



AMD 



AUGMENTED HAIR IN DEUS EX UNIVERSE PROJECTS: TRESSFX 3.0

JASON STEWART (AMD)
URIEL DOYON (EIDOS-MONTRÉAL)
THURSDAY, MARCH 5, 2015

HIGH-LEVEL AGENDA



- ▲ Brief TressFX overview
- ▲ TressFX in the Dawn Engine
 - New engine from Eidos-Montréal
 - Cornerstone for all Deus Ex Universe projects
- ▲ TressFX 3.0 library
 - Update to AMD's TressFX example implementation
 - Maya plugin
 - Viewer and runtime library (with full source)
 - Fur support
 - Skinning
 - Latest optimizations



A BRIEF HISTORY



- ▲ TressFX began as a collaboration between AMD and Crystal Dynamics
- ▲ First used in Tomb Raider
 - PC and consoles
 - Set new quality bar for hair in games
- ▲ Optimized for AMD GCN architecture
 - Radeon HD 7000 or later
 - PS4, Xbox One
- ▲ AMD is now also collaborating with Eidos-Montréal
 - Started with Tomb Raider code
 - Improvements and additions integrated into Dawn Engine
 - Will be used in future Deus Ex Universe projects

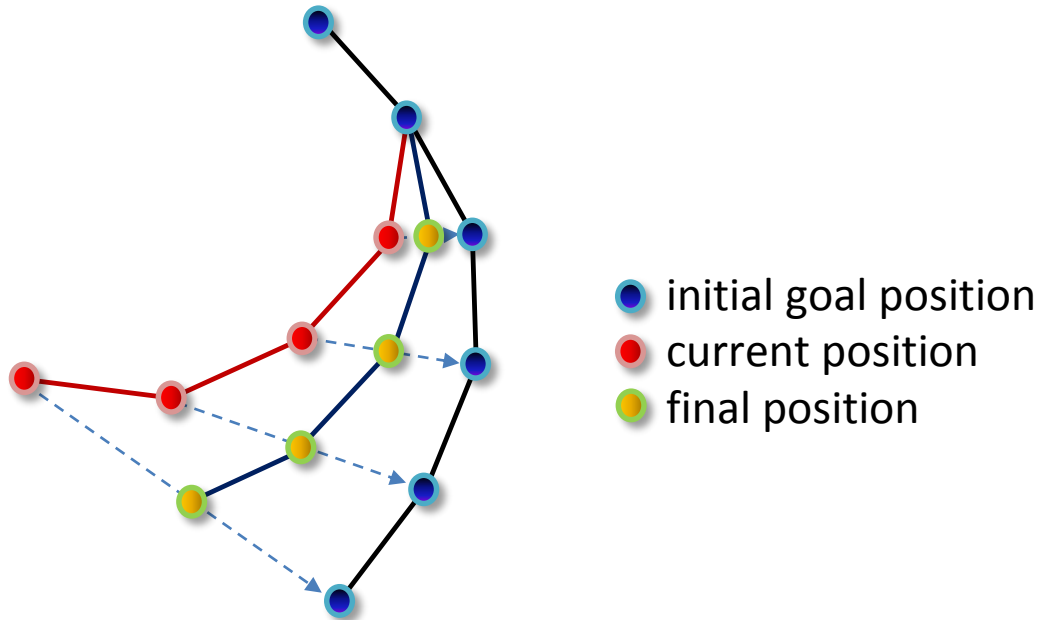
**CRYSTAL
DYNAMICS**
SQUARE ENIX



TRESSFX OVERVIEW



- ▲ Two parts to TressFX
 - Physics simulation on the GPU using compute shaders
 - High-quality rendering
- ▲ Simulates and renders individual hair strands

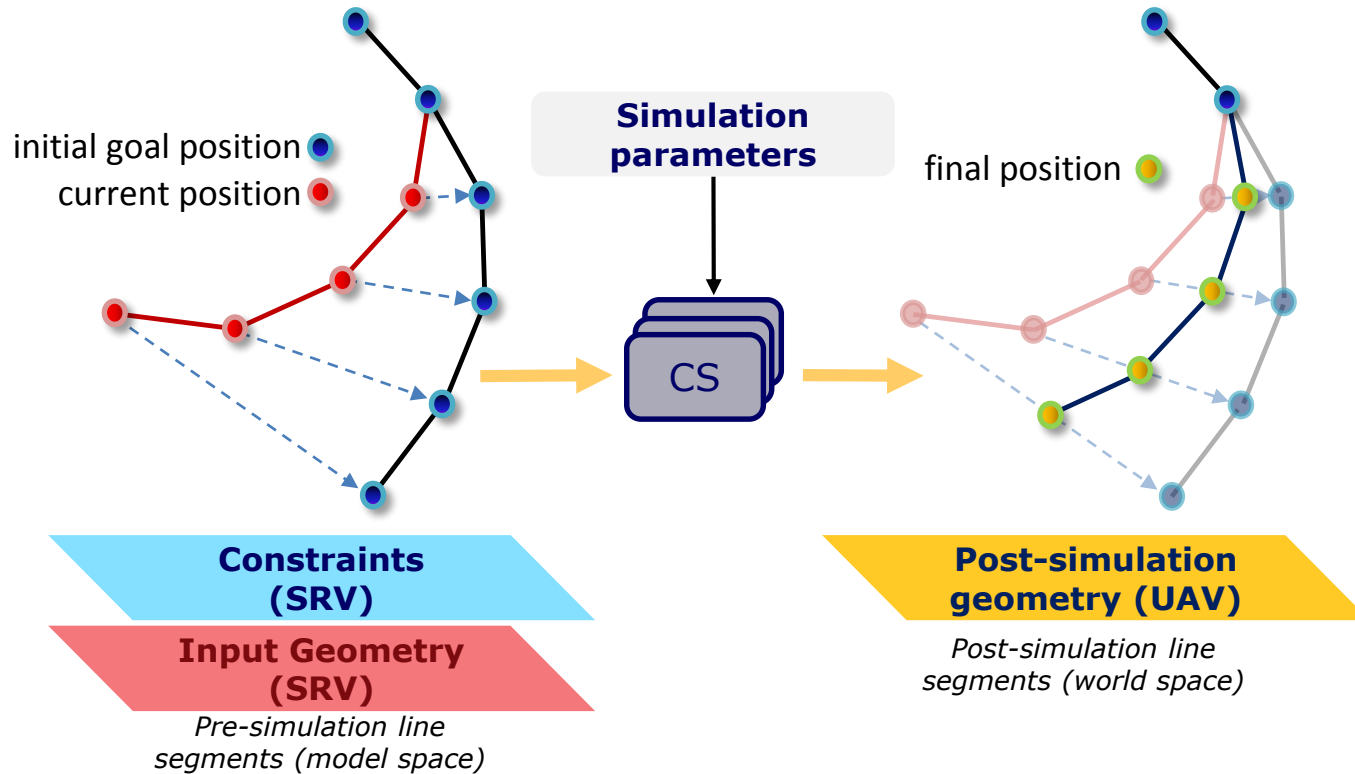


SIMULATION



RENDERING

TRESSFX SIMULATION



SIMULATION COMPUTE SHADERS

- Edge length constraint
- Local shape constraint
- Global shape constraint
- Model Transform
- Collision Shape
- External Forces (wind, gravity, etc.)

TRESSFX RENDERING



Good Lighting + Anti-Aliasing

+ Volume Shadows

+ Transparency

PRESENTATOR

URIEL DOYON



AUGMENTED HAIR IN DEUSEX UNIVERSE PROJECTS



PureHair.

PureHair Overview

- *Data Model*
- *Hair Strands*
- *Simulation*
- *Translucency*
- *Lighting*
- *Conclusion*

DATA MODEL

AUGMENTED HAIR IN DXU



PureHair: Data Model

For the *Dawn Engine*

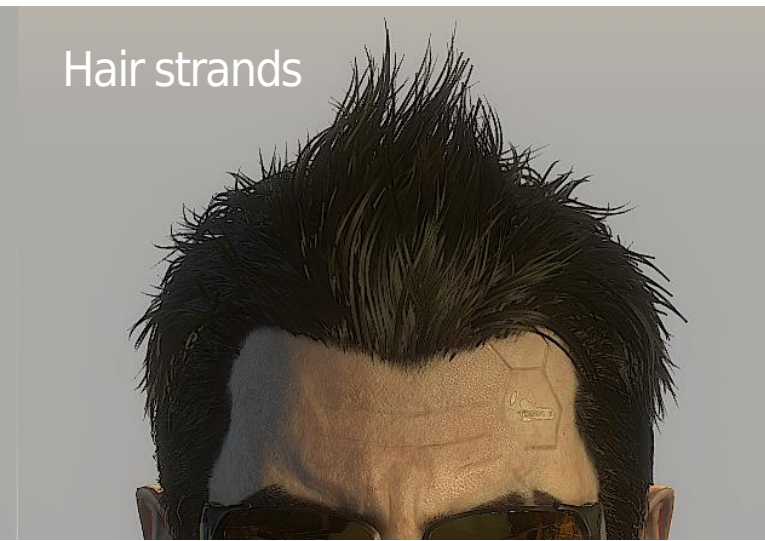
Asset Types:

- Textured head
- Hair meshes
- Hair strands

Textured head



Hair strands



Hair mesh



Hair strands



Both



Textured Head

A simple texture making the transition between the skin and the hairs.

Usefulness

- 1 Makes a smooth transition between the skin and other hair assets.
- 2 Fills any gap that could exist in other hair assets, giving the illusion that there are plenty of hairs on the head.
- 3 Low GPU cost because of opaque nature.

Limitations

- 1 Has no hair-like silhouette.
- 2 Unrealistic lighting.
- 3 100% rigid.

Hair Meshes

A mesh made of cards or banana leaves, using translucent textures.

Usefulness

- 1 Uses realistic hair lighting.
- 2 Handles small deformations well (from shader effect, simulation or animation).
- 3 Has moderate rendering cost, depending on overdraw.

Limitations

- 1 Requires sorting if used with alpha blended planes.
- 2 Heavy deformations will create unacceptable artefacts (like texture stretching and primitive popping).
- 3 Takes significant time to author.

Hair Strands

Billboarded cylinder-like hair strands used to render splines (TressFX).

Usefulness

- 1 Amazing look.
- 2 Handles any kind of deformations well.
- 3 Easy to author with current DCCs.

Limitations

- 1 Harder to texture because of how billboarding and stretching is handled.
- 2 Does not provide any surface normal for light attenuation.
- 3 Higher GPU cost.

HAIR STRANDS

AUGMENTED HAIR IN DXU

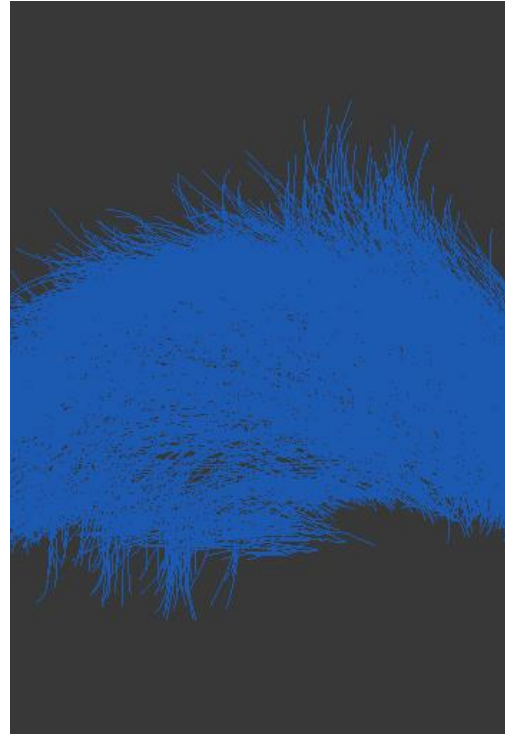


PureHair: Hair Strands

Content Creation

Haircuts can be authored by DDC tools like Hair and Fur, or Shave and Haircut.

Those tools will generate wisps, that can be converted into splines (and then rendered as strands).



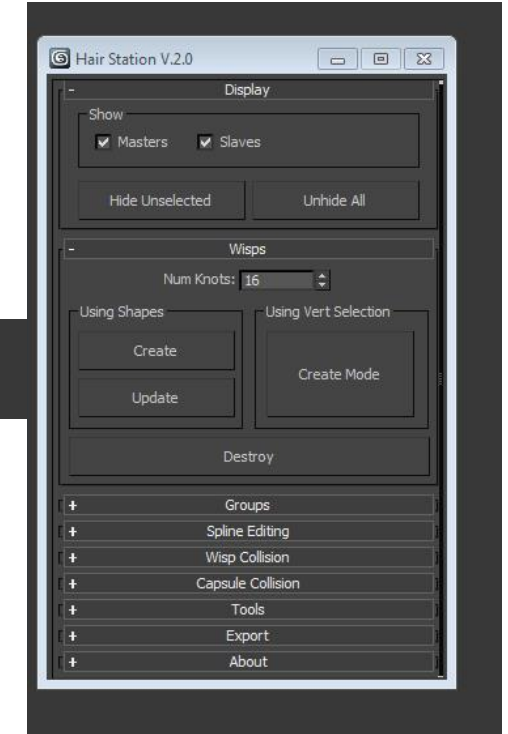
Authoring Tool

Hair Station

Our plugin for both Maya and 3DS Max.

Accelerates repetitive tasks.

Exports metadata for splines and wisps used for simulation and rendering.



Plugin for 3D softwares

PureHair: Hair Strands

Wisp Data

Wisp data is required at runtime and must be exported into the engine:

Keep the DCC wisps if available, or make new ones by grouping splines manually.

For each wisp, we generate a master strand.



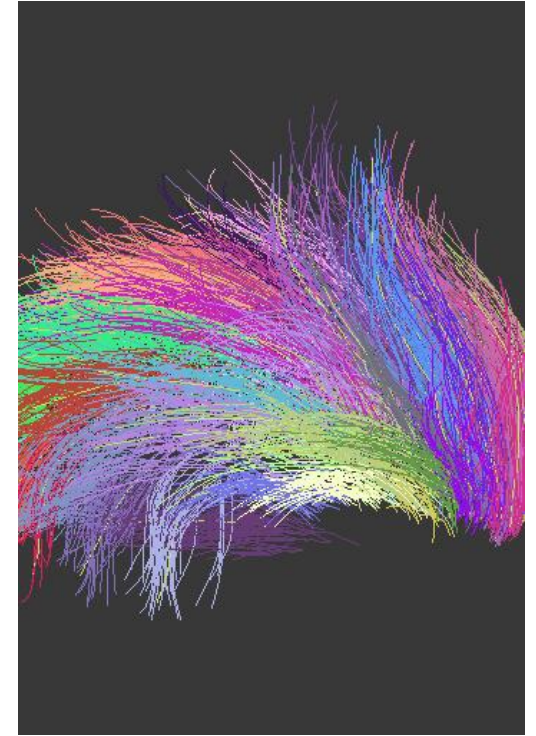
Master strands (one per wisp)

Master to slave strands skinning (also in *TressFX 3.0*)

Much less strands to simulate

Allows shape preserving LODs

Prevents simulation from breaking too much the haircut (from gravity & displacements)



Slave strands (many per wisp)

PureHair: Hair Strands

Shape preserving LODs:

- Progressively reduce the number of strands within each wisp while increasing each strand width.
- Since LOD is applied per wisp, the overall shape is preserved.



LOD 0



LOD 1



LOD 2



LOD 3

SIMULATION

AUGMENTED HAIR IN DXU



PureHair: Simulation

Fixed length enforcing (à la Muller):

- Preserves the length of each strands.
- Iterates on all the vertex of a strand, starting from the root vertex.

Variable number of vertex per master and slave strands:

- Between 4 to 32 vertex per strands.
- Each master strand within a group must have the same number of vertices.
- Slave strands can have any number of vertices.



Master Slave



Hairs When Moving



Effect Of Gravity



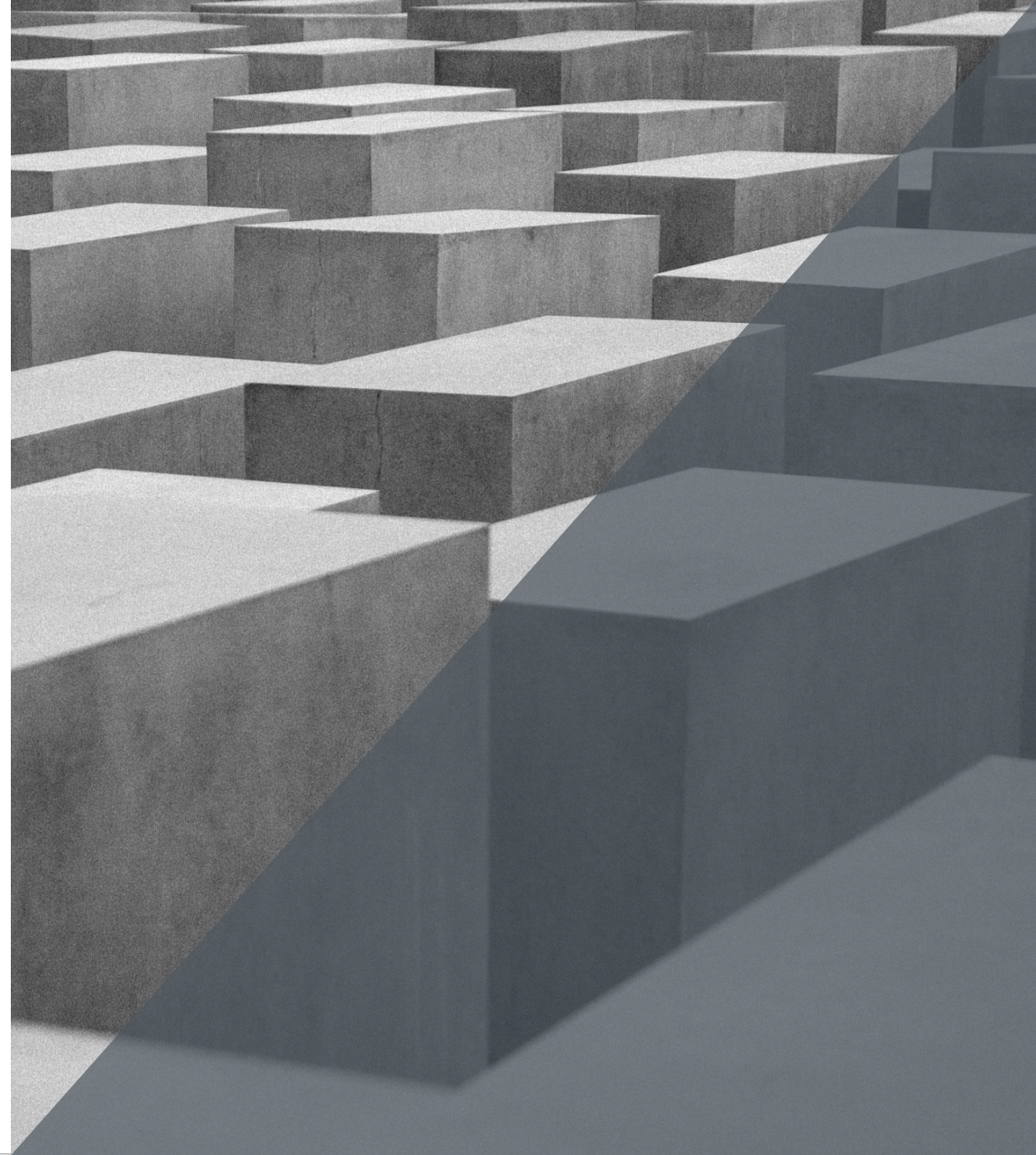
PureHair: Simulation

Wind:

- Build a cone around the wind direction.
- Each hair has a slightly different wind vector, that also changes between frame.
- Makes the hairs spread nicely.

World space simulation:

- World space simulation create issues on fast moving objects.
- We scale the root motion delta position and feed it to the simulation.
- It is then applied to the previous frame positions, to nullify partially the motion.



PureHair: Simulation

Wisp Interactions:

- Allows slave strands to be affected by surrounding master strands if they are close enough.
- Stateless approach simplifies LOD handling and view frustum changes.
- We tried a dynamic implementation where we interpolated each slave toward its closest neighbor, but it made unnatural transitions.
- We ended up choosing a static approach where each slave is only allowed to move toward its closest master strand in the reference position.



TRANSLUCENCY

AUGMENTED HAIR IN DXU



PureHair: Translucency

Importance of translucency:

- Research at AMD confirms that this is the most important element to get right.
- Alpha blend reflects the thinness of hair strands.
- Per pixel linked list OIT has been proven to be an efficient and good solution for hair assets.

Issues with translucency:

- Triangle/fragment sorting is harder to solve with hair assets than with other particles effects.
- Lighting for translucency usually involves less optimal engine paths.
- Translucency can have quality issues with depth based techniques like motion blur and depth of field.

PureHair: Translucency

Opaque rendering:

- Fast
- Noisy



PureHair: Translucency

Opaque with AA post-process:

- Good at close range.
- Does not fix the blob effect at medium range.
- [Video](#)

Alpha to coverage (multisampling):

- Research done at AMD for *TressFX* has shown that it is slower than PPLL OIT, mostly because alpha to coverage disables early depth.
- Does not accumulate translucency correctly.
- Only supports a small number of alpha values.

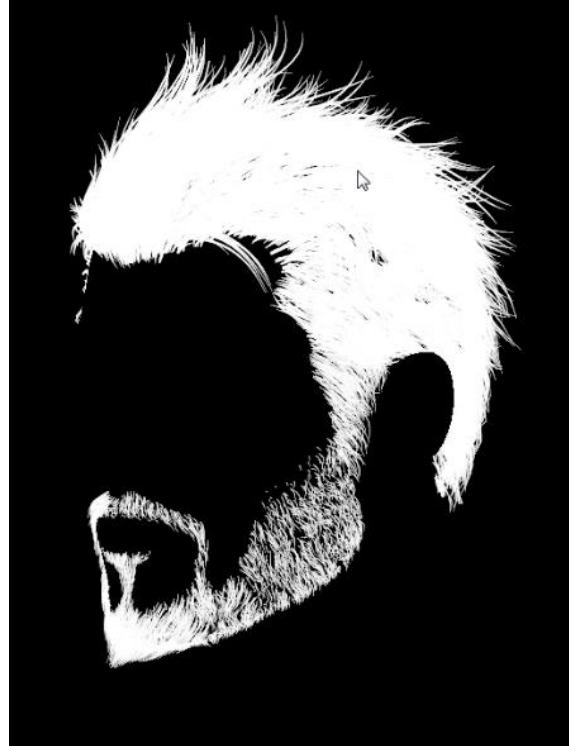
Weighted Blended Order Independent Translucency:

- Only works well with low opacity primitives ($A < .3$)

PureHair: Translucency

Opaque with translucency accumulation:

- Requires 2 render passes.
- Noisy inner hairs.



PureHair: Translucency

Per Pixel Linked List (order independent translucency):

- Requires space for the node buffer, around 200Mb in 1080p with 12 bytes per node and $K_{\text{overdraw}}=8$.
- More overdraw will make the resolve color more expensive.



PureHair: Translucency

PPLL node with deferred lighting:

- Tangent & Normal
- Color, Alpha
- Spec noise & intensity
- Lighting settings id
- Depth & Next entry
- 16 bytes per node

PPLL node with forward lighting:

- Color, Alpha
- Depth & Next entry
- 12 bytes per node

$$\text{Size}_{PPLL} = \text{Size}_{Node} \times \text{Resolution} \times K_{overdraw}$$

PureHair: Translucency

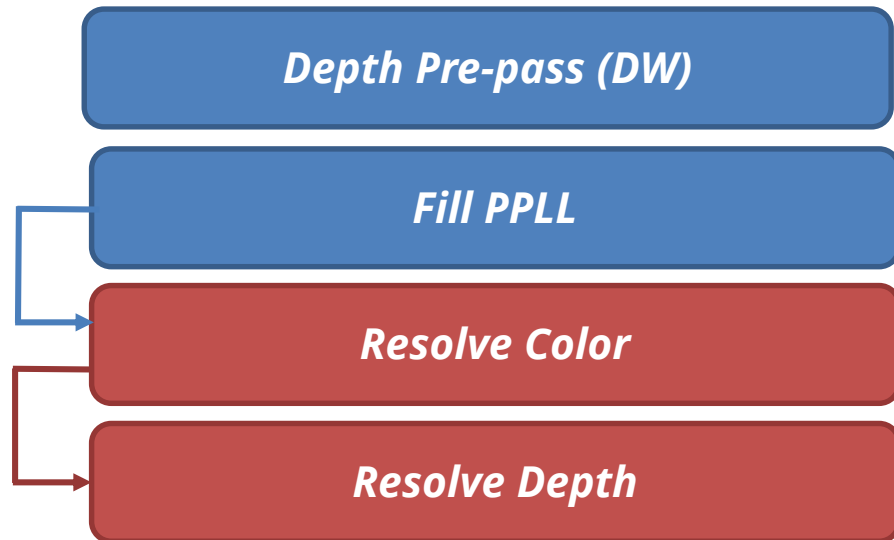
Handling overflow with the PPLL node buffer:

- Do nothing.
- Use an adaptive alpha threshold to progressively blend toward opaque rendering.
- Render and resolve the screen in tiles.
- Render in multiple passes, where each pass starts where the previous pass left.

PureHair: Translucency

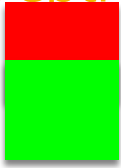
PPLL with depth pre-pass (also in *TressFX* 3.0):

- Optimizes the PPLL implementation if the depth pre-pass cost benefits the other passes.
- Usually a win when using LODs.
- Allows smaller K_{overdraw} for the same quality (~ 3).



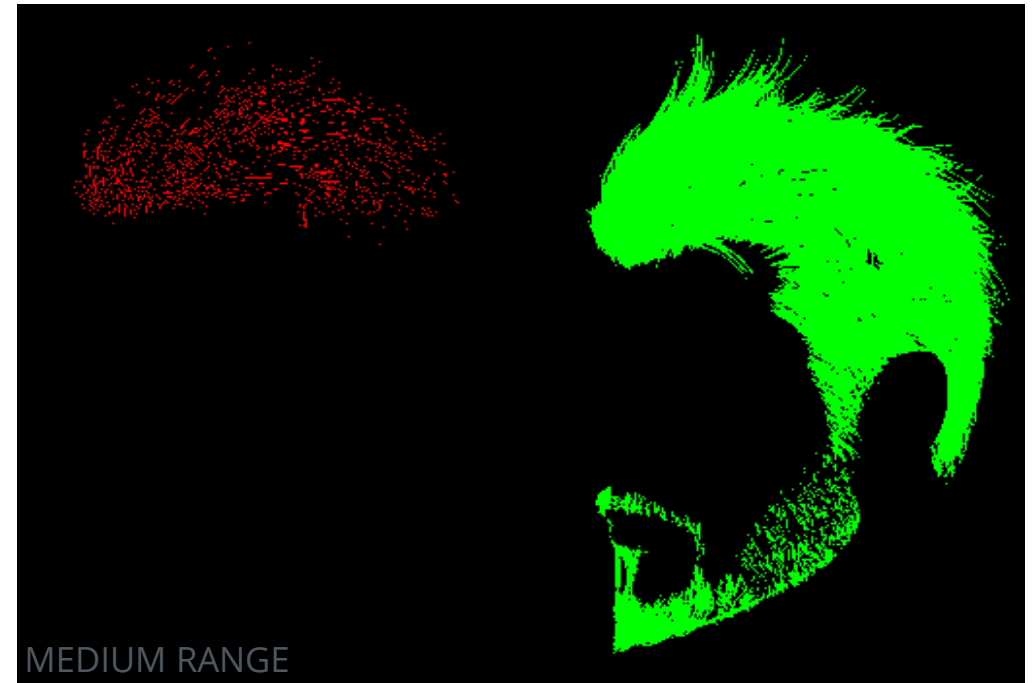
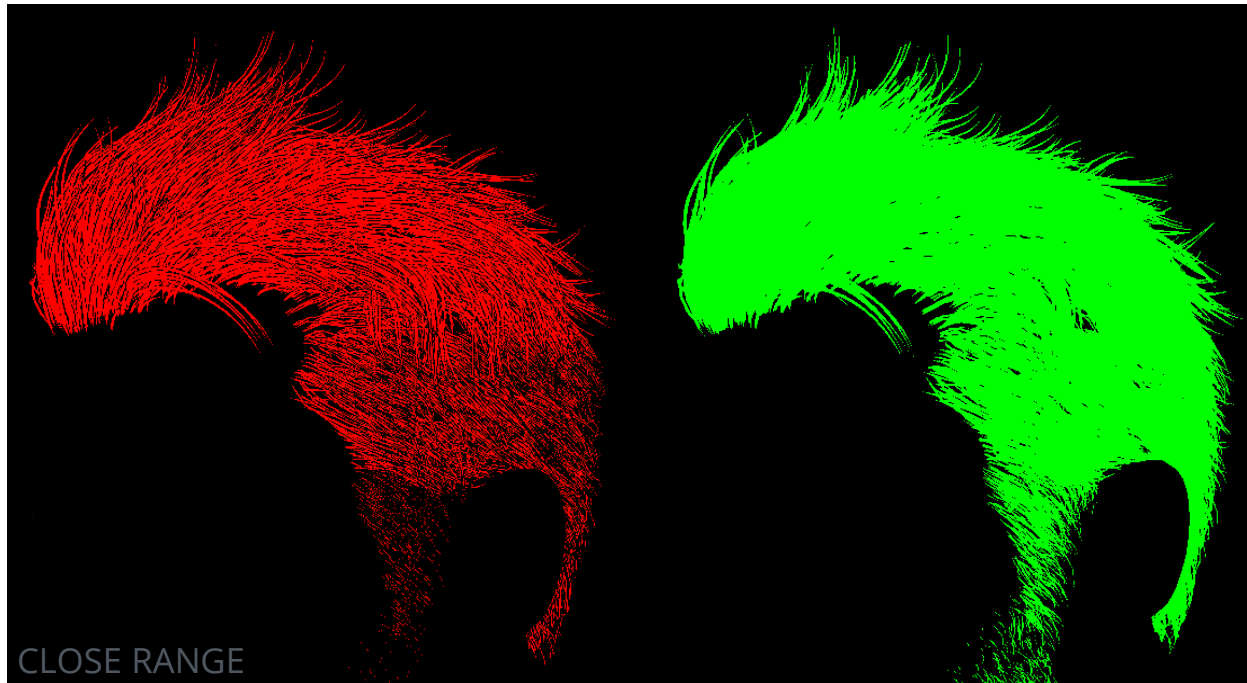
PureHair: Translucency

Depth pre-pass and resolve depth at different distances:



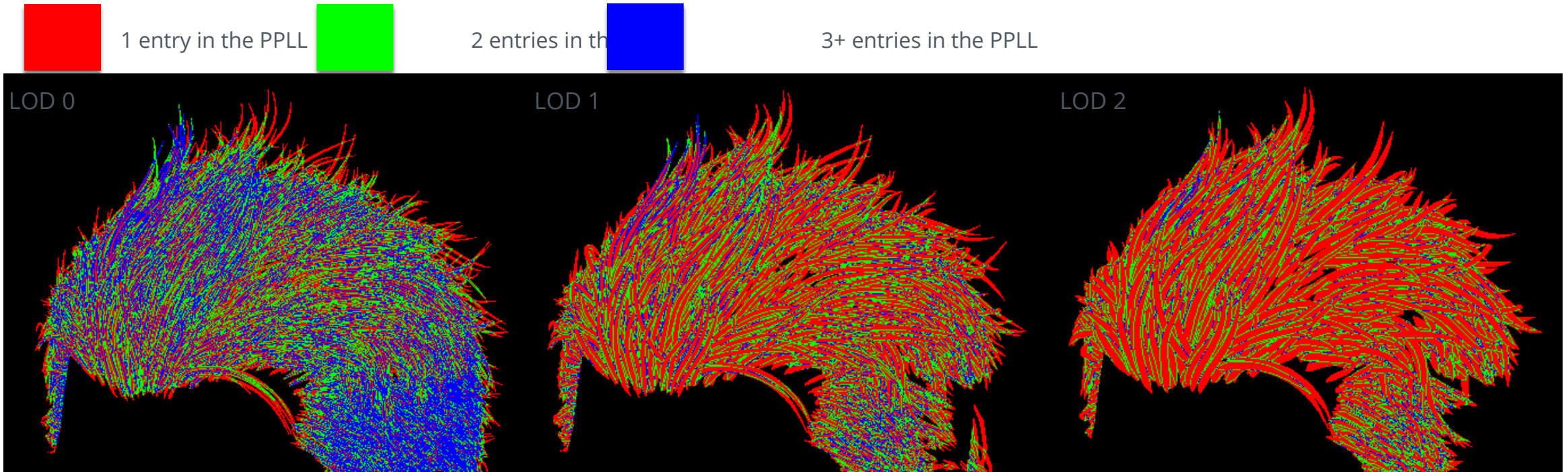
Depth pre-pass writes depth where $A=1$

Resolve depth writes depth where $A_{cum} > .25$



PureHair: Translucency

PPLL usage and LODs with the depth pre-pass:



With LODs, blending the first 2 fragments is enough for the color.

For alpha, it is still preferable to use the cumulative result.

PureHair: Translucency

Depth peeling:

- Use atomics to compute closest depth 0 and 1
- Fill nodes only if they are entry 0 or 1
- Similar performance than PPLL with depth pre-pass
- Half memory cost with no possible PPLL overflow



PureHair: Translucency

Depth peeling pseudo code:

```
[earlydepthstencil]
DepthPeeling()
{
    ...
    uint uDepth = asfloat(fDepth);

    If (A > .02)
    {
        uint uDepth0 = 0, uDepth1 = 0;
        Depth01UAV.InterlockedMin(uAddress, uDepth, uDepth0);

        // If the first fragment is opaque, we don't need the second.
        uDepth = alpha > .98f ? uDepth : max(uDepth, uDepth0);

        Depths01UAV.InterlockedMin(uAddress + 4, uDepth, uDepth1);
    }
    return 1 - A; // Blend mode : multiply
}
```

PureHair: Translucency

Depth peeling pseudo code:

```
ResolveDepth() : DEPTH
{
    ...
    return A > 0.75 ? (uDepth1 ? uDepth1 : uDepth0) : MAX_FLOAT;
}
```

```
[earlydepthstencil]
FillNode()
{
    ...
    uint uDPIndex = (uDepth == uDepth0) ? 0 :
                    ((uDepth == uDepth1) ? 1 : 0xFF);

    if (uDPIndex <= 1 && A > .02)
    {
        StoreFragment(uAddress + uDPIndex, ...);
    }
}
```

PureHair: Translucency

Depth peeling VS Per pixel linked list with depth pre-pass:

- Performance comparison on a Radeon HD 7970, using Adam Jensen, cinematic LOD:

	FullScreen	Near	Far *
PPLL & Depth Pre-pass	1.774	1.004	0.852
Resolve PPLL	2.059	0.422	0.146
Depth Peeling	1.452	0.892	0.833
Resolve DP	1.907	0.338	
0.098			

* In milliseconds

Note: Depth Peeling is faster at close range in part because the PPLL depth pre-pass does not have early depth, and also because data access have better locality. Still this is very hardware specific.

LIGHTING

AUGMENTED HAIR IN DXU



PureHair: Lighting

Lighting probes:

- Evaluates the lighting of some lights as constant terms over the haircut bounds.
- Stores directional intensities using cubemaps or spherical harmonics.
- Reduces the lighting cost significantly since most lights fall into this strategy.

Tangent based lighting:

- Use tangent based lighting, instead of normal based lighting.
- Especially visible in the diffuse term because strands are billboarded.

PureHair: Lighting

Normal based occlusion:

- Need directional occlusion from surrounding hairs and geometry.
- Shadowmaps help but most lights don't have them, and their resolution and precision can be an issue for hairs.
- For hair strands, we use the normal from a sphere mapped on the character's head.



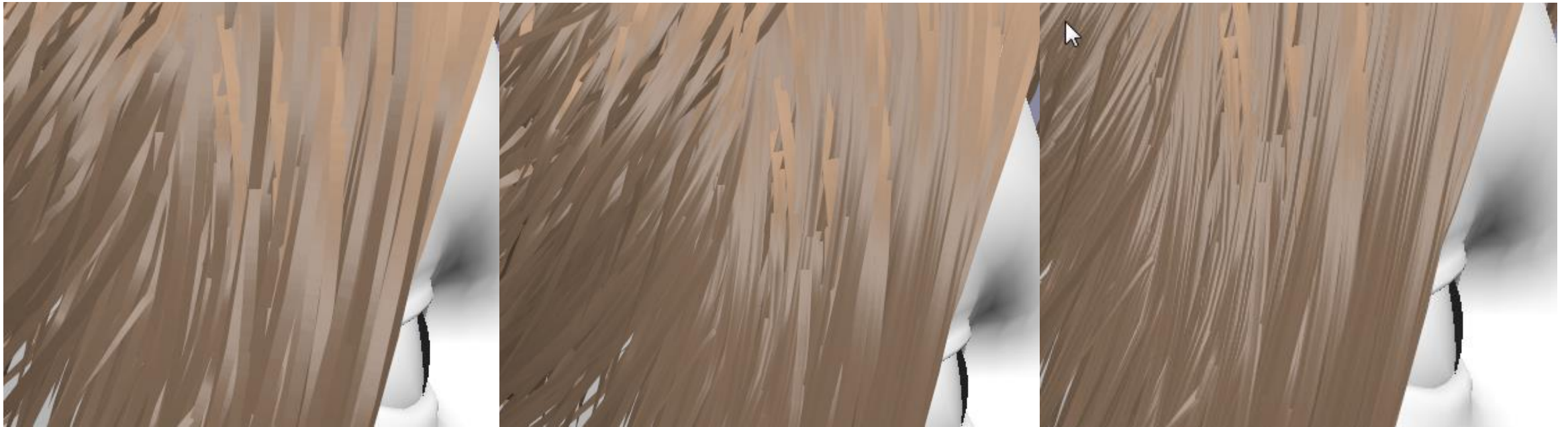
PureHair: Lighting

Dual specular highlight:

- Most significant lighting feature of the hair lighting model.
- Adding tangent noise will break the "audio tape" look.

Fresnel factor:

- More important than back scattering.
- Extra term in the lighting equation.



CONCLUSION

AUGMENTED HAIR IN DXU



PureHair: Conclusion

LABS conclusion after experimentation:

- Translucency on hairs should be limited to tips and edges for both hair meshes and hair strands.
- For hand strand assets, using LODs is of critical importance to ensure that most strands will be several pixels in width.
- With opaque-ish hairs, only the first 2 samples are significant for the color, allowing K_{overdraw} to be set as low as 3 (for PPLL budget).
- This also allows a novel depth peeling approach, which takes half memory cost compared to equivalent PPLL implementation.
- Lighting should be tangent based, but normals should still be used for computing occlusion factors.
- *TressFX* looks great and can be used on all platforms.

SPECIAL THANKS

JEAN-NORMAND BUCCI
LABS R&D | DIRECTOR

ANTON MICHELS
LABS R&D | 3D PROGRAMMER

NICHOLAS FRÉCHETTE
LABS R&D | SR. SYSTEM PROGRAMMER

PETER SIKACHEV
LABS R&D | 3D PROGRAMMER

FRANCIS MAHEUX
LABS R&D | SR. TECHNICAL ARTIST

SAMUEL DELMONT
LABS R&D | 3D PROGRAMMER

NICOLAS LONGCHAMPS
LABS R&D | SR. TECHNICAL ARTIST

DAVID GALLARDO
DXU | PROGRAMMING DIRECTOR

ROMAIN CHAGOT
DXU | SR. SYSTEM PROGRAMMER

JACQUE CHOI
DXU | SR. CHARACTER ARTIST

EVGENII FOKIN
DXU | SR. CHARACTER ARTIST

LAURA GALLAGHER
DXU | SR. CHARACTER ARTIST

MORTEN MIKKELSEN
CRYSTAL DYNAMICS | PRINCIPAL SOFTWARE ENGINEER

SZE JONES
CRYSTAL DYNAMICS | SR. CHARACTER ARTIST

SPECIAL MENTIONS

DAVID ANFOSSI
JONATHAN-JACQUES BELLETÈTE
JULIEN BOUVRAIS
RÉMI DRIANCOURT
MARTIN DUBEAU

BRIAN HORTON
DARRELL GALLAGHER
JOSHUA SALEM
GARY SNETHEN
JASON STEWART

NICOLAS THIBIEROZ
ISABELLE TREMBLAY
ANDRE VU
FREDERIC RICHARD



**CRYSTAL
DYNAMICS**

SQUARE ENIX®

AMD



TRESSFX 3.0 LIBRARY

TRESSFX SAMPLE HISTORY



▲ TressFX 1.0

- Example implementation (with full source)

▲ TressFX 2.x

- Simulation performance improvements
 - Master and slave strands
 - Compute shader optimizations
 - 1.6 ms → 0.3 ms
- Rendering performance improvements
 - Deferred rendering
 - Distance adaptive LOD
 - Pixel shader optimizations
 - 1.7 ms → 1.2 ms
 - With LOD, 0.3 ms (or lower) at a distance



TRESSFX 3.0 PREVIEW

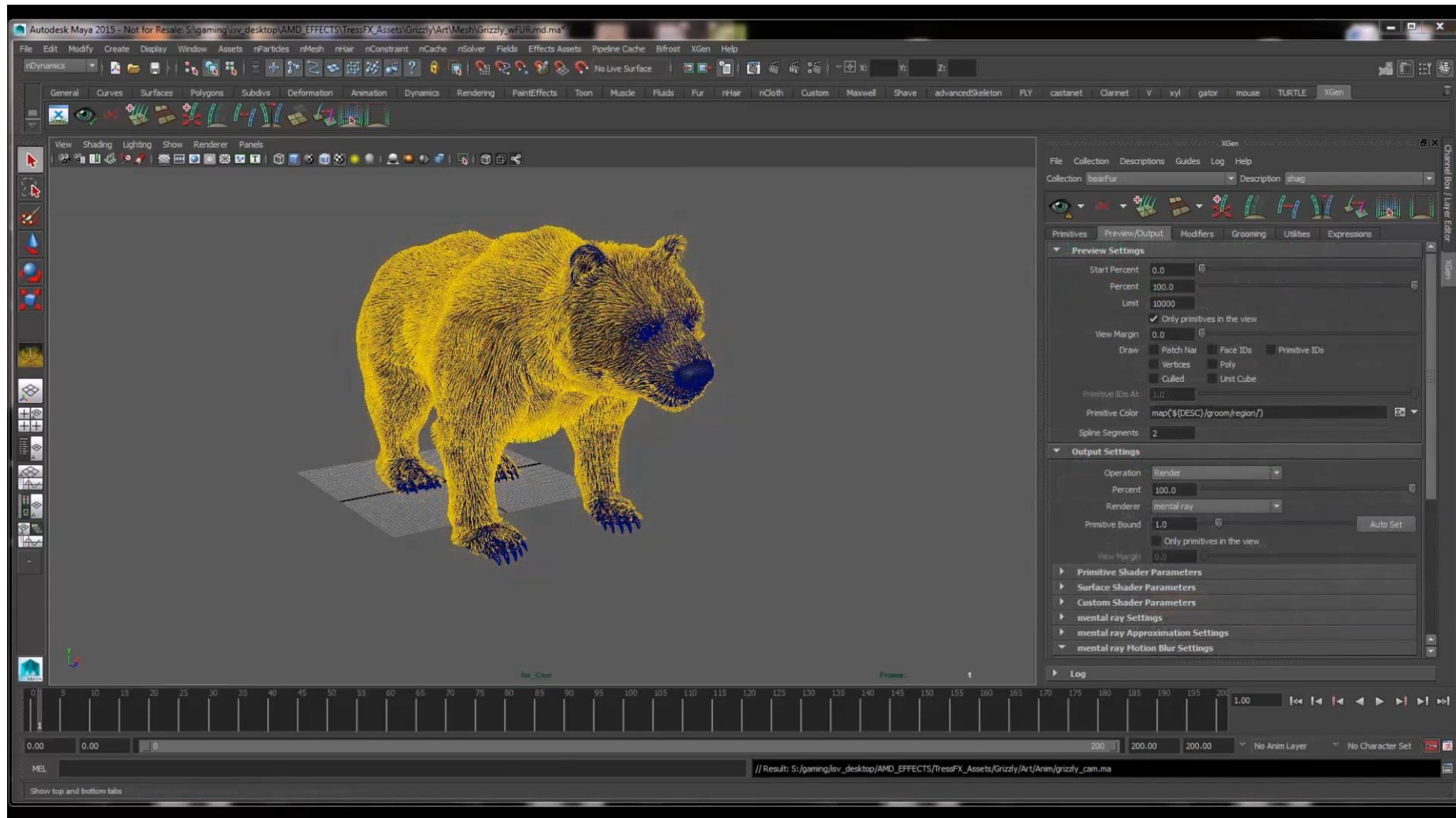


▲ TressFX 3.0

- Maya plugin
- Viewer and runtime library (with full source)
- Fur support
- Skinning
- Latest optimizations
- Free
- Coming soon



TRESSFX 3.0 MAYA PLUGIN



TRESSFX 3.0 VIEWER AND RUNTIME LIBRARY



- ▲ Viewer provided
 - Preview TressFX assets
- ▲ Runtime reorganized into a library
 - Easier integration
- ▲ Full source
 - Free
 - No black boxes

```
// TressFXRender.cpp
//
// Renders the hair in two passes. The first pass fills an A-buffer by rendering the
// hair geometry into a per-pixel linked list which keeps all of the overlapping fragments.
// The second pass renders a full screen quad (using a stencil mask set in the first pass
// to avoid unnecessary pixels) which reads fragments from the per-pixel linked list
// and blends the nearest k fragments (K-buffer) in back to front order.
//
//-----
void TressFXRender::RenderHair(ID3D11DeviceContext* pd3dContext)
{
    // Get original render target and depth stencil view
    TIMER_Begin( 0, L"ABufferFill" );
    ID3D11RenderTargetView* pRTV = DXUTGetD3D11RenderTargetView();
    ID3D11DepthStencilView* pDSV = DXUTGetD3D11DepthStencilView();

    // render hair

    const UINT dwClearDataMinusOne[1] = {0xFFFFFFFF};
    pd3dContext->ClearUnorderedAccessViewUint(m_pHeadPPLL_UAV, dwClearDataMinusOne);

    // Clear stencil buffer to mask the rendering area
    // Keep depth buffer for correct depth and early z
    pd3dContext->ClearDepthStencilView(pDSV, D3D10_CLEAR_STENCIL, 1, 0);

    ID3D11UnorderedAccessView* pUAV[] = {m_pHeadPPLL_UAV, m_pPPLL_UAV, NULL, NULL, NULL, NULL, NULL};
    UINT pUAVCounters[] = { 0, 0, 0, 0, 0, 0, 0 };
    pd3dContext->OMSetRenderTargetsAndUnorderedAccessViews(1, &pRTV, pDSV, 1, 7, pUAV, pUAVCounters);

    // disable color write if there is no need for fragments counting
    pd3dContext->OMSetBlendState(m_pColorWritesOff, 0, 0xffffffff);

    // Enable depth test to use early z, disable depth write to make sure required layers won't be clipped out in early z
    pd3dContext->OMSetDepthStencilState(m_pDepthTestEnabledNoDepthWritesStencilWriteIncrementDSS, 0x00);

    // Pass 1: A-Buffer pass
    if(m_hairParams.bAntialias)
    {
        if(m_hairParams.strandCopies > 1)
            RenderHairGeometry(pd3dContext, m_pVSRenderHairAAstrandCopies, m_pPSABuffer_Hair, m_hairParams.density, false, m_hairParams.strandCopies);
        else
            RenderHairGeometry(pd3dContext, m_pVSRenderHairAA, m_pPSABuffer_Hair, m_hairParams.density, false, 1);
    }
    else
    {

```

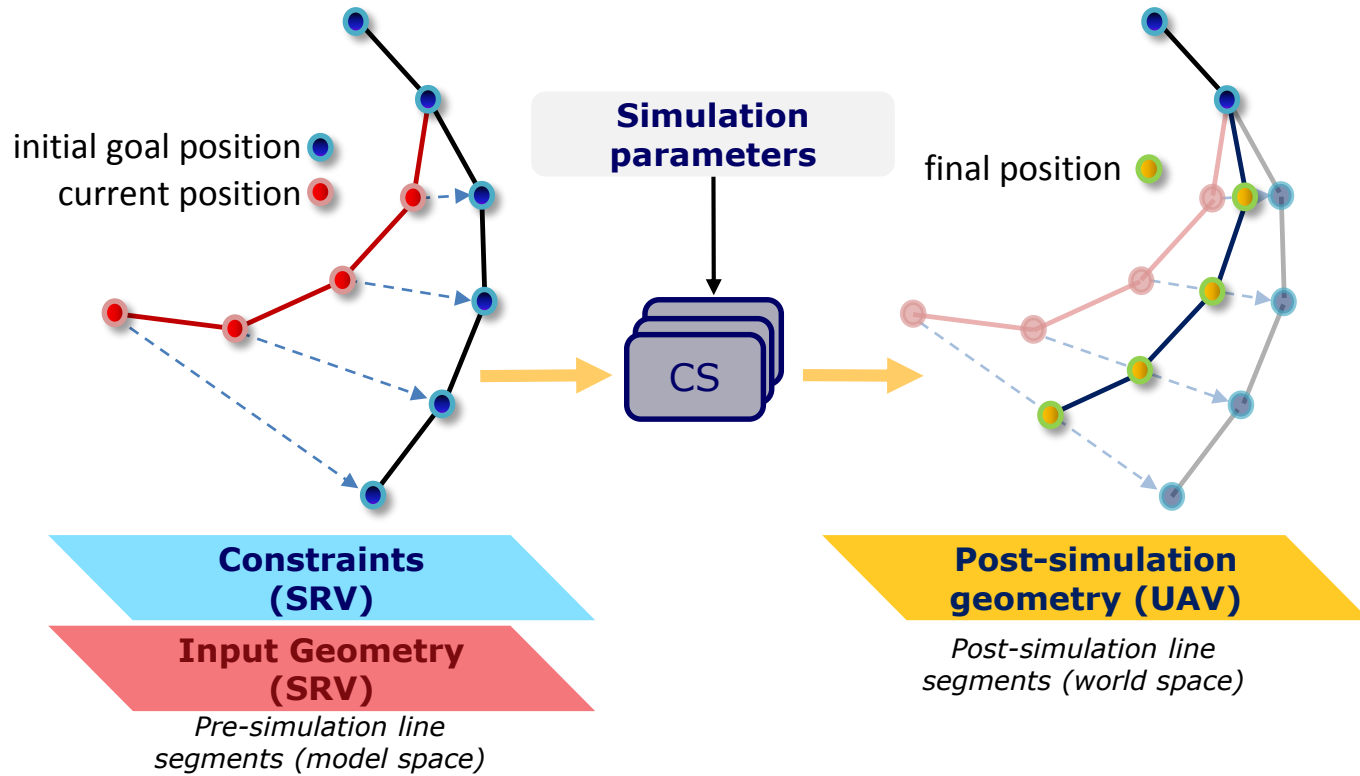
TRESSFX 3.0 FUR SUPPORT



- ▲ Core principles are the same
 - Simulation on GPU using compute
 - Good lighting
 - Anti-aliasing
 - Volume shadows
 - Transparency
- ▲ Texture coordinates
 - Allows variation in fur color
- ▲ Skinning
 - A simple head transform was enough for human hair
 - Fur requires skinning support



TRESSFX 3.0 FUR SIMULATION



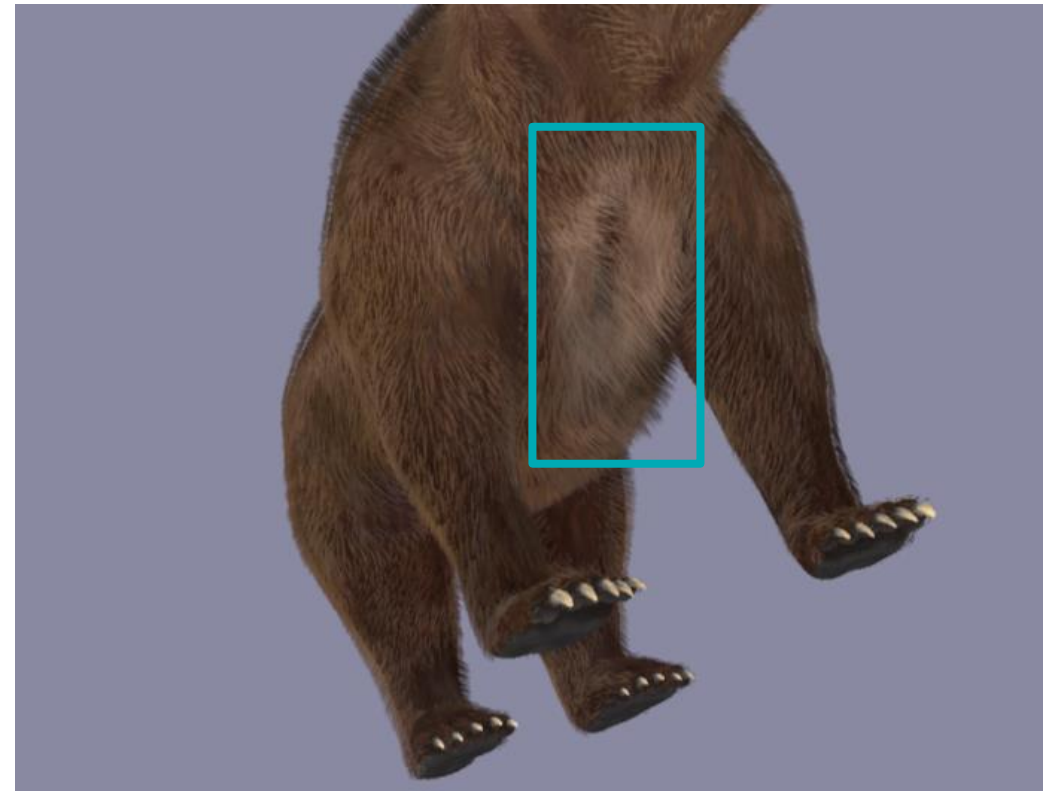
SIMULATION COMPUTE SHADERS

- Edge length constraint
- Local shape constraint
- Global shape constraint
- Model Transform
- Collision Shape
- External Forces (wind, gravity, etc.)

TRESSFX 3.0 TEXTURE COORDINATES

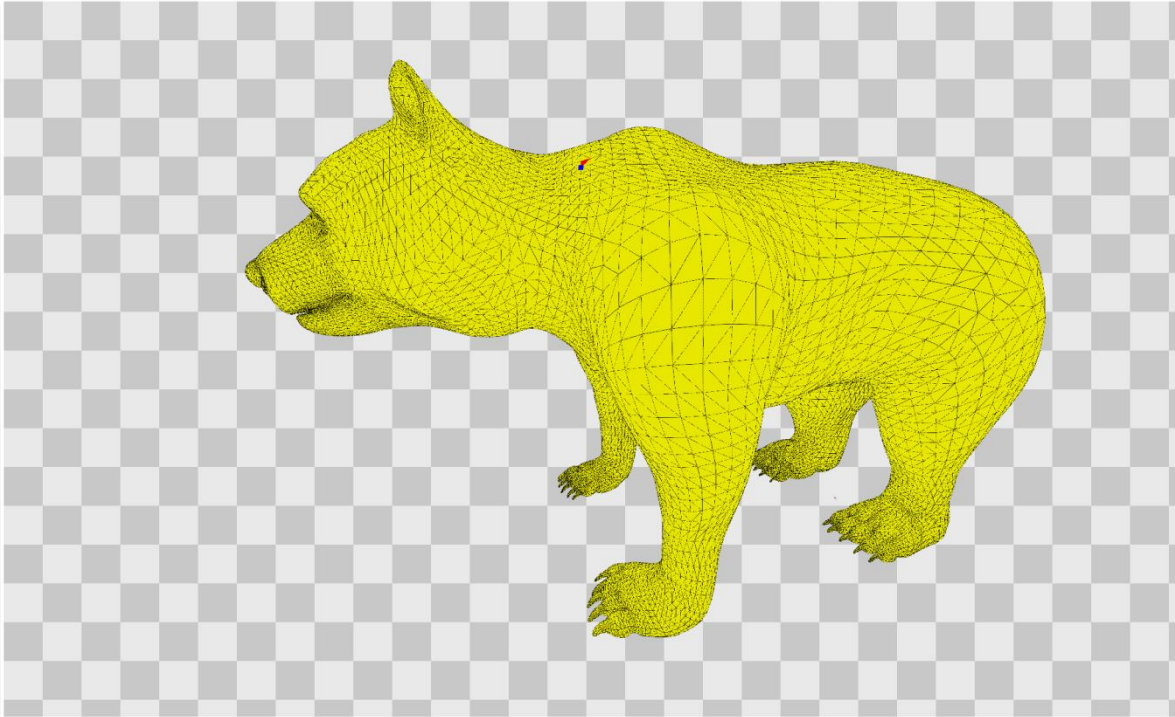


TEXTURED MESH

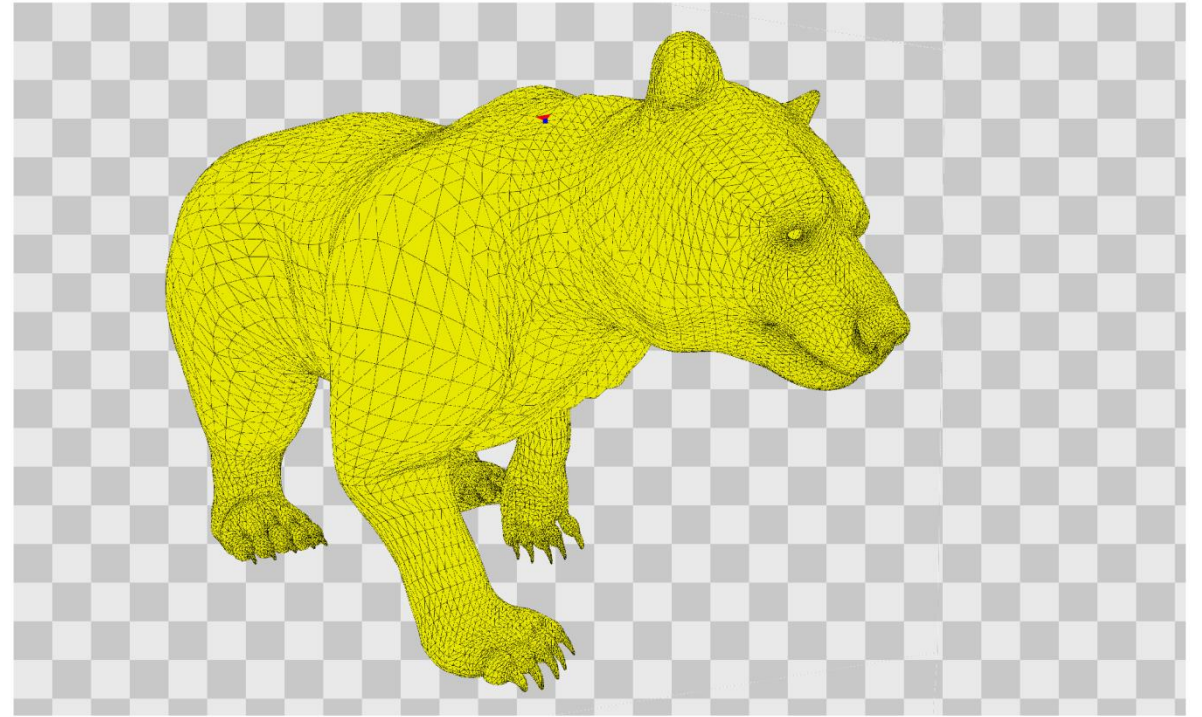


FUR PICKS UP COLOR VARIATION FROM TEXTURE

TRESSFX 3.0 SKINNING



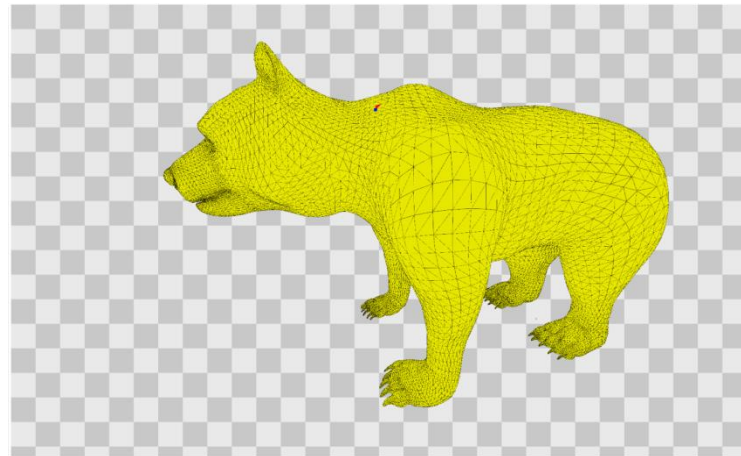
BIND POSE



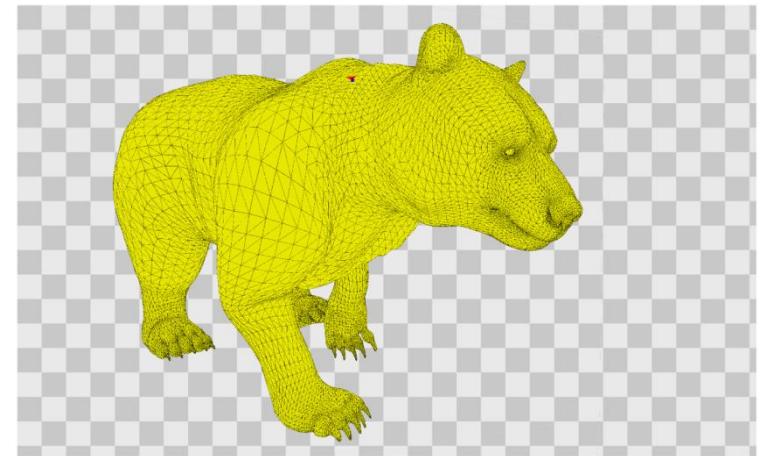
SKINNED

TRESSFX 3.0 SKINNING

- ▲ How to get skinned vertex position data to TressFX library?
- ▲ Up to you
- ▲ TressFX viewer currently uses Stream Out
 - DirectX 11
 - No geometry shader
 - See “Getting Started with the Stream-Output Stage” on MSDN
- ▲ UAVs at vertex shader stage
 - DirectX 11.1+
 - DirectX 12

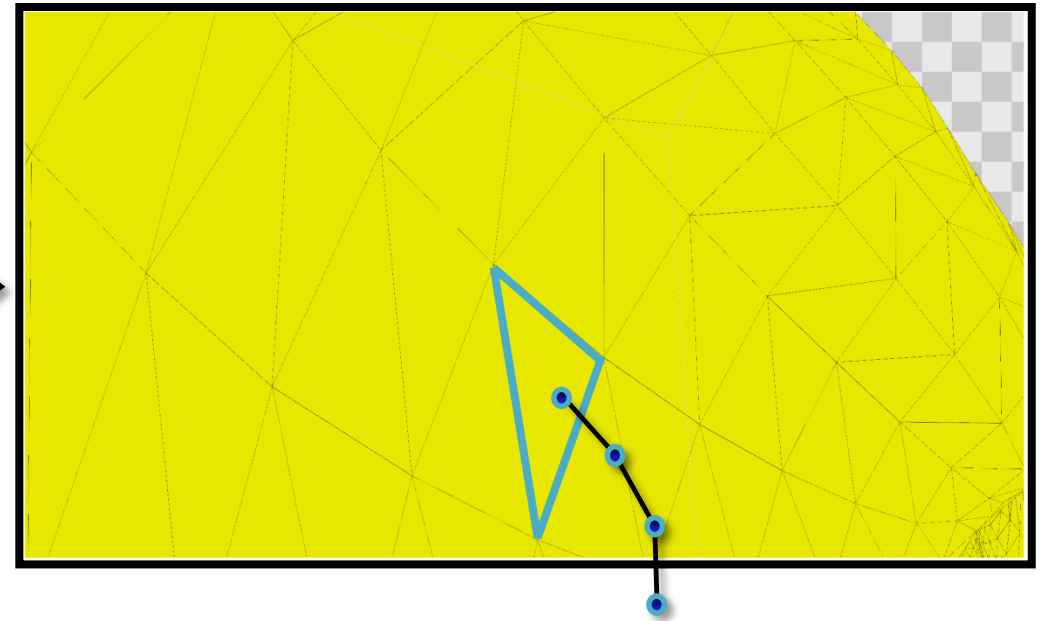
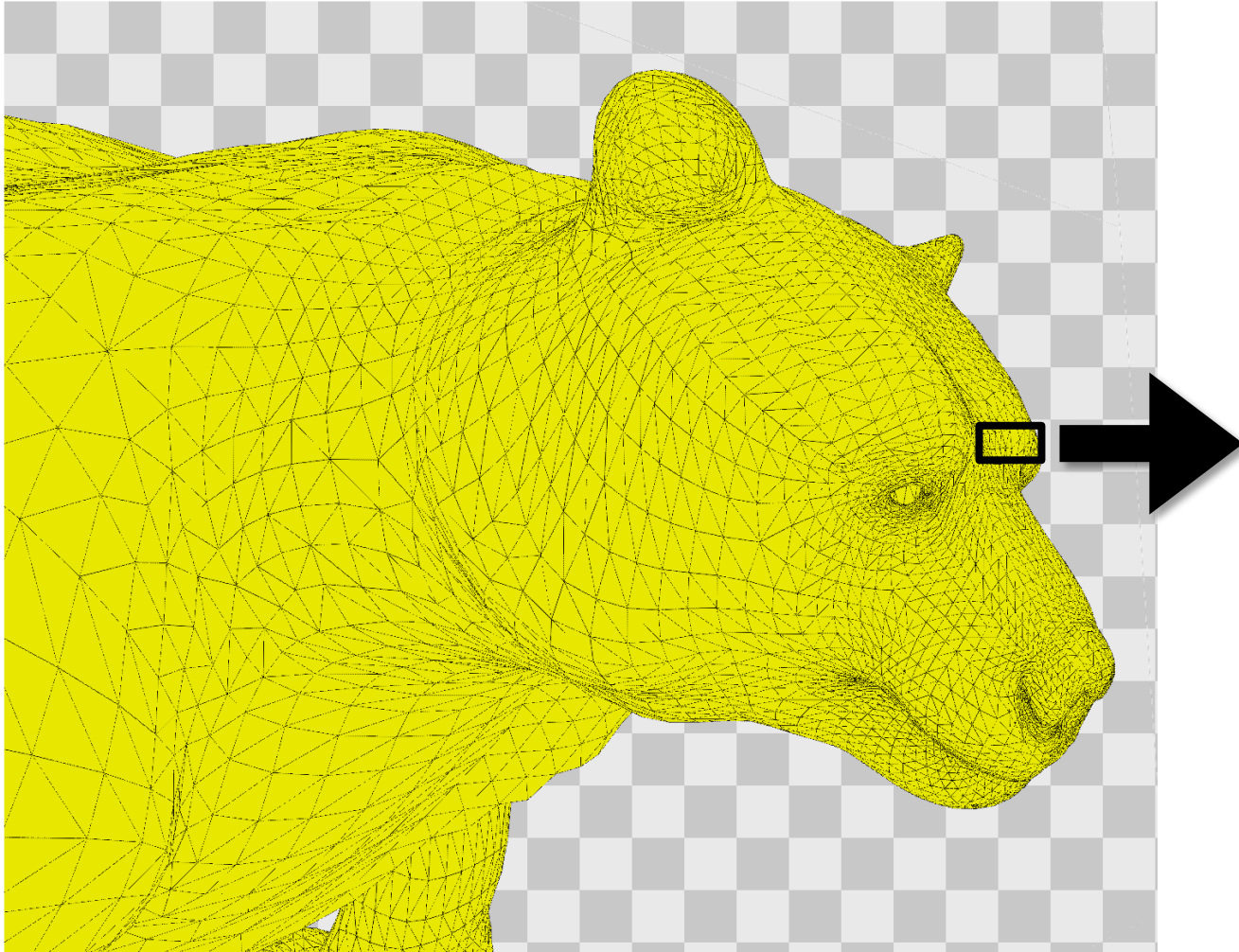


BIND POSE



SKINNED

TRESSFX 3.0 SKINNING

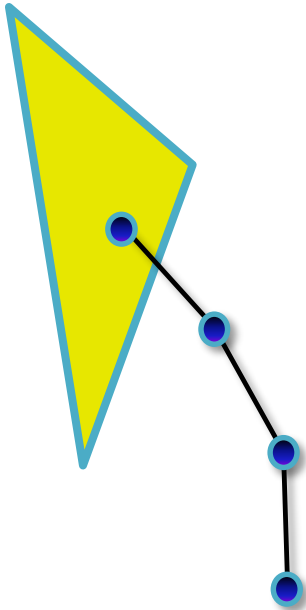


- ▲ Hair-to-mesh mapping
 - Exported from Maya plugin

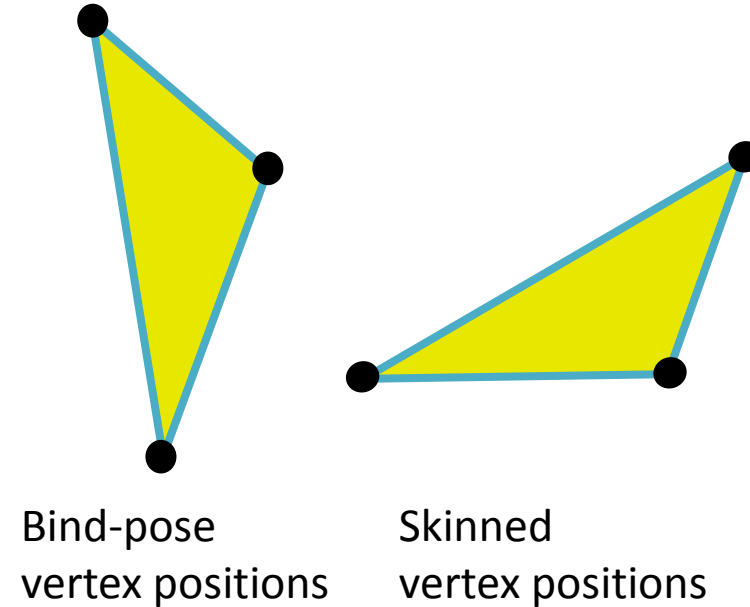
TRESSFX 3.0 SKINNING



- ▲ Use hair strand index to get triangle index



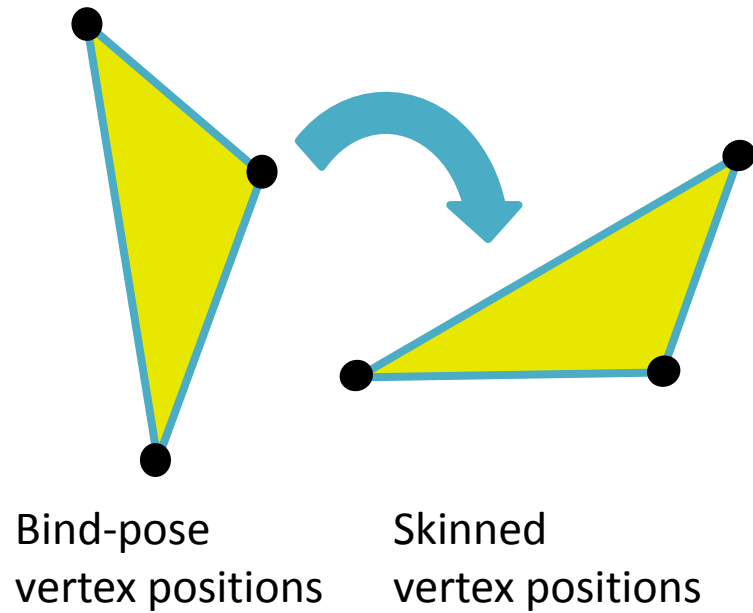
- ▲ Use triangle index to get bind-pose verts and skinned verts



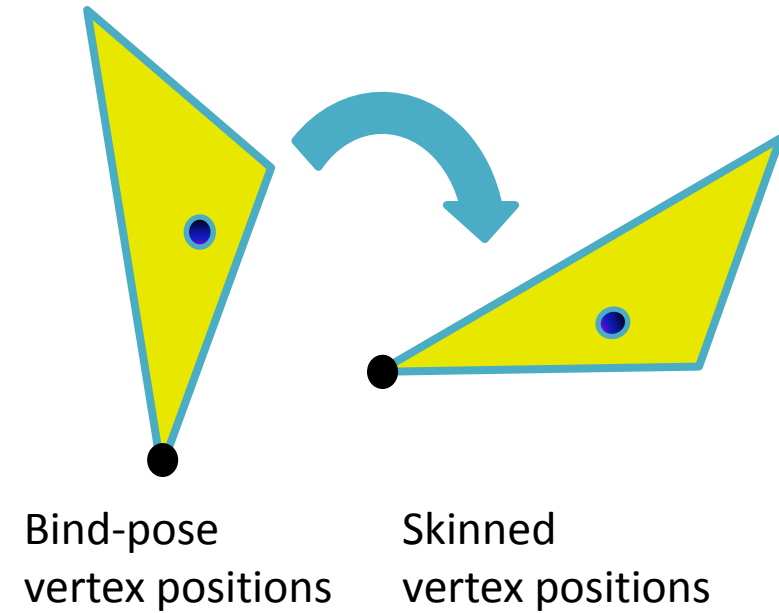
TRESSFX 3.0 SKINNING



- ▲ Calculate transform from bind-pose to skinned

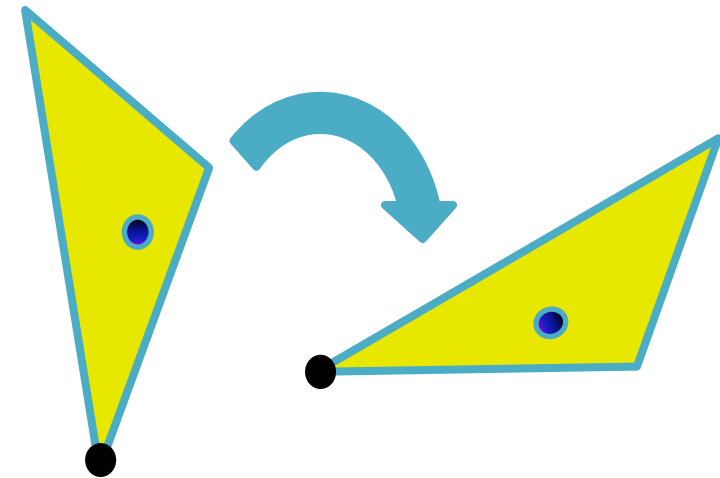


- ▲ Use barycentric coordinates for hair root to calculate final hair transform



TRESSFX 3.0 SKINNING

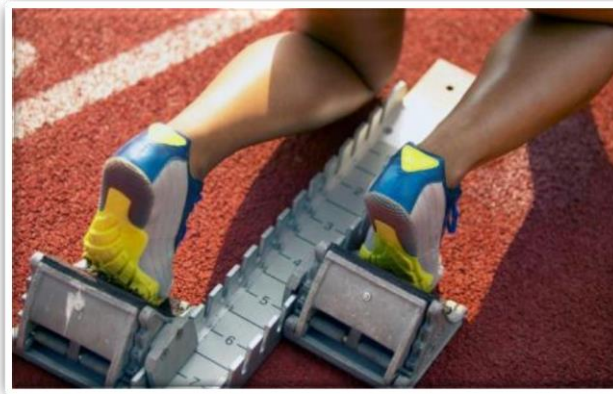
- ▲ Yeah, okay, but why not just skin the hair directly?
- ▲ This way doesn't impose any requirements on how the game engine does the animation update
 - Morph targets/blend shapes
 - Whatever, we just need the updated vertex positions
- ▲ But may code a fast path for ordinary skinning with max 4 bones



TRESSFX 3.0 OPTIMIZATIONS



- ▲ Already in TressFX 2.2
 - Master and slave strands
 - Distance adaptive LOD
 - Deferred rendering
 - Lots of shader optimizations
- ▲ Coming in TressFX 3.0
 - Depth pre-pass
 - Adjust K_{overdraw}
 - More shader optimizations



ACKNOWLEDGEMENTS



CRYSTAL
DYNAMICS

SQUARE ENIX

- | | | |
|-------------------|---------------------|----------------------|
| ▲ Bill Bilodeau | ▲ Jay McKee | ▲ Jean-Normand Bucci |
| ▲ Dongsoo Han | ▲ Dan Roeger | ▲ Uriel Doyon |
| ▲ Takahiro Harada | ▲ Joshua Salem | ▲ David Gallardo |
| ▲ Timothy Heath | ▲ Nicolas Thibieroz | ▲ Rodney Lelu |
| ▲ Karl Hillesland | ▲ Jason Yang | ▲ Andre Vu |

DISCLAIMER & ATTRIBUTION



The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

ATTRIBUTION

© 2015 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, Radeon, and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names are for informational purposes only and may be trademarks of their respective owners.

DEUS EX, the DEUS EX logo, TOMB RAIDER, CRYSTAL DYNAMICS, the CRYSTAL DYNAMICS logo, EIDOS, the EIDOS logo, the EIDOS-MONTRÉAL logo, and LARA CROFT are registered trademarks or trademarks of Square Enix Ltd. SQUARE ENIX and the SQUARE ENIX logo are registered trademarks or trademarks of Square Enix Holdings Co., Ltd.

PS4 is a trademark of Sony Computer Entertainment, Inc.

Xbox One is a trademark of Microsoft Corporation.

AMD

