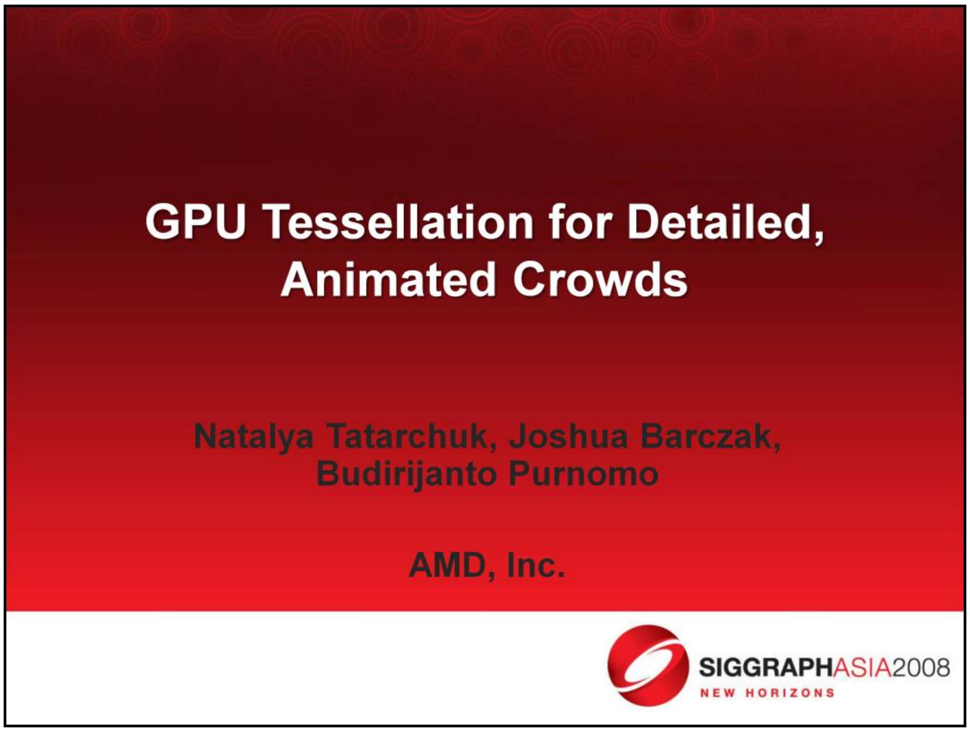
SIGGRAPHASIA2008

NEW HORIZONS



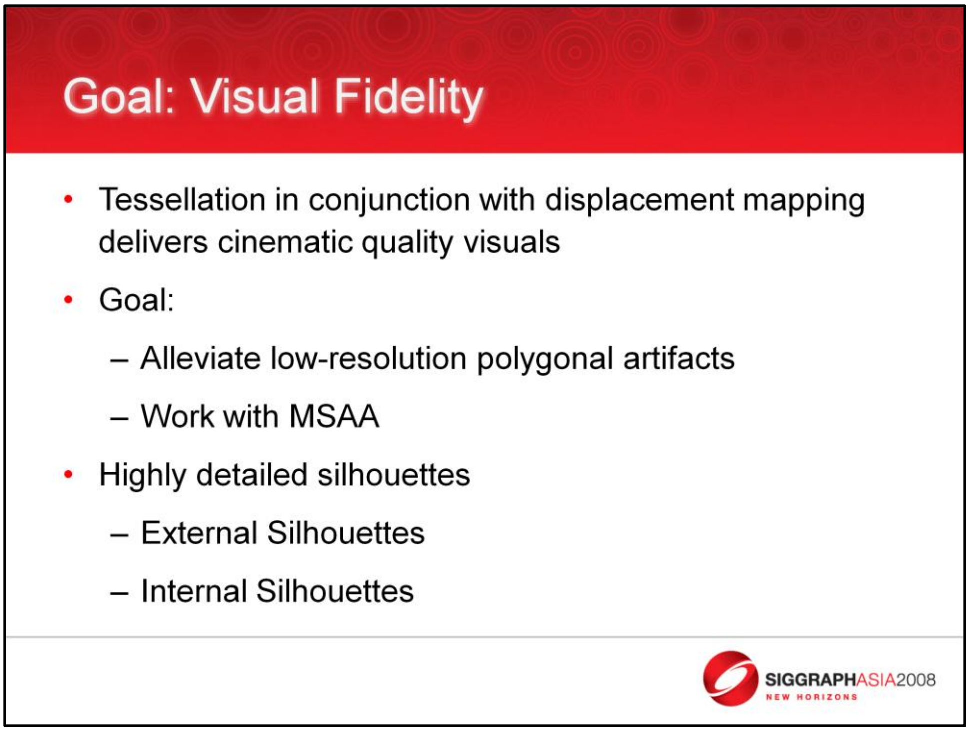
GPU Tessellation for Detailed, Animated Crowds

Natalya Tatarchuk, Joshua Barczak, Budirijanto Purnomo

AMD, Inc.

SIGGRAPHASIA2008 NEW HORIZONS

Hello! My name is Natalya Tatarchuk, and today I'll share with you our approaches for using GPU tessellation to render crowds of animated, detailed characters. This presentation builds upon our methods for GPU scene management for crowd rendering.



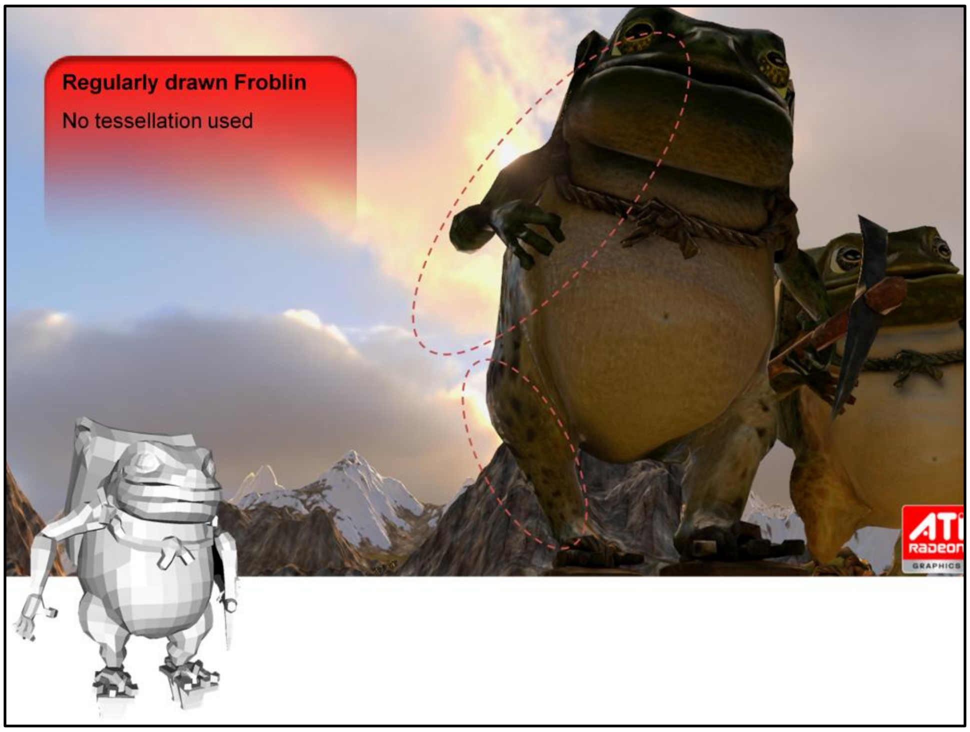
Goal: Visual Fidelity

- Tessellation in conjunction with displacement mapping delivers cinematic quality visuals
- Goal:
  - Alleviate low-resolution polygonal artifacts
  - Work with MSAA
- Highly detailed silhouettes
  - External Silhouettes
  - Internal Silhouettes

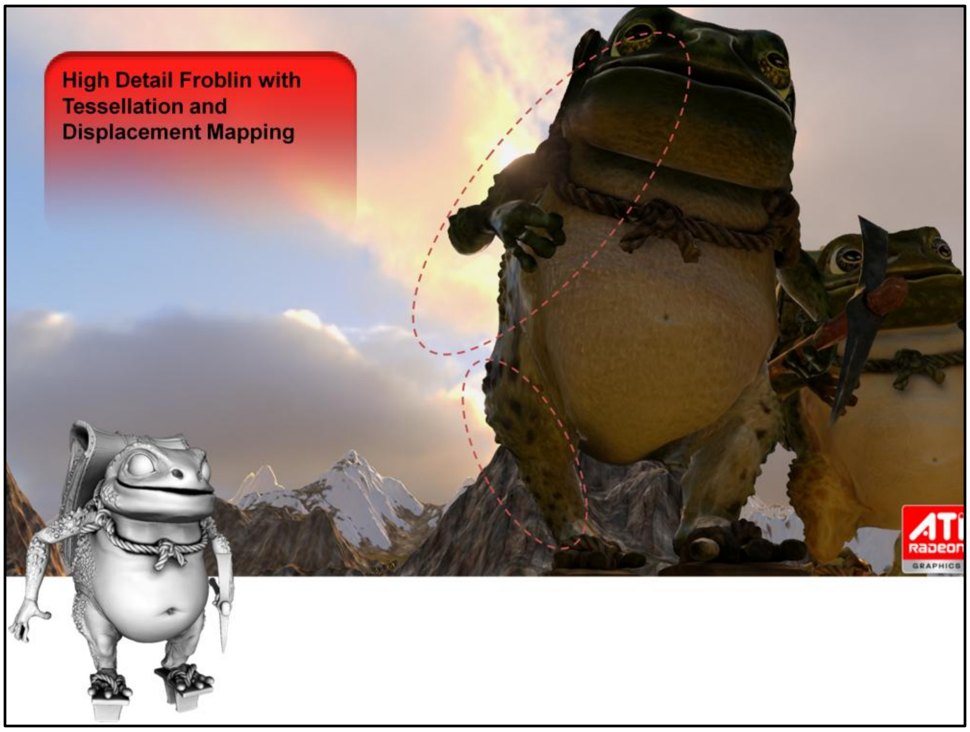
SIGGRAPHASIA2008
NEW HORIZONS

One of the goals of our work has been to increase visual fidelity for rendering the characters. The result needs to match the artistic vision for the character – and, ideally, surpass it!

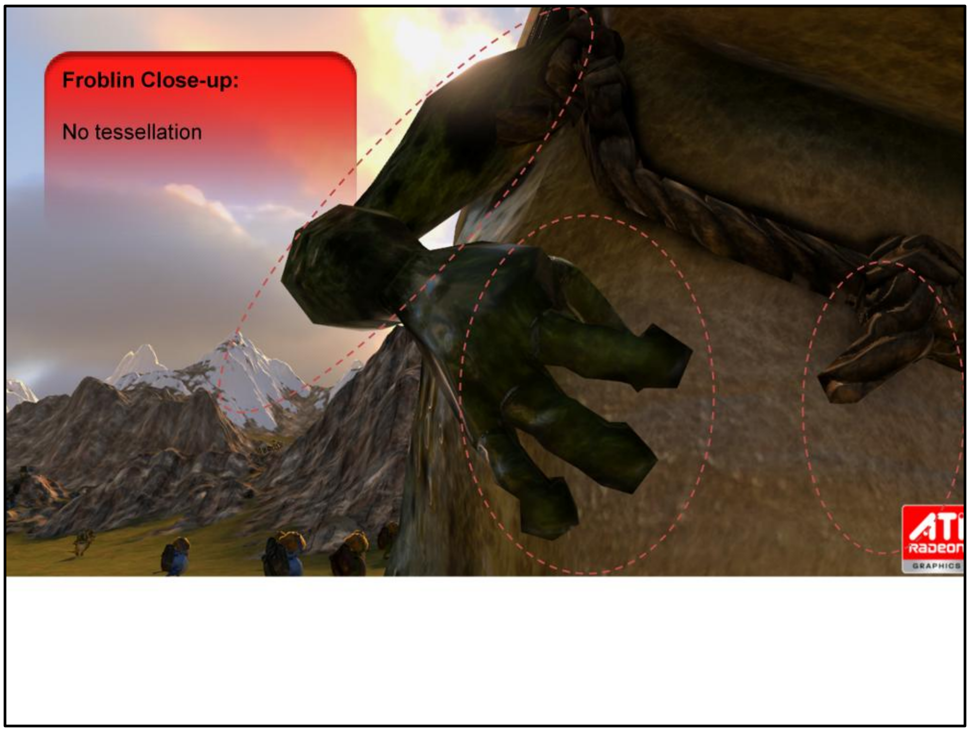
We sought a technique that allows detailed internal and external silhouettes, but that works coherently and seamlessly with MSAA. The use of GPU tessellation in conjunction with displacement mapping allows us to get closer to this goal.



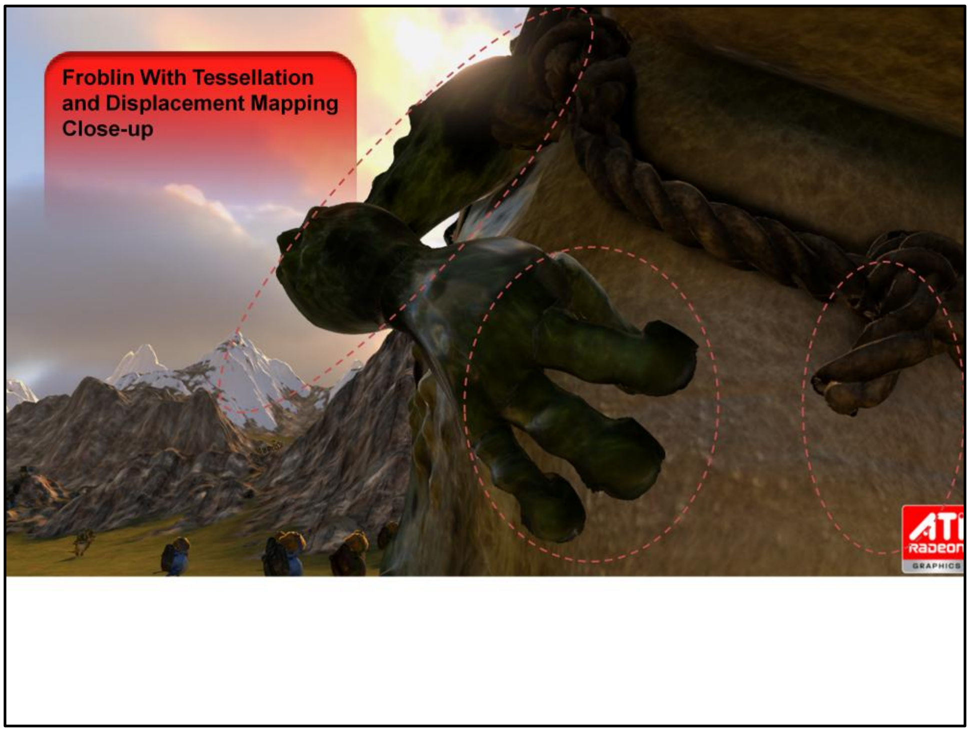
Here is an example of a character designed to meet the needs of current games – it's a low resolution mesh (around 5K triangles) for our Frog Goblin character, the Froblin. Notice the coarse silhouettes and lack of detail in the highlighted regions.


High Detail Froblin with Tessellation and Displacement Mapping
ATI
Radeon
GRAPHICS

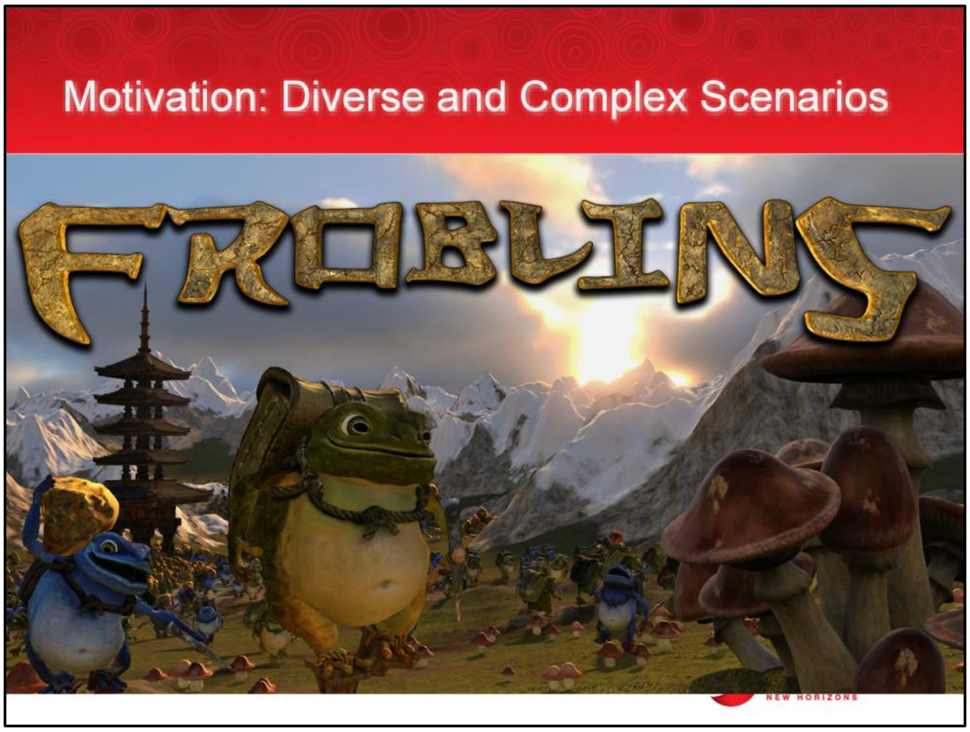
Using tessellation on characters (and other parts of the environment) allows superior detail and high quality animation. You can instantly see the difference in the amount of fine scale detail such as the bumps on his skin in this shot.



Here is a close up so that you can see the real difference between a conventional rendering without tessellation

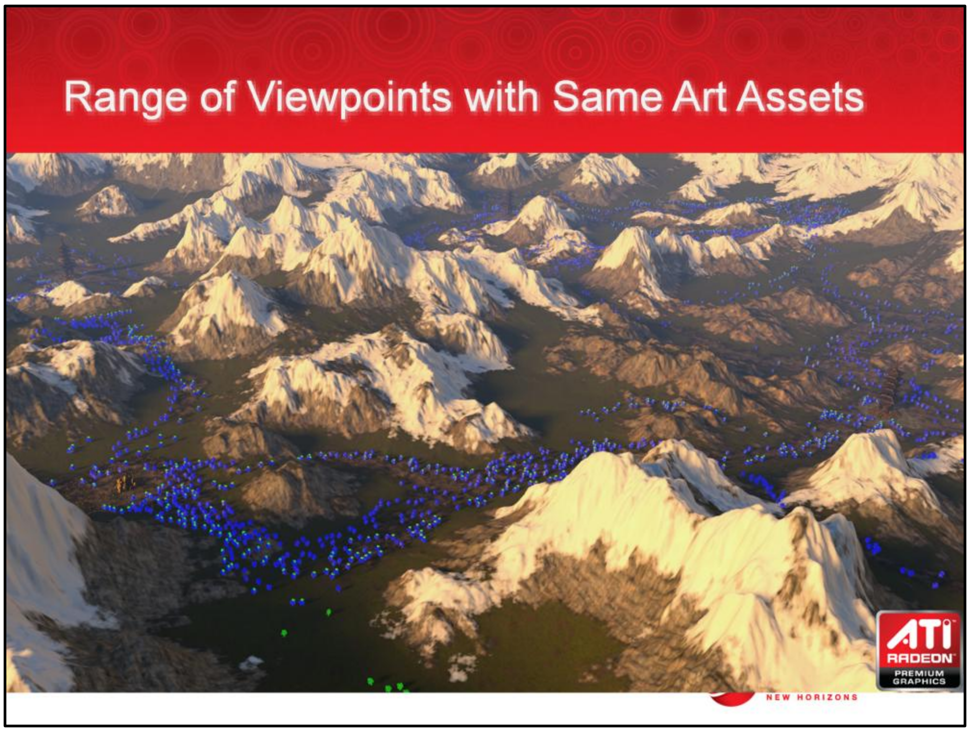


And with using GPU tessellation and displacement mapping. We definitely start getting a much better feel for the warts and wrinkles on this character’s skin.



Motivation: Diverse and Complex Scenarios

We wanted to be able to increase visual fidelity in variety of complex scenarios, not only in cases of rendering a single character. In our dynamic interactive environment, from the *Froblins* demo, we have a large world, full of thousands of characters, simulated directly on GPU.



Range of Viewpoints with Same Art Assets

As such, we had to be able to support a range of viewpoints with the same visual fidelity. Here you see a far away (bird's eye) shot with thousands of characters.

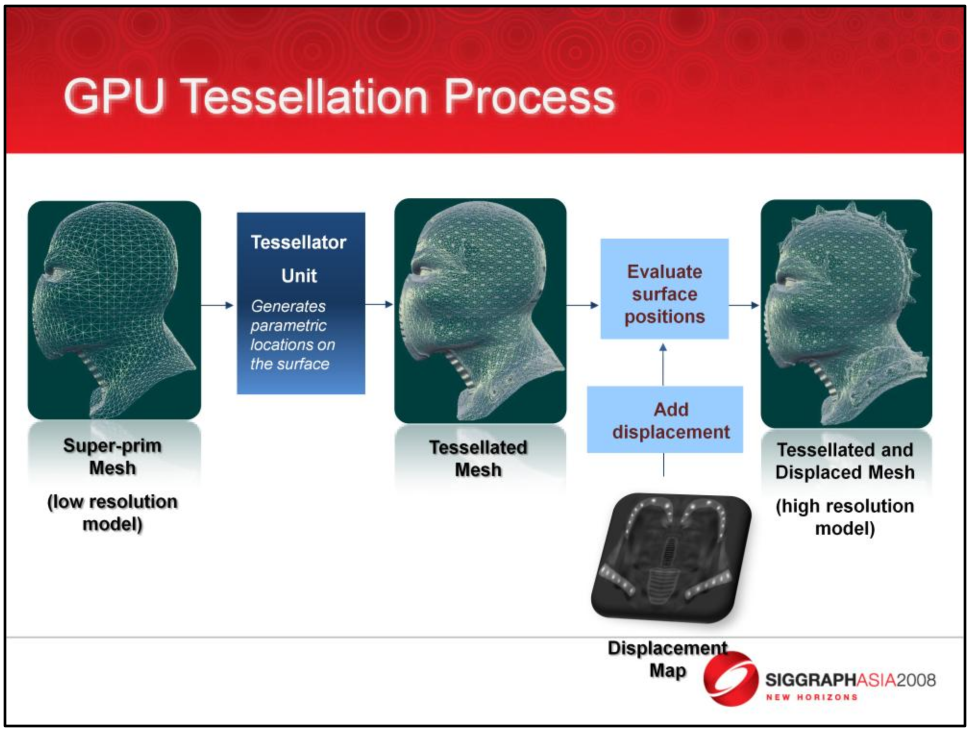
Range of Viewpoints with Same Art Assets

Displacement mapping is added to capture all the fine scale details of the character.

ATI RADEON PREMIUM GRAPHICS
NEW HORIZONS

However, when we would get close to any of these characters, we wanted to see a great deal of details on them – without losing significant performance or having to swap in new meshes.

Our solution takes advantage of GPU tessellation available on a number of recent commodity GPUs.



GPU Tessellation Process

We designed an API for a GPU tessellation pipeline taking advantage of hardware fixed-function tessellator unit available on recent consumer GPUs. **We start by rendering a low resolution mesh** (also referred to as *control cage*).

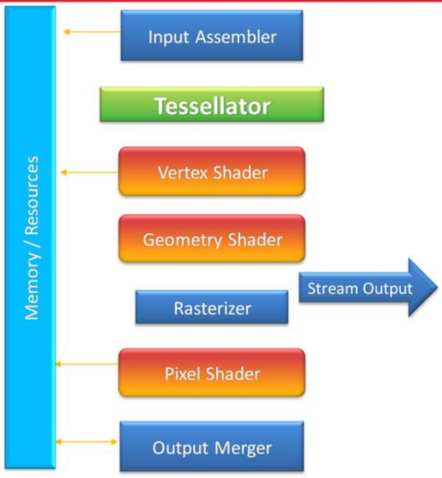
The tessellator unit generates parametric coordinates on the tessellated surface (the *uv*s) and topology connectivity for subdivided input primitives amplifying the original data up to 411 times.

The generated vertex data is directly consumed by the vertex shader invoked for each new vertex.

The super-primitive vertex IDs and barycentric coordinates are used to **evaluate the new surface position**.

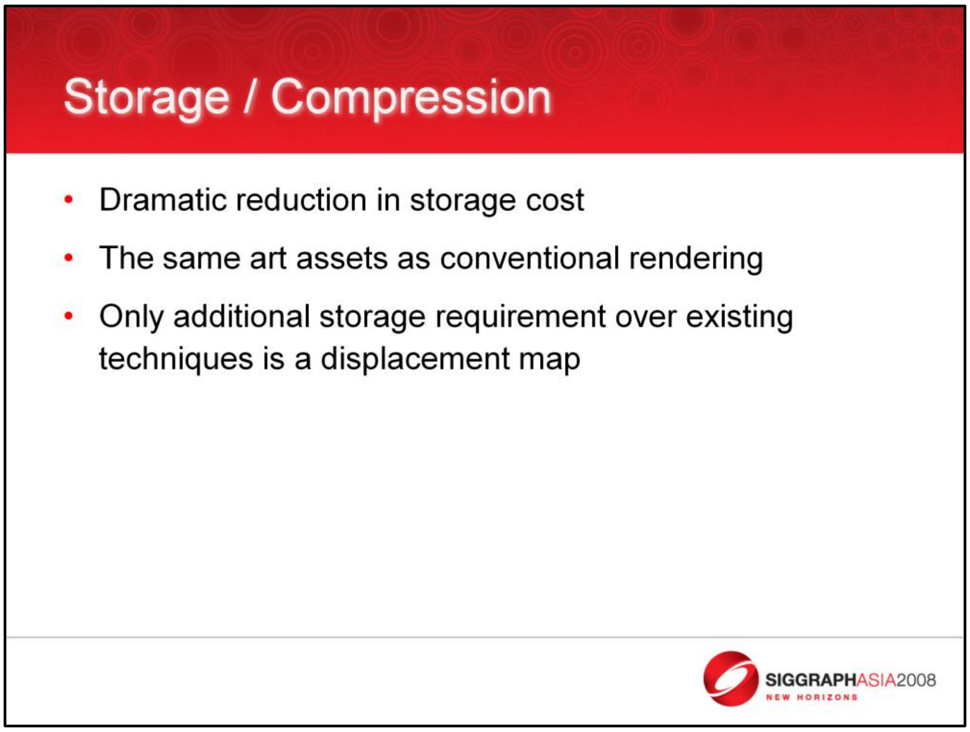
The amount of amplification can be controlled either by a per draw call tessellation level or by dynamically computing tessellation factors per-primitive edge for the input mesh.

GPU Tessellation Pipeline: Direct3D 10



SIGGRAPHASIA2008 NEW HORIZONS

With Direct3D 10 API and beyond, we can combine tessellation with several additional features, such as geometry shaders, stream out, and instancing. We can also use these features for an animation and transformation pass for the low-resolution control cage.

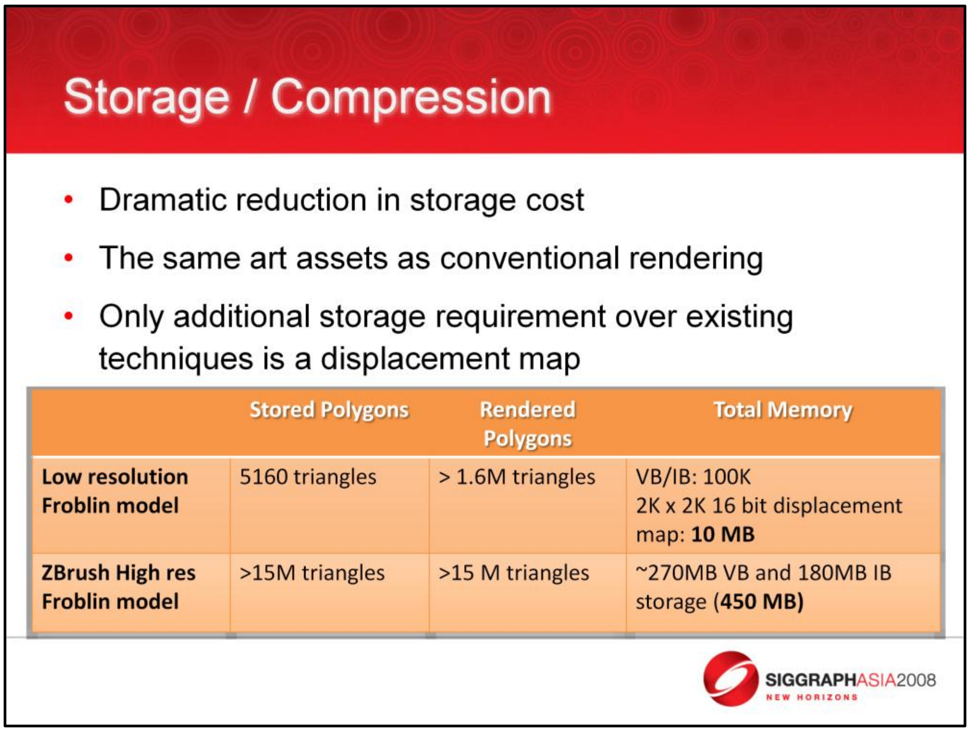

Storage / Compression

- Dramatic reduction in storage cost
- The same art assets as conventional rendering
- Only additional storage requirement over existing techniques is a displacement map

SIGGRAPHASIA2008
NEW HORIZONS

There several important advantages to using GPU tessellation. We can design one set of assets to use with and without GPU tessellation (with an addition of a displacement map). The latter can be used on systems without GPU tessellation to do complex per-pixel lighting effects.

Fine geometric detail are captured by the displacement map. Animation data is only stored for the control cage (the low resolution mesh).

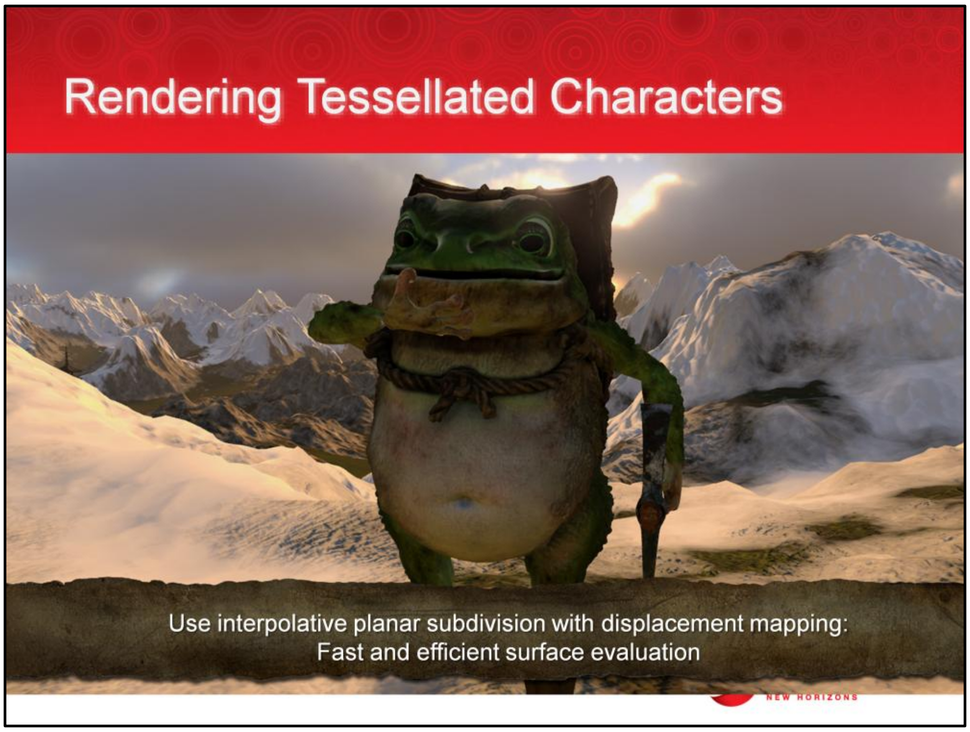


Storage / Compression

- Dramatic reduction in storage cost
- The same art assets as conventional rendering
- Only additional storage requirement over existing techniques is a displacement map

	Stored Polygons	Rendered Polygons	Total Memory
Low resolution Froblin model	5160 triangles	> 1.6M triangles	VB/IB: 100K 2K x 2K 16 bit displacement map: **10 MB**
ZBrush High res Froblin model	>15M triangles	>15 M triangles	~270MB VB and 180MB IB storage (**450 MB)**

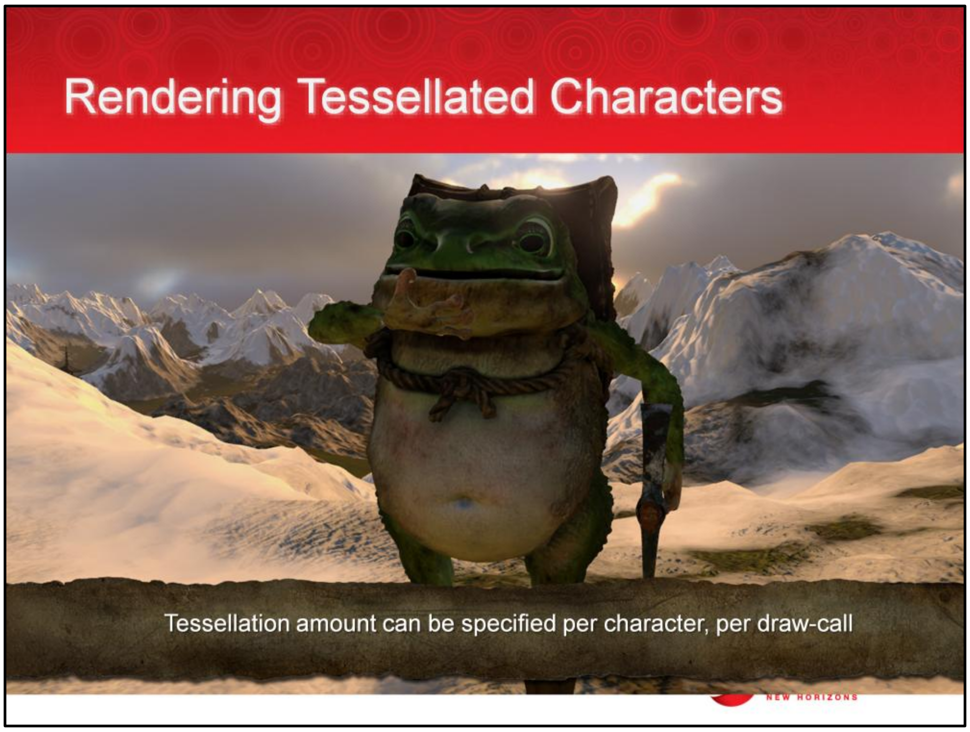
Thus we can think of hardware tessellation as an effective form of geometry compression. We can see this in the table where we compare the footprint for a GPU tessellation-ready model (a little over 10 MB) with a comparable high resolution model for the full character mesh (450 MB). We see that for a modest increase in memory footprint, we dramatically increase the total polygonal count for the rendered mesh when using GPU tessellation.



Rendering Tessellated Characters

Use interpolative planar subdivision with displacement mapping:
Fast and efficient surface evaluation

We use interpolative planar subdivision with displacement to efficiently render our highly detailed characters. The benefits of this approach (as opposed to higher order surface evaluation) is extremely fast computation of new surface positions. Additionally, the interpolation happens on triangular domain, which allows us to use the same low-resolution mesh assets as traditional rendering for the control cage.



Rendering Tessellated Characters

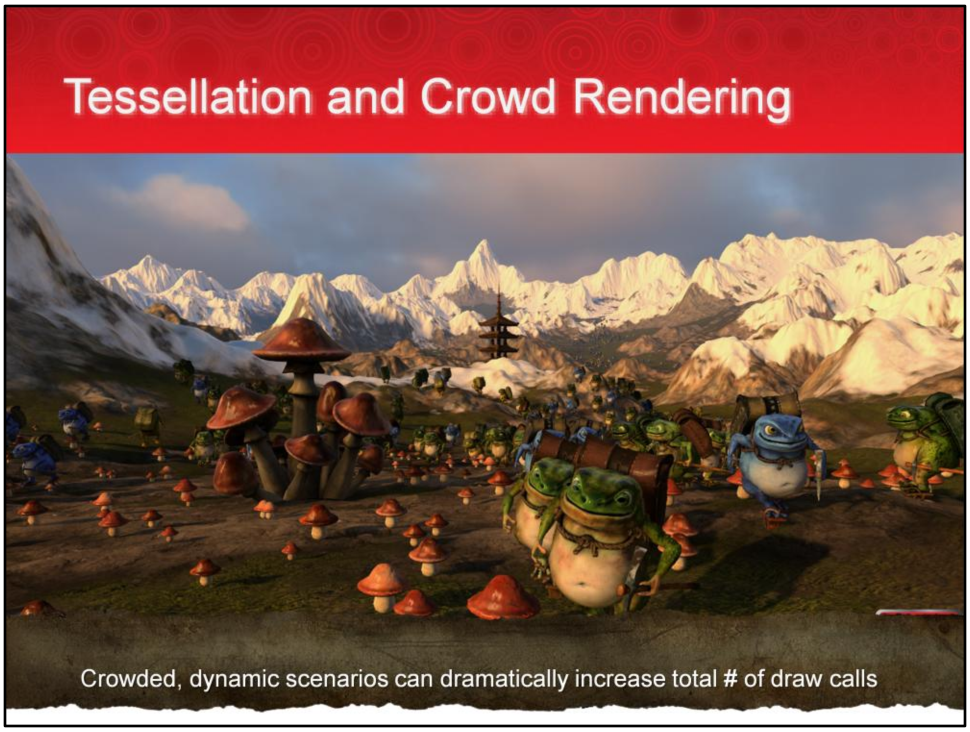
Tessellation amount can be specified per character, per draw-call

We can use tessellation to control how fine we are going to subdivide this character’s mesh.

We specify tessellation level, controlling the amount of amplification, per draw-call.

We can use the information about character location on the screen or other factors to control the desired amount of details.

Per-draw call tessellation level specification works well for rendering individual characters, such as this contemplative Froblin here.

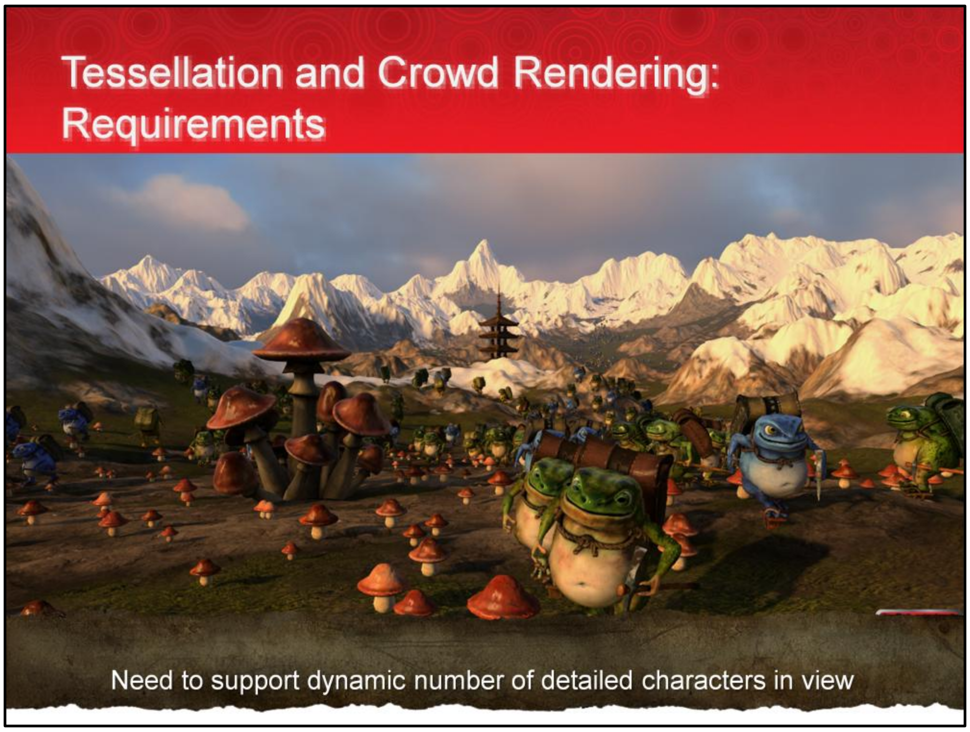


Tessellation and Crowd Rendering

Crowded, dynamic scenarios can dramatically increase total # of draw calls

However, this approach doesn’t necessarily scale well for dynamically simulated scenarios, as in this picture (and our environment).

If we use this strategy, the amount of draw-calls can increase drastically in crowded scenarios, as the number of tessellated characters we wish to render increases.



Tessellation and Crowd Rendering: Requirements

Need to support dynamic number of detailed characters in view

In many scenarios, we may not know the exact number of tessellated characters in a given view. This is particularly important in cases where the simulation happens on the GPU, and changes interactively, with no a priori control.

Tessellation and Crowd Rendering: Requirements

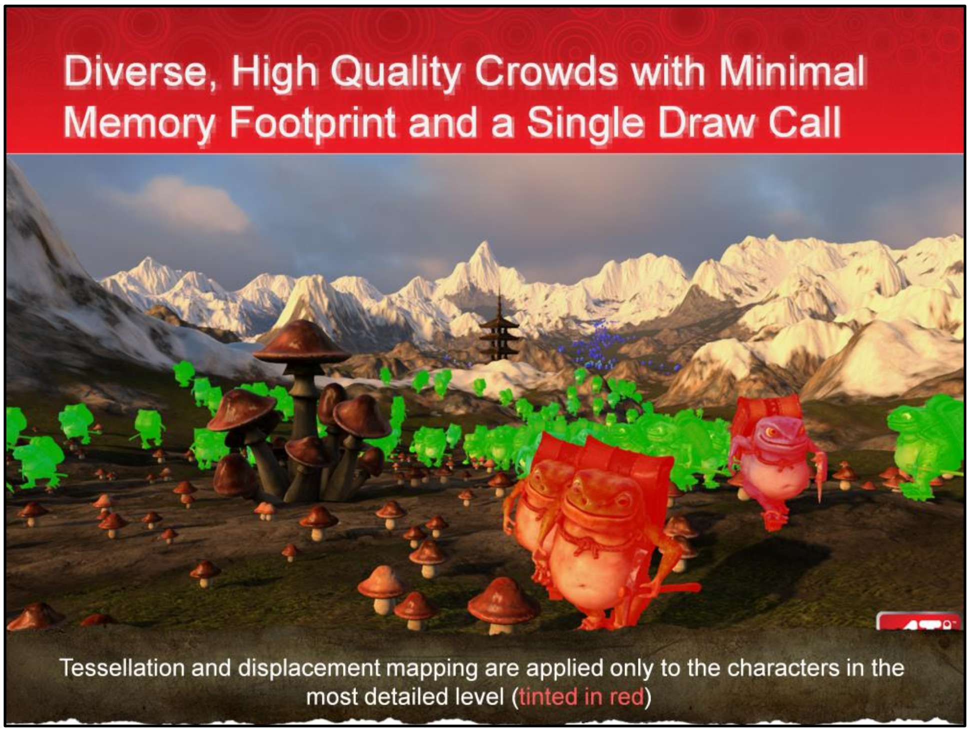
Need stable frame rate, regardless of the number of tessellated characters in view

We need to be able to render high quality detailed characters, even if we suddenly enter extremely crowded areas, without bringing the application down to a crawl.

Tessellation and Crowd Rendering: Solution

Combine tessellation and **instancing** for efficient rendering

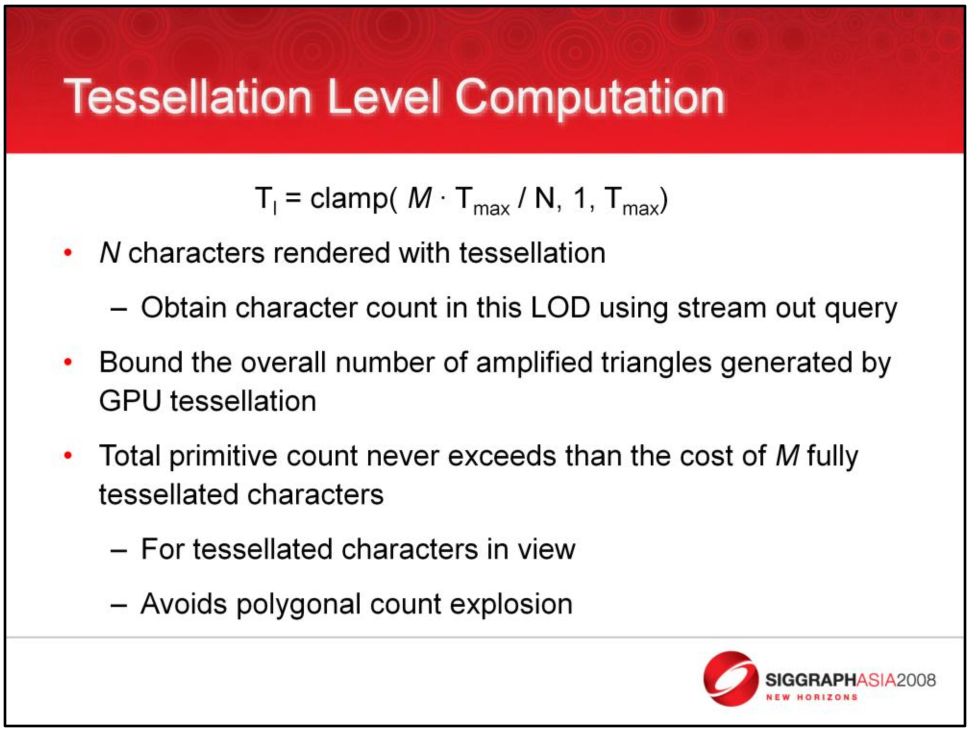
To address this challenge, we utilize DirectX 10.1 features for level of detail management to render our froblins as an army of **instanced characters**.



Diverse, High Quality Crowds with Minimal Memory Footprint and a Single Draw Call

Tessellation and displacement mapping are applied only to the characters in the most detailed level (tinted in red)

We can use this method along with stream out buffers and geometry shaders for GPU scene management and texture arrays to create visually interesting and varied crowd of characters. Here in the example, we note that the creatures tinted with red are rendered with GPU tessellation, the green froblins are rendered with conventional rendering and the blue froblins use simplified geometry.



Tessellation Level Computation

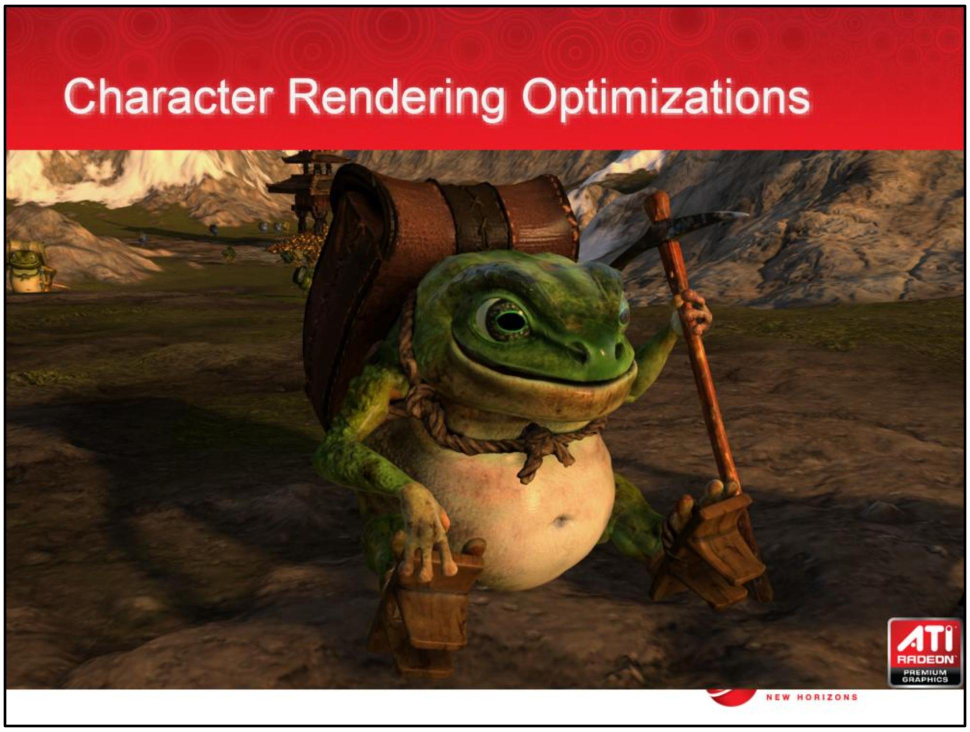
$$T_l = \text{clamp}(\ M \cdot T_{max} / N,\ 1,\ T_{max})$$

- *N* characters rendered with tessellation
  - Obtain character count in this LOD using stream out query
- Bound the overall number of amplified triangles generated by GPU tessellation
- Total primitive count never exceeds than the cost of *M* fully tessellated characters
  - For tessellated characters in view
  - Avoids polygonal count explosion

SIGGRAPHASIA2008
NEW HORIZONS

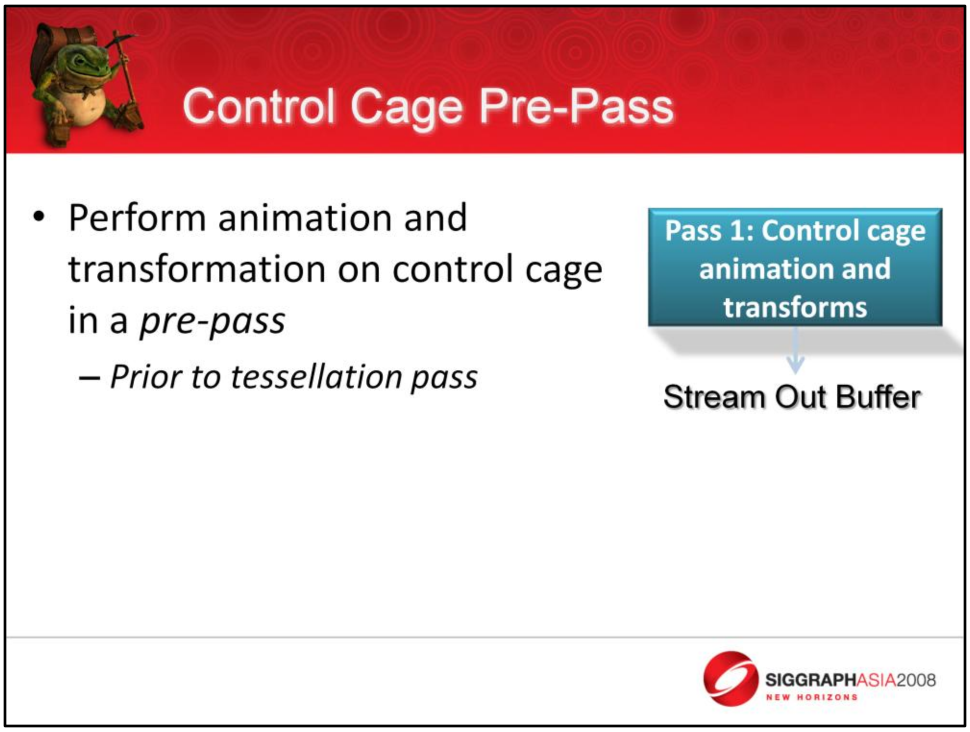
The tessellation level is calculated as follows:

Here, T_i is the tessellation level to be used for character instances in the first detail level, *N* is the number of character instances in the first detail level, and T_{max} is the maximum tessellation level to use for a single character. This scheme effectively bounds the number of triangles created by the tessellator, and ensures that the primitive count will never increase by more than the cost of *M* fully tessellated characters. If there are more than *M* such characters in the view frustum, this scheme will divide the tessellated triangles evenly among them. While this can lead to slight visual popping as the size of the crowd changes dramatically from one frame to the next, in a lively scene with numerous animated characters this popping is very hard to perceive.



Character Rendering Optimizations

When we render animated characters with subdivision, we need to perform animation calculations on the control mesh (the superprimitives), and then interpolate between the animated superprimitive vertices. A brute force approach of transforming and animating the superprimitive vertices in the evaluation shader wastes performance and bandwidth due to redundant computations – all newly generated tessellated vertices would perform the same computations as on the original vertices. Because hardware tessellation can generate millions of additional triangles, it is essential to minimize the amount of per-vertex computations post-tessellation, and to perform animation calculations only once per control vertex.



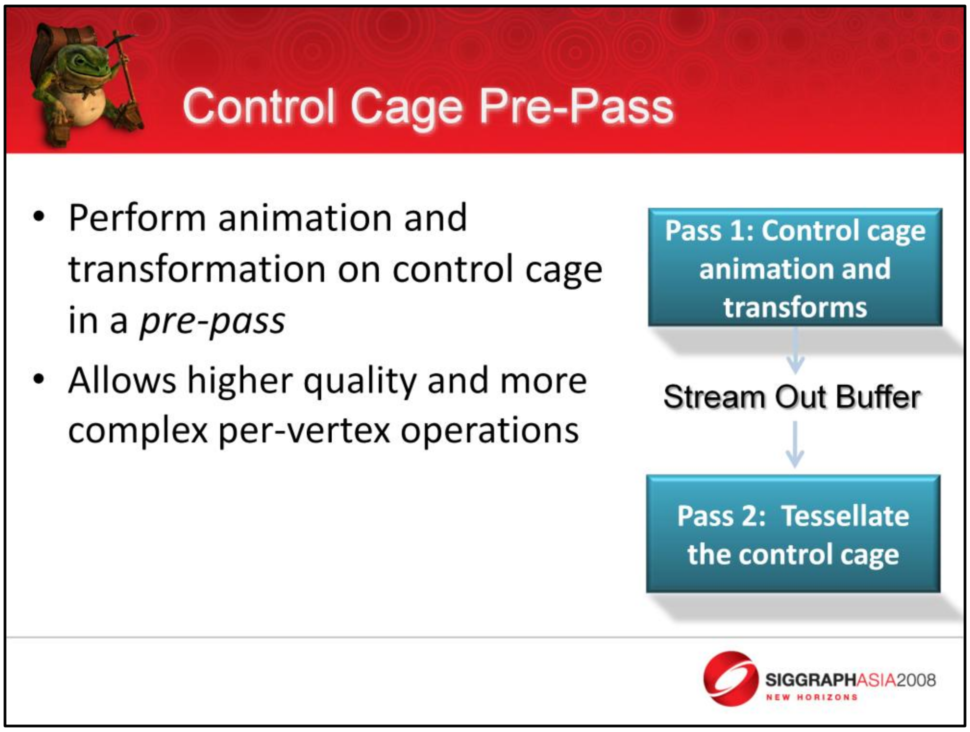
Control Cage Pre-Pass

- Perform animation and transformation on control cage in a *pre-pass*
  - *Prior to tessellation pass*

Pass 1: Control cage animation and transforms
Stream Out Buffer
SIGGRAPHASIA2008
NEW HORIZONS

We improve performance with a multi-pass approach for rendering out animated characters. We compute control cage pre-pass, where we can compute all relevant computations for the original low resolution mesh, such as animation and vertex transformations. This method is general and takes advantage of Direct3D® 10 *stream out* functionality.

In the first pass we perform input mesh animation and transformations; rendering the base mesh vertices as instanced sets of point primitives, skinning them, and streaming out the results. Since we are performing this computation on the coarse input mesh, we can afford higher quality skinning as well as significantly reduce geometry transform cost.

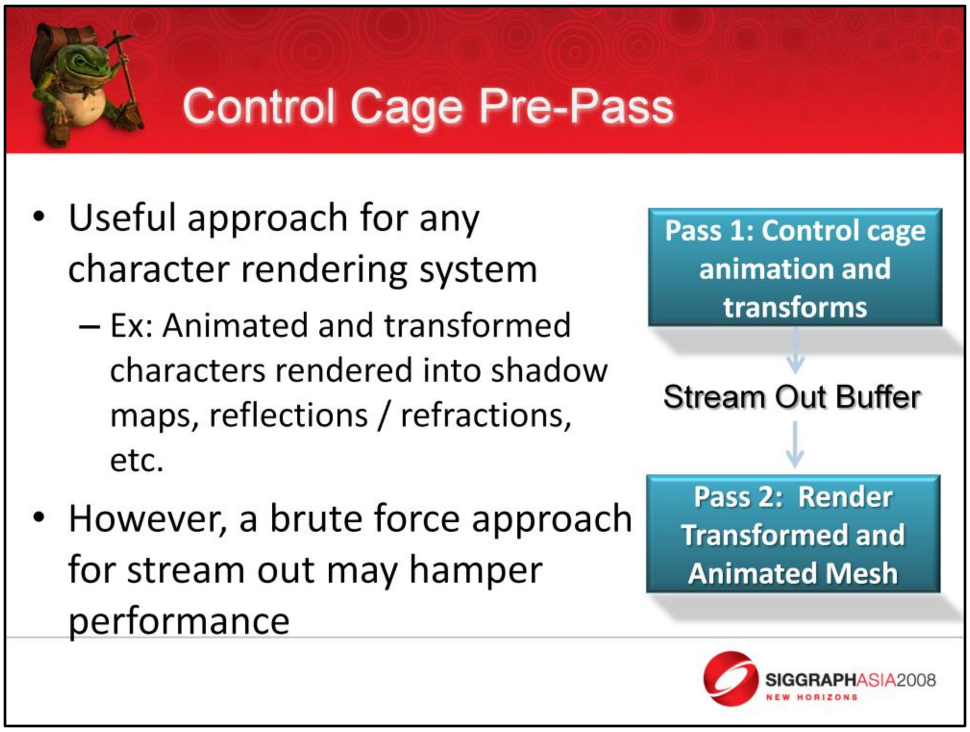


Control Cage Pre-Pass

- Perform animation and transformation on control cage in a *pre-pass*
- Allows higher quality and more complex per-vertex operations

Pass 1: Control cage animation and transforms
Stream Out Buffer
Pass 2: Tessellate the control cage
SIGGRAPHASIA2008
NEW HORIZONS

In the next pass, we tessellate the already animated and transformed mesh, using the original input primitive vertices' vertex ID and instance ID to retrieve and interpolate the transformed vertices from the stream out buffer, and apply displacement mapping.

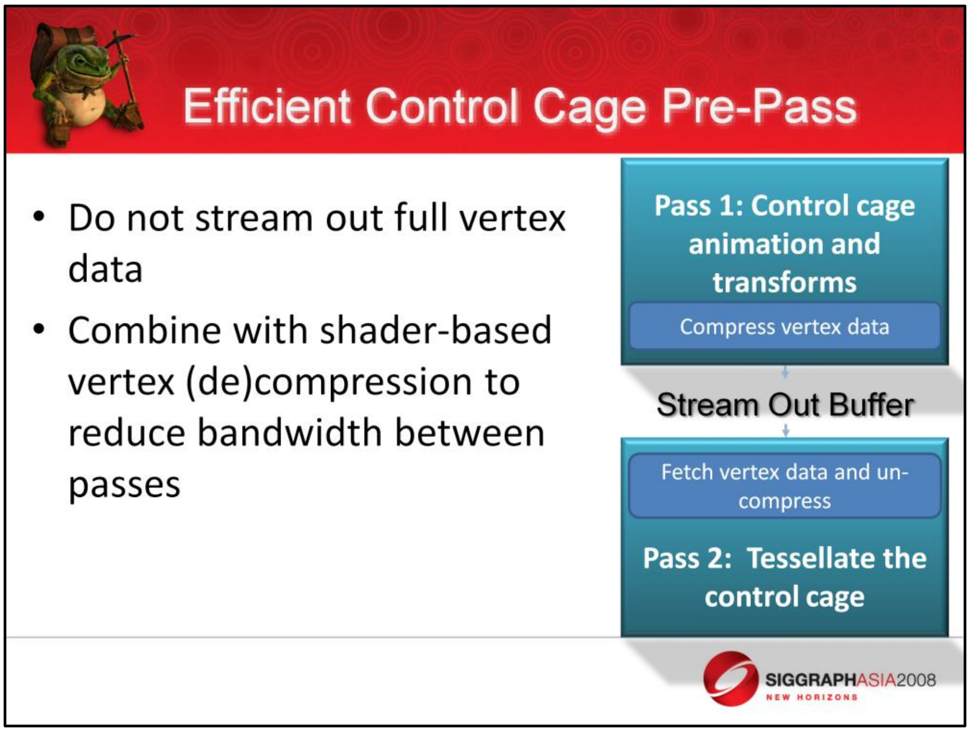


Control Cage Pre-Pass

- Useful approach for any character rendering system
  - Ex: Animated and transformed characters rendered into shadow maps, reflections / refractions, etc.
- However, a brute force approach for stream out may hamper performance

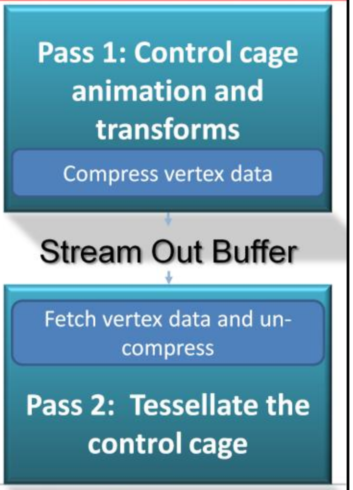
Pass 1: Control cage animation and transforms
Stream Out Buffer
Pass 2: Render Transformed and Animated Mesh
SIGGRAPHASIA2008
NEW HORIZONS

Note that using this multi-pass method for control cage rendering is beneficial not only for rendering tessellated characters, but for any rendering pipeline where we wish to reuse results of expensive vertex operations multiple times. For example, we can use the results of the first pass for our animated and transformed characters for rendering into shadow maps and cube maps for reflections.



Efficient Control Cage Pre-Pass

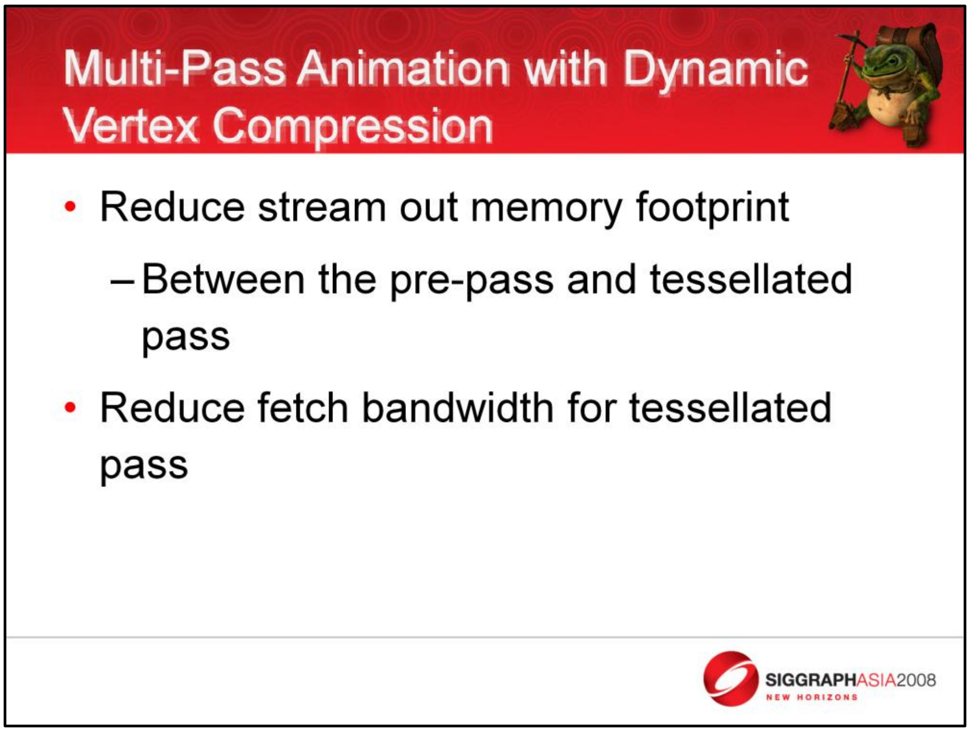
- Do not stream out full vertex data
- Combine with shader-based vertex (de)compression to reduce bandwidth between passes


Pass 1: Control cage animation and transforms
Compress vertex data
Stream Out Buffer
Fetch vertex data and un-compress
Pass 2: Tessellate the control cage

Although it is helpful to stream and re-use the animation calculations, this alone is not fully effective.

The vertex data will be streamed at full precision, and the evaluation shader must still pay a large cost in memory bandwidth and fetch instructions to retrieve it.

We augmented our control cage multi-pass method with vertex compression and decompression.

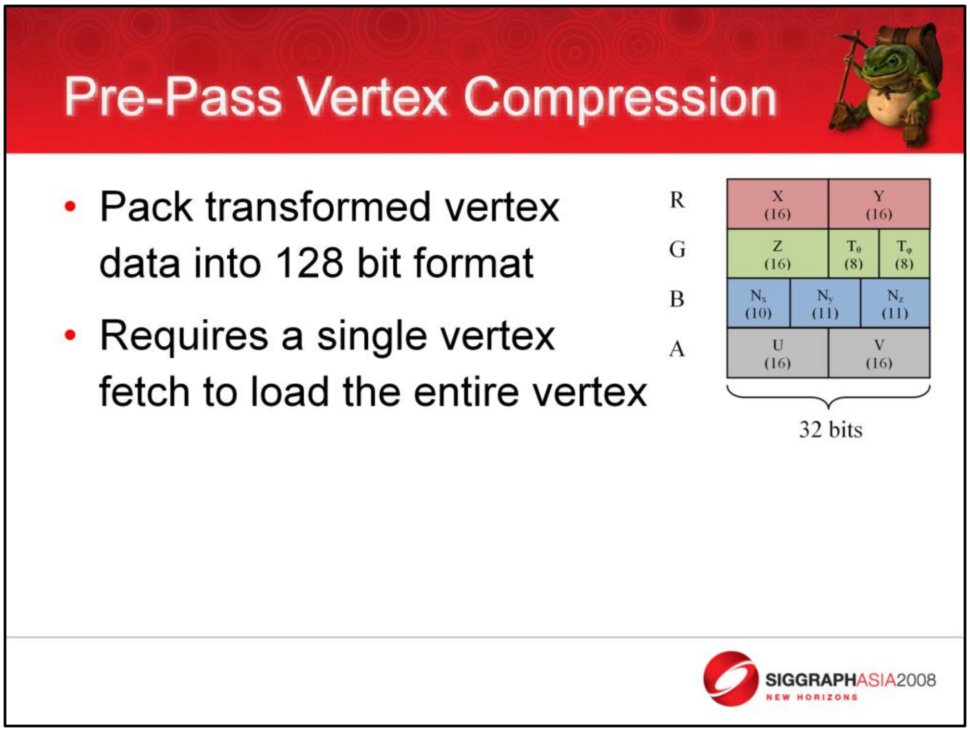


Multi-Pass Animation with Dynamic Vertex Compression

- Reduce stream out memory footprint
  - Between the pre-pass and tessellated pass
- Reduce fetch bandwidth for tessellated pass

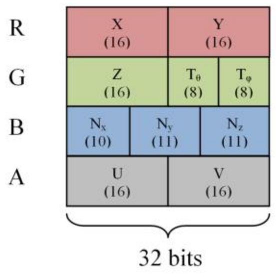
SIGGRAPHASIA2008 NEW HORIZONS

This modification helps reduce the amount of memory being streamed out per character, as well as reduce vertex fetch and vertex cache reuse for the evaluation shader.



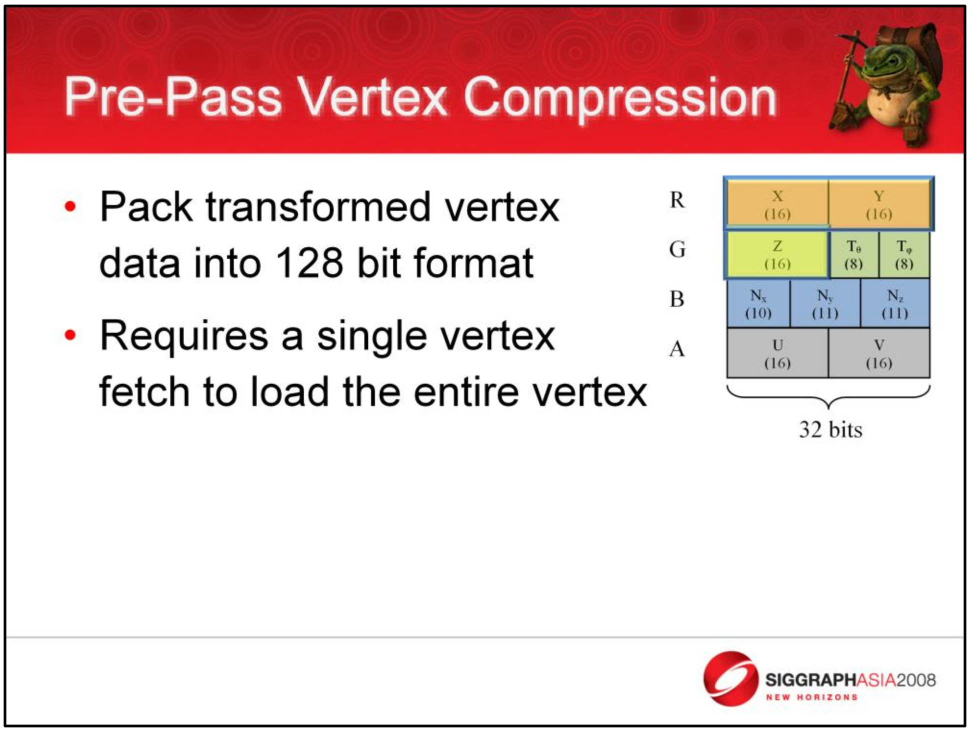
Pre-Pass Vertex Compression

- Pack transformed vertex data into 128 bit format
- Requires a single vertex fetch to load the entire vertex



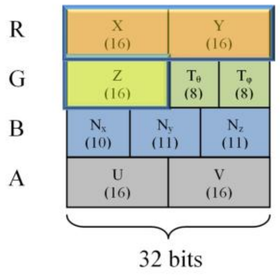
SIGGRAPHASIA2008
NEW HORIZONS

We use a compression scheme to pack the transformed vertices into a compact 128-bit format, allowing the tessellation pass to load a full set of vertex data using only one vertex fetch.



Pre-Pass Vertex Compression

- Pack transformed vertex data into 128 bit format
- Requires a single vertex fetch to load the entire vertex

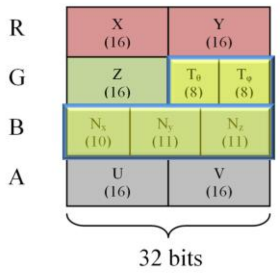


SIGGRAPHASIA2008 NEW HORIZONS

We compress **vertex positions** by expressing them as fixed-point values which are used to interpolate the corners of a sufficiently large bounding box that is local to each character.

Pre-Pass Vertex Compression

- Pack transformed vertex data into 128 bit format
  - Only one vertex fetch to load the entire vertex

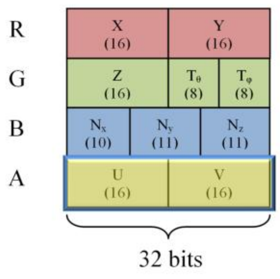


SIGGRAPHASIA2008 NEW HORIZONS

We can **compress the tangent frame** by converting the basis vectors to spherical coordinates and quantizing them. Spherical coordinates are well suited to compressing unit length vectors, since every compressed value in the spherical domain corresponds to a unique unit-length vector.

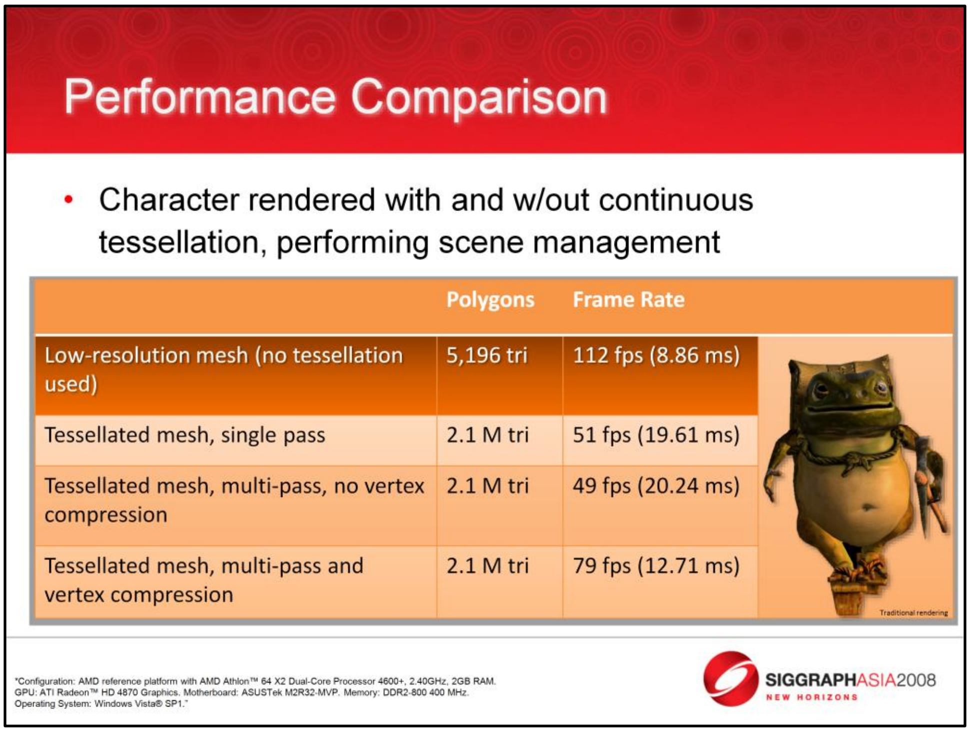
Pre-Pass Vertex Compression

- Pack transformed vertex data into 128 bit format
  - Only one vertex fetch to load the entire vertex



SIGGRAPHASIA2008 NEW HORIZONS

Texture coordinates are compressed by converting the *uv* coordinates into a pair of fixed-point values, using whatever bits are left. To ensure acceptable precision, this requires that the *uv* coordinates in the model to the 0-1 range, with no explicit tiling of textures by the artist.



Performance Comparison

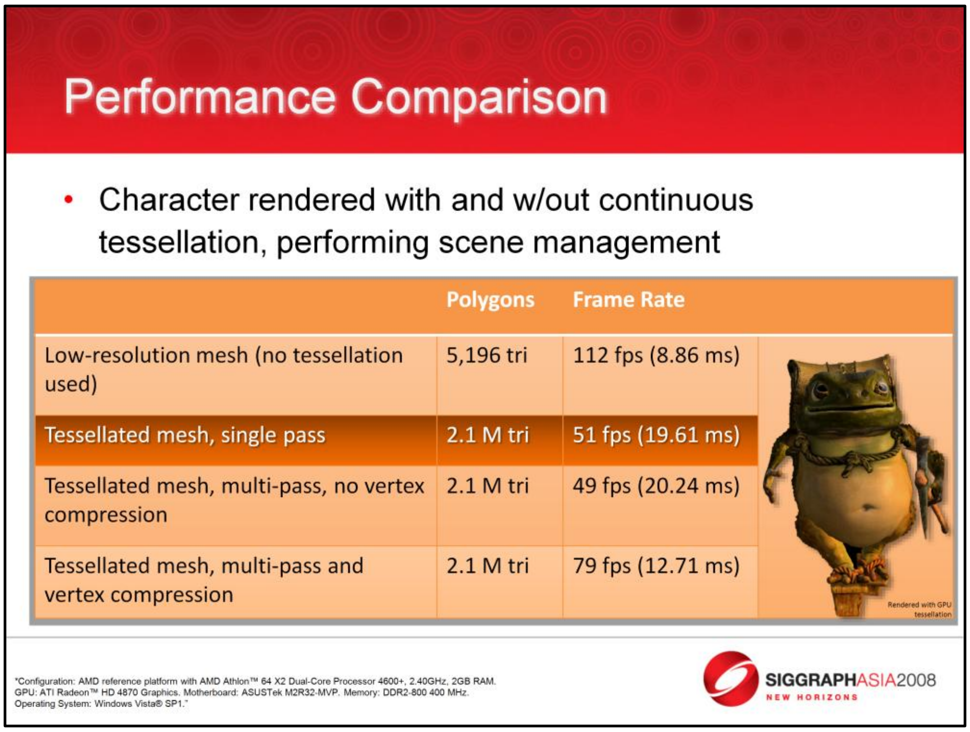
- Character rendered with and w/out continuous tessellation, performing scene management

	Polygons	Frame Rate
Low-resolution mesh (no tessellation used)	5,196 tri	112 fps (8.86 ms)
Tessellated mesh, single pass	2.1 M tri	51 fps (19.61 ms)
Tessellated mesh, multi-pass, no vertex compression	2.1 M tri	49 fps (20.24 ms)
Tessellated mesh, multi-pass and vertex compression	2.1 M tri	79 fps (12.71 ms)

*Configuration: AMD reference platform with AMD Athlon™ 64 X2 Dual-Core Processor 4600+, 2.40GHz, 2GB RAM. GPU: ATI Radeon™ HD 4870 Graphics. Motherboard: ASUSTek M2R32-MVP. Memory: DDR2-800 400 MHz. Operating System: Windows Vista® SP1."

SIGGRAPHASIA2008 NEW HORIZONS

Here we have performance analysis for different methods of rendering our character, starting from conventional (no GPU tessellation rendering).



Performance Comparison

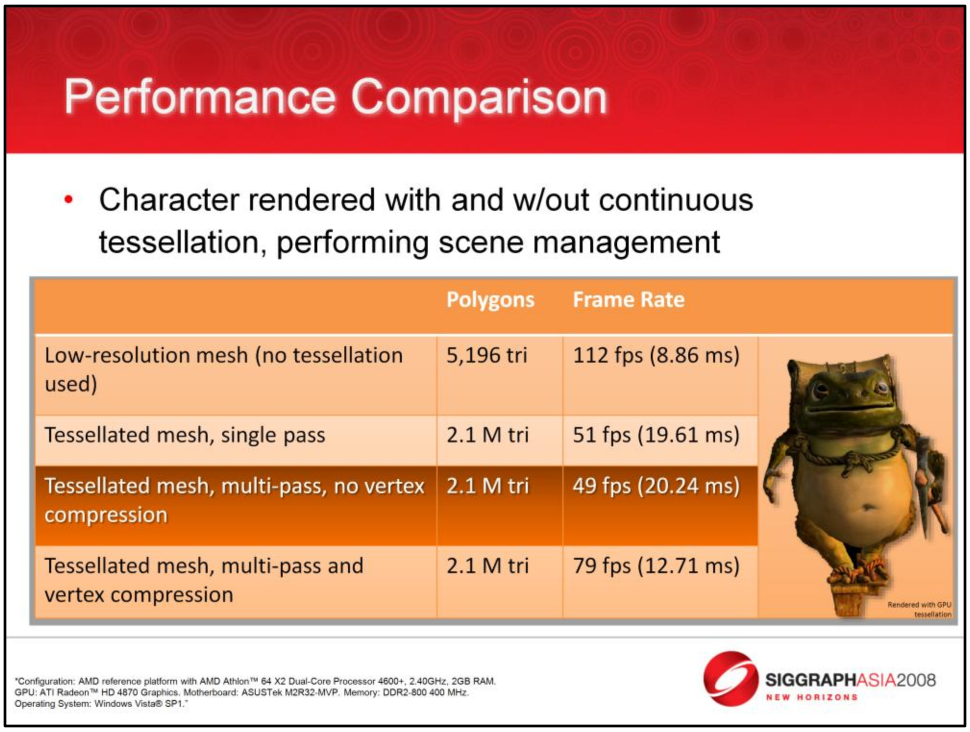
- Character rendered with and w/out continuous tessellation, performing scene management

	Polygons	Frame Rate	
Low-resolution mesh (no tessellation used)	5,196 tri	112 fps (8.86 ms)	Rendered with GPU tessellation
Tessellated mesh, single pass	2.1 M tri	51 fps (19.61 ms)	
Tessellated mesh, multi-pass, no vertex compression	2.1 M tri	49 fps (20.24 ms)	
Tessellated mesh, multi-pass and vertex compression	2.1 M tri	79 fps (12.71 ms)	

"Configuration: AMD reference platform with AMD Athlon™ 64 X2 Dual-Core Processor 4600+, 2.40GHz, 2GB RAM. GPU: ATI Radeon™ HD 4870 Graphics. Motherboard: ASUSTek M2R32-MVP. Memory: DDR2-800 400 MHz. Operating System: Windows Vista® SP1."

SIGGRAPHASIA2008
NEW HORIZONS

We notice that brute force conversion to GPU tessellation reduces performance (as compared to rendering the input low resolution mesh) to about 50% while dramatically increasing the overall quality of character rendering (as seen in this image).



Performance Comparison

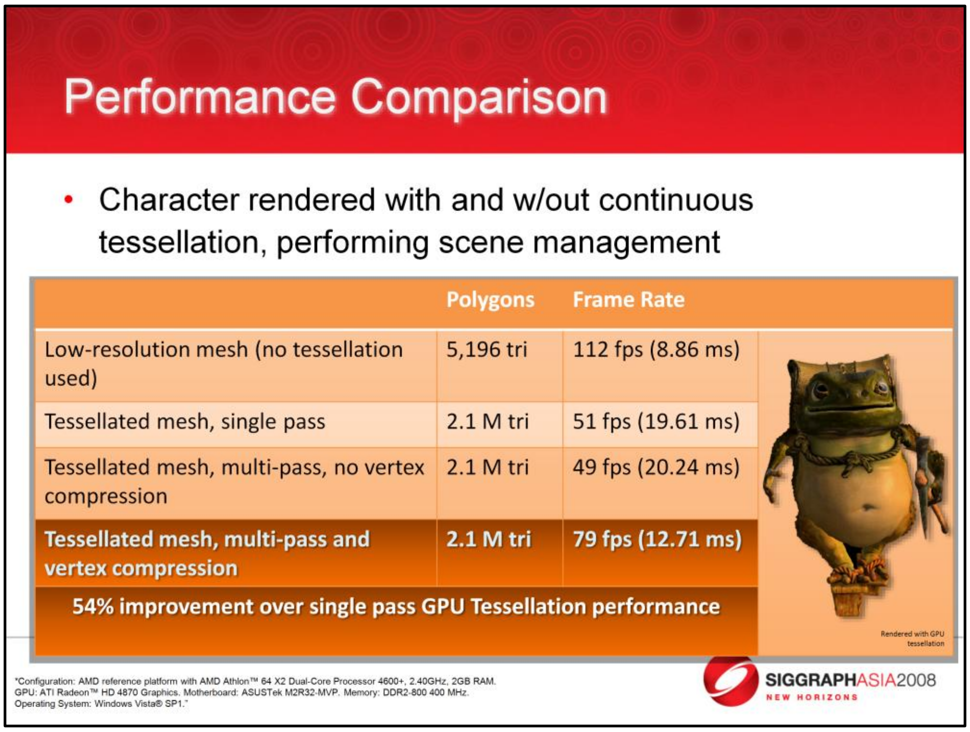
- Character rendered with and w/out continuous tessellation, performing scene management

	Polygons	Frame Rate	
Low-resolution mesh (no tessellation used)	5,196 tri	112 fps (8.86 ms)	
Tessellated mesh, single pass	2.1 M tri	51 fps (19.61 ms)	
Tessellated mesh, multi-pass, no vertex compression	2.1 M tri	49 fps (20.24 ms)	
Tessellated mesh, multi-pass and vertex compression	2.1 M tri	79 fps (12.71 ms)	

"Configuration: AMD reference platform with AMD Athlon™ 64 X2 Dual-Core Processor 4600+, 2.40GHz, 2GB RAM. GPU: ATI Radeon™ HD 4870 Graphics. Motherboard: ASUSTek M2R32-MVP. Memory: DDR2-800 400 MHz. Operating System: Windows Vista® SP1."

SIGGRAPHASIA2008
NEW HORIZONS

A straight-forward conversion of GPU tessellation to multi-pass rendering doesn't offer any performance improvement, as we notice here.



Performance Comparison

- Character rendered with and w/out continuous tessellation, performing scene management

	Polygons	Frame Rate
Low-resolution mesh (no tessellation used)	5,196 tri	112 fps (8.86 ms)
Tessellated mesh, single pass	2.1 M tri	51 fps (19.61 ms)
Tessellated mesh, multi-pass, no vertex compression	2.1 M tri	49 fps (20.24 ms)
Tessellated mesh, multi-pass and vertex compression	**2.1 M tri**	**79 fps (12.71 ms)**
54% improvement over single pass GPU Tessellation performance		

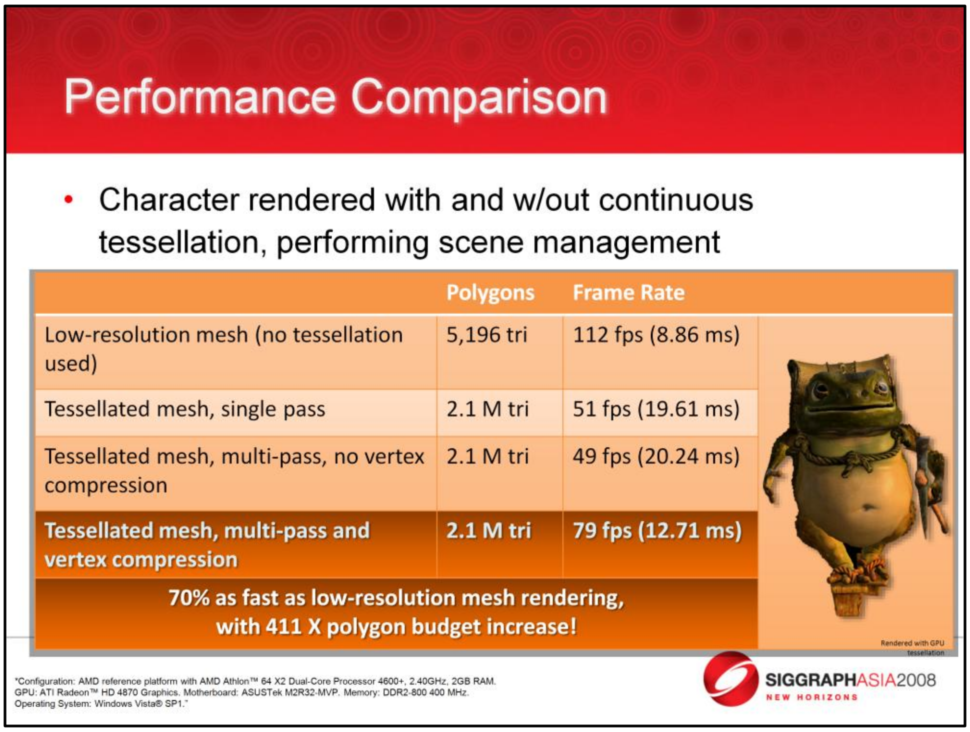
Rendered with GPU tessellation

"Configuration: AMD reference platform with AMD Athlon™ 64 X2 Dual-Core Processor 4600+, 2.40GHz, 2GB RAM. GPU: ATI Radeon™ HD 4870 Graphics. Motherboard: ASUSTek M2R32-MVP. Memory: DDR2-800 400 MHz. Operating System: Windows Vista® SP1."

SIGGRAPHASIA2008 NEW HORIZONS

However, although the compression scheme requires additional ALU cycles for both compression and decompression, this is more than compensated for by the reduction in memory bandwidth and fetch operations in the evaluation shader.

The multi-pass GPU tessellation approach with shader-based vertex compression provides 54% performance increase over the single pass GPU tessellation performance.



Performance Comparison

- Character rendered with and w/out continuous tessellation, performing scene management

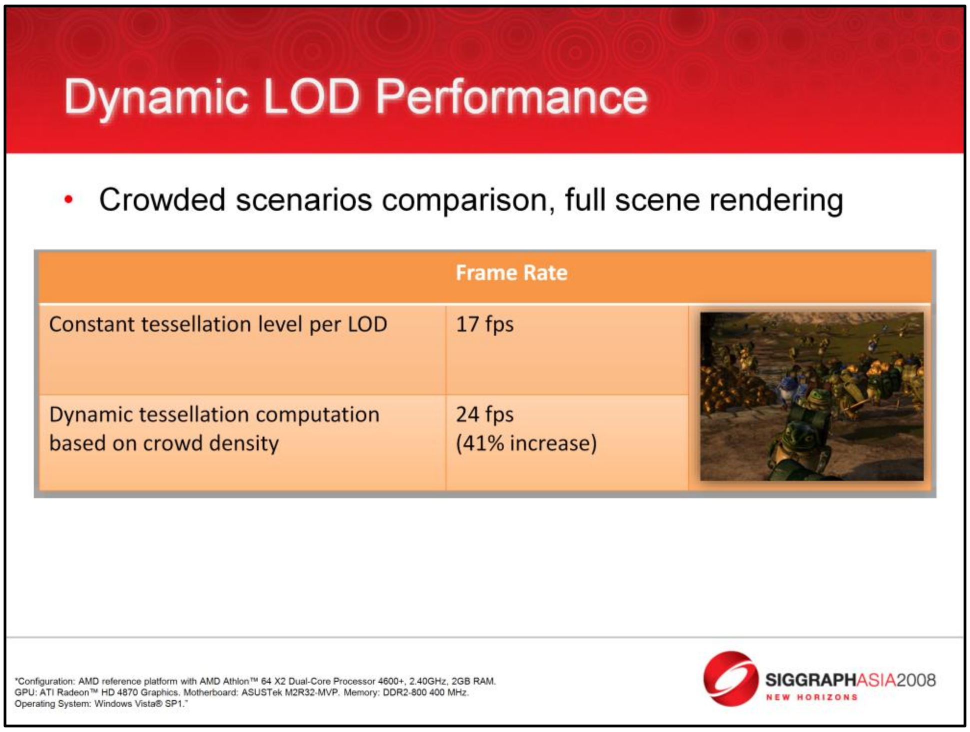
	Polygons	Frame Rate
Low-resolution mesh (no tessellation used)	5,196 tri	112 fps (8.86 ms)
Tessellated mesh, single pass	2.1 M tri	51 fps (19.61 ms)
Tessellated mesh, multi-pass, no vertex compression	2.1 M tri	49 fps (20.24 ms)
Tessellated mesh, multi-pass and vertex compression	**2.1 M tri**	**79 fps (12.71 ms)**

70% as fast as low-resolution mesh rendering, with 411 X polygon budget increase!

"Configuration: AMD reference platform with AMD Athlon™ 64 X2 Dual-Core Processor 4600+, 2.40GHz, 2GB RAM.
GPU: ATI Radeon™ HD 4870 Graphics. Motherboard: ASUSTek M2R32-MVP. Memory: DDR2-800 400 MHz.
Operating System: Windows Vista® SP1."

SIGGRAPHASIA2008 NEW HORIZONS

Furthermore, we notice that this method is 70% as fast as low-resolution mesh rendering, while rendering over 411 times MORE polygons for our high-quality character!



Dynamic LOD Performance

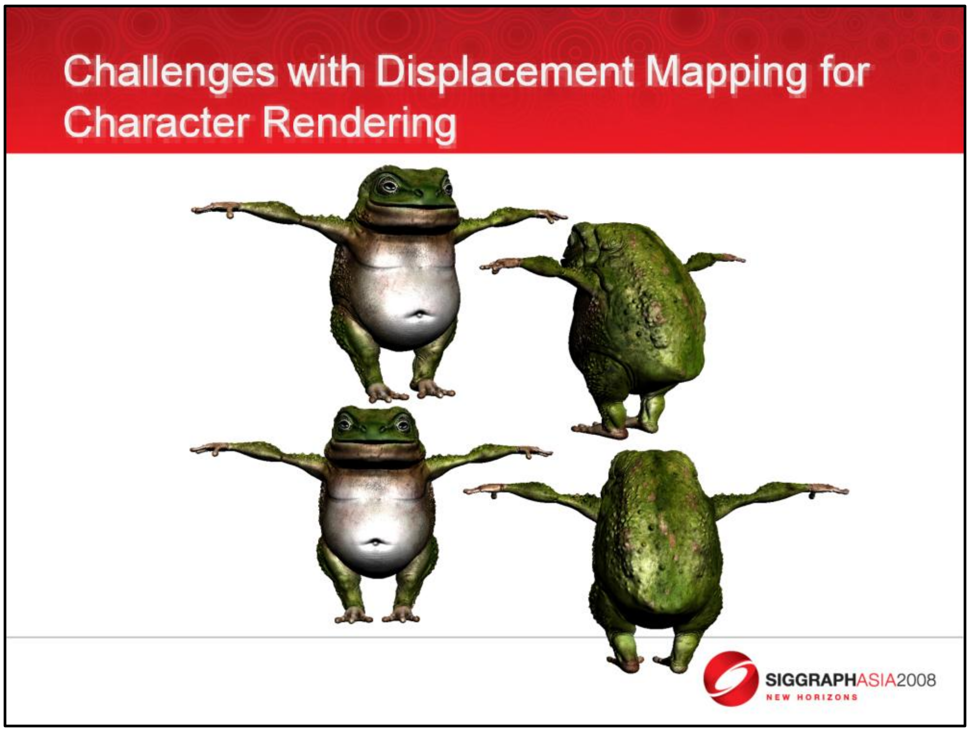
- Crowded scenarios comparison, full scene rendering

	Frame Rate	
Constant tessellation level per LOD	17 fps	
Dynamic tessellation computation based on crowd density	24 fps (41% increase)	

"Configuration: AMD reference platform with AMD Athlon™ 64 X2 Dual-Core Processor 4600+, 2.40GHz, 2GB RAM. GPU: ATI Radeon™ HD 4870 Graphics. Motherboard: ASUSTek M2R32-MVP. Memory: DDR2-800 400 MHz. Operating System: Windows Vista® SP1."

SIGGRAPHASIA2008
NEW HORIZONS

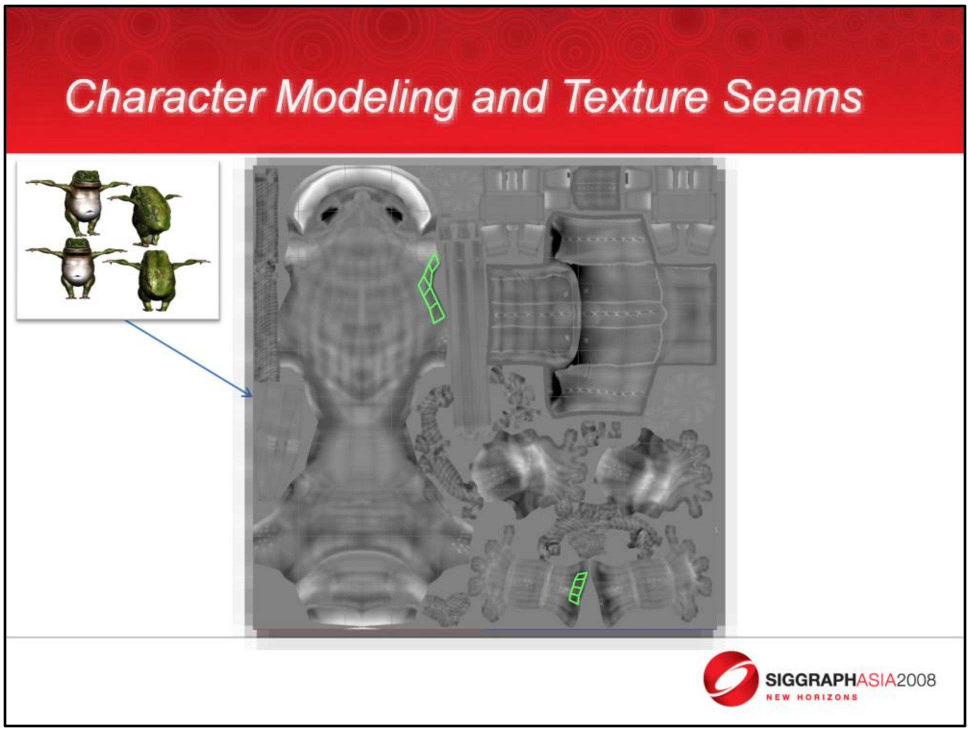
In this table, we compare performance of using dynamic method for computing tessellation amount based on crowd density versus a statically specified tessellation level. We notice that our method provides 41% increase in overall frame performance, which is impressive considering a huge number of other elements present in this frame (performing GPU simulation, rendering cascade shadows, post-processing, just to name a few).



Challenges with Displacement Mapping for Character Rendering

SIGGRAPHASIA2008
NEW HORIZONS

Next, we also wanted to mention a couple of production challenges that we've encountered while working on our algorithms. One particular aspect of rendering characters with displacement mapping is dealing with maps that contain texture *uv* borders as they frequently introduce texture uv seams.

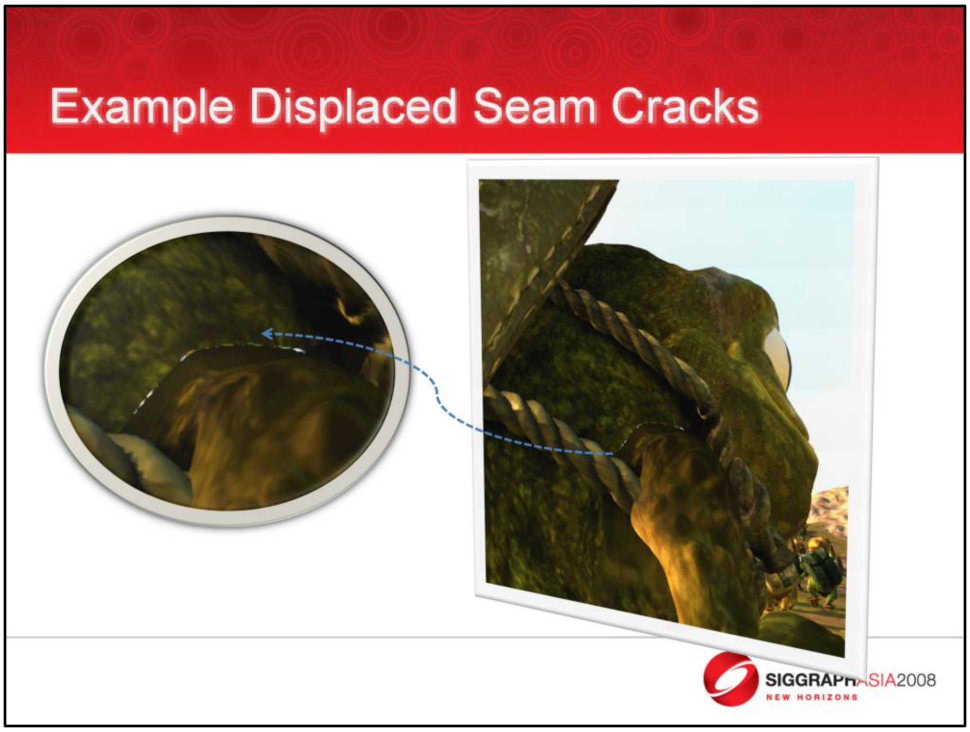


Character Modeling and Texture Seams

SIGGRAPHASIA2008
NEW HORIZONS

UV borders are rarely one-to-one in parameterization. Unless neighboring borders are laid out with the same orientations and lengths, displacing with these maps will introduce geometry cracks along the seams.

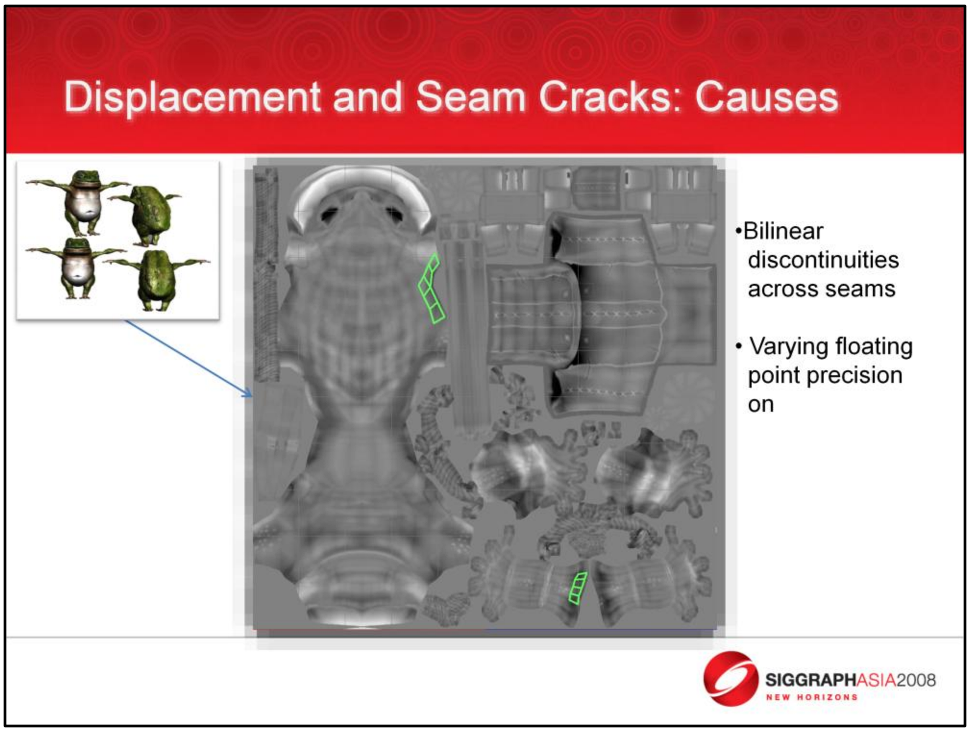
Here we highlighted the specific edges along the texture seam (in green). Note that the adjacent edges for this seam do not have uniform parameterization.



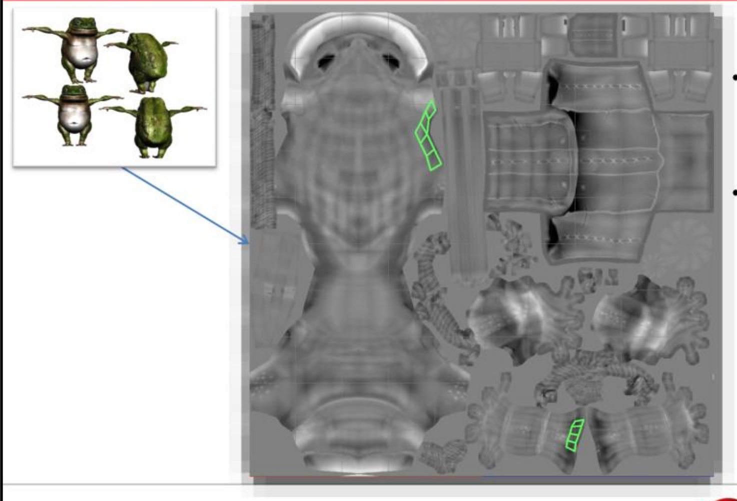
Example Displaced Seam Cracks

Example of a visible crack generated due to inconsistent values across the edges of displacement map for this character. This crack is generated along the seam that we highlighted in green on the previous slide.

Note that the images' contrast and brightness have been manipulated for higher contrast.



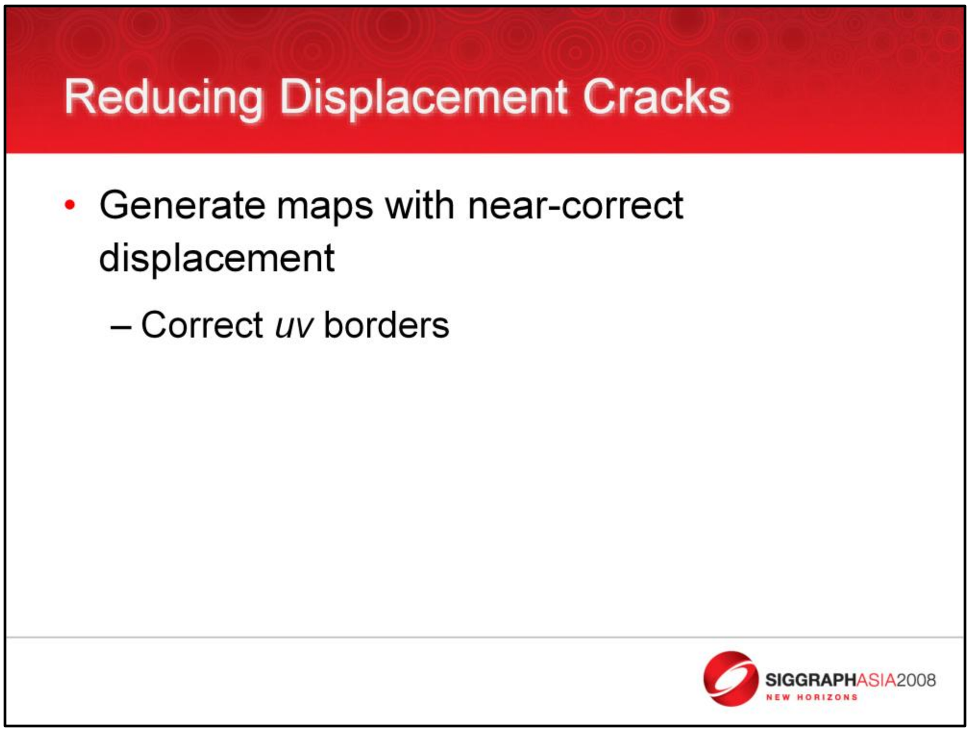
Displacement and Seam Cracks: Causes



- Bilinear discontinuities across seams
- Varying floating point precision on

SIGGRAPHASIA2008 NEW HORIZONS

Typically, with brute force displacement map generation, different floating point values are created across edges. Even if we had textured our character with tiled textures (however impractical of a concept), while this seamless parameterization alleviates bilinear artifacts, we still have to worry about floating point precision mismatch along the seams.

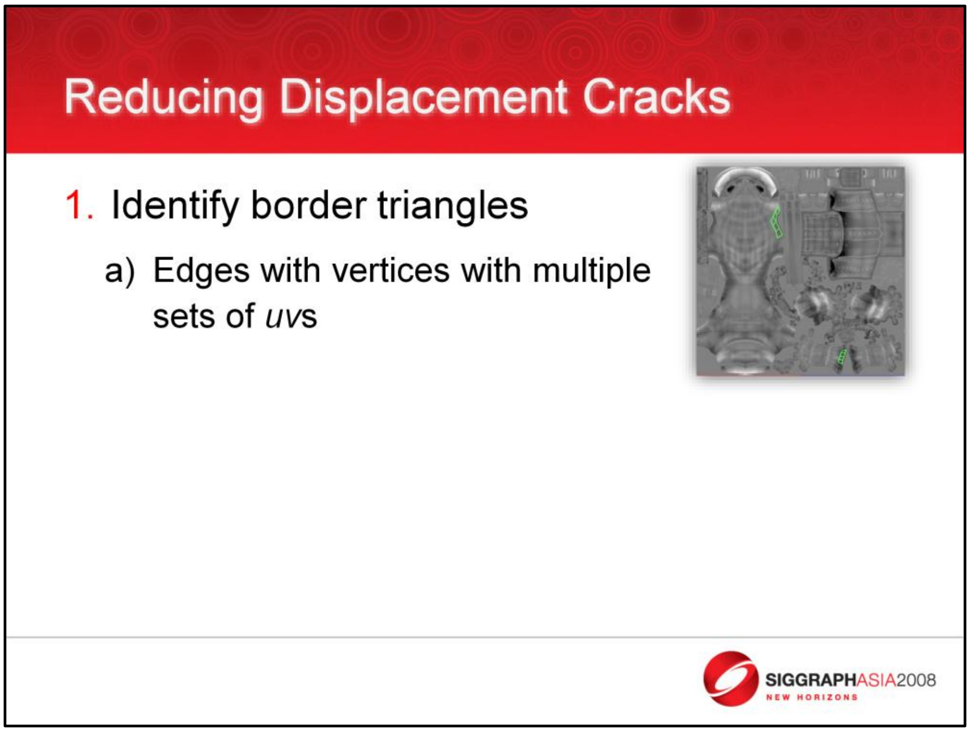


Reducing Displacement Cracks

- Generate maps with near-correct displacement
  - Correct *uv* borders

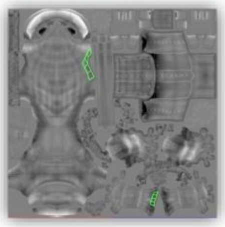
SIGGRAPHASIA2008 NEW HORIZONS

To solve this problem, we post-process our displacement maps by correcting all the texture uv borders as follows.



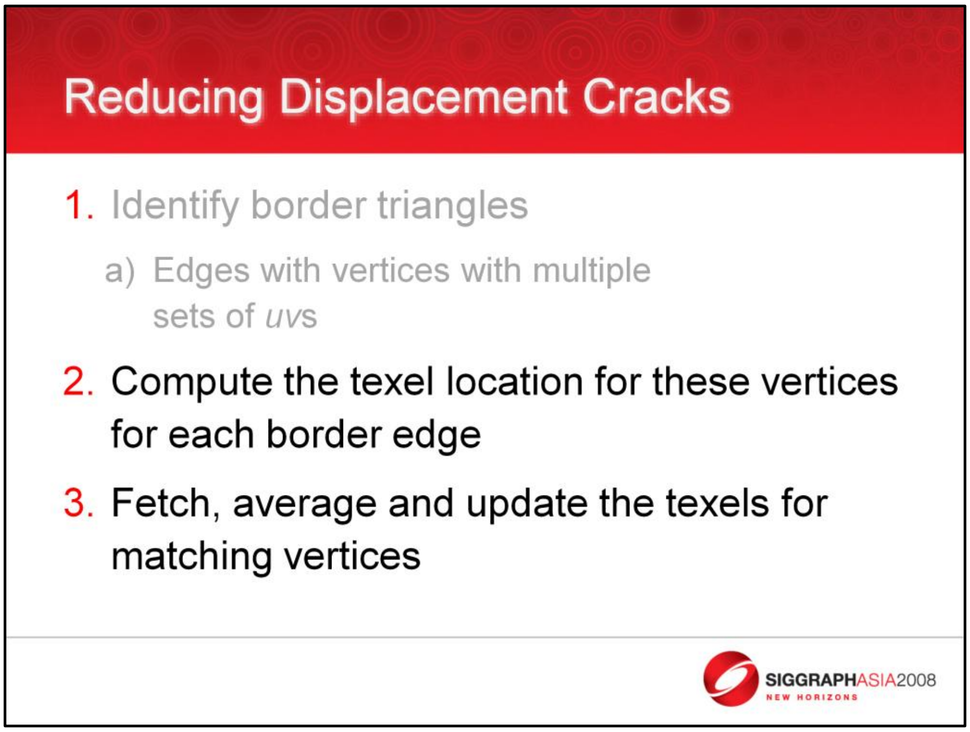
Reducing Displacement Cracks

1. Identify border triangles
   a) Edges with vertices with multiple sets of *uv*s



SIGGRAPHASIA2008 NEW HORIZONS

First, we identify the border triangle edges (i.e. edges that contain vertices with more than one set of texture coordinates).

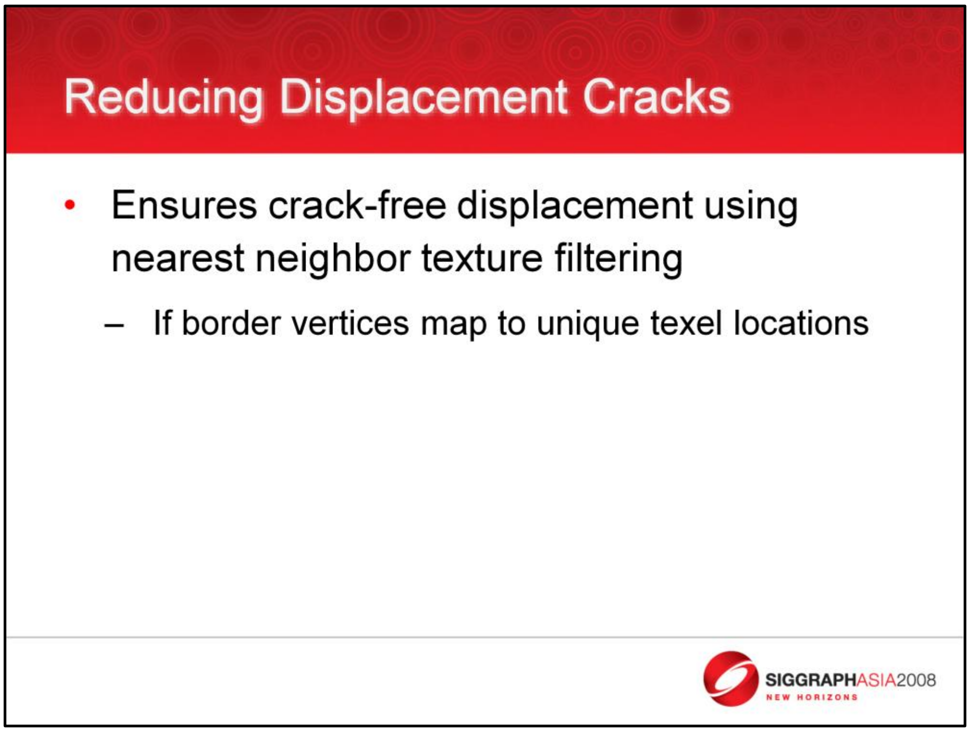


Reducing Displacement Cracks

1. Identify border triangles
   a) Edges with vertices with multiple sets of *uv*s
2. Compute the texel location for these vertices for each border edge
3. Fetch, average and update the texels for matching vertices

SIGGRAPHASIA2008
NEW HORIZONS

Then, for each border edge, we compute the texel locations for the vertices; fetch, average, and update the texels for matching vertices.

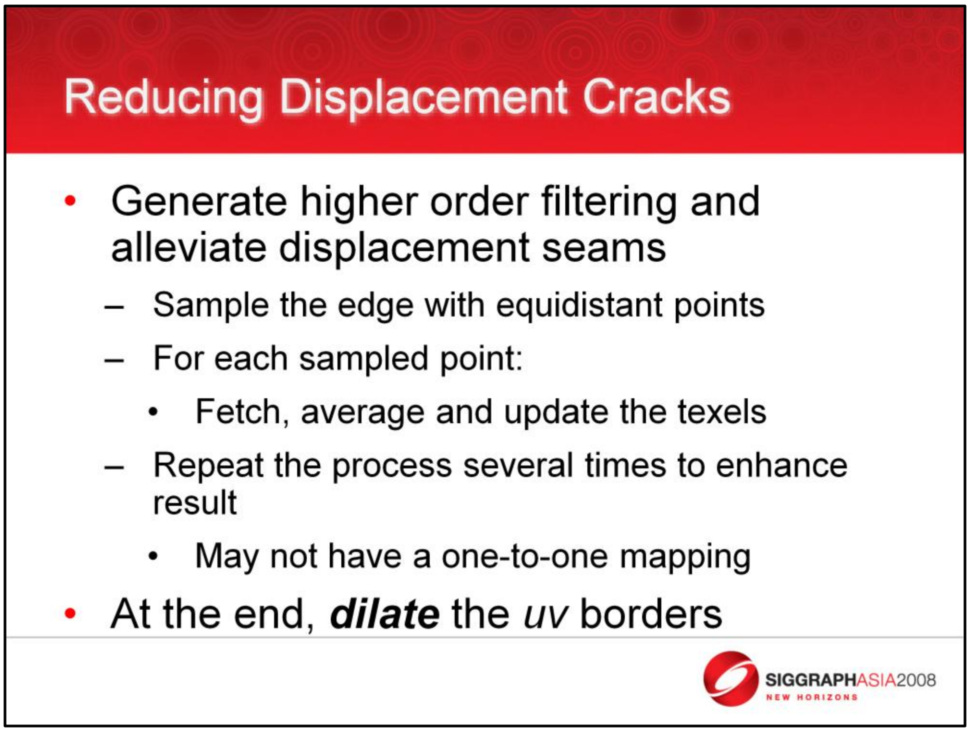


Reducing Displacement Cracks

- Ensures crack-free displacement using nearest neighbor texture filtering
  - If border vertices map to unique texel locations

SIGGRAPHASIA2008
NEW HORIZONS

As long as all these border vertices map to unique texel locations, we can ensure a crack-free displacement mapping using nearest neighbor texture filtering.

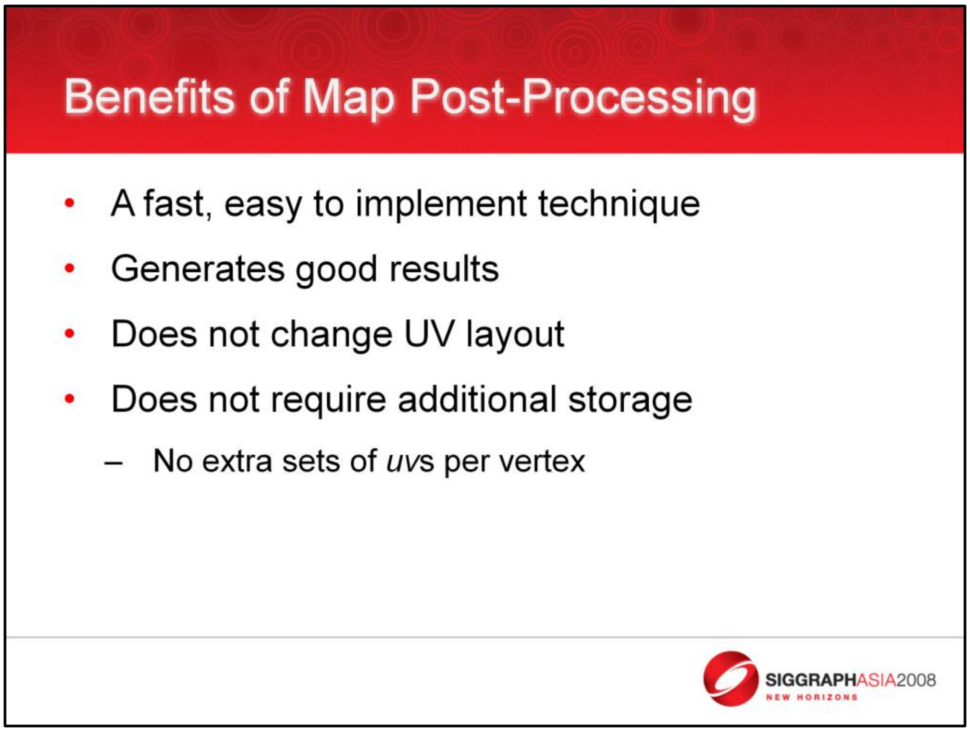


Reducing Displacement Cracks

- Generate higher order filtering and alleviate displacement seams
  - Sample the edge with equidistant points
  - For each sampled point:
    - Fetch, average and update the texels
  - Repeat the process several times to enhance result
    - May not have a one-to-one mapping
- At the end, ***dilate*** the *uv* borders

SIGGRAPHASIA2008
NEW HORIZONS

To improve *uv* seams for linear texture filtering and/or hardware tessellation, for each border edge, we sample the edge with equidistant points. Then, for each sampled point, we fetch, average and update the texels for matching points. Because the points might not map one-to-one, we repeat the above process several times to enhance the result. At the end, we dilate the uv borders.

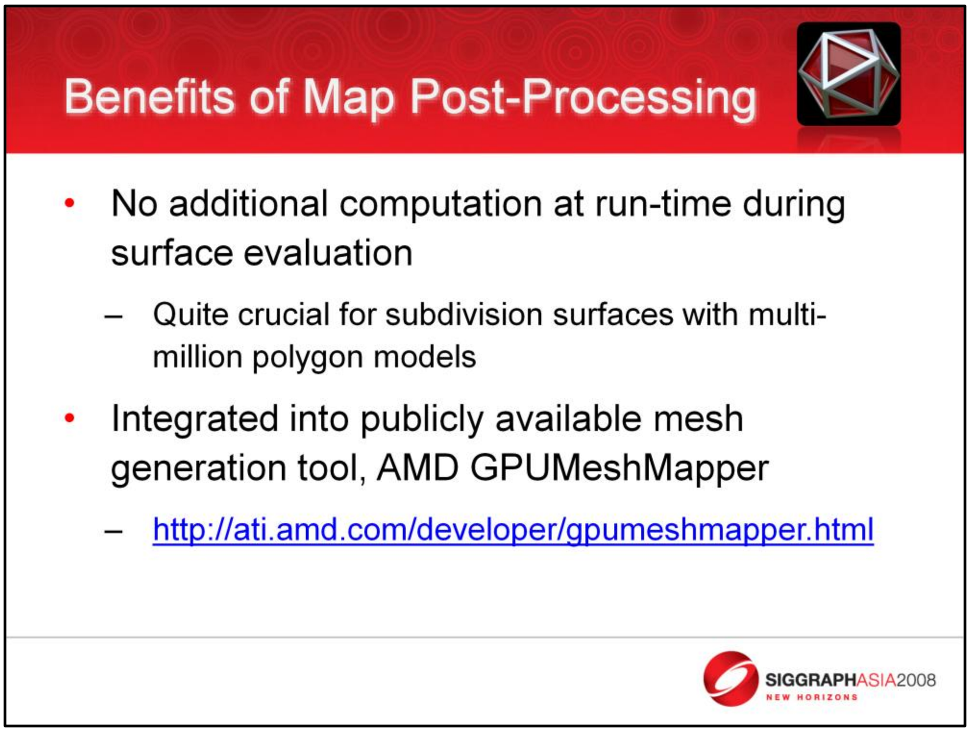


Benefits of Map Post-Processing

- A fast, easy to implement technique
- Generates good results
- Does not change UV layout
- Does not require additional storage
  - No extra sets of *uv*s per vertex

SIGGRAPHASIA2008
NEW HORIZONS

This technique is attractive because it is fast, simple to implement and it generates good results.

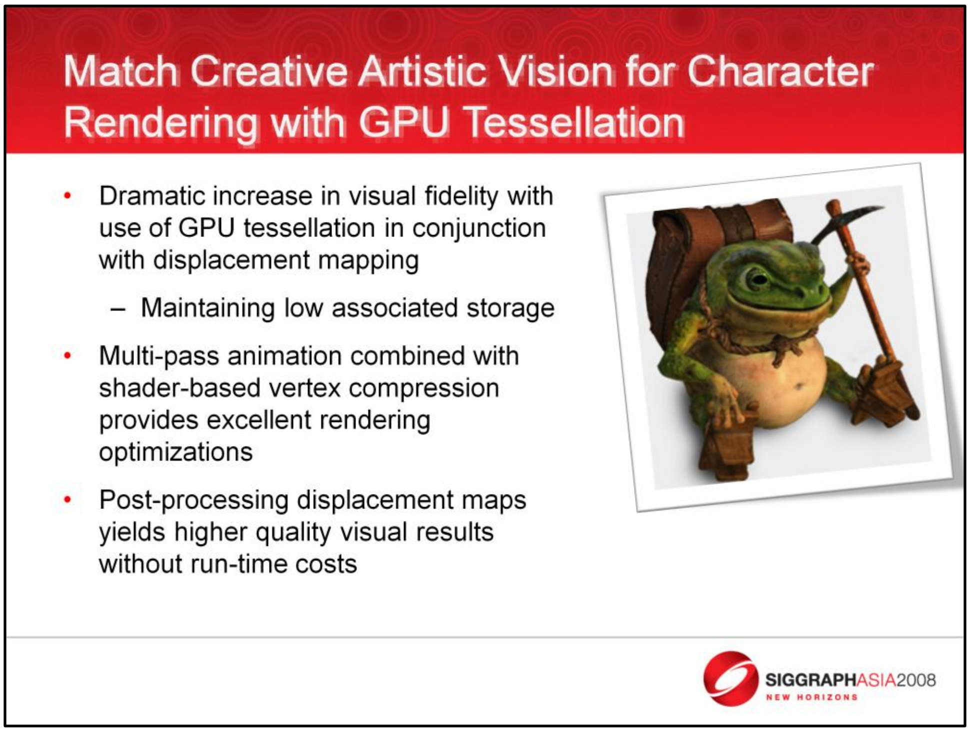


Benefits of Map Post-Processing

- No additional computation at run-time during surface evaluation
  - Quite crucial for subdivision surfaces with multi-million polygon models
- Integrated into publicly available mesh generation tool, AMD GPUMeshMapper
  - http://ati.amd.com/developer/gpumeshmapper.html

SIGGRAPHASIA2008 NEW HORIZONS

Furthermore, this method does not require additional computations at run-time, which is very important when evaluating surface positions for subdivision surface, which frequently may contain millions of triangles at render time!. This functionality is integrated into the freely available GPUMeshMapper tool.

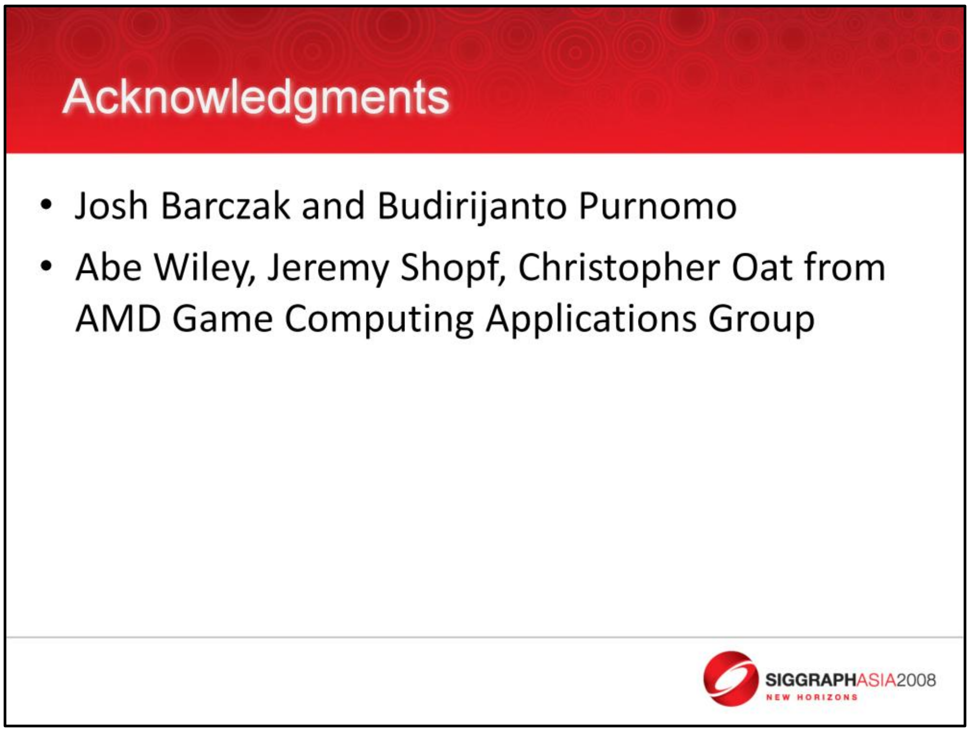


Match Creative Artistic Vision for Character Rendering with GPU Tessellation

- Dramatic increase in visual fidelity with use of GPU tessellation in conjunction with displacement mapping
  - Maintaining low associated storage
- Multi-pass animation combined with shader-based vertex compression provides excellent rendering optimizations
- Post-processing displacement maps yields higher quality visual results without run-time costs

SIGGRAPHASIA2008
NEW HORIZONS

To conclude, our methods provide a set of techniques allowing dramatic improvement in visual quality for our rendered characters, letting technology match the creative vision of the artists. Using GPU tessellation with displacement mapping and the multi-pass technique with vertex compression in the shaders allows excellent performance with tremendous jump in visual fidelity. Additionally, we also developed a method to increase the quality of generated displacement maps to use with our algorithms.

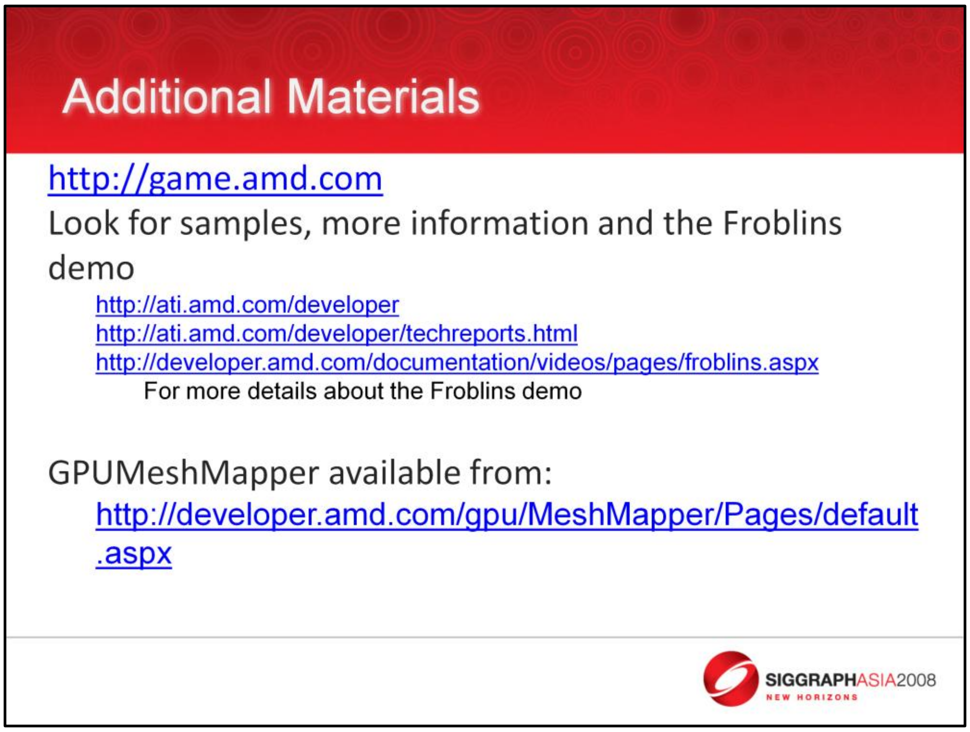


Acknowledgments

- Josh Barczak and Budirijanto Purnomo
- Abe Wiley, Jeremy Shopf, Christopher Oat from AMD Game Computing Applications Group

SIGGRAPHASIA2008 NEW HORIZONS

I'd like to thank some folks who contributed to the techniques we just described and who also worked on the Froblins demo.



Additional Materials

http://game.amd.com

Look for samples, more information and the Froblins demo

- http://ati.amd.com/developer
- http://ati.amd.com/developer/techreports.html
- http://developer.amd.com/documentation/videos/pages/froblins.aspx
  - For more details about the Froblins demo

GPUMeshMapper available from:

- http://developer.amd.com/gpu/MeshMapper/Pages/default.aspx

SIGGRAPHASIA2008 NEW HORIZONS

If you are interested in further information, here are some helpful links.

Questions?

SIGGRAPHASIA2008
NEW HORIZONS

Thank You!

SIGGRAPHASIA2008
NEW HORIZONS

SIGGRAPHASIA2008
NEW HORIZONS