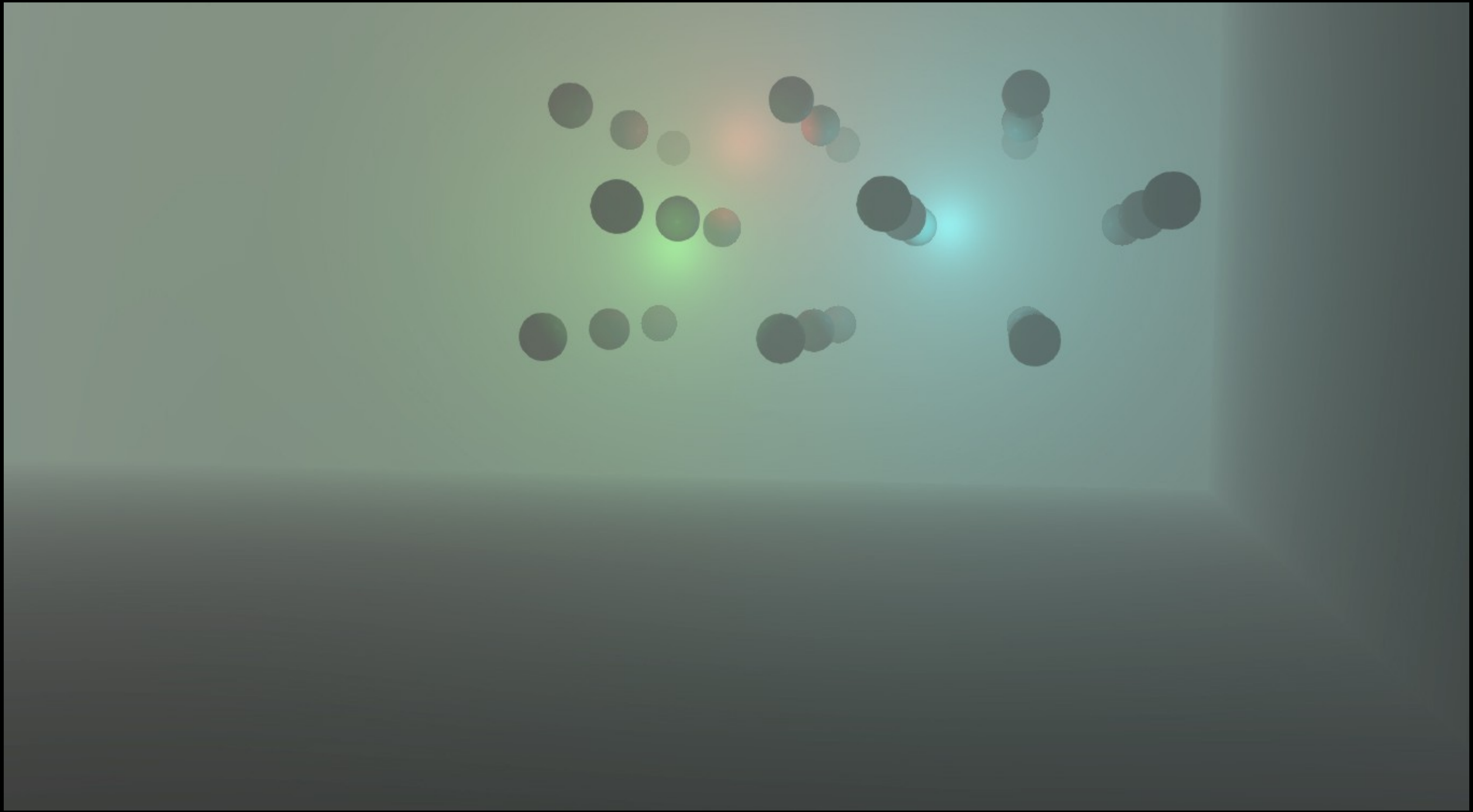
Fog = off



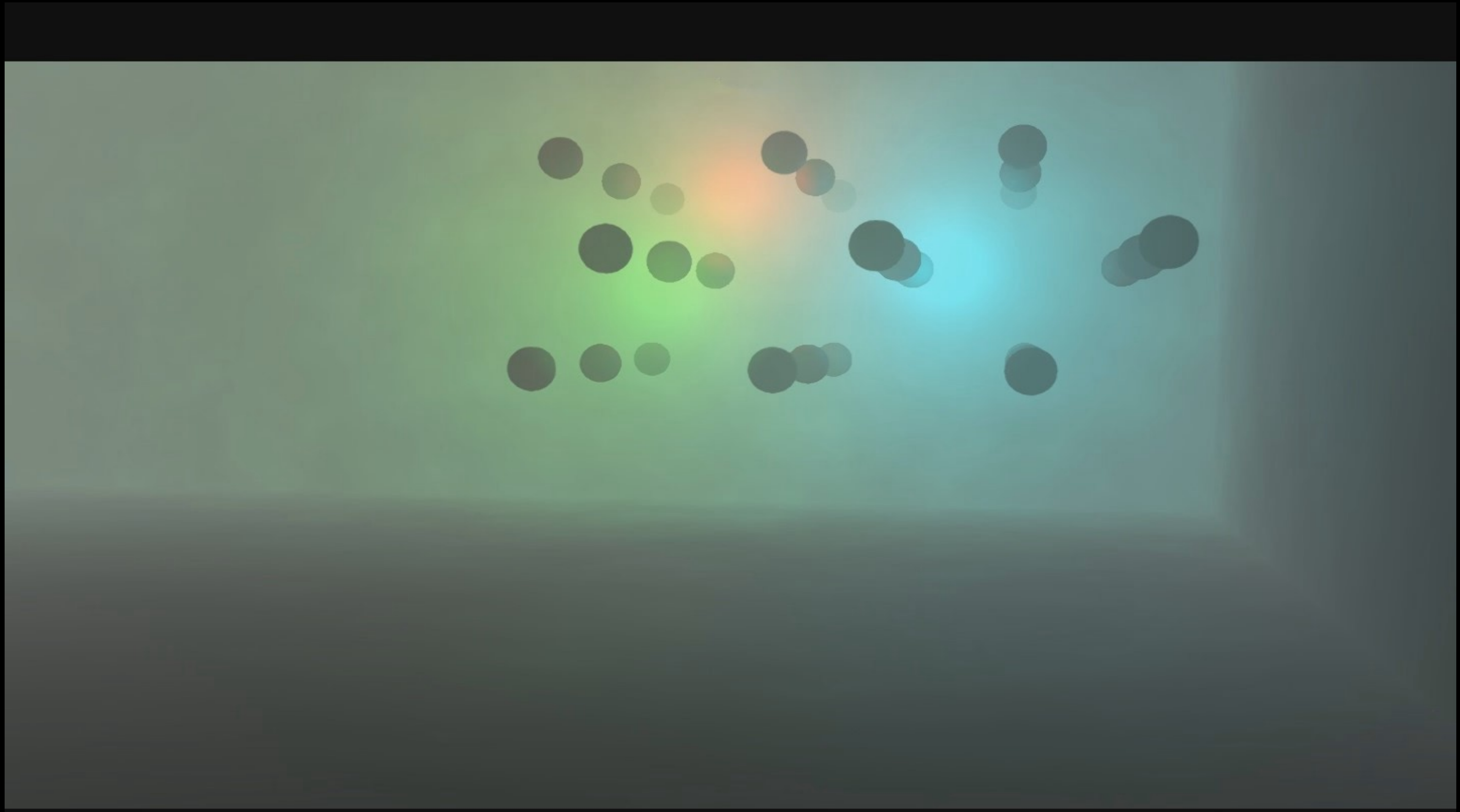
Fog = on [0.2 ms]



Add light interaction [0.5ms]



Add noise [2.6ms]



Fog = off

Fog = on

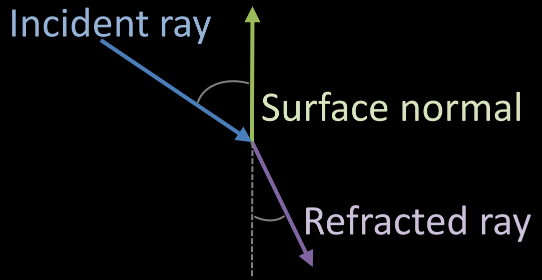
Special Effect:
Refraction

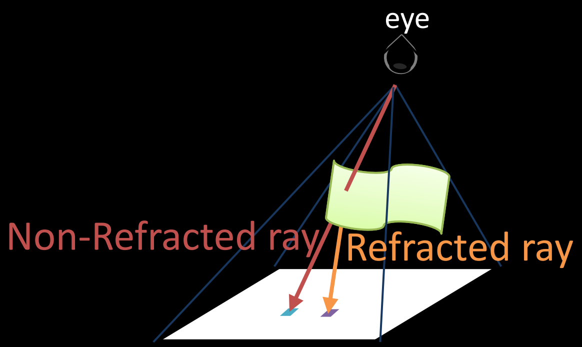
Goal

Make glass look like glass

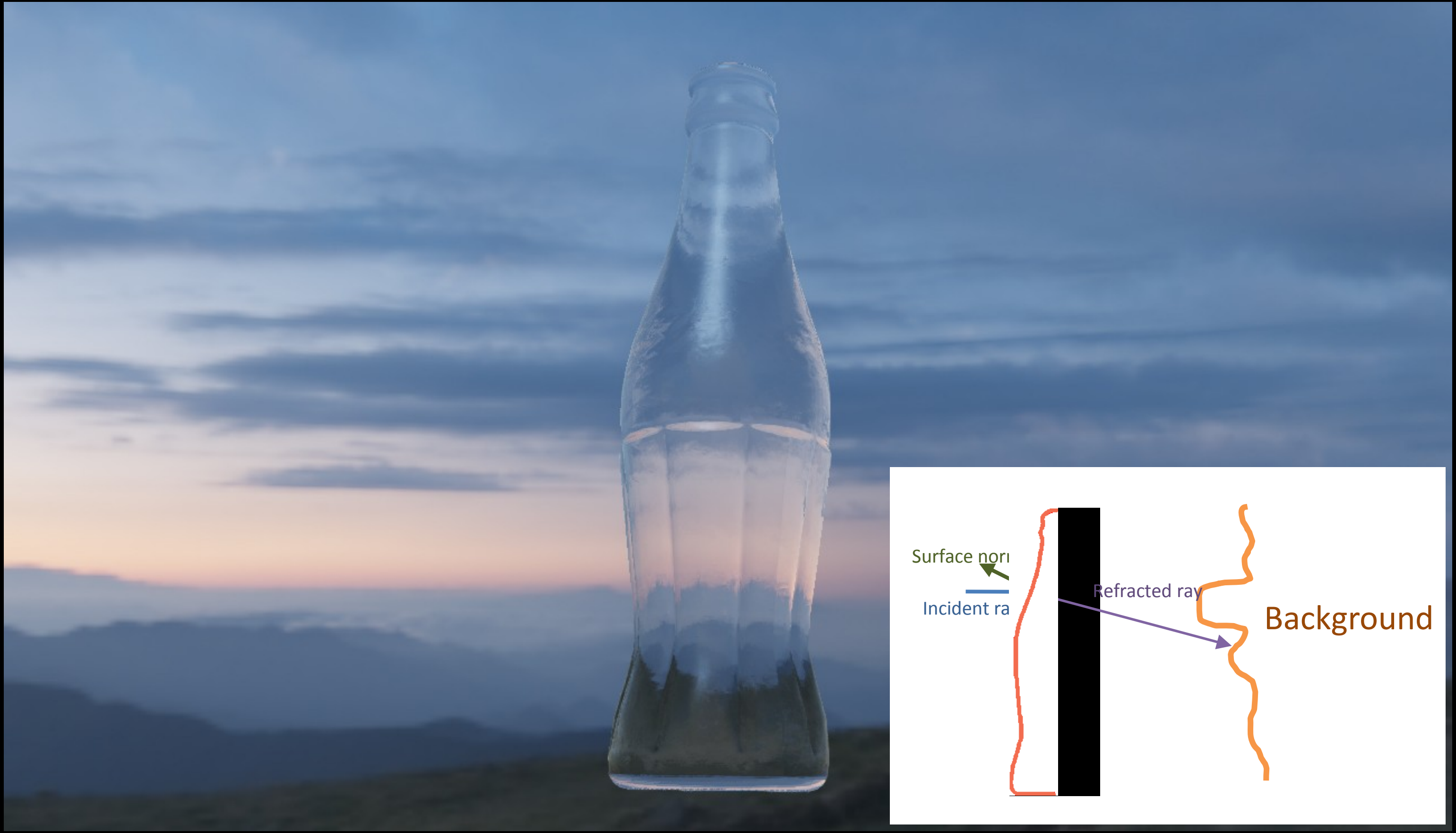
Basic Refraction Rendering

- Screen-space Technique
  - Render the scene geometries into a texture
  - Compute refracted ray
  - Intersect refracted ray with background
  - Get color at the intersection from the texture
- Based on "Interactive Image-Space Refraction of Nearby Geometry,"[Wayman 2005]

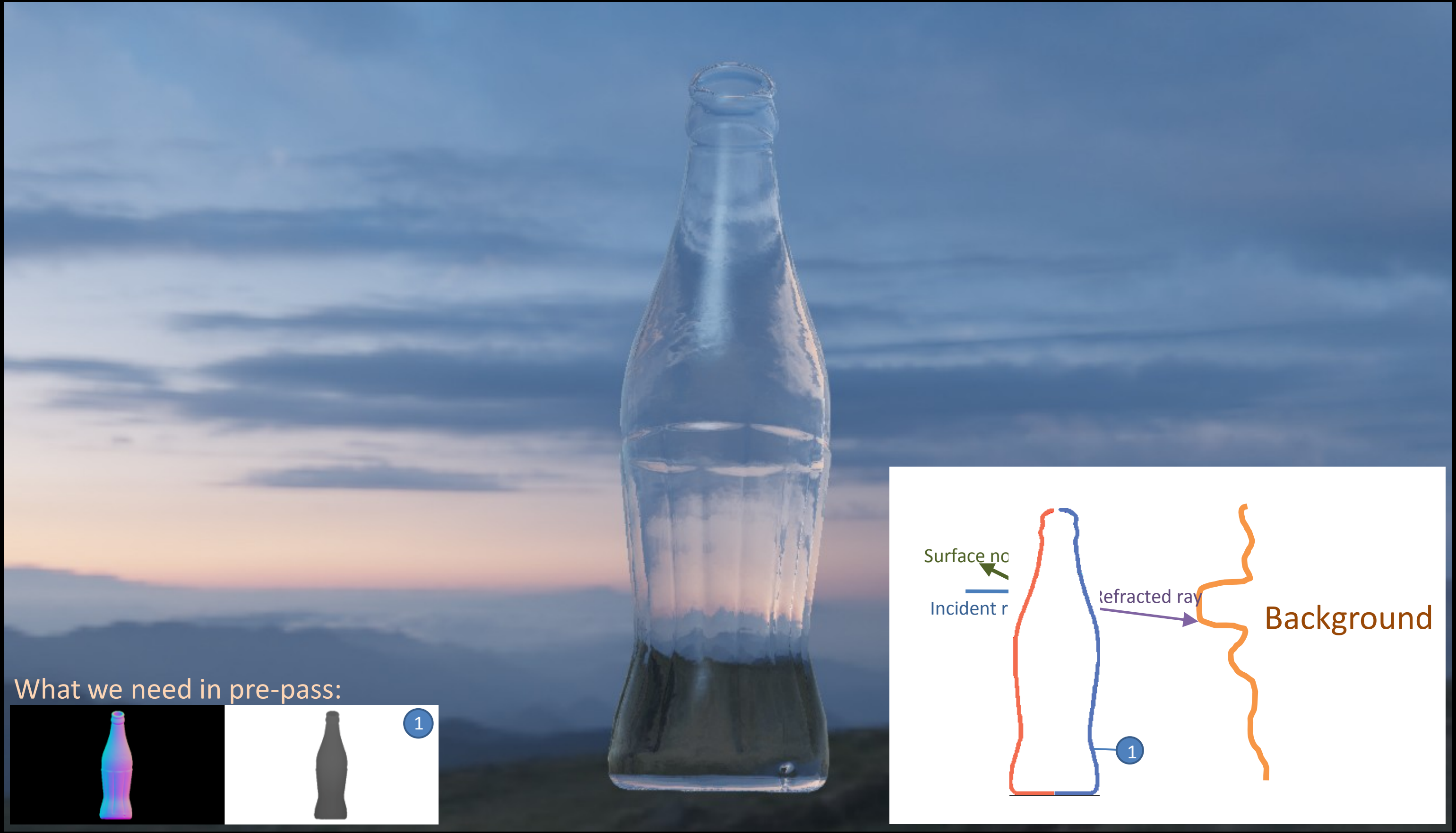




1st attempt: use only the front surface



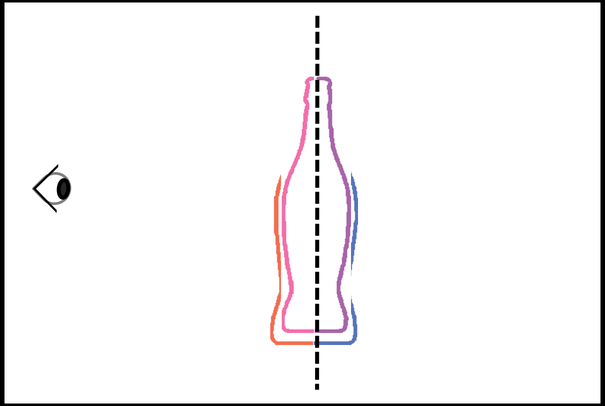
2nd attempt: also use the back

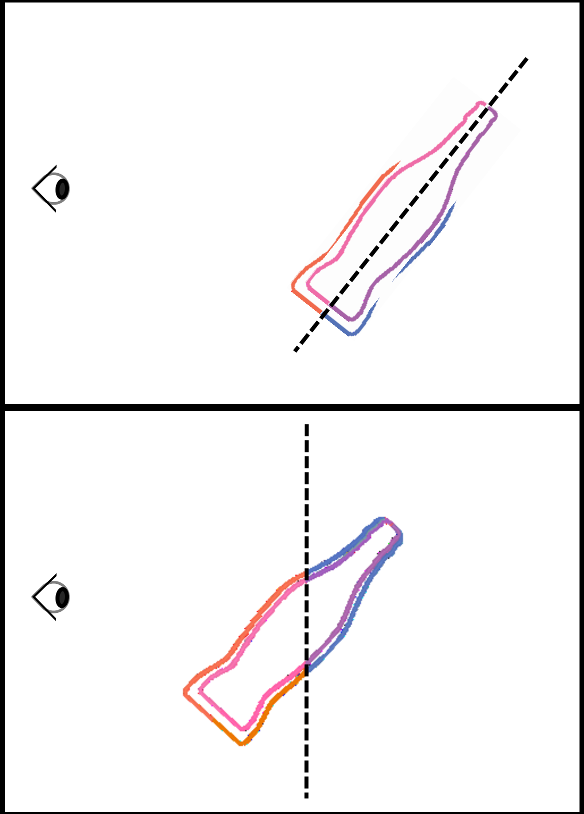


3rd attempt: use the inside

Splitting front, back & inside:1st attemp

correct





incorrect

Splitting front, back & inside:2nd attempt

		1 pass: Culling front Comparison greater-equal
		2 pass: Culling back Comparison greater-equal
		3 pass: Culling front Comparison less-equal
		Final pass: Culling back Comparison less-equal

Result from pure refraction

Adding specular

Adding cubemap reflection

Using a texture

No Refraction : Render Time = 1.5 ms

1st attempt : Render Time = 2.4 ms

2nd attempt : Render Time = 2.8 ms

3rd attempt : Render Time = 3.7 ms

Refraction: DEMO

Special Effect:
Particles

Particle System

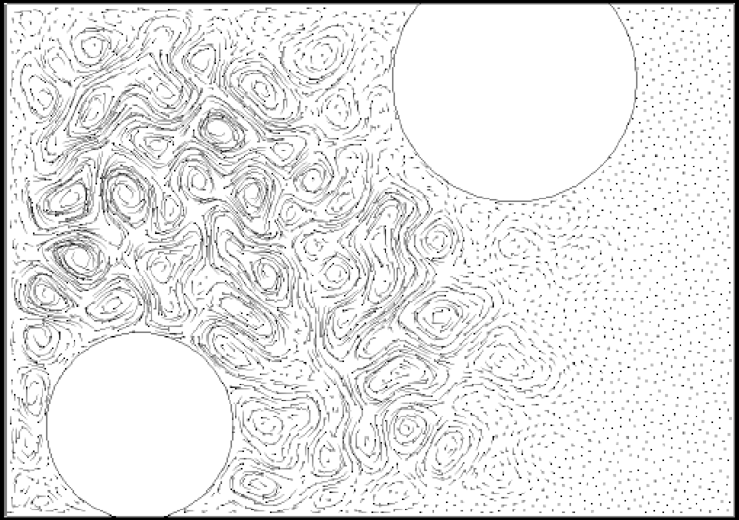
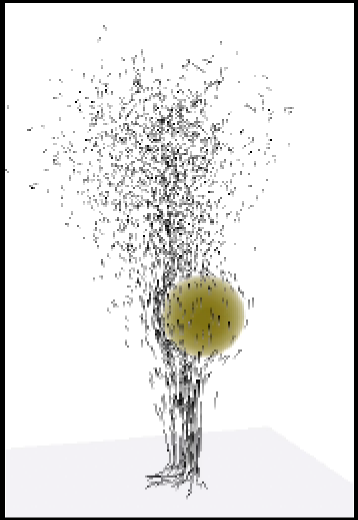
- Global Architecture
  – Simulate and render totally on GPU
- Simulation Features
  - External Forces
  - Collision with rigid objects
  - Vector Field
  - Cache
  - Curl Noise
  - Target Mesh
  - Locator

Particle System Simulation

- Birth : depend on emitter's parameters
  - Emitter type
  - Emitter's position
  - Direction and spread
  - Emit rate
  - Emitter's size
  - Speed and speed random
- Life : advance positions and velocities
  - Semi-Implicit Euler integration
  - External forces (gravity, wind, etc.)
  - Collision handling
- Death : when they live longer than their lifetime

Curl Noise

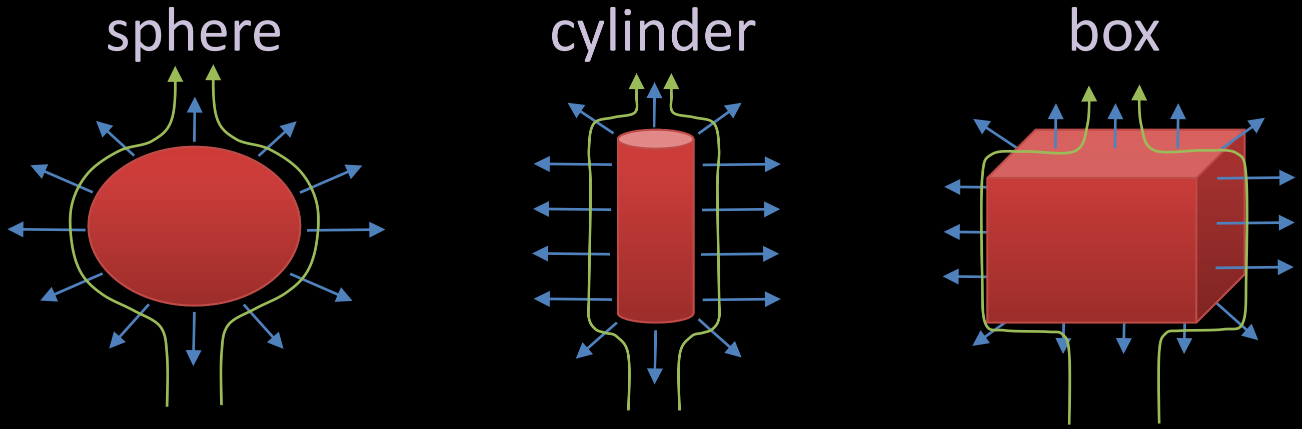
- "Curl-Noise for Procedural Fluid Flow" [Bridson *et al.* 2007]

Images from the paper

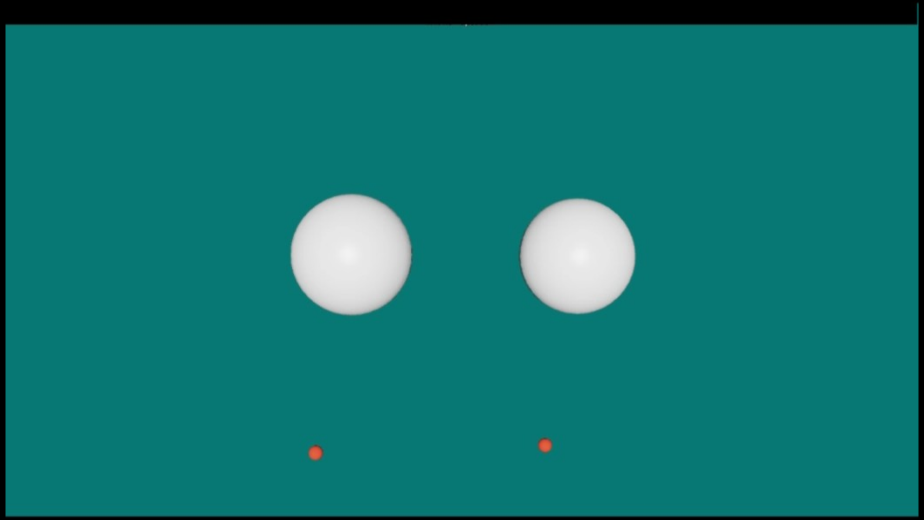
Collision with Rigid Objects

- Repulsion forces + vector fields



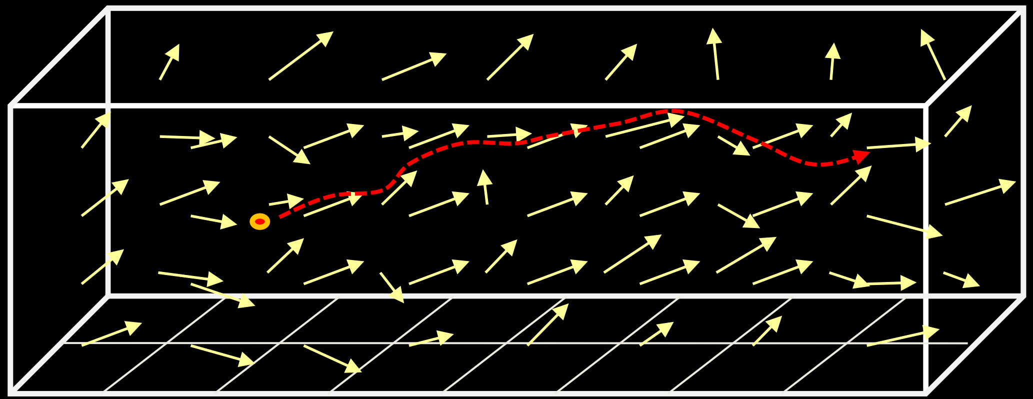
Collision with Rigid Objects

- Comparison between without(left) and with(right vector field)



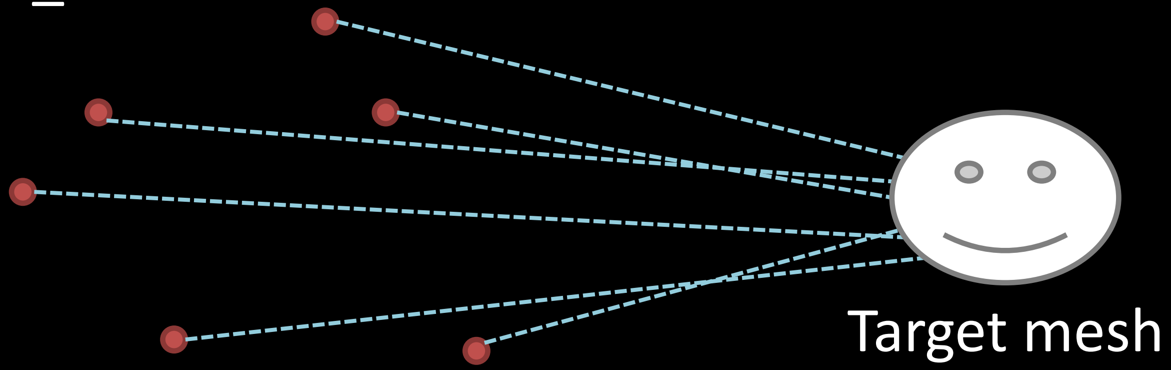
Velocity Field 3D Texture

- Add extra movement to dynamic systems



Target Mesh

- A mesh that particles follow or move towards.
- Particles move as if connected to the target by invisible springs.
- Particle cannot move towards the target faster than MAX_VEL_TO_GOAL

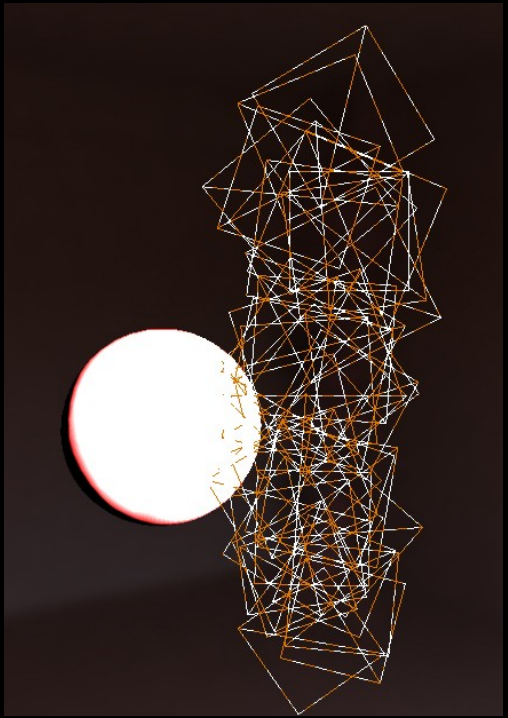


Particle System Rendering

- Types of Particles
  - Billboard : smoke, candle light, lightning, etc.
  - Mesh Instancing: bugs, cartridge, etc.
  - Blobby Object: flesh, blood and water

Billboard

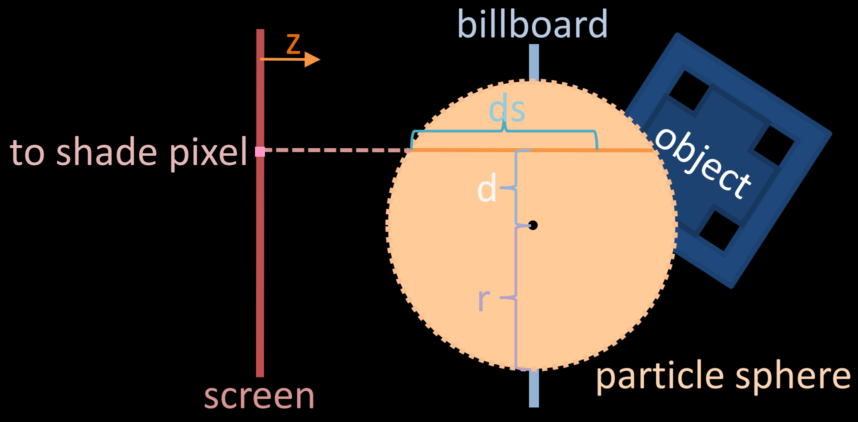
- Camera aligned, texture-mapped, partially transparent quad
- Sort is needed when using alpha blending
- Stretchable along velocity direction



Billboard: Candles

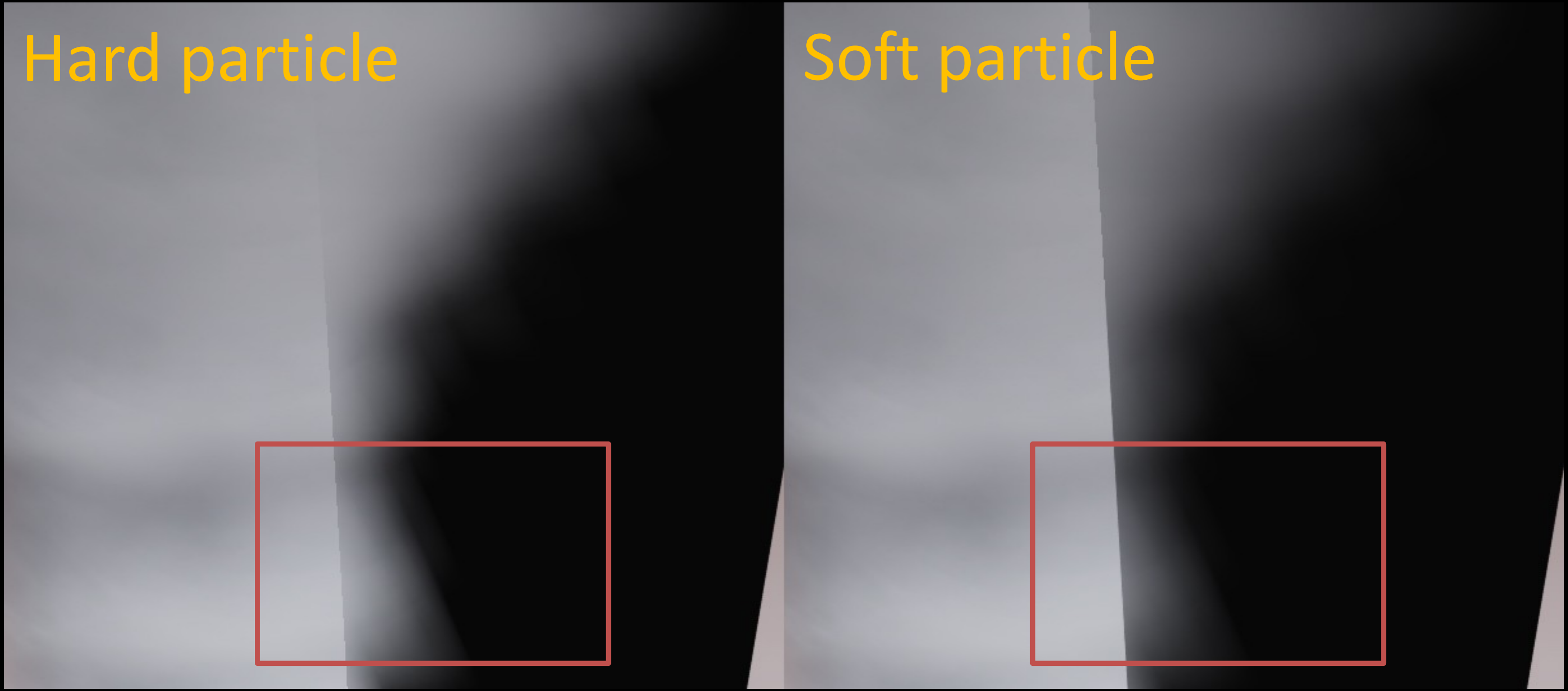
Billboard: Smoke

- Spherical billboards (soft particle)
  (based on "Spherical Billboards and their Application to Rendering Explosions [Umenhoffer *et al.* 2006]")

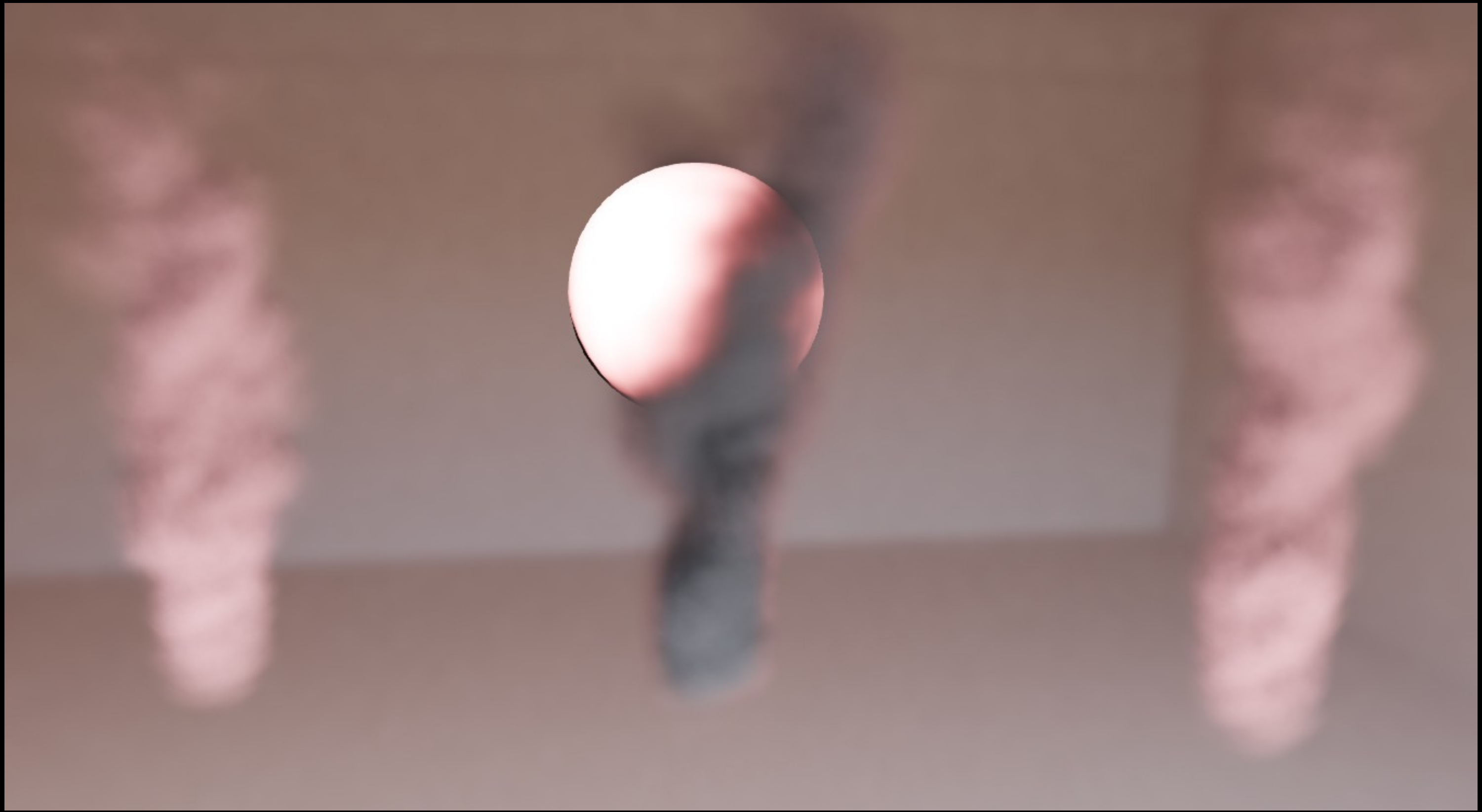


Alpha = 1 – exp(-density * (1 – d / r) * ds)
Alpha *= alpha from texture

Billboard: Smoke (soft particles)

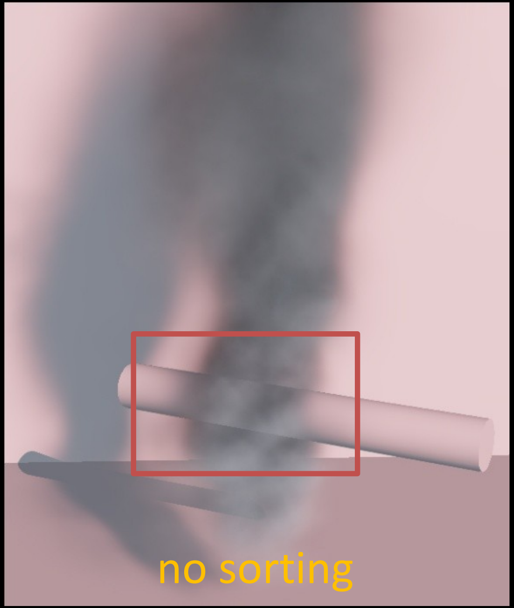


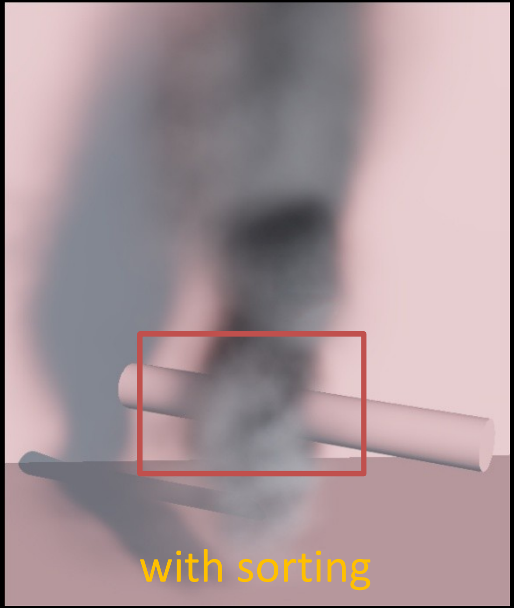
Billboard: Smoke (soft particles)



Billboard: Smoke (alpha blending)

- Proper alpha blending with sorting (Bitonic sort in our case)
- Plus 0.3 – 2 ms depended on the number of particles





Billboard: Smoke (light interaction)

Fourier Opacity Mapping

- “Fourier Opacity Mapping [Jansen and Bavoil 2010]”
- Purpose: To render volumetric shadows in cases where spatial opacity variations are smooth (e.g. smoke)
- Concept: Formulate the absorption function as a Fourier series

Fourier Opacity Mapping

- Implementation:
  - Render the Fourier coefficients to textures

$$\delta a'_{i,k} = -2\ln(1-\alpha_i)\cos(2\pi k d_i)$$
$$\delta b'_{i,k} = -2\ln(1-\alpha_i)\sin(2\pi k d_i)$$

  - Calculate the transmittance to generate the shadowing term

$$T(d) = \exp\left(-\left(\frac{a'_0}{2}d + \sum_{k=1}^{n}\frac{a'_k}{2\pi k}\sin(2\pi k d) + \sum_{k=1}^{n}\frac{b'_k}{2\pi k}\left(1-\cos(2\pi k d)\right)\right)\right)$$

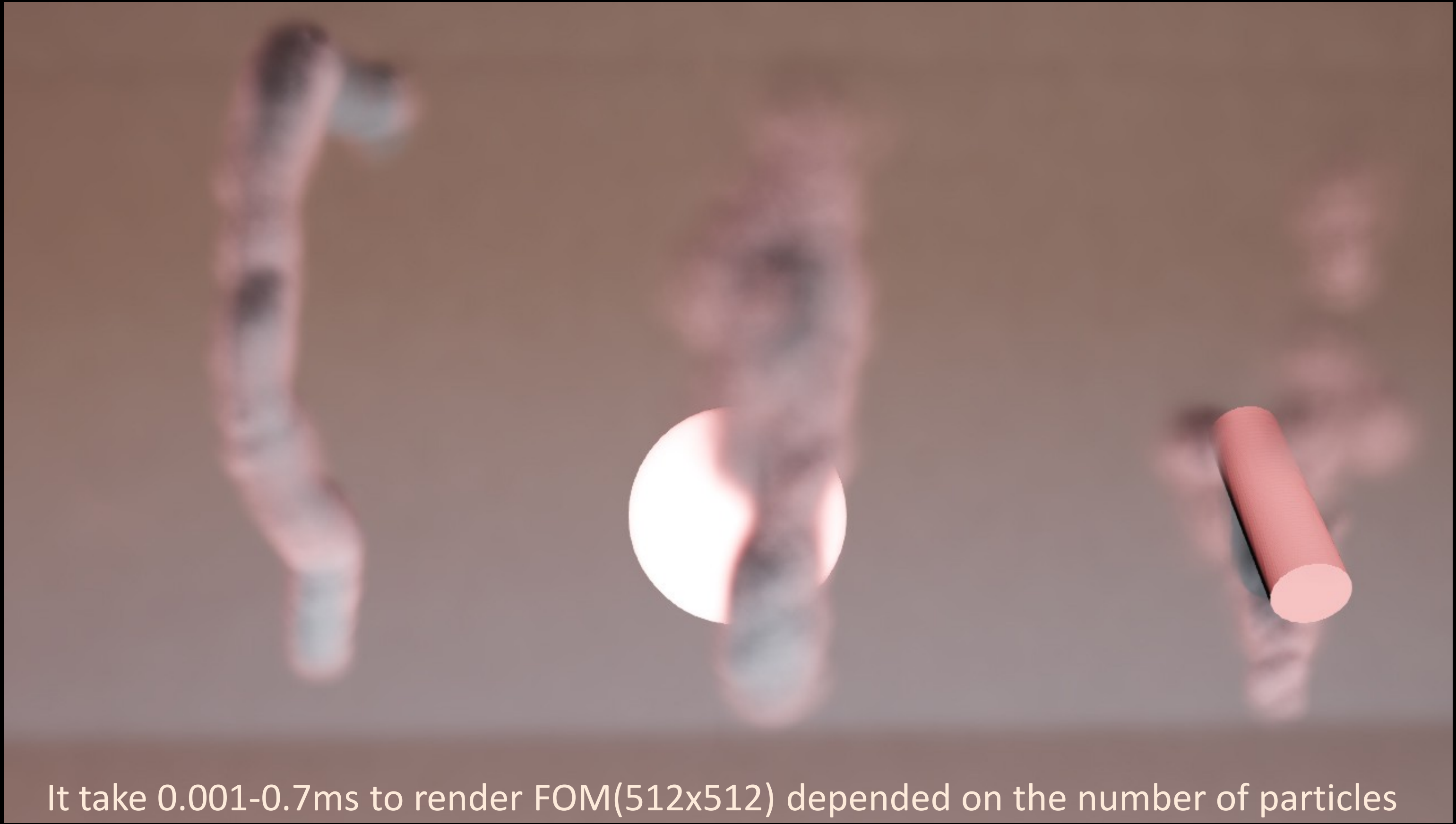
  - Apply shadow term to primitives

Fourier Opacity Mapping

Coefficient map with 7 coefficients

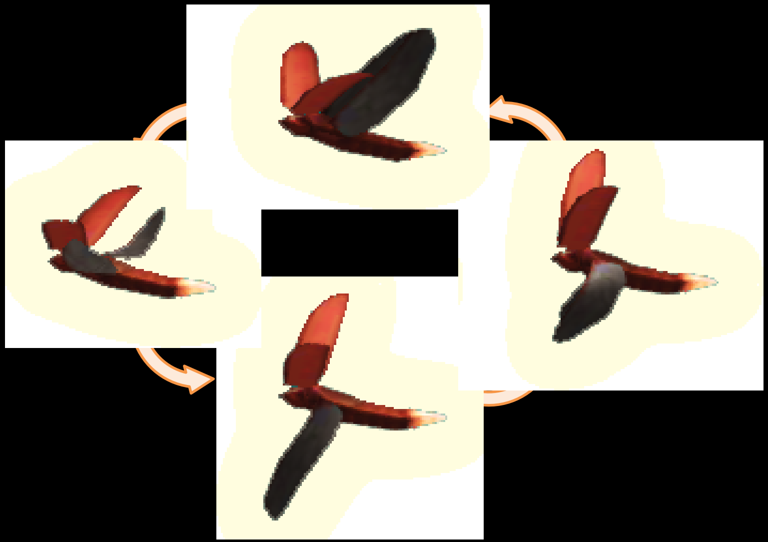
Fourier Opacity Mapping : OFF

Fourier Opacity Mapping : ON



Mesh Instancing

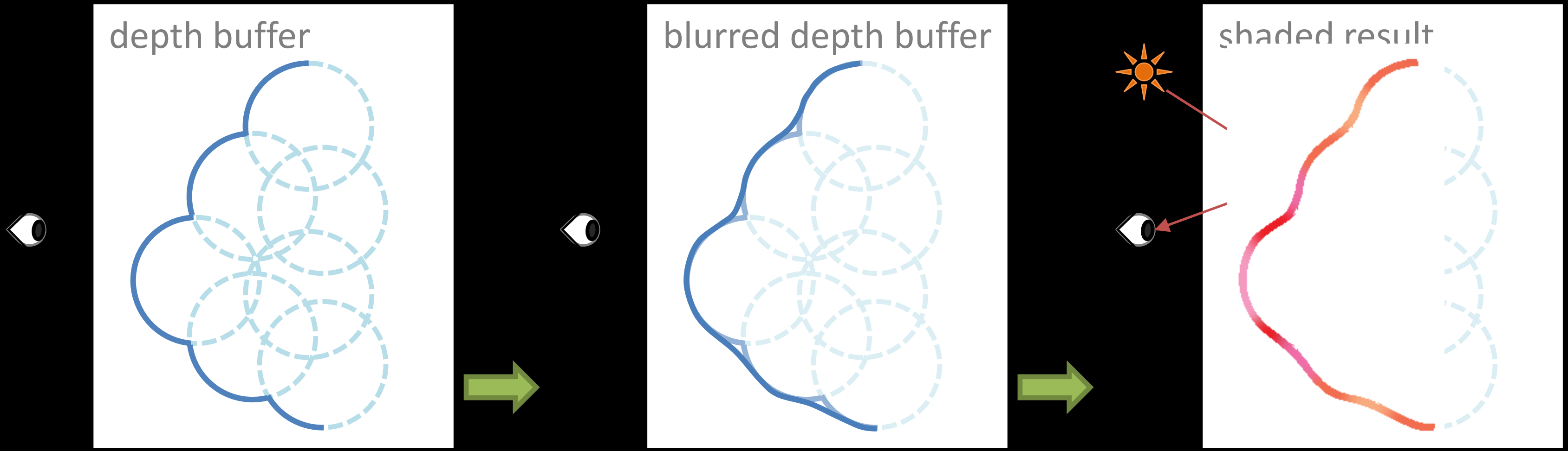
- Using mesh instancing to display particles
- A series of meshes could be used to display animation



Mesh Instancing

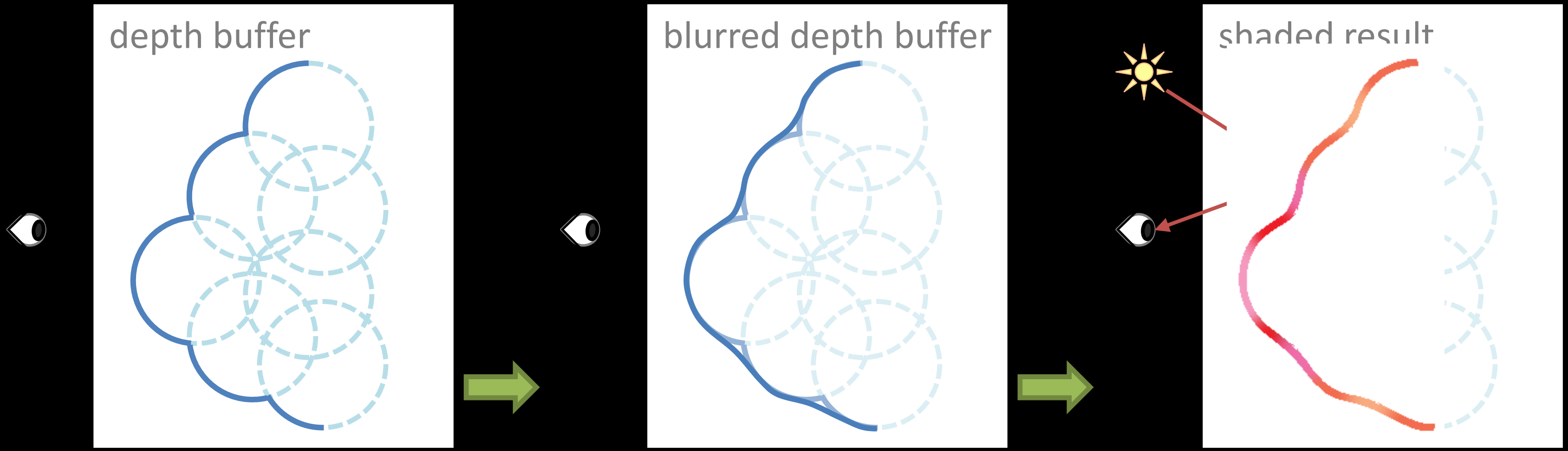
Opaque Blobby Object

- Screen-space Technique Based on:
  - “Screen Space Fluid Rendering with Curvature Flow” [Van der Laan *et at.* 2009]
  - “Screen Space Mesh” [Muller *et al.* 2007]



Opaque Blobby Object

① Render particles as spheres to depth buffer
② Blur depth buffer
③ Calculate position and normal at each pixel from depth buffer
④ Shade the pixel



Blood

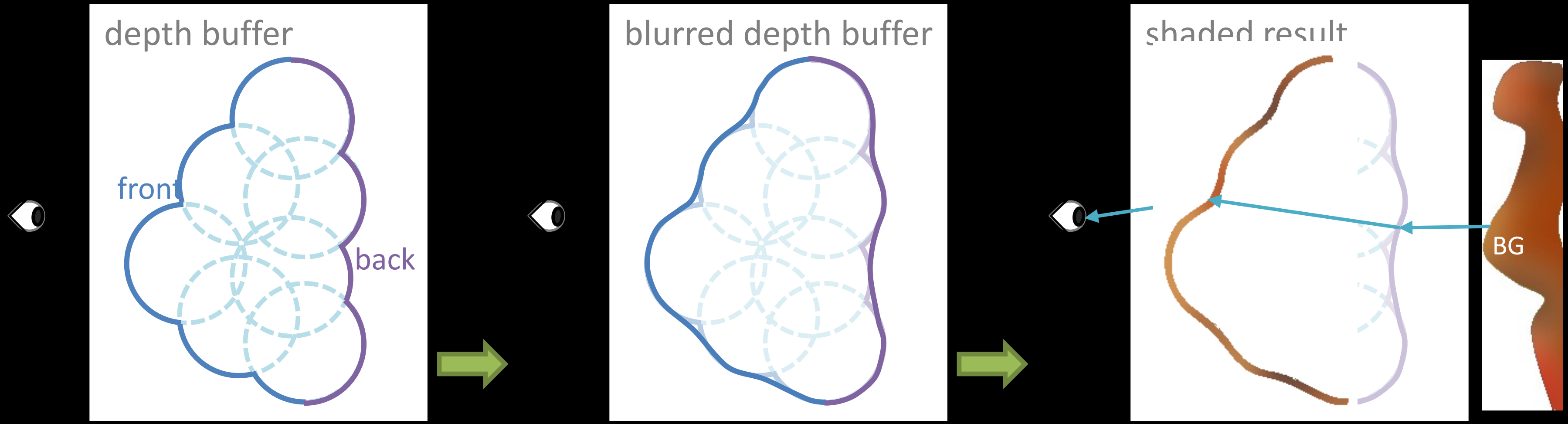
Blood

“Metaball” particles

"Metaball" particles: more blur

Transparent Blobby Object

① Render not only front depth but also back depth
② Do refraction if needed
③ Add foam color "A Layered Particle-Based Fluid Model for Real-Time Rendering of Water" [Bagar *et al.* 2010]
④ Apply some water coloring techniques



Water as a transparent object

Adding refraction

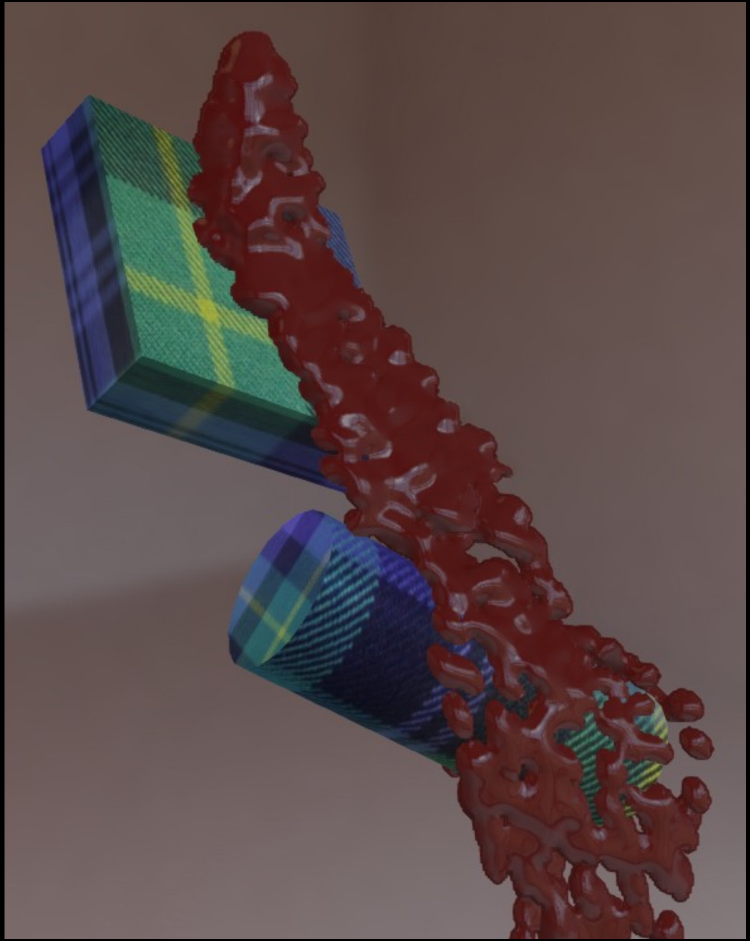
Adding specular

Adding foam

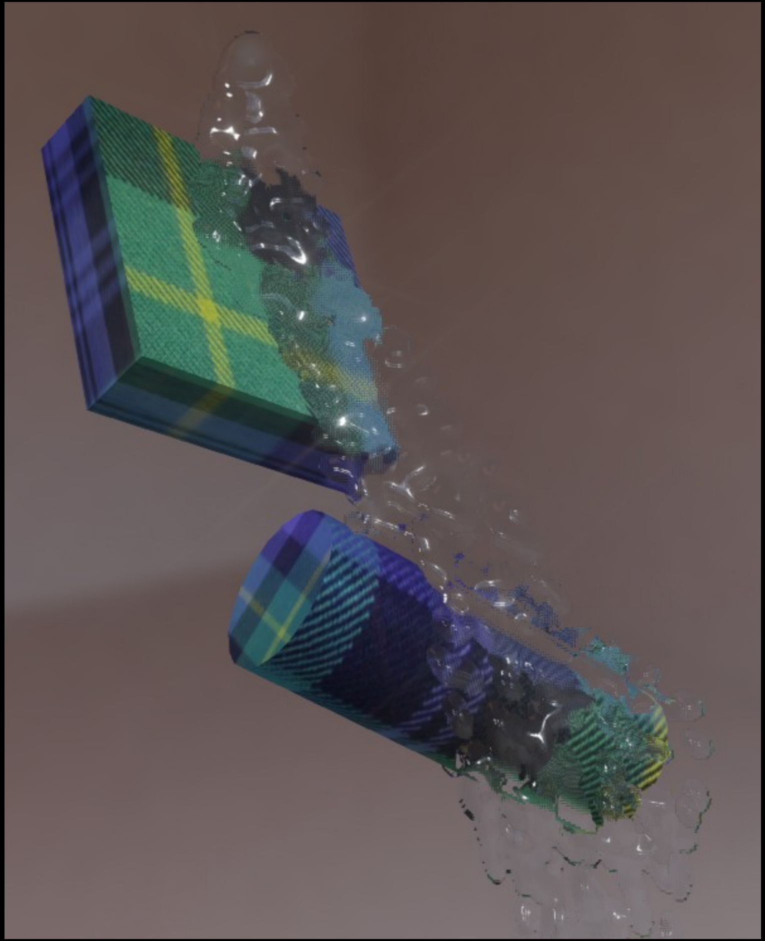
Adding cubemap reflection

Some post process glare...

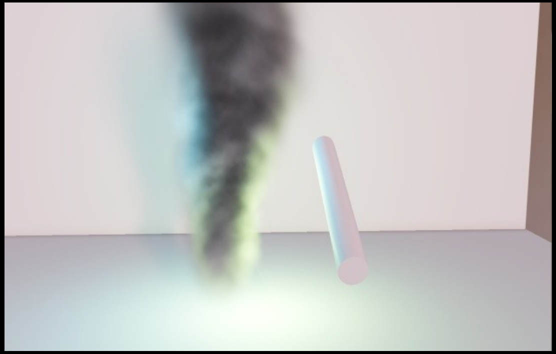
Opaque/Transparent Rendering Comparison

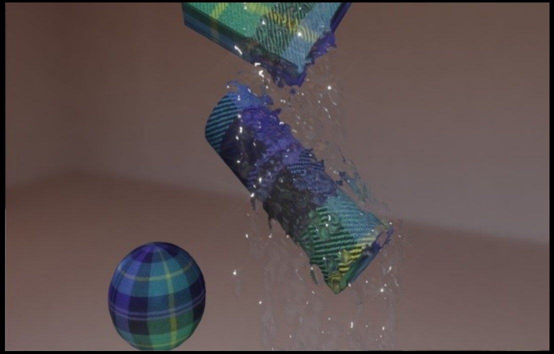


Slow down
≈ 15%



Particles: DEMO





The End


Questions?