ENHANCING GRAPHICS IN UNREAL ENGINE 3 TITLES USING NEW CODE SUBMISSIONS

Owen Wu
Developer Relations Engineer, AMD
Owen.wu@amd.com

AGENDA

- This presentation covers AMD code submissions that have been integrated into Unreal Engine 3
  - Tessellation
    - Phong Tessellation
    - Tessellation Optimizations
    - Performance considerations
  - Multi-monitor Support
    - Eyefinity support code
    - How to test multi-monitor support on a single monitor
  - Vertex Shader-Based Bokeh Depth Of Field (DOF)
    - Implementation
    - Performance considerations
  - Post-Process Full Screen Anti-Aliasing (FSAA)
    - MLAA support
    - Performance and quality comparison

POWERED BY UNREAL TECHNOLOGY

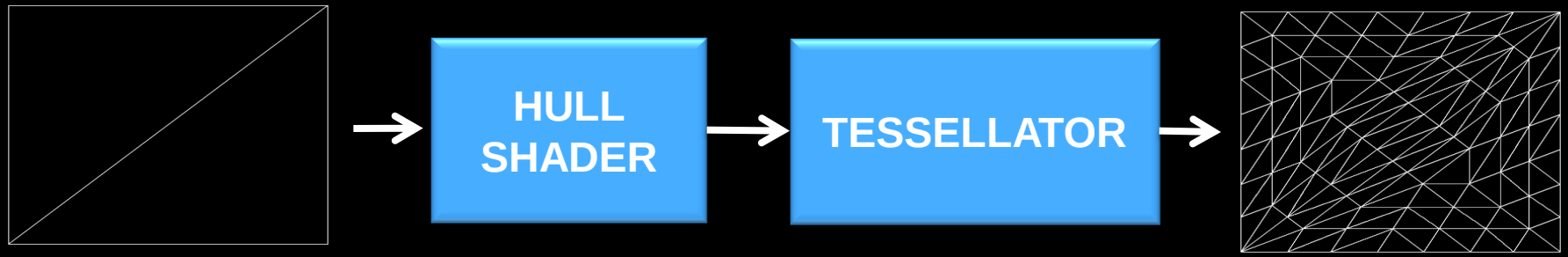
AMD

TESSELLATION

TESSELLATION TECHNIQUES

- Displacement mapping with flat tessellation
  - Requires a height map for displacement
  - Generates high-quality bumpy surface
- PN-Triangle tessellation
  - No displacement map required
  - Used to generate smooth silhouette
  - Hull shader for patch construction
- Phong tessellation
  - No extra displacement map required
  - Used to generate smooth silhouette
  - No patch construction required

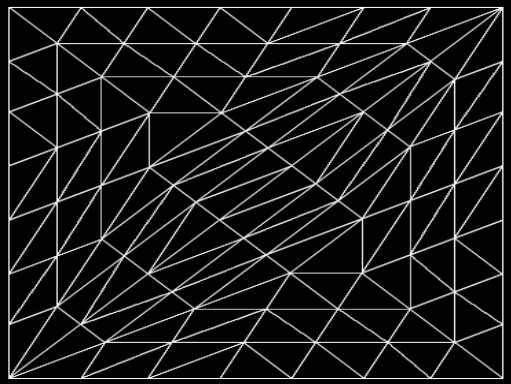
DISPLACEMENT MAPPING WITH FLAT TESSELLATION



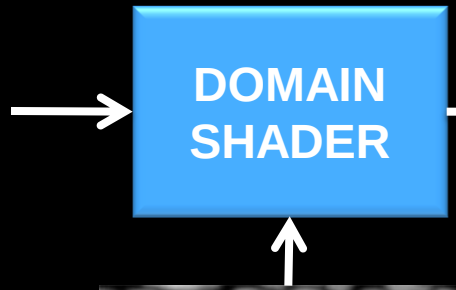
Triangle Patch Mesh

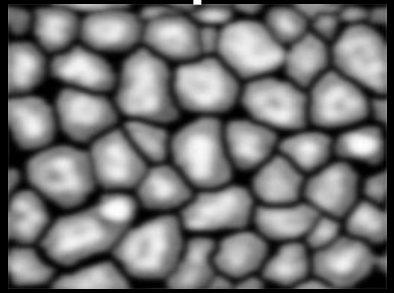
Tessellated Mesh

DISPLACEMENT MAPPING WITH FLAT TESSELLATION

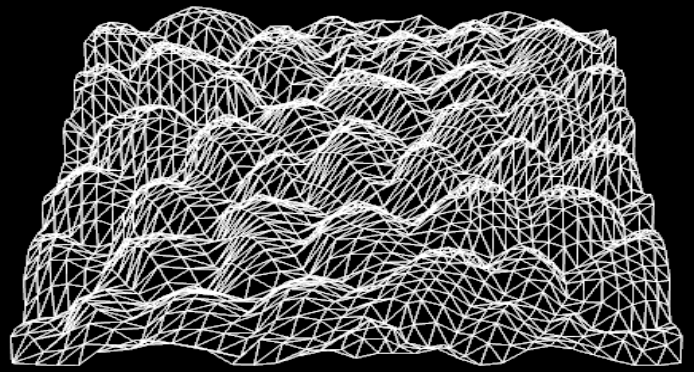


Tessellated Mesh





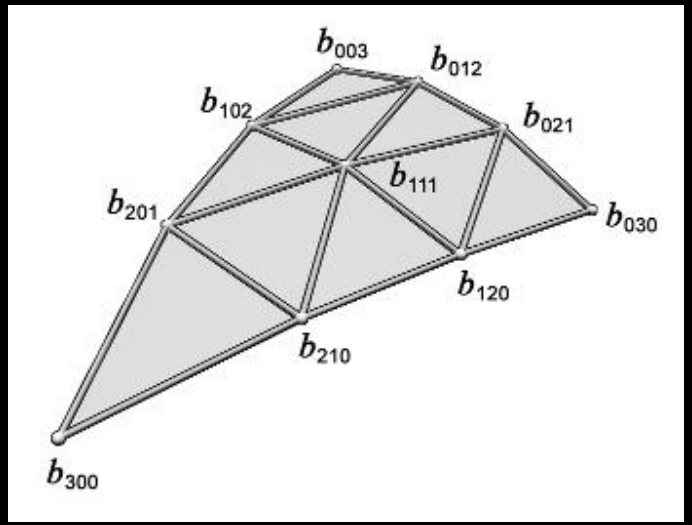
Displacement map



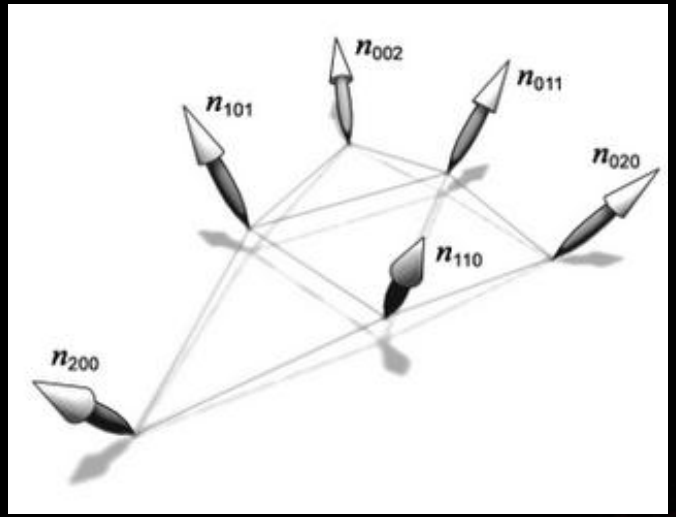
Displaced Mesh

PN-TRIANGLE TESSELLATION

- PN-triangles is a purely local scheme
- Construct a cubic Bezier patch according to the three vertex positions and normals of a triangle in the Hull shader



Control points of triangular Bezier patch

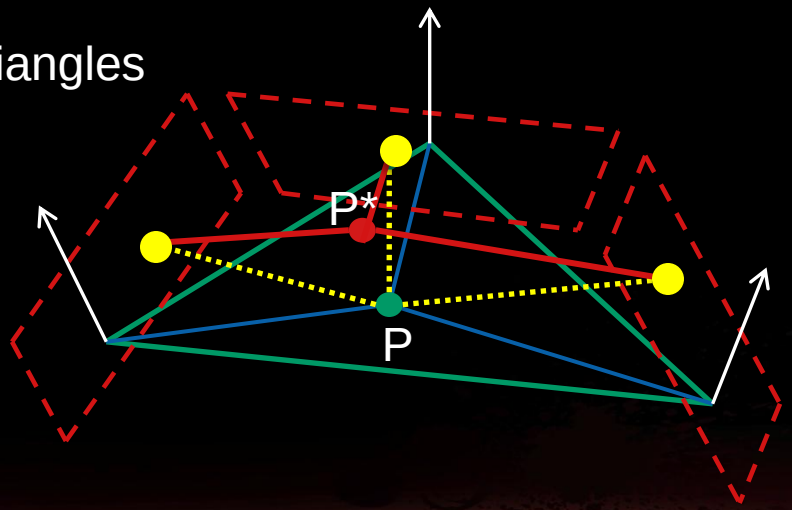


Normal component of PN-Triangle

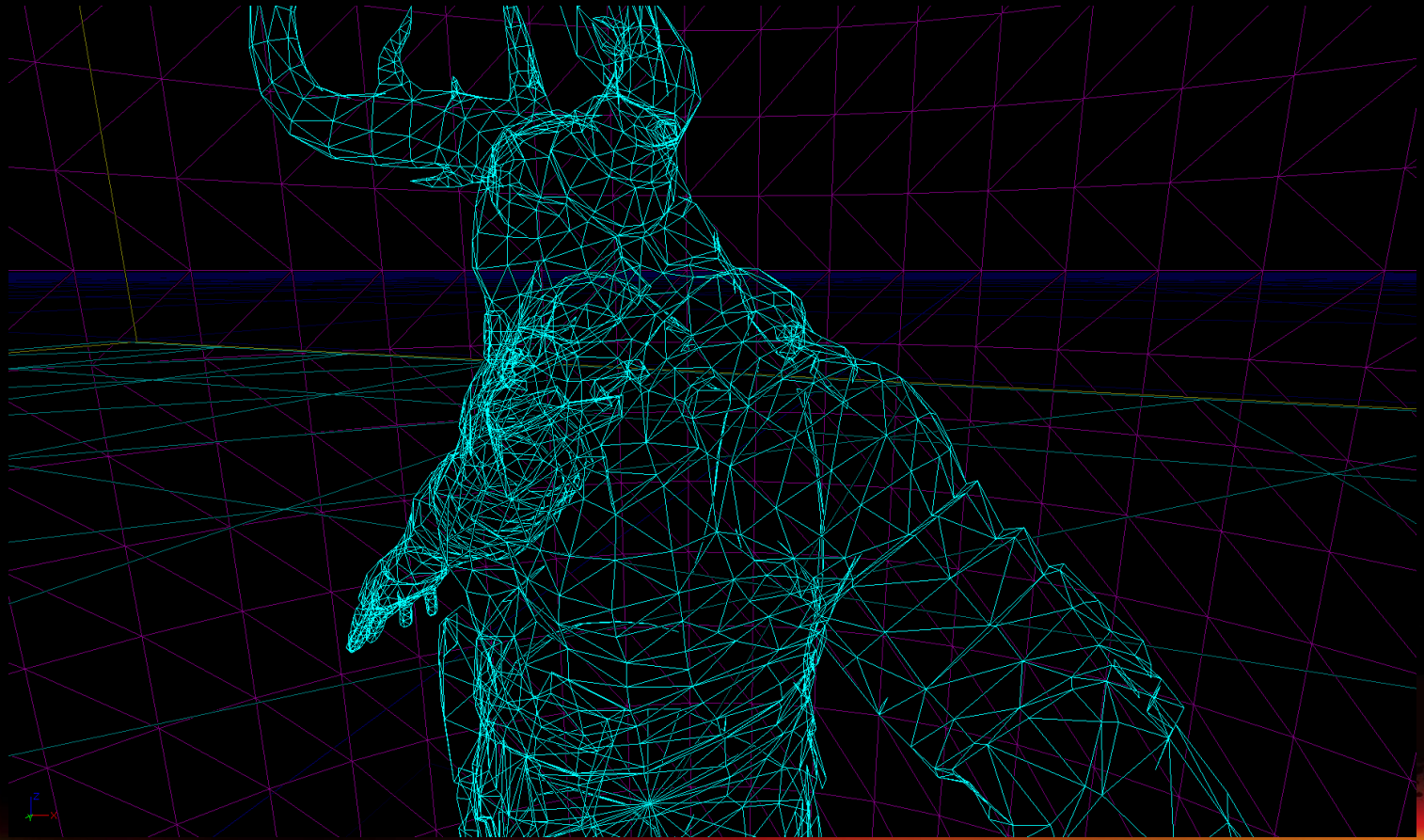
Pictures from "Curved PN Triangles" white paper, Vlachos et al.

PHONG TESSELLATION

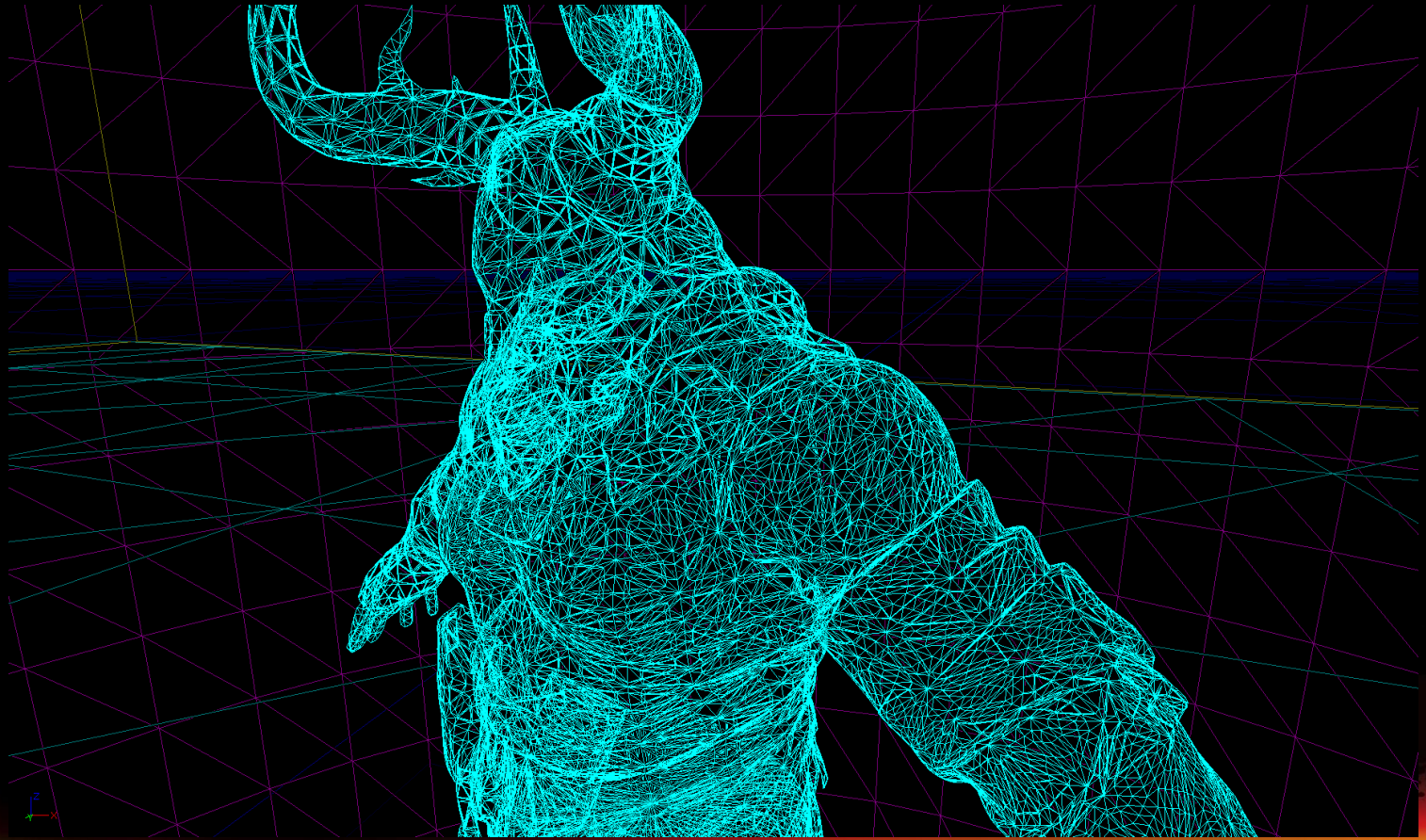
- Phong tessellation is also a purely local scheme
- No need to construct complex patch in Hull shader
  - Simple Hull shader, just pass over vertex position and normal
  - Simple shader == better performance
- Phong is simpler and more efficient than PN-Triangles
  - Yet produces very similar visual output
  - … at much better performance!



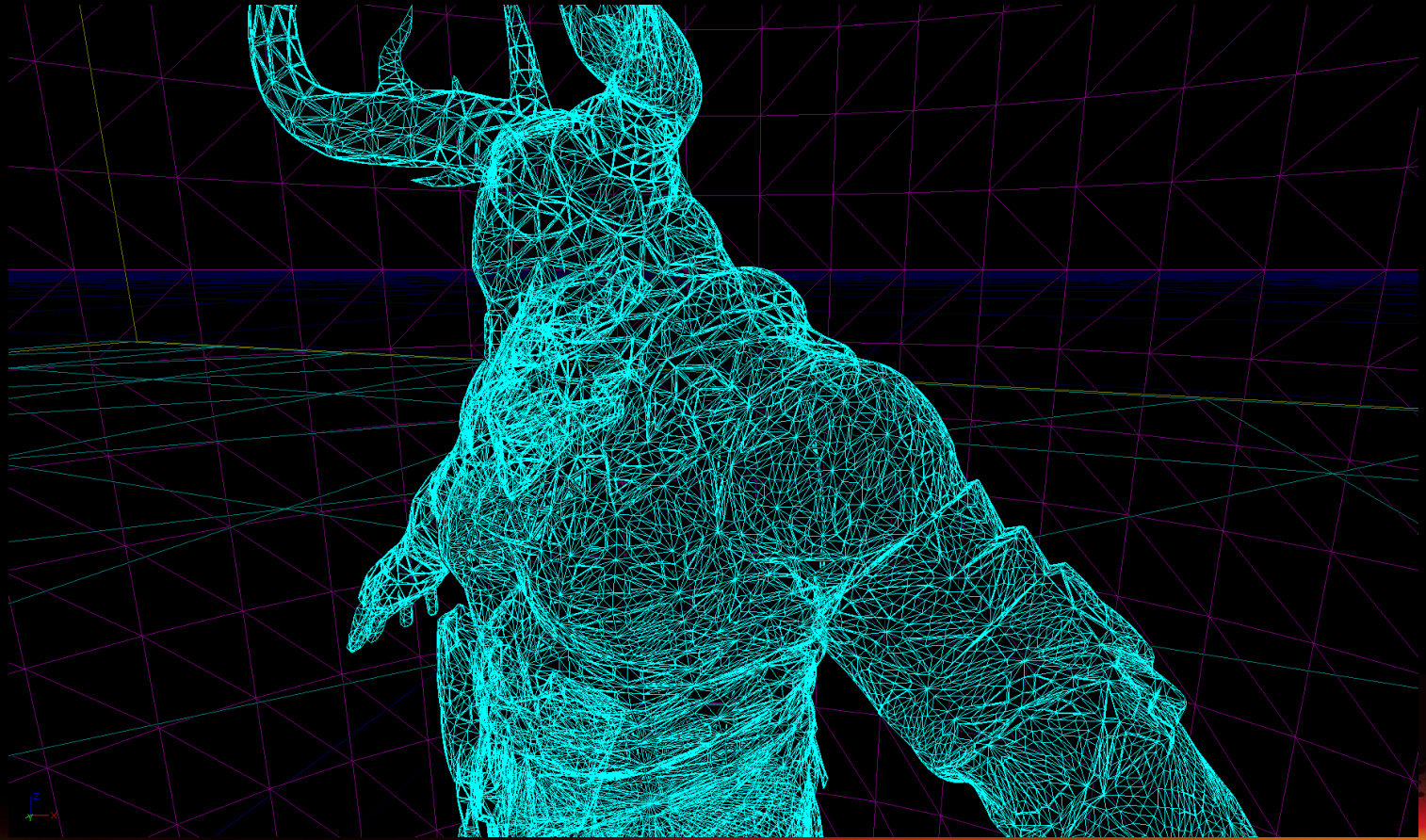
NO TESSELLATION



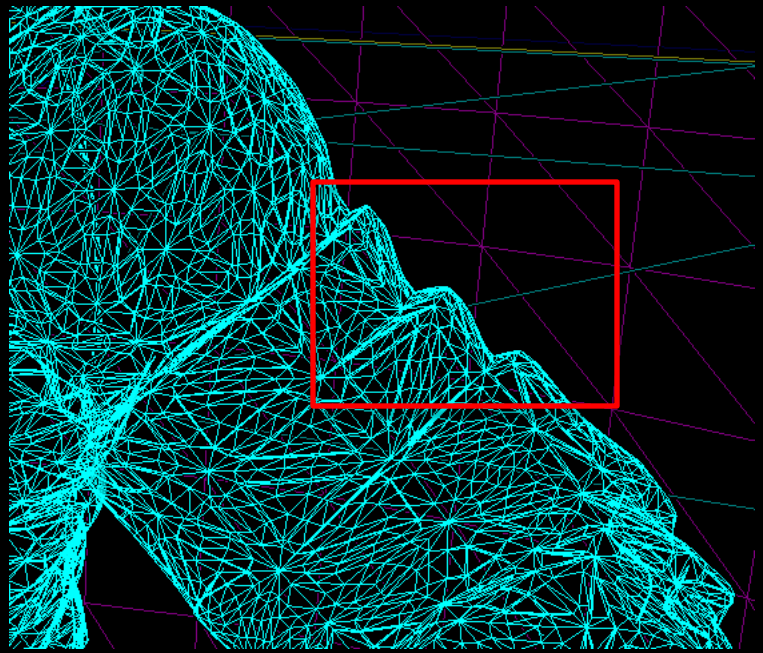
PN-TRIANGLE TESSELLATION



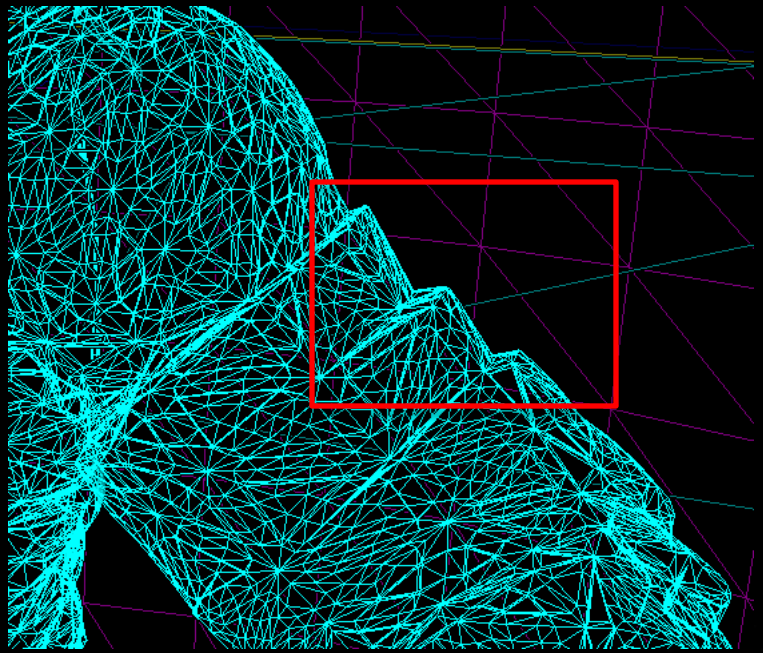
PHONG TESSELLATION



TESSELLATION COMPARISON



PN-Triangle Tessellation



Phong Tessellation

NO TESSELLATION

PN-TRIANGLE TESSELLATION

PHONG TESSELLATION

UE3 TESSELLATION FEATURES

- Tessellation modes
  - Flat Tessellation
  - PN-Triangle tessellation
  - Our submission adds Phong Tessellation mode into UE3
  - Can add displacement map to any tessellation mode
- Built-in adaptive factor based on triangle screen-space size
  - Allows constant and predictable performance
  - Avoids generation of very small triangles which are inefficient (<8 pixels)
  - More optimizations can be selected to improve performance further

TESSELLATION PERFORMANCE

AMD

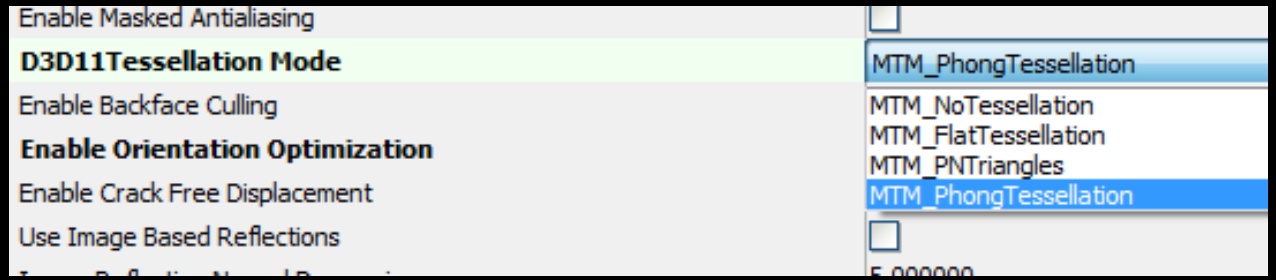
- Tessellation is not free!
  - Use with caution otherwise it can impact performance considerably
  - Should only be used where image quality can be improved
- Adaptive tessellation keeps tessellation requirements reasonable
  - Vary tessellation factors based on real-time metrics
  - UE3 implements screen-space adaptive optimization
- More aggressive optimizations are required to keep performance up

MORE AGGRESSIVE OPTIMIZATIONS

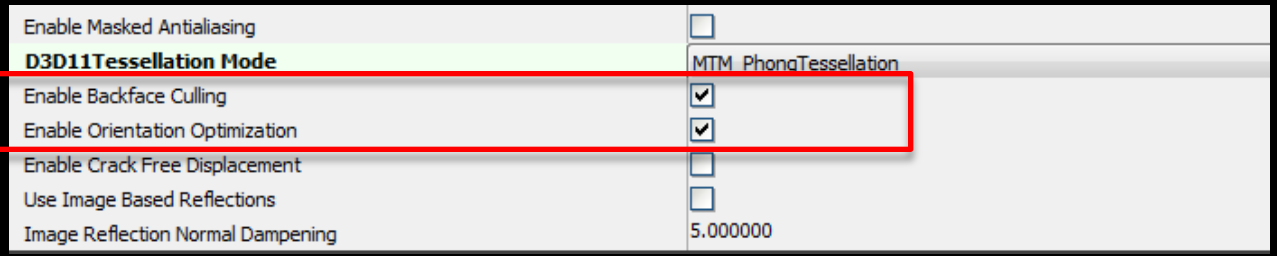
AMD

- Backface culling
  - Set tessellation factor to 0 on back-facing (i.e. invisible) triangles
  - Warning: some back facing triangle still contribute to a silhouette!
- View frustum culling
  - Don‘t waste tessellation power on invisible triangles
  - Set tessellation factor to 0 if the *whole* triangle patch is outside the view frustum
- Orientation-Adaptive Tessellation
  - Only silhouette patches contribute to silhouette enhancement
  - Silhouette patches therefore get higher tessellation factors

MATERIAL EDITOR



New Phong tessellation mode



Two new optimization options for all tessellation modes

UDKENGINE.INI

- Two new variables in UDKEngine.ini
  - TessellationBackfaceCullingThreshold
    - **If (dot(N, V) < -TessellationBackfaceCullingThreshold)**
      Tessellation factor = 0;
  - TessellationOrientationThreshold
    - **EdgeScale = 1.0f – abs(dot(N, V));**
    - **Tessellation factor =**
      (EdgeScale – TessellationOrientationThreshold) /
      (1.0 – TessellationOrientationThreshold);

PHONG TESSELLATION

No Tessellation

Phong Tessellation with backface culling and orientation adaptive factor

REAL-TIME DEMO

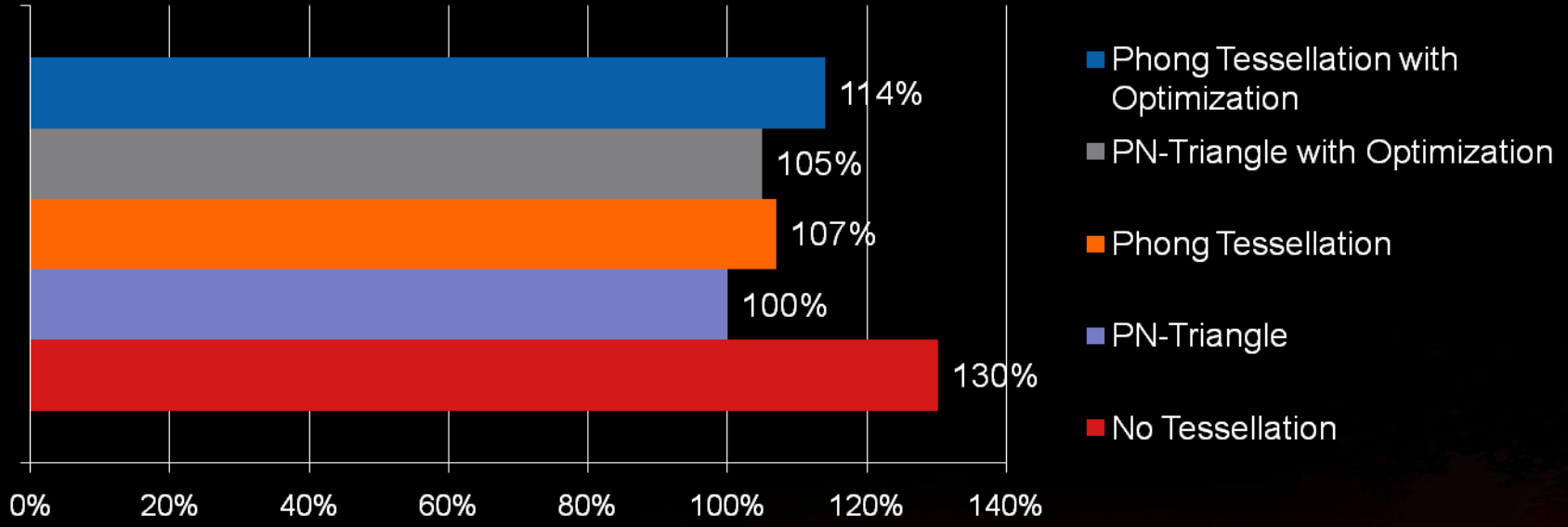
- How to activate Phong tessellation and optimizations in material editor
- Show real-time orientation adaptive optimization demo in UE3 editor

PERFORMANCE NUMBERS

- Tessellation performance on AMD Radeon HD6970

  BackfaceCullingThreshold=0.6

  OrientationThreshold=0.0



POWERED BY UNREAL TECHNOLOGY

AMD

MULTI-MONITOR

WHY SUPPORTING MULTI-MONITOR

AMD

- Multi-monitor configurations are becoming more common as a result of the affordability of LCD monitors!

- Technically easy to support in your titles (with some extra care for HUD)
  - Multi-Monitor is "just" a single larger render target from the programmer's perspective

- AMD's code submission for Eyefinity makes it even easier
  - Developers are able to test multi-monitor on a single monitor system

SINGLE MONITOR VS. MULTI-MONITOR

HOW TO SUPPORT MULTI-MONITOR

AMD

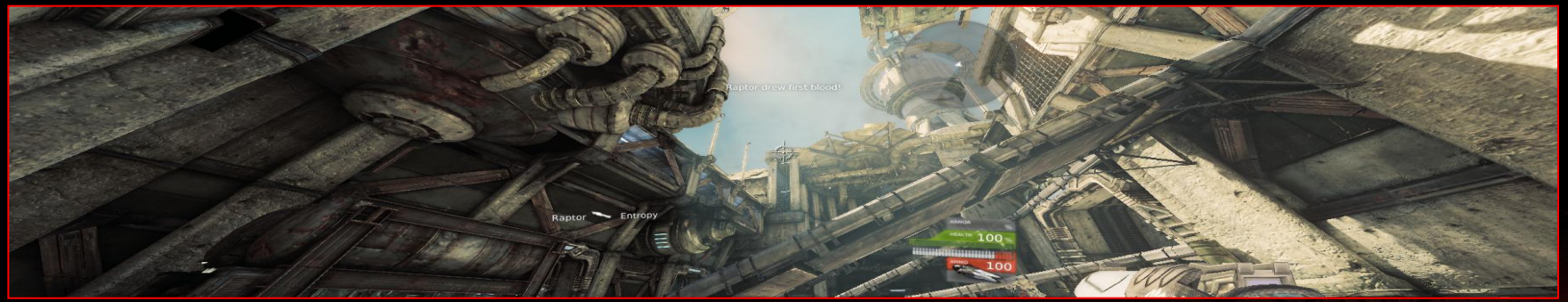
- Don’t block any special aspect ratio resolution (i.e. 5760x1200)
  - Game should be flexible with its supported resolutions
- Expand Field of View according to the resolution
  - Most common Multi-Monitor resolution is 3:1 landscape
  - Set vertical axis to a fixed FOV and let horizontal FOV expand with resolution
- Place HUD to the middle monitor
- Cut-scenes and movies should be played on the middle monitor and retain their original aspect ratio
  - Use bConstrainAspectRatio property of camera to keep the right aspect ratio for your movie
  - FOV expansion will be disabled if bConstrainAspectRatio is TRUE

WHAT AMD HAS DONE FOR YOU

- Use “AllowAMDEyefinity“ in UDKEngine.ini to enable Eyefinity support
- Expand FOV according to the resolution
  - Use "EyefinityFOVThreshold" in UDKEngine.ini to limit the Max FOV in X axis
- Place HUD to the middle monitor
  - Only works on Gfx HUD component
  - It detects the Eyefinity mode then place Gfx HUD component against middle monitor automatically
- Test multi-monitor support on a single monitor
  - Programmer fills out the window resolution and monitor configuration in C++ code
  - This feature is only activated on debug version
  - Convenient feature for developers if they don't have access to all Eyefinity configurations

SAMPLE SCREENSHOTS

SAMPLE SCREENSHOTS



REAL-TIME DEMO

- Demonstrate UDKGame without Eyefinity support
- Demonstrate how to test Eyefinity support on a single monitor machine

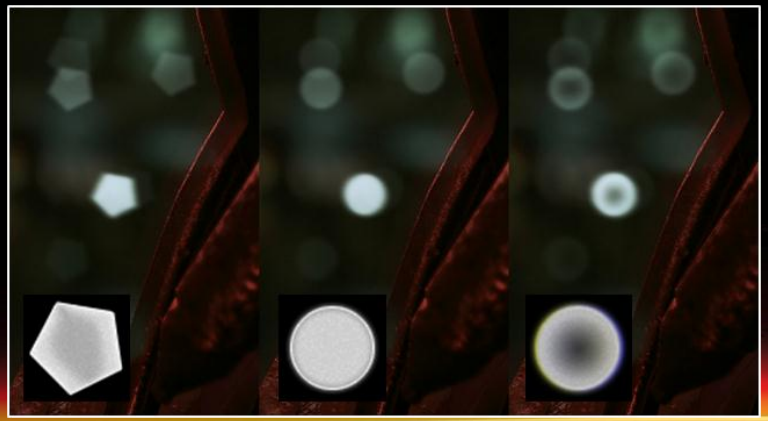
POWERED BY UNREAL TECHNOLOGY

AMD

VERTEX SHADER BASED BOKEH DEPTH OF FIELD

BOKEH DEPTH OF FIELD (DOF) IN UE3

- New stunning UE3 post-processing effect introduced last GDC
- In photography, **Bokeh** is the blur in out-of-focus areas of an image.
- UE3 Bokeh DOF uses the Geometry Shader (GS) to generate a massive number of point sprites to simulate Bokeh
  - Generates 1 to 4 Bokeh point sprites for every 4 pixels of half-resolution image
- UE3 renders Bokeh DOF into two layers (foreground and background) to avoid artifacts
- DirectX® 11 only



BOKEH DOF CODE SUBMISSION

AMD

- Generating a massive number of point sprites in the GS impacts performance!
- We moved it to the Vertex Shader (VS)
  - Performance improvement is quite large on some hardware
  - Visual results unchanged
- Now supports DirectX 9 level hardware after moving it to the VS
  - Cost a bit more video memory in DX9 mode to store vertex IDs
- Video memory footprint is the same in DirectX 11 mode
  - No actual vertex buffer is needed
- Triangle and Quad Bokeh supported

VERTEX IDS

DirectX 11:

- Use system-generated vertex IDs (SV_VertexID)
- Bind NULL Vertex Buffer
  - Vertices are generated in the vertex shader without buffer input

DirectX 9:

- Generate a vertex buffer with vertex ID attributes to emulate same functionality
- Generate (HalfResX/2)*(HalfResY/2)*3*4 vertices (for triangle Bokeh)
  - (HalfResX/2)*(HalfResY/2)*6*4 vertices for quad bokeh
- A new vertex buffer is created on resolution change

IMPLEMENTATION

AMD

- Render the vertex buffer (NULL on DX11) as a triangle list
- Use vertex ID to compute current Bokeh ID
  - Triangle number = (VertexID/3). (VertexID%3) is local triangle vertex index
  - If using quads: Quad number = (VertexID/6). (VertexID%6) is local quad vertex index
- Compute the Bokeh position and texture coordinates in Vertex Shader
  - For NULL Bokeh: place all vertices at the same position to skip rendering
  - (VertexID%12) to get the vertex index ID in a Bokeh group
    - Place all last 9 vertices at the same position to eliminate 3 triangles
  - (VertexID%24) for quad

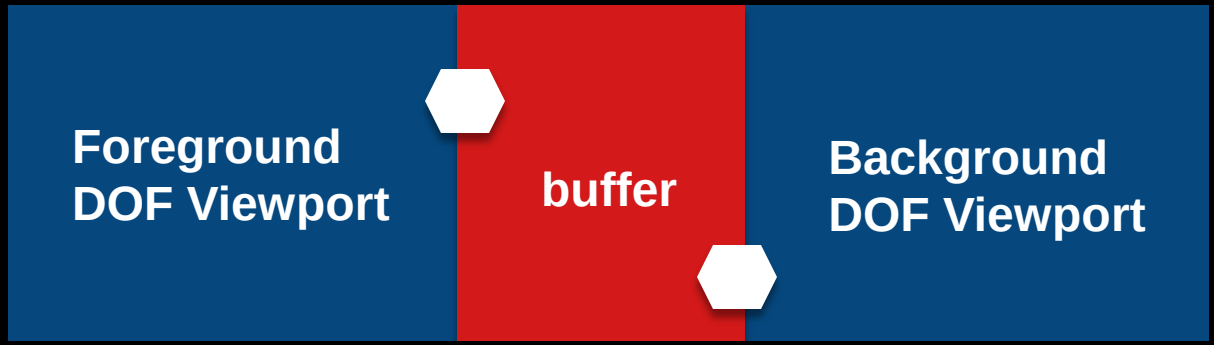
IMPLEMENTATION

- Shift and scale vertex position to simulate multiple viewports in DX11
  - Original code uses multi-viewport to render both foreground and background Bokeh into a single render target

  - Use clip() in pixel shader to simulate viewport clipping
    - Compute the clip distance in Vertex Shader

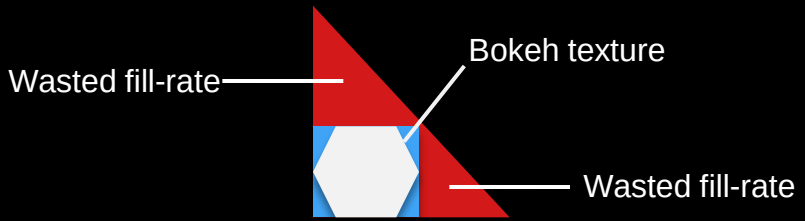
IMPLEMENTATION

- Alternate solution for simulating viewport clipping
  - Create a bigger render target with buffer in the middle
  - Faster, no clipping is needed
  - Costs a little more video memory
  - Bokeh may cross the buffer if it's bigger than the buffer



TRIANGLE OR QUAD?

- Should one use a triangle or quad to represent a point sprite?
- Using triangles can reduce vertex processing cost by 50% compared to using quads
- But triangle point sprite may not be a good solution for big Bokeh shapes
  - Rasterization/fill-rate power will be wasted on invisible pixels (red area)



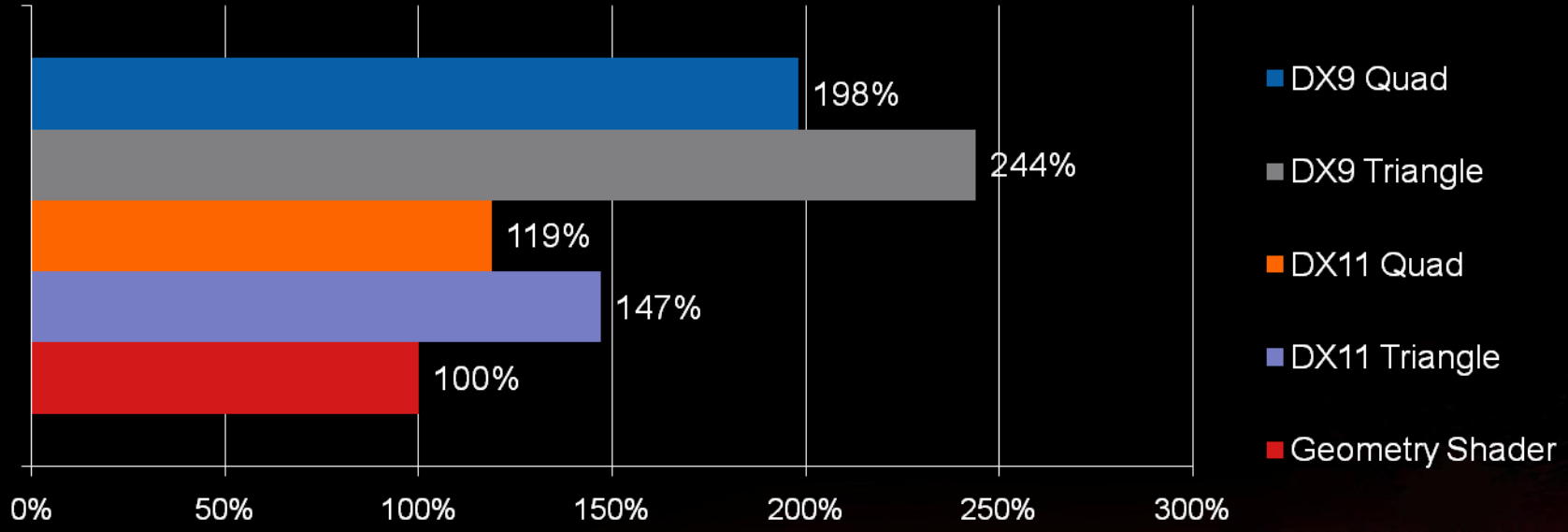
- We implemented both
- Simply use preprocess in C++ code to switch
  - #define __TRIANGLE_BOKEH__ 1

PERFORMANCE NUMBER

Performance test map : EpicCitadel
Small Bokeh setting (not fill-rate bound for triangle Bokeh)

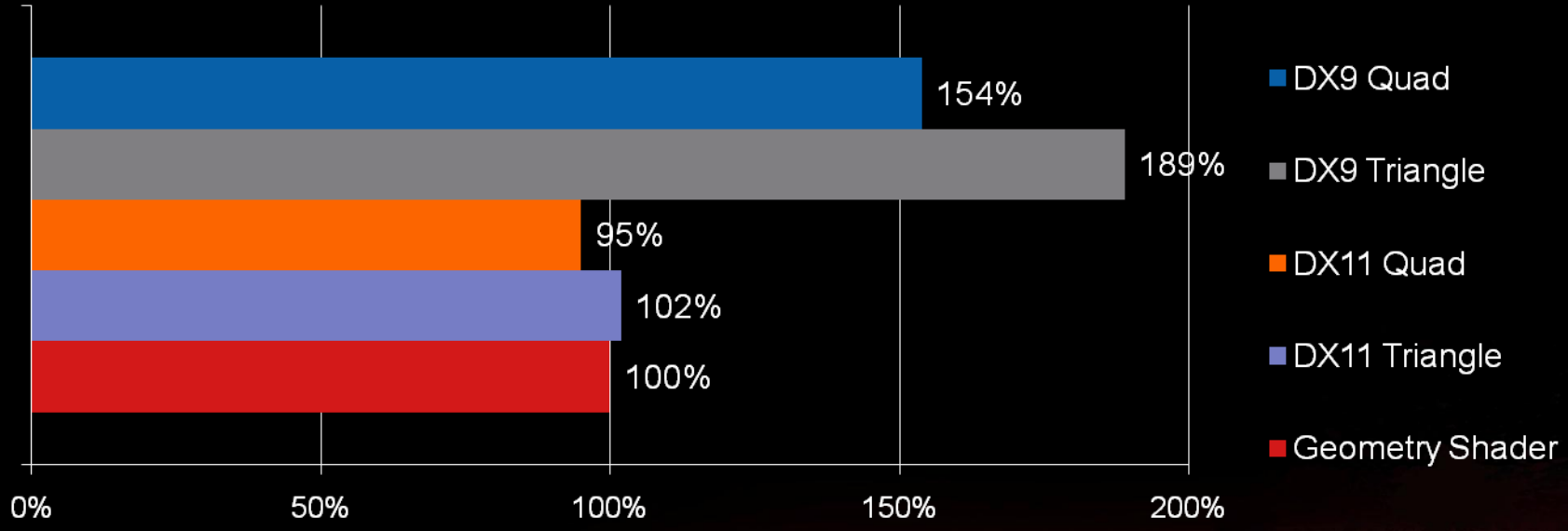
PERFORMANCE NUMBER

- Bokeh DOF on AMD Radeon HD 6970 @ 1920x1080



PERFORMANCE NUMBER

- Bokeh DOF on GTX 580 @ 1920x1080



POWERED BY UNREAL TECHNOLOGY

AMD

POST-PROCESS FULLSCREEN ANTI-ALIASING (FSAA)

POST-PROCESS FSAA

- Why using Post-Process Fullscreen Anti-Aliasing?
  - MSAA doesn't work for deferred shading when using DX9 level hardware
  - MSAA is expensive with deferred shading (performance and memory footprint)
  - MSAA doesn't work with transparent textures (alpha-tested)
  - Post-Process FSAA does not add any complexity to the rendering pipeline
  - Easy to change, modify or optimize without adverse effects on other rendering stages
  - It can also work with MSAA
- UE3 supports two types of post-process FSAA since the July 2011 build
  - Fast Approximate Anti-Aliasing (FXAA)
  - Morphological Anti-Aliasing (MLAA)

COMPARISON

COMPARISON

No AA

MLAA

FXAA OVERVIEW

AMD

- Single-pass post-processing
- No extra render target required
- Tends to detect too many edges and prone to blur non-edge pixels

MLAA OVERVIEW

- Three passes post-processing
  - 1st Pass : Detect edges
  - 2nd Pass : Compute edge length
  - 3rd Pass : Blend edge color according to the edge type and length
- Needs two extra render targets
  - One is for storing edge mask
  - One is for storing edge length
- Can detect edges pretty well so that only edge pixels are anti-aliased
- Adjustable edge detection level
  - It's good for performance tuning

FXAA AND MLAA COMPARISON – EDGE DETECTION

Edge pixels detected by MLAA (in red)

Edge pixels detected by FXAA (in red)

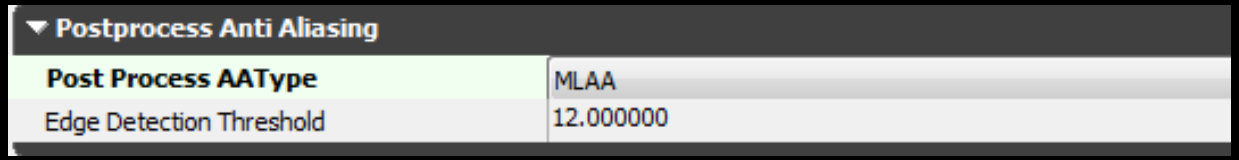
FXAA AND MLAA COMPARISON – EDGE DETECTION (ZOOMED-IN)

Edge pixels detected by MLAA (in red)

Edge pixels detected by FXAA (in red)

UE3 POSTPROCESS AA

- UE3 supports both from July 2011 build
- Can be activated in uberpostprocess node



- MLAA needs to be activated from UDKEngine.ini file
  - bAllowPostProcessAA = True
  - Default is OFF
- Good AA solution for deferred shading
- Supports both DX9 and DX11

CONCLUSION

- Tessellation
  - New Phong tessellation mode which generates similar visual output to PN-Triangle but at much better performance
  - New optimization options for all tessellation modes
- Multi-monitor
  - Automatic FOV expansion and HUD placement
  - Simulate multi-monitor on single monitor system
- Vertex Shader Based Bokeh DOF
  - Huge performance improvement
  - Support both DirectX 9 and DirectX 11
- Post-processing FSAA
  - Already in UE3 since the July 2011 build
  - More efficient option over MSAA when using deferred shading

QUESTIONS?

- Send your feedback/suggestion to owen.wu@amd.com

CODE SUBMISSION DOWNLOAD LINKS

- https://udn.epicgames.com/pub/Three/LicenseeCodeSubmissions/Eyefinity.rar
- https://udn.epicgames.com/pub/Three/LicenseeCodeSubmissions/VSBokehDOF.rar

REFERENCES

- Vlachos Alex, Jorg Peters, Chas Boyd and Jason L. Mitchell. "Curved PN Triangles". Proceedings of the 2001 Symposium interactive 3D graphics (2001).
- Tamy Boubekeur, Marc Alexa. Phong Tessellation. ACM Trans. Graph (2008).
- Alexander Reshetov, Intel. Morphological Antialiasing http://visual-computing.intel-research.net/publications/papers/2009/mlaa/mlaa.pdf
- Timothy Lottes, NVIDIA. FXAA Whitepaper. http://developer.download.nvidia.com/assets/gamedev/files/sdk/11/FXAA_WhitePaper.pdf

AMD FUSION[12] DEVELOPER SUMMIT (AFDS)

June 11-14, 2012
amd.com/afds

AFDS gives you direct access to some of the world's authorities and the most detailed training in heterogeneous computing, OpenCL™, OpenGL, DirectCompute, and C++ AMP.

10 Technical Tracks

1) Heterogeneous Computing
2) Web Technologies
3) Cloud Computing
4) Gaming and Consumer Graphics
5) Innovative Client Experiences
6) Multimedia Processing
7) Professional Graphics & Visual Computing
8) Programming Languages and Models
9) Programming Tools
10) Security

Industry Leading Keynotes

- Phil Rogers, AMD
- Tom Malloy, Adobe®
- Dr. Amr Awadallah, Cloudera
- Mark Papermaster, AMD
- Phil Pokorny, Penguin Computing

SPECIAL OFFER FOR GDC ATTENDEES

June 11-14, 2012
amd.com/afds

Be one of the first 100 GDC attendees to register for AFDS and save $100.

REGISTER TODAY

Just use this code when you register:

GDC100

DISCLAIMER

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. THE MATERIAL IS PROVIDED “AS IS.”

AMD SPECIFICALLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT OF ANY INTELLECTUAL PROPERY RIGHT. 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.

Trademark Attribution

AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names used in this presentation are for identification purposes only and may be trademarks of their respective owners.