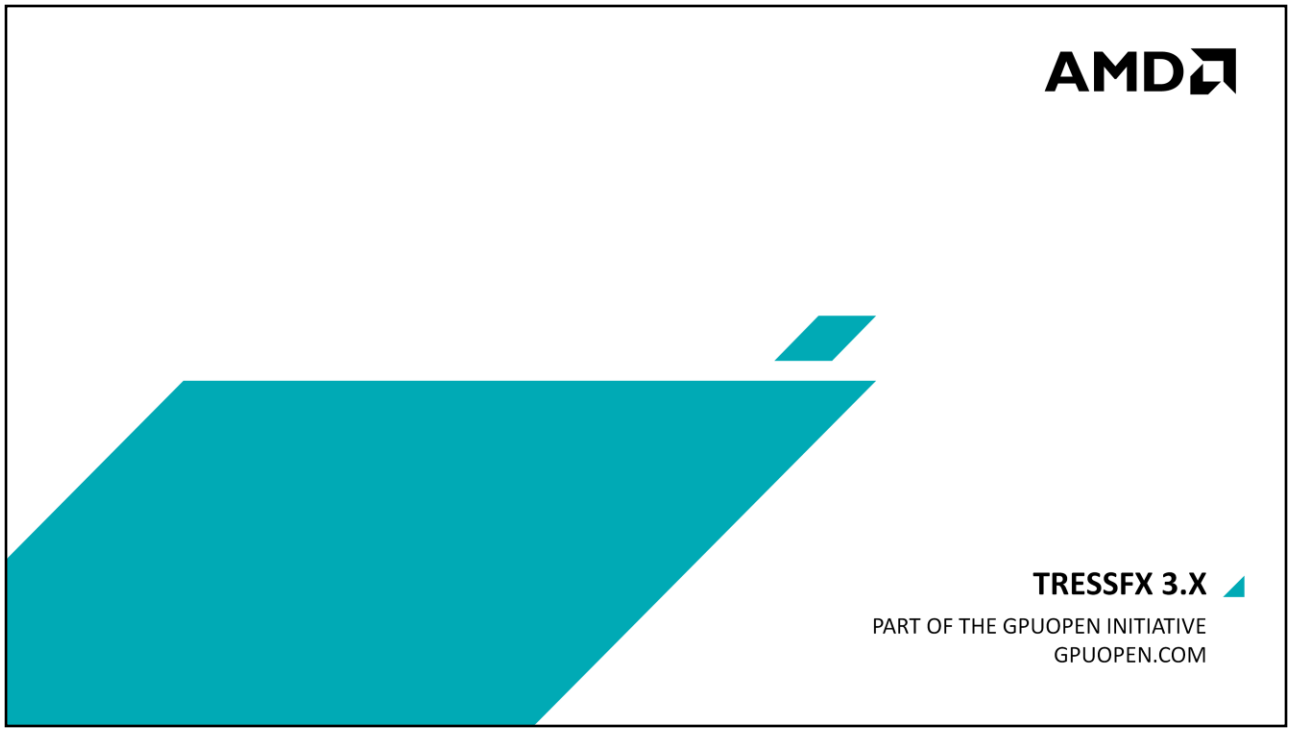

AMD
TRESSFX 3.X
PART OF THE GPUOPEN INITIATIVE
GPUOPEN.COM

HIGH-LEVEL CONTENTS

AMD

- Brief TressFX overview
- TressFX 3.x library
  - Update to AMD's TressFX example implementation
  - Maya plugin
  - Viewer and runtime library (with full source)
  - Fur support
  - Skinning
  - Future optimizations
  - New memory-friendly OIT method (ShortCut)

A BRIEF HISTORY

AMD

- 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
  - Consoles
- 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

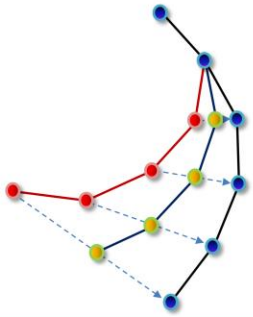
SQUARE ENIX

TressFX is very high-quality real-time hair for games
Eidos-Montréal is a Square Enix company, as is Crystal Dynamics.
Being a Square Enix company, they had access to the Tomb Raider code
Started with that and have been enhancing and improving it

TRESSFX OVERVIEW

AMD

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



initial goal position
current position
final position

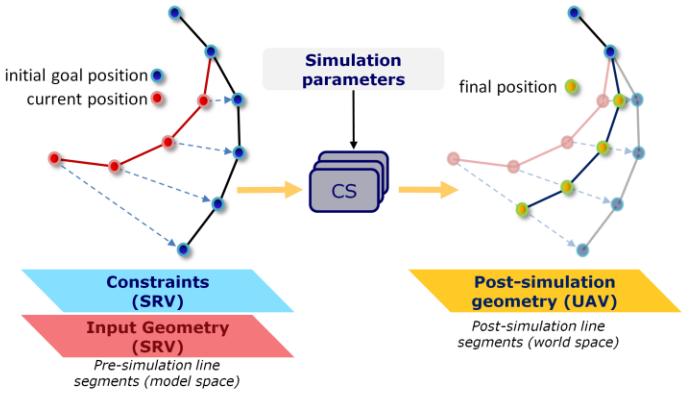
SIMULATION

RENDERING

4 | TRESSFX 3.X | GPUOPEN

TressFX is very high-quality real-time hair for games
But how does it work?
At a high level, there are two parts

TRESSFX SIMULATION



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

Start with standard Verlet position update to account for forces such as gravity, with some damping
Then various constraints, to tune the behavior of the hair and to make the simulation behave
The example shows the global shape constraint, which tries to pull the hair verts back to their initial positions

TRESSFX RENDERING

AMD

Good Lighting + Anti-Aliasing + Volume Shadows + Transparency

Hair shading (good lighting) using Marschner dual highlight approach and Kajiya-Kay lighting model
AA is not hardware MSAA. Calculate coverage in pixel shader and convert to alpha value
Simplified Deep Shadow Map technique
Order-independent transparency (OIT) using a per-pixel linked list (PPLL)

TRESSFX 3.X LIBRARY

TRESSFX SAMPLE HISTORY

AMD

- 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

~22k hair strands in the example on the slide
Updates to this point have been mostly focused on perf improvements

TRESSFX 3.X

AMD

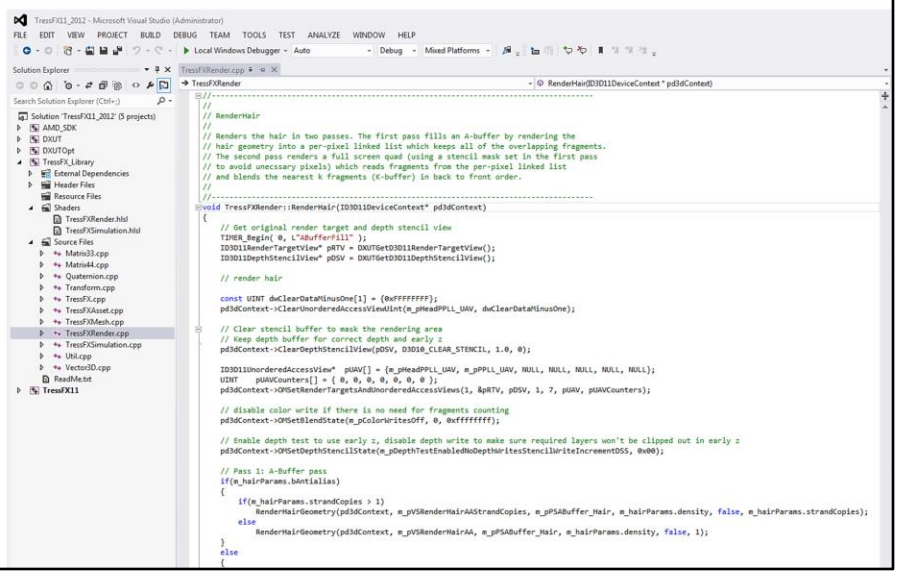
- TressFX 3.0
  - Maya plugin
    - with full source
  - Viewer and runtime library
    - with full source
  - Fur support
  - Skinning
  - Free
- TressFX 3.1
  - New ShortCut OIT method
    - Lower memory
    - Potentially better performance
    - But at slightly reduced quality vs. standard PPLL TressFX

Whereas previous updates have been mostly focuses on perf, TressFX 3.0 is a larger update
It's a big update that we hope will be exciting and useful to game developers

TressFX 3.1 adds a new order-independent transparency (OIT) method called ShortCut

TRESSFX 3.X VIEWER AND RUNTIME LIBRARY

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



Viewer provides an example of how to hook into the runtime
And then also lets you preview your exported results from Maya

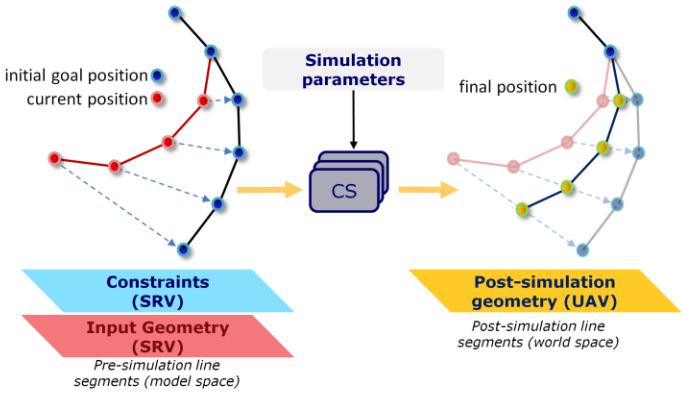
TRESSFX 3.X FUR SUPPORT

AMD

- 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

Core principles are the same as for hair
But does require some additional features
In existing implementations of TressFX, the hair is one color.
Fur color varies across the animal's body
And need animation support

TRESSFX 3.X FUR SIMULATION



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

Global shape constraint and collision shape might not be needed for fur

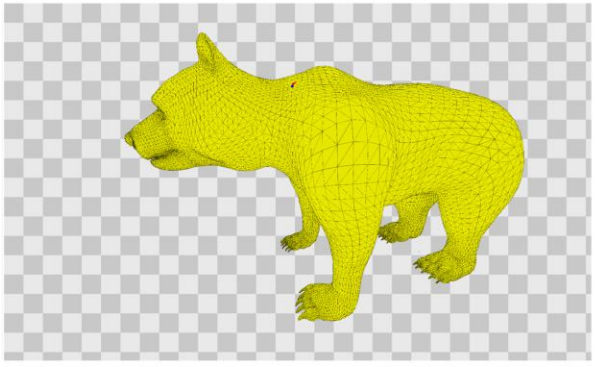
TRESSFX 3.X TEXTURE COORDINATES

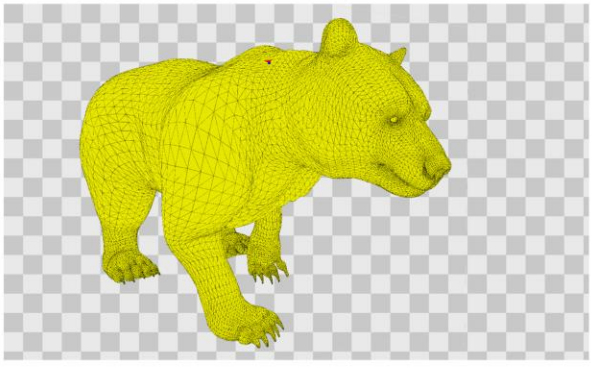
AMD

Fur picks up color from the underlying texture on the model

TRESSFX 3.X SKINNING

AMD





BIND POSE

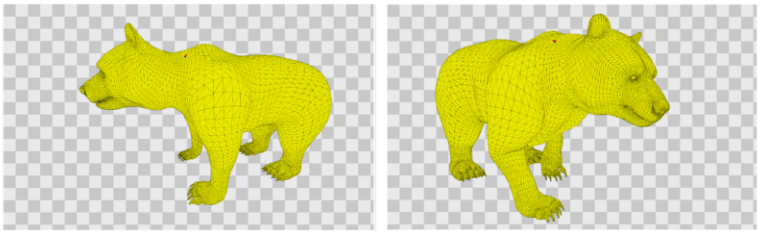
SKINNED

Game engine does animation update and vertex skinning as usual
Then, bind pose vertex positions and current skinned vertex positions are fed into the TressFX runtime

TRESSFX 3.X SKINNING

AMD

- 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

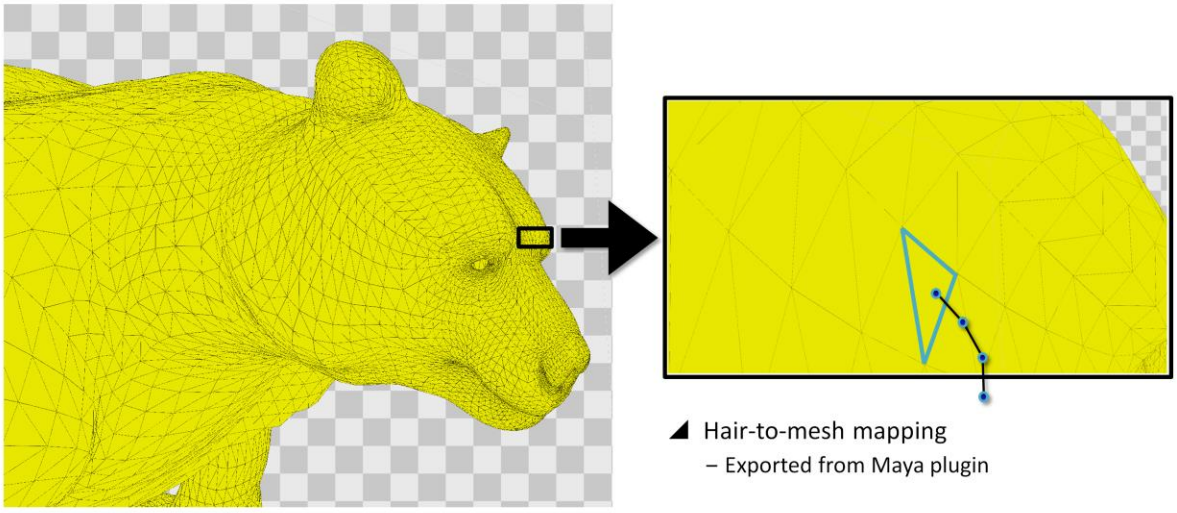


BIND POSE SKINNED

To use the SO stage without using a geometry shader, call ID3D11Device::CreateGeometryShaderWithStreamOutput and pass a pointer to a vertex shader to the pShaderBytecode parameter.

TRESSFX 3.X SKINNING

AMD



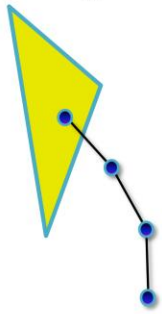
- Hair-to-mesh mapping
  - Exported from Maya plugin

Okay, so the engine does its animation and skinning update as usual. Then what?
The TressFX data from the exporter has hair-to-mesh mapping. Specifies the mesh triangle to which a hair strand belongs.

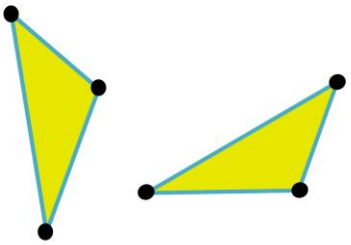
TRESSFX 3.X SKINNING

AMD

- Use hair strand index to get triangle index



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



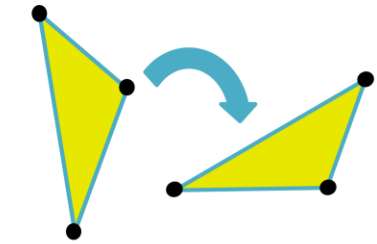
Bind-pose vertex positions

Skinned vertex positions

TRESSFX 3.X SKINNING

AMD

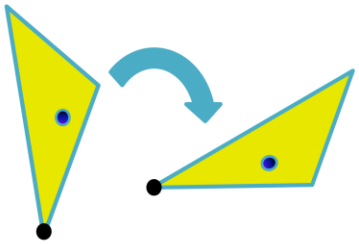
- Calculate transform from bind-pose to skinned



Bind-pose vertex positions

Skinned vertex positions

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



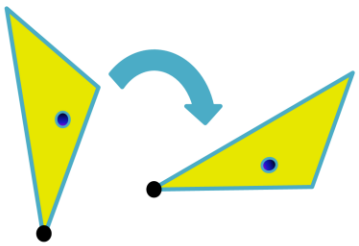
Bind-pose vertex positions

Skinned vertex positions

TRESSFX 3.X SKINNING

AMD

- 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.X OPTIMIZATIONS

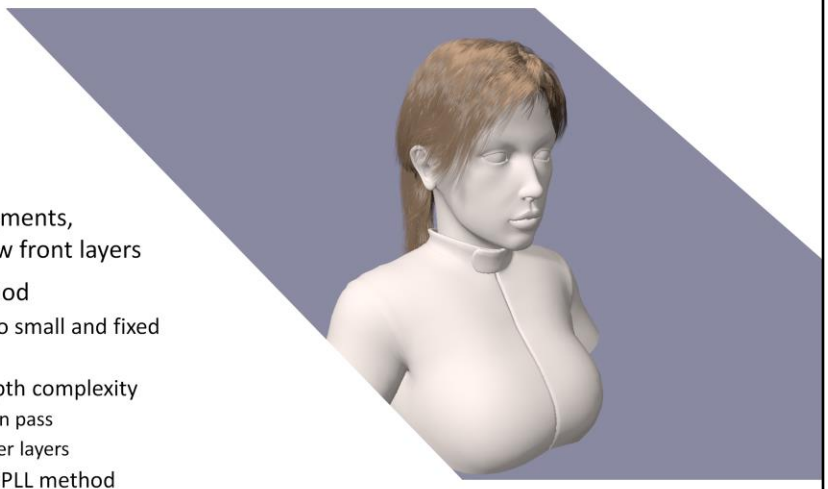
AMD

- Already in TressFX 2.2
  - Master and slave strands
  - Distance adaptive LOD
  - Deferred rendering
  - Lots of shader optimizations
- Coming in TressFX 3.x
  - Depth pre-pass
  - Adjust $K_{overdraw}$
  - More shader optimizations

TRESSFX 3.1 *SHORTCUT* RENDERING

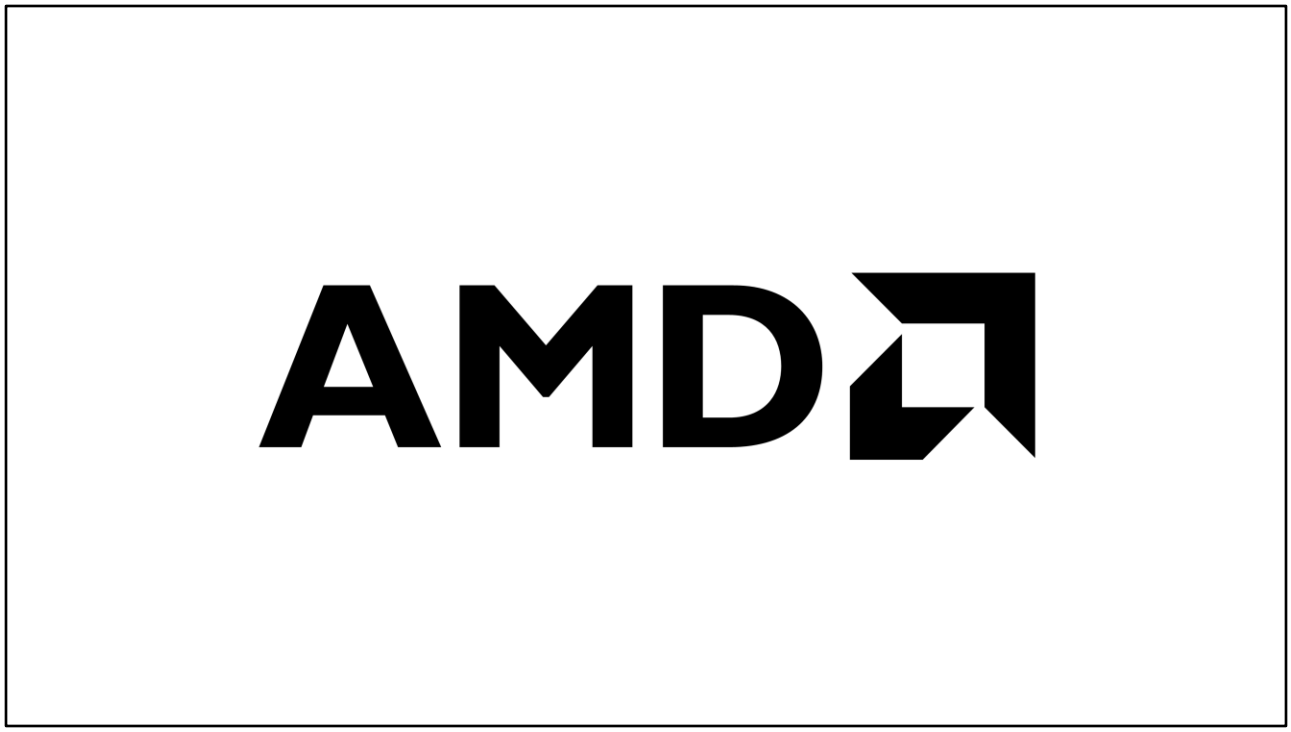
AMD

- A second OIT implementation provided with the 3.1 release
  - Inspired by Eidos-Montréal's GDC 2015 talk
  - Enabled in the sample with the ShortCut checkbox
- Alpha is computed from all hair fragments, while color is based on shading a few front layers
- Comparison with current PPLL method
  - Memory requirements reduced due to small and fixed number of layers
  - Relative performance depends on depth complexity
    - Requires an extra geometry and screen pass
    - Potential savings from processing fewer layers
  - Result not quite the same quality as PPLL method



- Eidos-Montréal talk available at http://developer.amd.com/resources/documentation-articles/conference-presentations/
- Four main steps of ShortCut:
  - Render hair geometry, using a sequence of InterlockedMin calls to update the list of nearest fragments.
  - Screen space pass that puts the *k*th nearest depth in the depth buffer for early z culling in the next step.
  - Render hair geometry again. Shade the fragment and put it in the appropriate slot. [earlydepthstencil] focuses shading cost on the front *k*.
  - Screen space pass that does the final blending.
- Implementation supports *k* = 2 or 3

DISCLAIMER & ATTRIBUTION


AMD