Vulkanised 2026
The 8th Vulkan Developer Conference
San Diego, USA | February 9-11, 2026

Path-Traced Hair Rendering in Indiana Jones and the Great Circle

Sergei Kulikov, MachineGames
Jiho Choi, NVIDIA

Agenda

- Introduction
- Geometry
- Rendering
- Performance
- Q&A

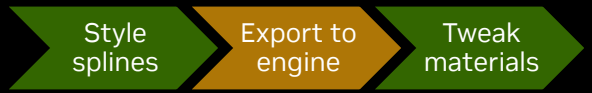
Introduction

Why strands?

- Game with a heavy focus on characters and cinematic experience
  - ~4 hours of cutscenes
  - Characters must look as good as possible
- Making hair cards is time consuming
  - Don’t want to make card-based fallback for characters with strand hair

Hair cards authoring

Hair strands authoring



MOTOR engine

- Derived from idTech 7
- Vulkan renderer on PC
- 60 FPS target on PC and consoles (Xbox Series S/X, PS5)
  - Handhelds target 30 FPS
- Requires HW ray tracing support for indirect lighting
- Has “Full Ray Tracing” mode on PC (path tracing)
  - Everything needs to be in BVH, including hair
  - Problematic with strands (without LSS support)
    - No hair in reflections
    - No shadows on surrounding geometry


RTX HAIR OFF
Indiana Jones © & ™ 2026 Lucasfilm Ltd.
E Skip


RTX HAIR ON
Indiana Jones © & ™ 2026 Lucasfilm Ltd.
E Skip


RTX HAIR ON
75
50
35
Indiana Jones © & ™ 2026 Lucasfilm Ltd.


RTX HAIR OFF
Indiana Jones © & ™ 2026 Lucasfilm Ltd.
E Skip


RTX HAIR ON
Indiana Jones © & ™ 2026 Lucasfilm Ltd.
E Skip


RTX HAIR OFF
75
50
35
Indiana Jones © & ™ 2026 Lucasfilm Ltd.


RTX HAIR ON
75
50
35
Indiana Jones © & ™ 2026 Lucasfilm Ltd.


RTX HAIR OFF
Indiana Jones © & ™ 2026 Lucasfilm Ltd.


RTX HAIR ON
Indiana Jones © & ™ 2026 Lucasfilm Ltd.

Pipeline overview (rasterized hair)

CPU Setup
- Gather models
- Calculate LODs
- Emit dispatches

Simulation
- Update positions
- Update density voxel grids
- Optional (enabled per asset)

Skinning
- Skin to triangles
- Write post-transform cache

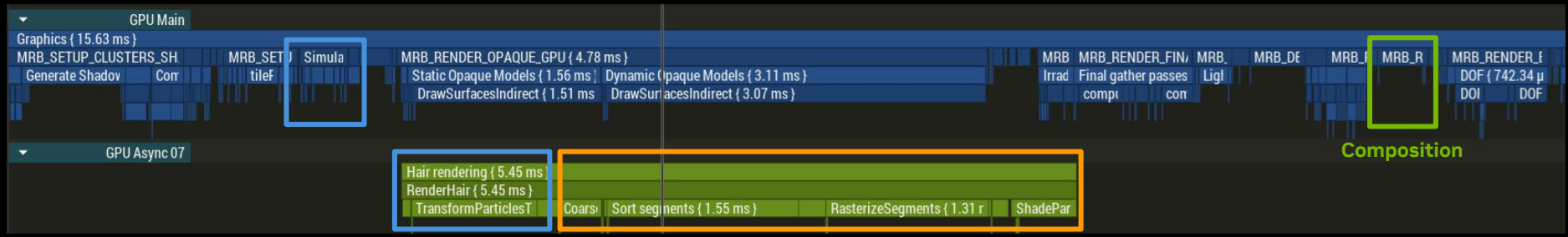
Culling
- Frustum culling
- Occlusion culling
- Hat culling
- Write lists of visible segments

Rasterization
- SW rasterization
- 1 lane per segment
- Write to visibility atlas

Shading
- Strand-space
- 1 shading point per segment
- Write to separate buffer

Composition
- 1 quad per groom asset
- Sorted back to front
- Together with other transparent objects
- Write depth and motion

* hair timings here do not reflect actual workload size because of async compute overlap with opaque lighting
Measured on NVIDIA GeForce RTX 2070



LODs

Randomized strand order, discard tail of the strand data to do LODs [Taillandier20]

Bound maximum number of strands per asset (on asset load)

Compensate strand number reduction by increasing thickness

LODs

Bound number of segments per area (per frame, based on model bounding box area)

Limit frame workload to 2 million segments (for all models combined)

Skip every other vertex in strand for distant models

For path tracing:

- Distance-based LOD calculations (can’t use projected area for models outside view frustum)
- X4 less strands for offscreen models (compared to on-screen)
- No vertex skipping (to avoid changing topology of a strand)

Skinning

Pre-skin all parent meshes

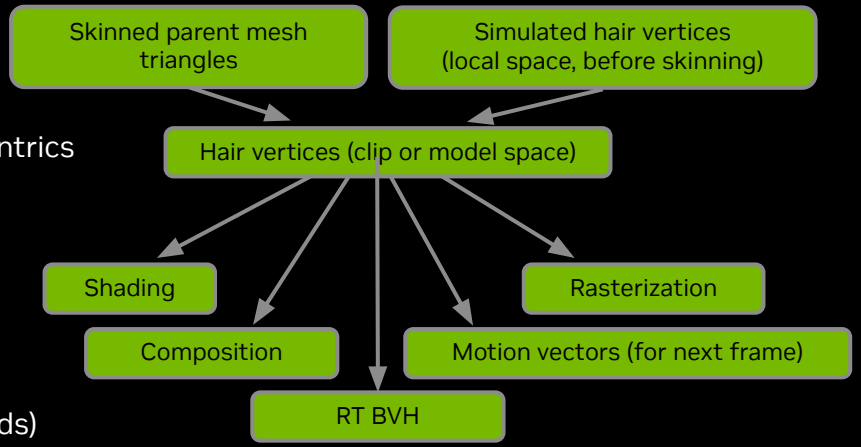
Pre-skin & transform all strand vertices

Skin directly to triangles of parent mesh, store triangle barycentrics and compressed tangent frame for each strand

Store post-transformed results in global cache (before division by w)

32MB for 2 million vertices (1 million / 16MB on low spec)

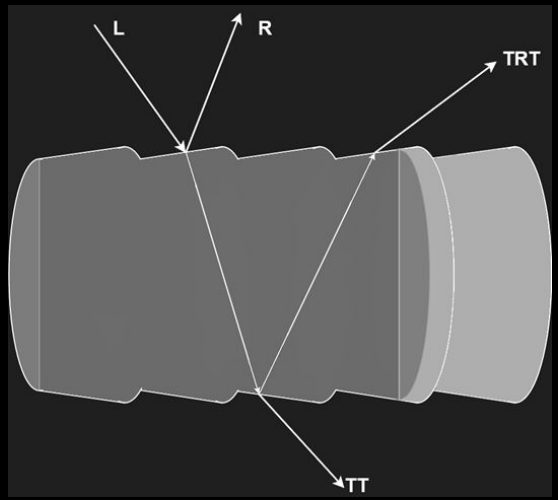
For path tracing also cache model space position (for BVH builds)




```
struct hairRootSkinningInfo_t {
    uint32 idxTriangle;
    uint32 barycentrics; // packed x and y, z is reconstructed
    uint32 triangleNormal; // packed x and y, octahedral encoding
    uint32 triangleTangent; // packed x and y, octahedral encoding
};
```

Rasterization and shading

- Conservative software rasterizer
  - Perfect antialiasing
  - 3 top hair fragments in each pixel
- Per-vertex shading
  - 3 paths - Reflection (R), Transmission-Transmission (TT) and Transmission-Reflection-Transmission (TRT)
  - [Karis16] and [Tafuri19] approximations for single scattering
  - In-house approximation for multiple scattering
  - Baked near-field self-shadowing
  - Low-res volumetric shadows from hair



Path-Traced Hair

Path-Traced Hair

- Strand-based hair geometries in Ray Tracing Acceleration Structures (=BVH)
- Trace Rays over hair geometries with multiple bounces

Pros
- Ray-Traced Hair Reflections
- Ray-Traced Hair Shadows and GI
- Ray-Traced Hair Multi-Scattering
- No hard tricks

Cons
- Performance / Memory

(BVH: Bounding Volume Hierarchy)

Geometry

Hairs In BVH

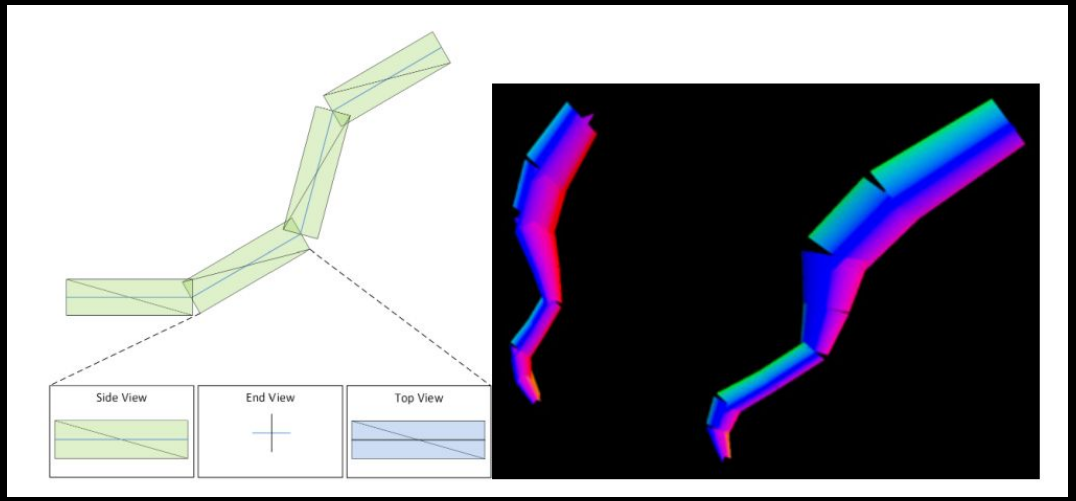
- Game uses strand-based hair without hair-card fallback
- Strand-based hairs are drawn as camera-facing quads
  - No world-space triangle geometries ready
- Without hairs in BVH
  - No hairs in RT shadows and RT reflections
- Options:
  1. Intersection Shaders → suboptimal performance
  2. Convert strand-based hair into triangle mesh → Rectangular Tube or DOTS
  3. New primitive type with HW support → LSS

DOTS(Disjoint Orthogonal Triangle Strips)

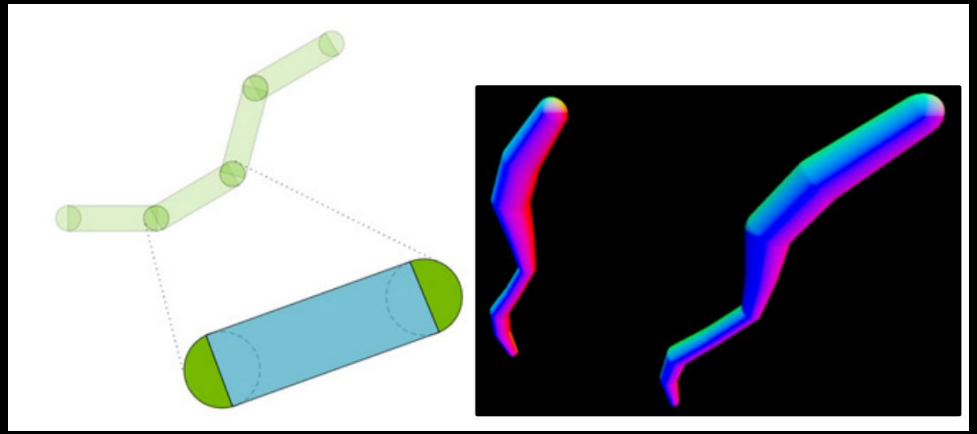
To add hair geometries in BVH

One hair segment:
- Two orthogonal quads
- = 8 vertices
- = 4 triangle primitives

(Not used in the released game)



LSS (Linear Swept Spheres)



NVIDIA Blackwell+ GPUs

One hair segment:
- 1 LSS primitive
- = 2 positions, 2 radius
- shared with next primitive

LSS is faster than DOTS and uses less VRAM than DOTS

Vulkan
- VK_NV_ray_tracing_linear_swept_spheres

D3D12
- With nvapi

How to integrate LSS

1. Prepare your strand-based hair buffers (position, radius, and index)
2. Build BLAS with LSS primitives

```
VkAccelerationStructureGeometryKHR::geometryType = VK_GEOMETRY_TYPE_LINEAR_SWEPT_SPHERES_NV;
// Rather than VK_GEOMETRY_TYPE_TRIANGLES_KHR
```

3. Build TLAS with hair instance mask added (to filter hair)
4. Add the flag to ray tracing pipline

```
VK_PIPELINE_CREATE_2_RAY_TRACING_ALLOW_SPHERES_AND_LINEAR_SWEPT_SPHERES_BIT_NV
```

(BLAS = Bottom-Level Acceleration Structure)

(TLAS = Top-Level Acceleration Structure)

How to integrate LSS

- In RayGen shader
  - TraceRay() with hair instance mask included
  - (SkipBuiltinPrimitivesNV / SkipTrianglesKHR ray flag to filter out)
  - There is no front-/back-face culling with LSS (no winding orders)
- In ClosestHit shader
  - Compute hair normal vector
  - Lighting (Forward+) and return outLighting as payload
  - LSS and sphere use different HitKind from triangles
- `NVIDIA Vulkan Ray Tracing Tutorials (v2.0)`
  - `github.com/nvpro-samples/vk_raytracing_tutorial_KHR`

LSS primitives for BVH build

- Positions and radius are packed with 4 x float32
  - Positions : 3 float32 (model space)
    - Precision is important for hair
  - Radius: 1 float32
- Index buffer
  - Format: uint32 (single global vertex buffer)
  - With VK_RAY_TRACING_LSS_INDEXING_MODE_SUCCESSIVE_NV
  - Only one index is needed for one LSS primitive
  - The second vertex index is just N+1
- No end caps
  - Hard to see holes
  - No noticeable perf / vram diff

Builds/Refits

- LOD changes → Topology changes
  - Degenerate primitives by 0 radius
  - Change primitive count
- So we had
  - No Refits
  - No BLAS compaction

Rendering

Trace Primary Ray for Hair Only

- No gbuffer hair depth/normal ready (raster in blending mode)
- Due to different LOD between raster vs PT hair
- Raster hair doesn’t output same depth/normal as LSS
- Raster hair uses alpha-blending for short hairs
  - PT hair decreases the root radius for similar effect

Trace Primary Ray for Hair Only

- In a separate visibility buffer render pass
- Outputs depth, normal, primitiveIndex, and modelIndex
- When pixels are within hair AABB
- TraceRay to hit the hair
  - Trace with hair only mask (for perf)
  - Compare hair depth with gbuffer depth to cull

RTX Character Rendering SDK

- Used BSDF (Bidirectional Scattering Distribution Function) from NVIDIA RTX Character Rendering SDK

  github.com/NVIDIA-RTX/RTXCR
- Added multipliers to minimise the visual difference between raster vs PT hair
- Near-field Chiang BSDF vs Far-field BSDF (=used this for less noise)

Near-field Chiang BSDF

Far-field BSDF

RayGen Shader

- Max 6 bounces (+primary ray)
- Sample new ray from Far-field BSDF : PrepareRay()
  - New ray direction can be from R, TT or TRT lobe
- Trace ray with the new ray direction
  - ClosestHitShader returns the radiance
- Shares the same PT loop as non-hair
  - PrepareRay() implementation is different

```
bounceNum = hitDesc.isHair ? 6 : 2; // For hair, use max 6 extra bounces than the primary rays
PrepareRay( hitDesc.isHair, ... ); // Sample new ray direction based on BSDF sampling, R/TT/TRT probability and roughnesses
for( uint16_t bounceId = 0; bounceId < bounceNum; ++bounceId ) { // Iterate for multi bounces
  TraceRay(); // Trace rays to see if it hits something
  hitDesc = InitHitDesc( ... ); // Decode hit material and shading result from the CHS
  pathDesc.radiance.xyz += half3( hitDesc.shading * float3( throughput ) ); // Accumulate the shading
  bounceId = shouldTermintePath ? bounceNum : bounceId; // Ray miss (sky/emissive) or no throughput then terminate
  ray = PrepareRay( hitDesc.isHair, ... ); // Decide which direction for the next journey
}
```

PrepareRay() for Hairs

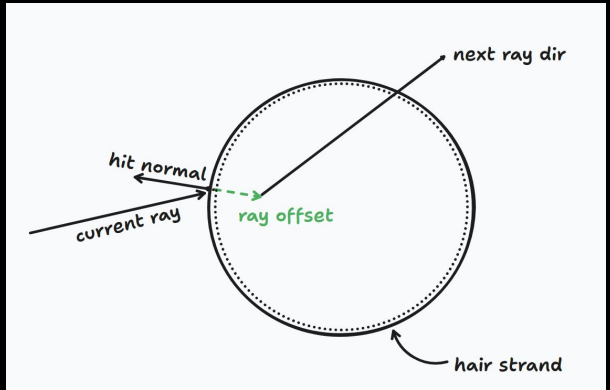
1. Unpack hair model ID and primitive ID from the payload
2. Decode hair material parameters
3. Sample BSDF → new ray direction with PDF
4. If next ray goes through the hair,
   - Offset ray origin

```
PT_PREPARE_SHADING_SAMPLE( ... ) // Output hairSample
RT_HAIR_SETUP_BSDF_MATERIAL( TBN, viewVectorLocal, lightVectorLocal, hairSample, hairMaterialInteraction,
                             hairInteractionSurface );
bool continueTrace = RTXCR_SampleFarFieldBcsdf( hairInteractionSurface, hairMaterialInteraction, viewVectorLocal, h,
                                                lobeRandom, rand2, sampleDirection, bsdfSpecular, bsdfDiffuse, bsdfPdf );
bsdfWeight = bsdfDiffuse + bsdfSpecular;
if ( !continueTrace ) {
  throughput = _half3( 0 );
  return false;
}
bsdfWeight /= bsdfPdf;
throughput *= bsdfWeight;
// New sample direction
ray.direction.xyz = normalize( MatrixMul( TBN, sampleDirection ) );
```

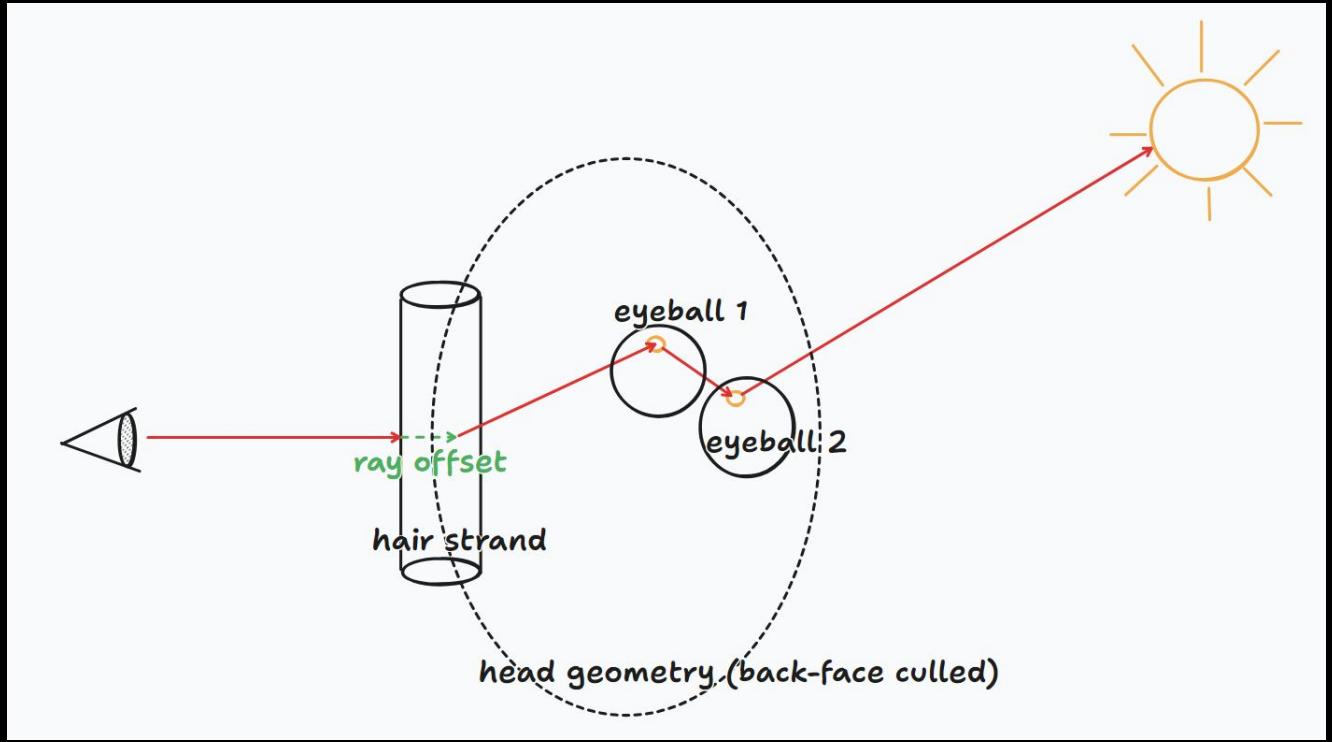
Offset Ray Origin

- To avoid self occlusion
- Too big offset (> the strand width) can cause artifact
- LSS hair is always back-face culled in TraceRay (= no inside-to-outside hit)

```
// Refraction requires the ray offset to go in the opposite direction
const bool transition = dot( hitNormal, ray.direction.xyz ) <= 0.0f;
const float3 offsetNormal = transition ? -hitNormal : hitNormal;
// moving the ray origin slightly forward can avoid passing the head geometry when the hair // is
attached to the head geometry
const float extraOffsetDistance = transition ? thickness * 0.1f : 1e-5f;
ray.origin = hitPos + offsetNormal * extraOffsetDistance;
```



Offset Ray Origin: Bug

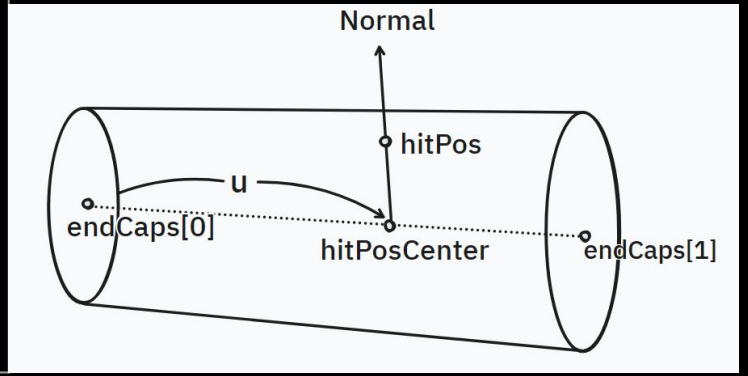


Ray Hit Normal Vector in Closest Hit Shader

To compute normal vector from hit position of hair,

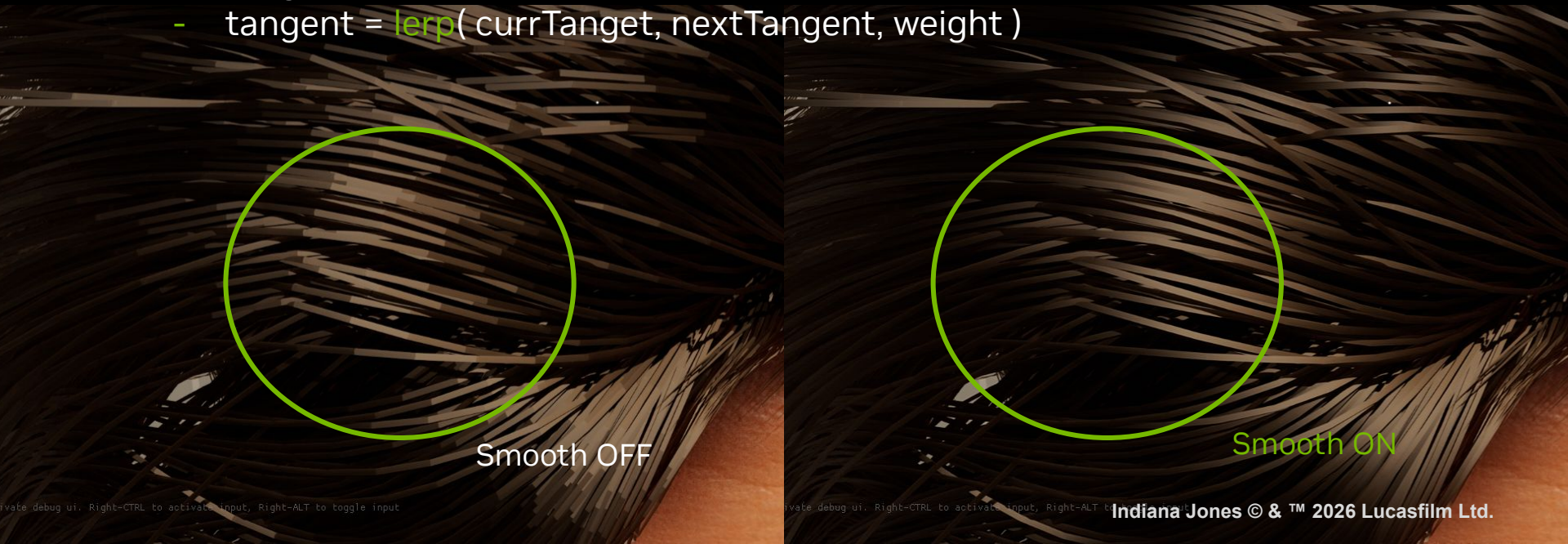
- Using the u(or x) parameter from '`hitAttributeEXT vec2 baryCoord`'
- Compute projected hit position along centerline (= hitPosCenter)
- Then, normal vector = hitPos - hitPosCenter

```
hitAttributeEXT float uVal; // hit position along LSS midsection. [0 to 1]

// Returns the position of the two endcaps of LSS
vec3 endCaps[2] = { gl_HitLSSPositionsNV[0], gl_HitLSSPositionsNV[2] };

// In object space
const vec3 hitPosSurface = ( gl_ObjectRayDirectionEXT * gl_HitTEXT ) +
                           gl_ObjectRayOriginEXT;
// Interpolated position in midsection
const vec3 hitPosCenter = lerp( endCaps[ 0 ], endCaps[ 1 ], uVal );
vec3 rayHitNormalInObjectSpace = normalize( hitPosSurface - hitPosCenter );

// In world space
float rayHitNormalInWorldSpace = mat3( gl_ObjectToWorldEXT ) *
                                 rayHitNormalInObjectSpace;
```



Smooth Tangent Normal

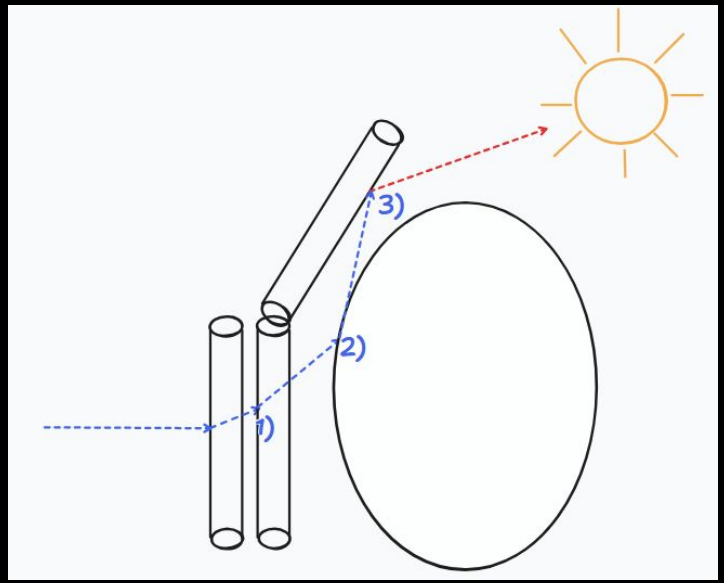
- LSS segments are not curved. In closer view, lighting looks flat for longer hairs
- Interpolate tangent/normal from **current** segment to **next** segment
  - weight = u from barycentric
  - tangent = lerp(currTanget, nextTangent, weight)



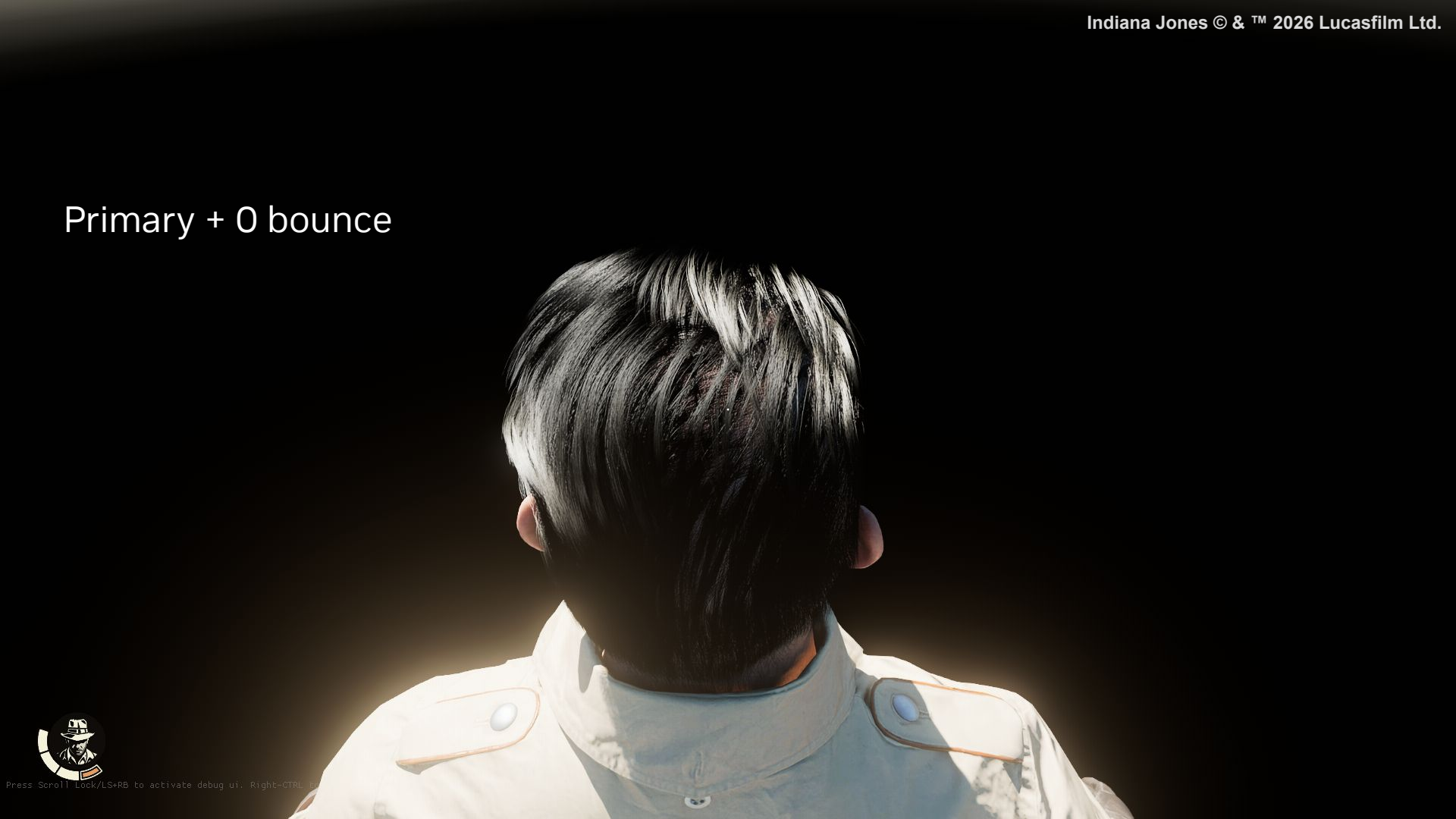
Multi-Bounce Scattering

Multi-Bounce Scattering

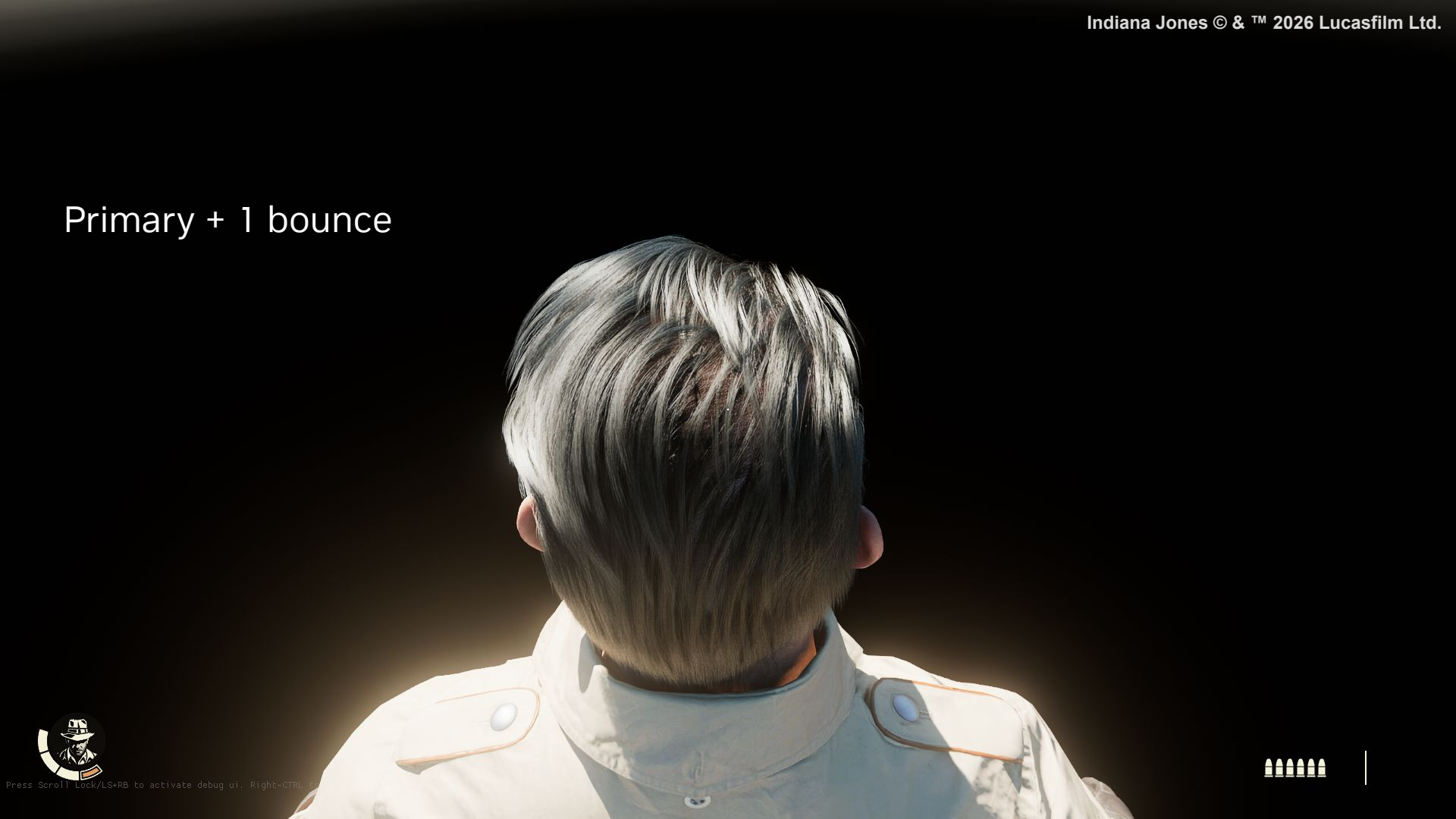
- Intra-strand scattering is done by Far-field BSDF using R/TT/TRT
- Inter-strand scattering is done by shooting more rays
- Max 6 (excluding primary ray) bounces for path-tracing loop:
  1) Strand → Strand
  2) Strand → Opaque geometry
  3) Opaque geometry → Strand



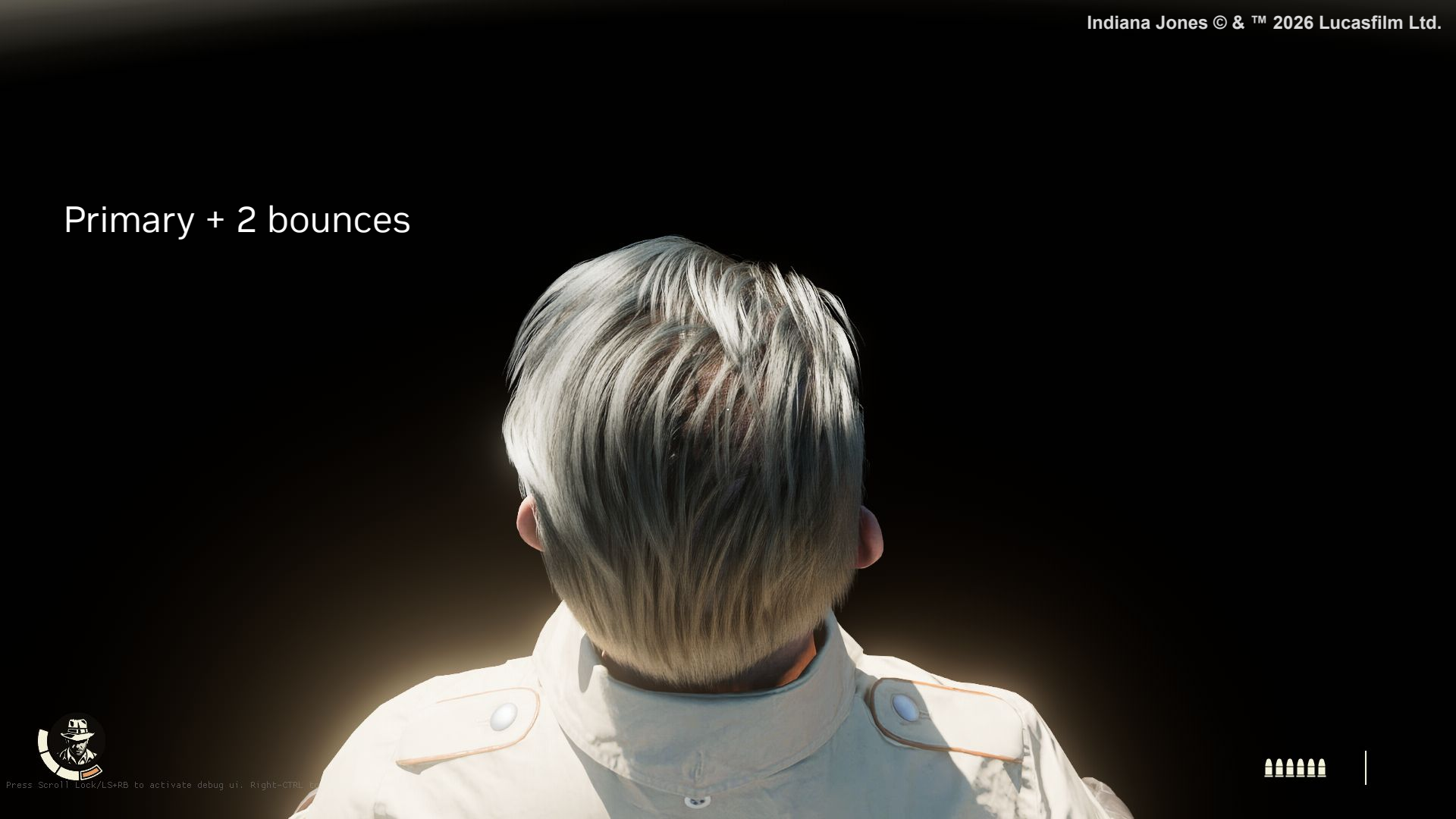
Primary + 0 bounce



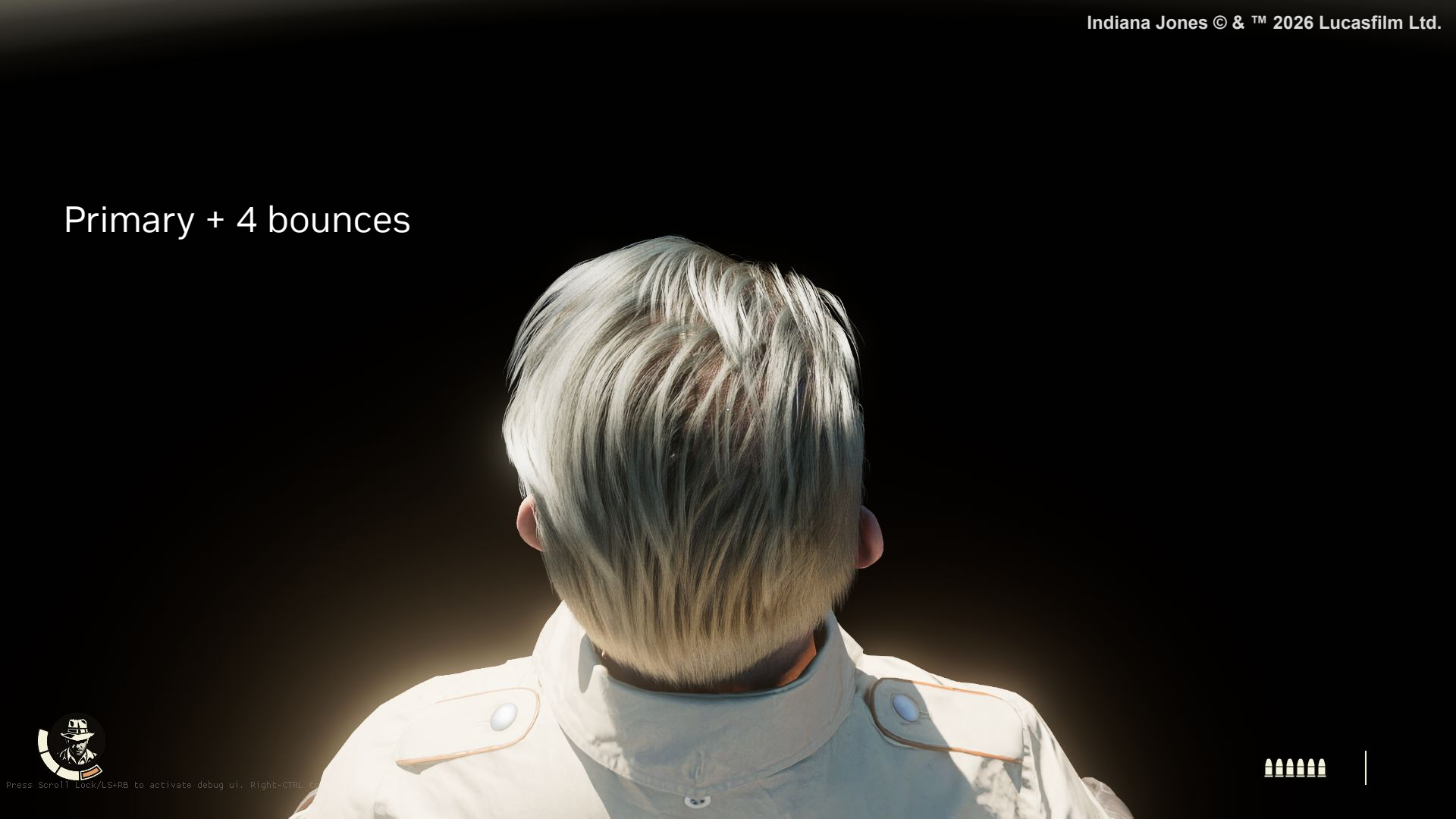
Primary + 1 bounce



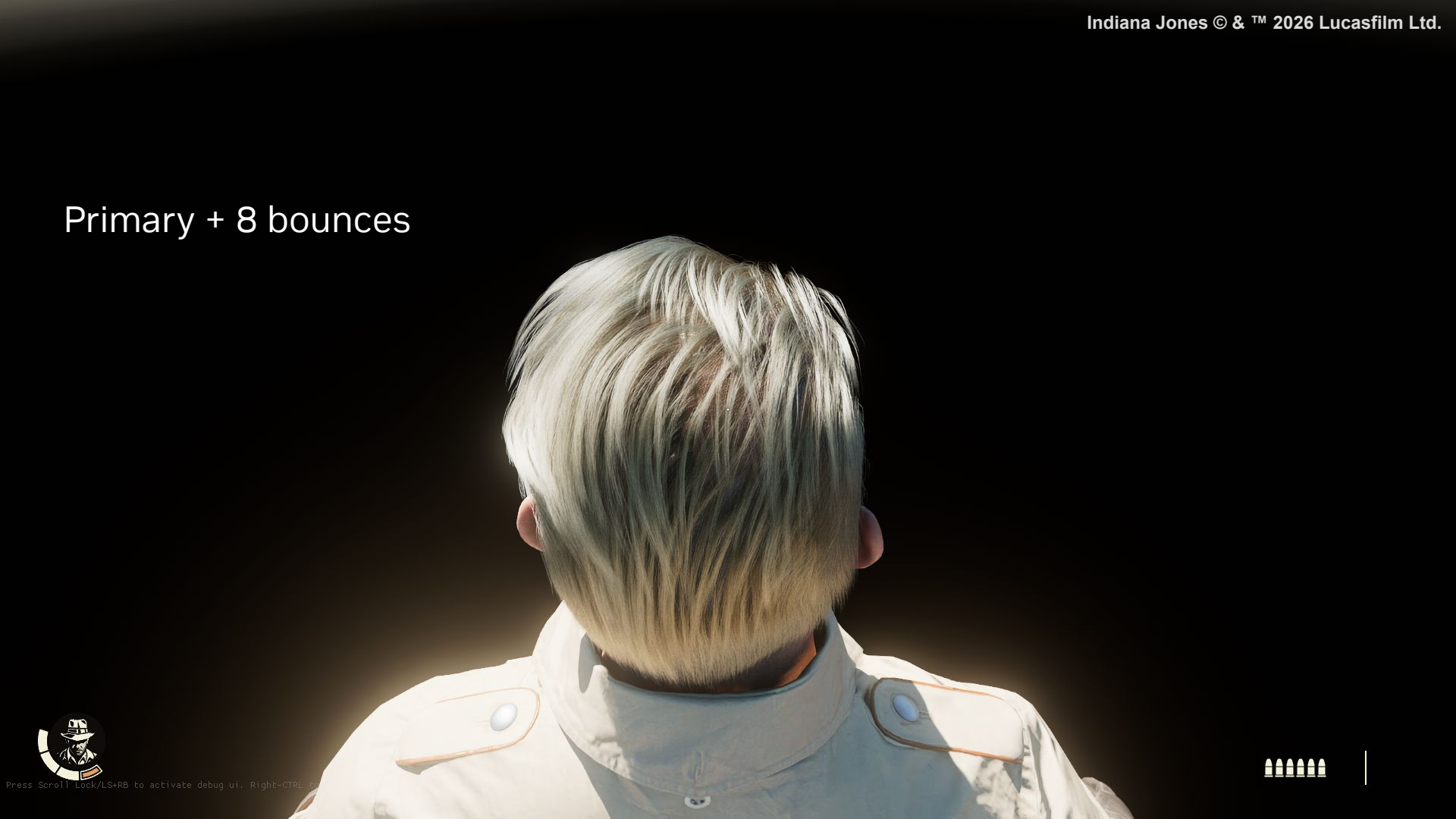
Primary + 2 bounces



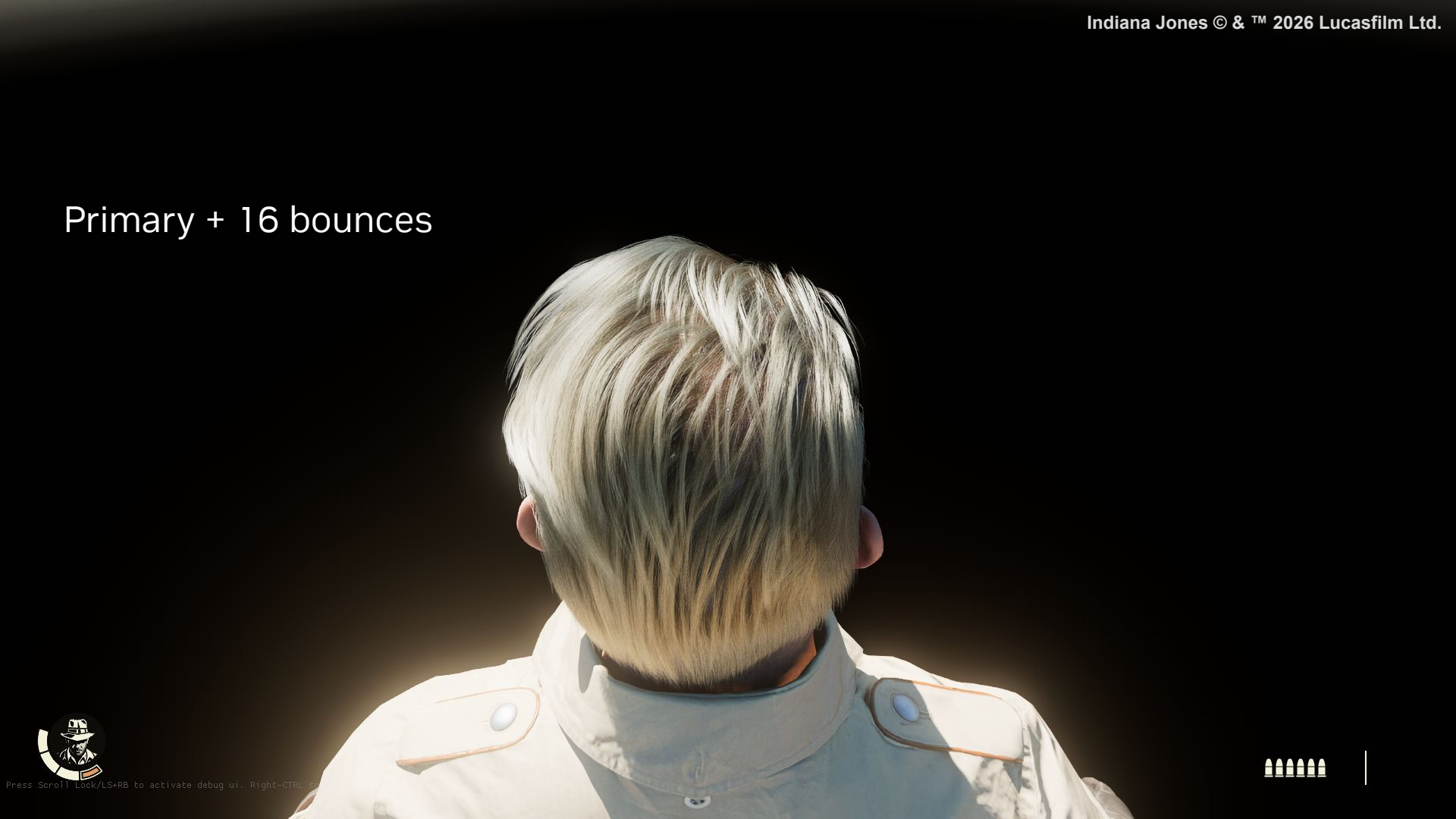
Primary + 4 bounces



Primary + 8 bounces



Primary + 16 bounces



Path-Traced Hair Reflection

RTX HAIR OFF

No hair in path-traced reflection


RTX HAIR ON
Hairs in path-traced reflection
Indiana Jones © & ™ 2026 Lucasfilm Ltd.
Press Scroll Lock/LS+RB to activate debug ui. Right-CTRL to activate input.

DLSS-RR

720p → 1080p

- Albedo
- Specular Albedo
- Normal / Roughness
- Motion Vectors

Performance

Performance Comparison

	PT Hair OFF	DOTS Hair	LSS Hair (vs DOTS)
Hair Primary Ray	0 ms	1.71 ms	0.7 ms (2.4x faster)
Path Tracing Main	1.79 ms	3.52 ms	3.35 ms (4.8% faster)
BLAS Build	1.07 ms	2.03 ms	1.23 ms (39% faster)
Hair BLAS VRAM	0	155 MB	43 MB (3.6x less)

(1440p DLSS-RR Auto, RTX 5080)

Acknowledgement

MachineGames

- Jim Kjellin
- Magnus Auvinen
- Andreas Larsson
- Nicholas Siren
- Mathias Lindner
- Patrik Willbo
- Mariusz Macieja
- Engine & Character Art teams

Michael Wynne

Jorge Luna (The Multiplayer Group)

idSoftware Engine team

NVIDIA

Pawel Kozlowski, Xiangshun Bei, Pierre Moreau, Magnus Andersson, Martin Stich, Vikram Kushwaha, Jussi Rasaren, Juho Marttila, Louis Bavoil, Ivan Povarov, Oleg Arutiunian, Fedor Gatov, Vladimir Mulabaev, Dajuan Mcdaniel, Dean Bent, Jon Story, Eric Reichley, Jiayin Cao, Tion Thomas, Tanki Zhang, Chris Lentini, Wen Su, Eric Enderton, Keith Morley, David Hart, Barbara Wolfers, Greg Muthler, Joshua Noel, John Burgess, Steven Parker

References

[Taillandier20] – Robin Taillandier, Jon Valdes (2020). “Every Strand Counts: Physics and Rendering Behind Frostbite’s Hair”
[Karis16] – Brian Karis (2016). “Physically Based Hair Shading in Unreal”
[Tafuri19] – Sebastian Tafuri (2019). “Strand-based Hair Rendering in Frostbite”

Q&A

END OF SLIDES