

learn network inspire

Game Developers
Conference

08



February 18-22, 2008
San Francisco

www.gdconf.com



Future-Proof Games with Real-Time Tessellation

Natalya Tatarchuk
Game Computing Applications Group (O-CTO)
AMD Graphics Products Group



WWW.GDCONF.COM





Recent Evolution of Rendering

- ⌚ For the last decade, real-time shading has significantly increased in quality
- ⌚ Great strides in lighting and shadowing techniques
 - ⌚ Normal mapping is ubiquitous in games
 - ⌚ Inverse displacement mapping is present in the latest top graphics-rich games



WWW.GDCONF.COM



The Need for Geometric Detail Hasn't Gone Away

- ③ We can express subtle shading details in variety of ways
 - ③ Shadowing, self-occlusion and surface features
- ③ However, there are always the tell-tale signs giving it all away...

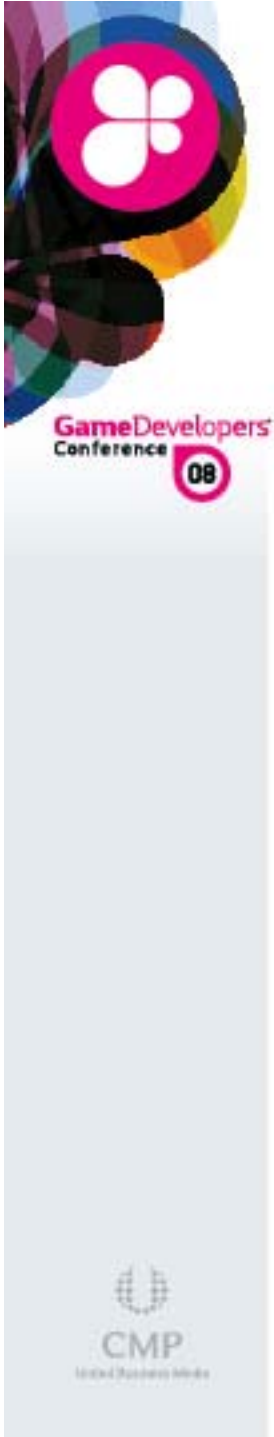


Bridging the Gap on Visual Quality for Games

- ③ Inflection points for games: **content generation and rendering**
 - ③ Character modeling with detail and deformation



Image courtesy of Valve



Bridging the Gap on Visual Quality for Games

- ⌚ Inflection points for games: **content generation and rendering**
 - ⌚ Character modeling with detail and deformation



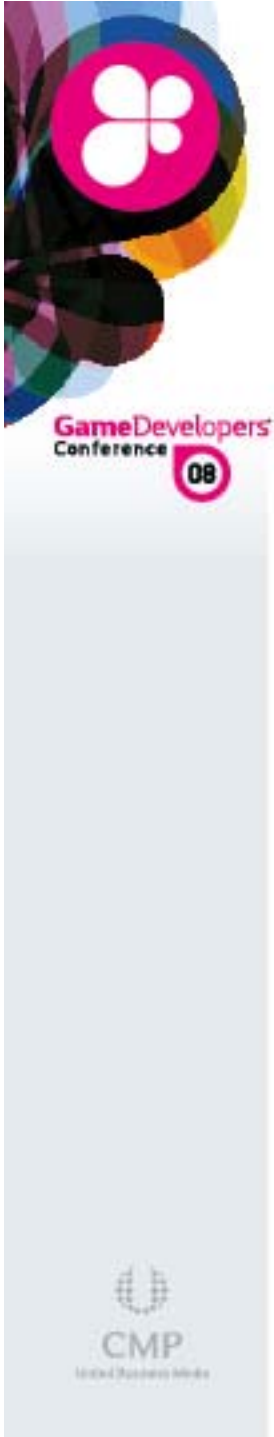
Game Developers
Conference

A Game Frame vs. a Movie Frame Comparison

Per-Frame	Shrek	Typical Game
Content size	100 M polygons	200-500K polygons
Animation quality	350 bones skinned on the CPU	32-40 bones skinning on GPU
Rendering time	8000 sec a frame on a Pentium IV	0.015 sec or less a frame



A frame from the Shrek [Yee04, PDI/Dreamworks]



Bridging the Gap on Visual Quality for Games

- ④ Transition to deformable subdivision surfaces with displacement
 - ④ Much higher quality

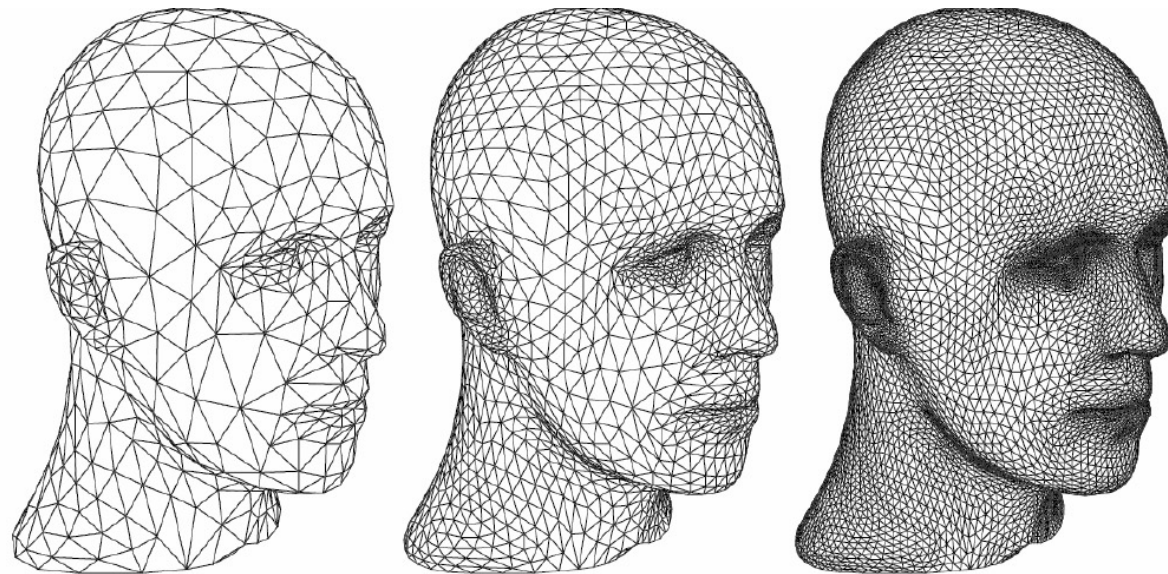


Those Costly Meshes...

- ④ Meshes are an inherently expensive representation
 - ④ Need to store a fair amount of data (positions, *uvs*, animation)
 - ④ Lots of research has gone into LOD schemes
- ④ Vertex transform cost is incurred regardless of object-viewer proximity
 - ④ Expensive for complex shaders or animated / detailed objects
- ④ Latest GPUs' unified shader architecture has much more efficient geometric processing
 - ④ But memory storage and fetch bandwidth is still a big concern
 - ④ Especially for animation
 - ④ Large meshes reduce vertex cache reuse

Tessellation Process

- Start with a polygon mesh
- Recursively apply subdivision rule



[Zorin and Schröder, 2000]



Game Developers
Conference

08

The Advantages of Using Tessellation

- ④ Compression of vertex data
- ④ Scalability with a knob



CMP
United Business Media

WWW.GDCONF.COM



The Advantages of Using Tessellation

- ④ Compression of vertex data
- ④ Scalability with a knob
- ④ Stable performance



GameDevelopers
Conference



The Advantages of Using Tessellation

- ④ Amplification of animation data
- ④ Allows providing data to GPU at coarser resolution while rendering at high resolution



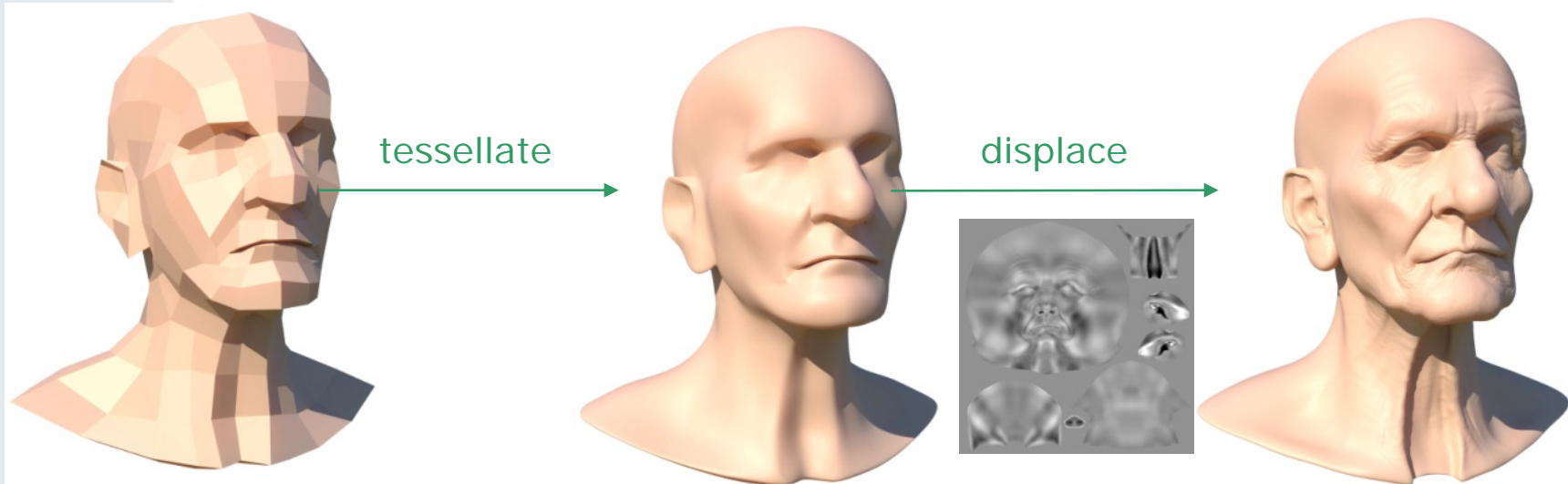
CMP
United Business Media

WWW.GDCONF.COM



The Advantages of Using Tessellation

③ Displacement mapped surfaces become first class citizens



Model concept courtesy of Valve

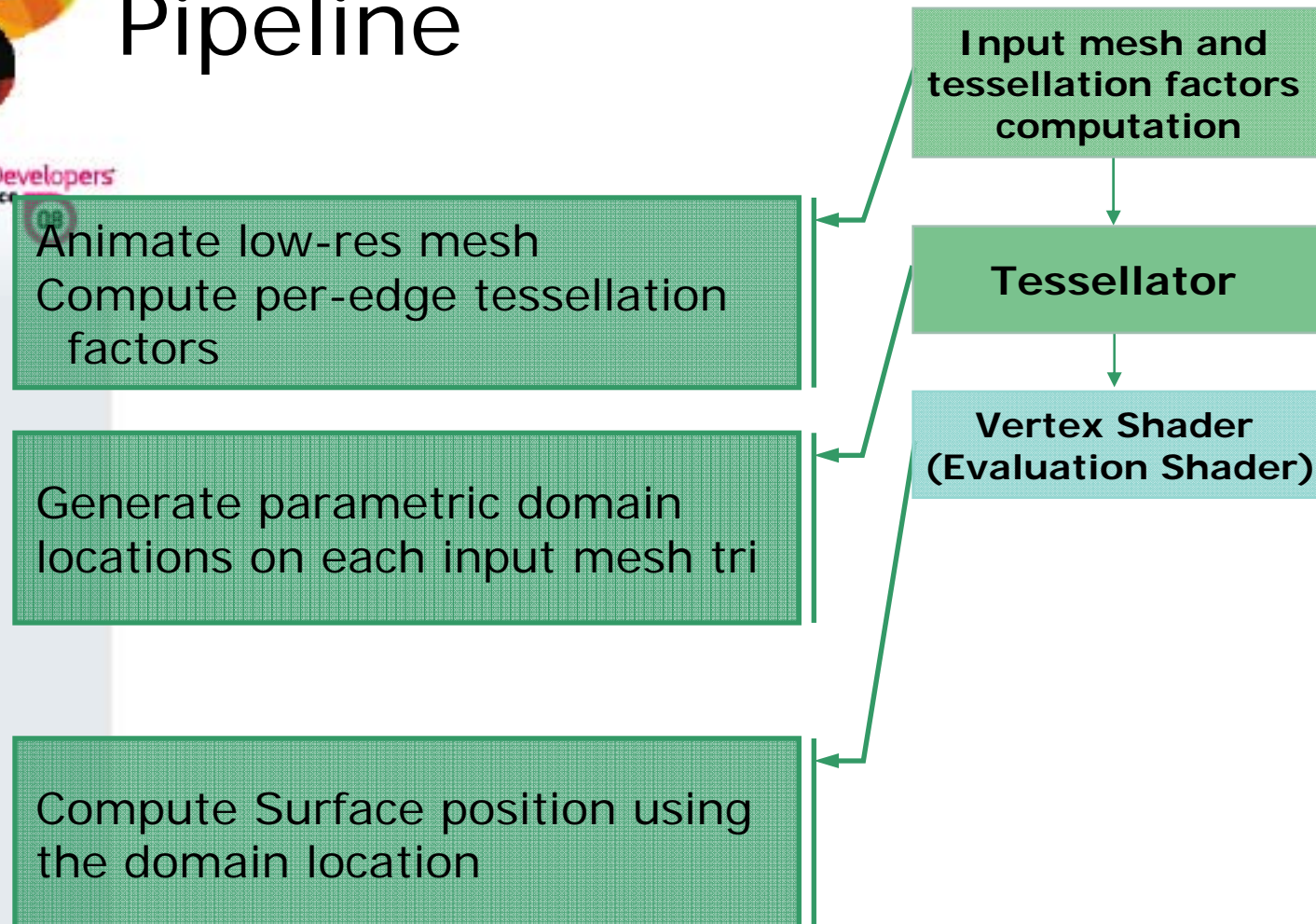


WWW.GDCONF.COM



Game Developers
Conference

Efficient GPU Tessellation Pipeline





Game Developers
Conference 08

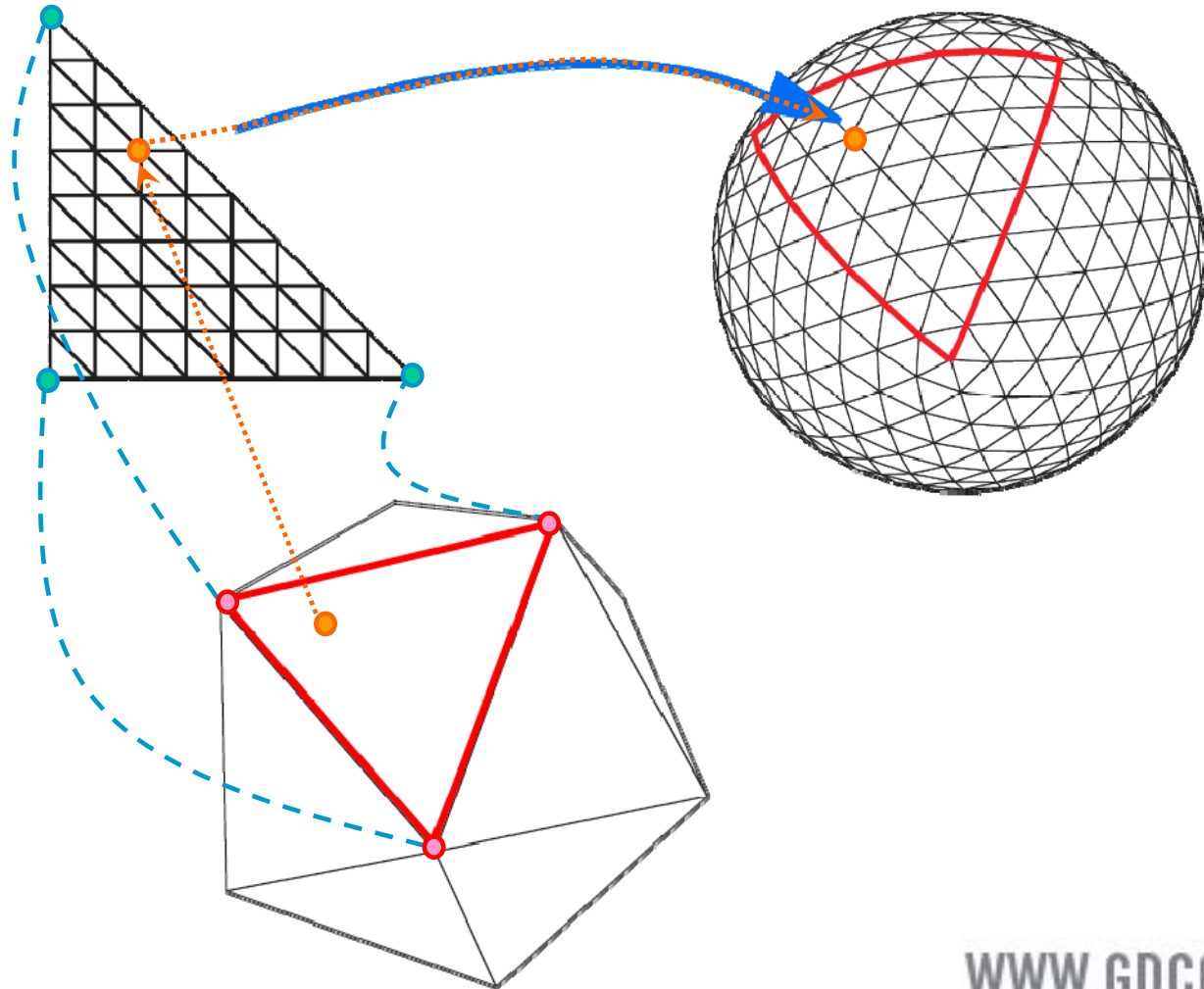
Tessellator Generates Domain Locations

- ④ (u,v) s on the domain surface
 - ④ Barycentric coordinates for a triangle
 - ④ Parametric coordinates for quads or lines
- ④ Efficiently passed to the vertex shader for evaluation
- ④ Resulting tessellation is watertight



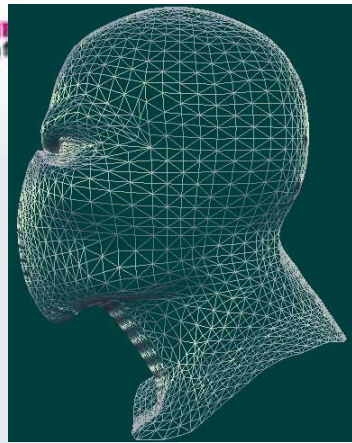
Game Developers
Conference 08

Domain Parametrization





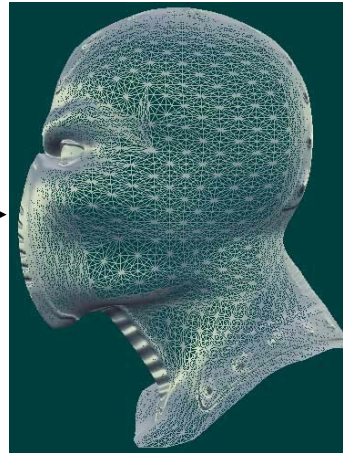
The Vertex Shader is Used as a Surface Evaluation Shader



Super-prim Mesh



Tessellator



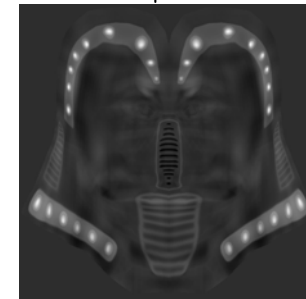
Tessellated Mesh



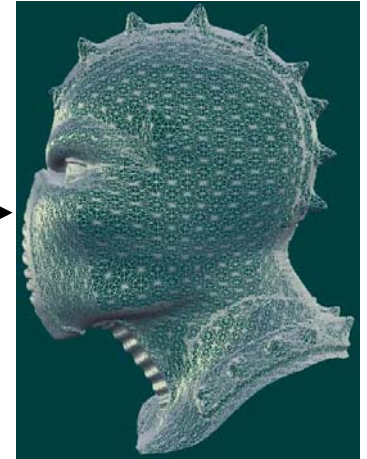
(Evaluation Shader)

Vertex
Shader

Sampler



Displacement
Map



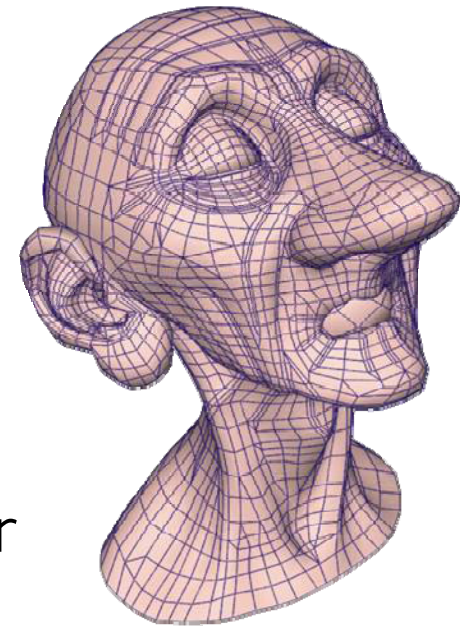
Tessellated and
Displaced Mesh

Evaluate Variety of Surfaces

Interpolative subdivision

Higher order surfaces

- ③ Bezier
- ③ N-Patches
- ③ B-Spline, NURBs, NUBs
- ③ Loop, Catmull-Clark and other subdivision surfaces



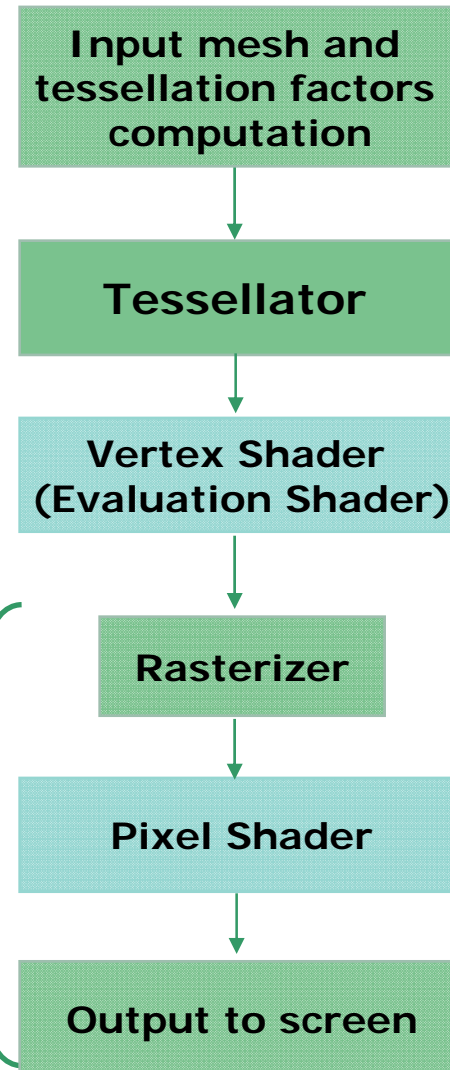
Catmull-Clark surfaces for a cinematic character [DeRose98]

Select the type that you need by providing your own evaluation shader



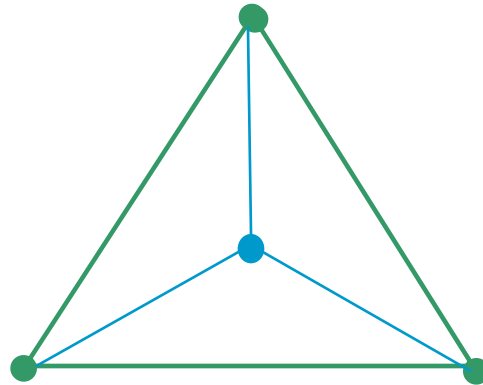
Efficient GPU Tessellation Pipeline

Process geometry, rasterize and render resulting high resolution mesh

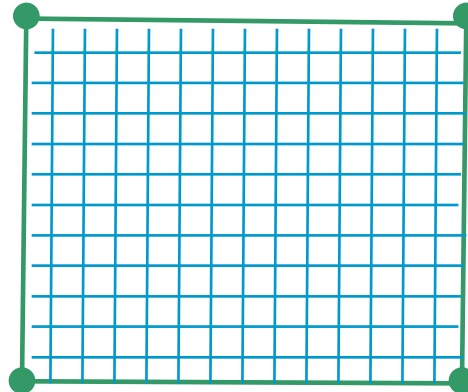


SuperPrimitives Types

⦿ Triangles



⦿ Quads



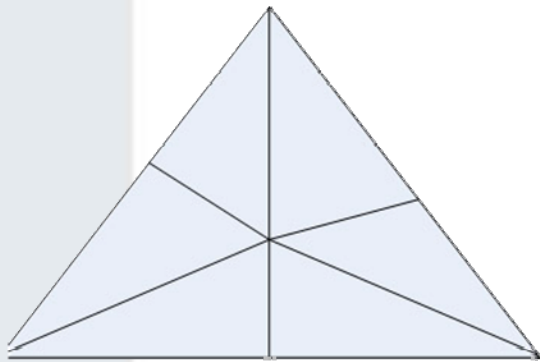
⦿ Lines



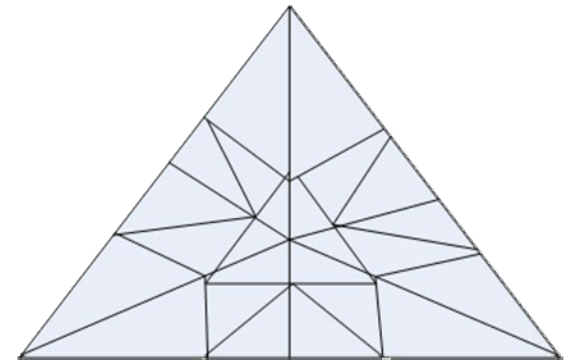


Use Continuous Tessellation

- ⌚ Specify floating point tessellation level per-draw call
- ⌚ Eliminates popping as vertices are added through tessellation



Level 1.0

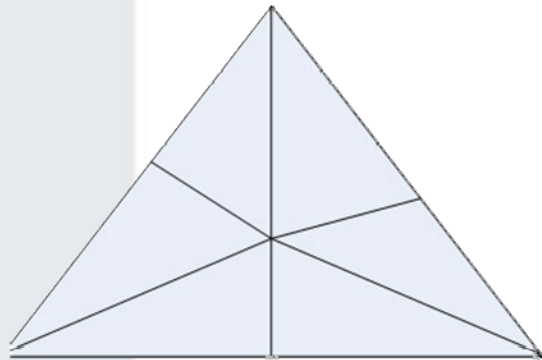


Level 2.0

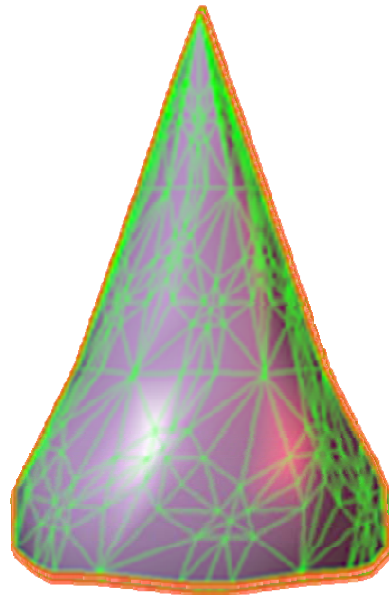


Continuous Tessellation

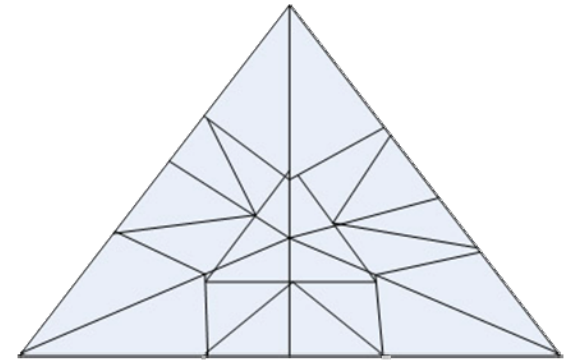
⦿ Watertight tessellation



Level 1.0



Level = 1.0



Level 2.0

[4:52:05] Enabling Manual Camera

Press H for Help

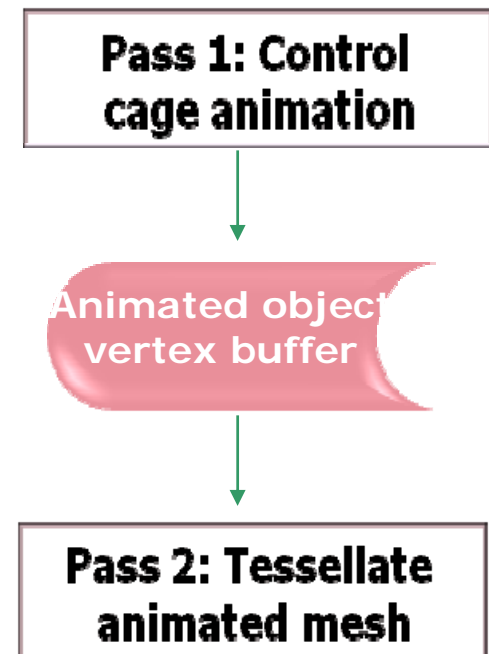
High Quality Characters Rendering

- ⊗ Straight-forward LOD scheme: Get up close to a character and see a lot of details
- ⊗ Start by rendering low-resolution character
 - ⊗ That's the control cage
- ⊗ Tessellate and displace for finer details
- ⊗ Uses the same art assets as when rendering without tessellation
 - ⊗ Displacement map
- ⊗ Generate displacement maps with a new AMD tool – GPUMeshMapper
 - ⊗ More on this tool in “Tessellation in the Low Poly World” session on Wednesday



Higher Quality Animation at Lower Cost with Tessellation

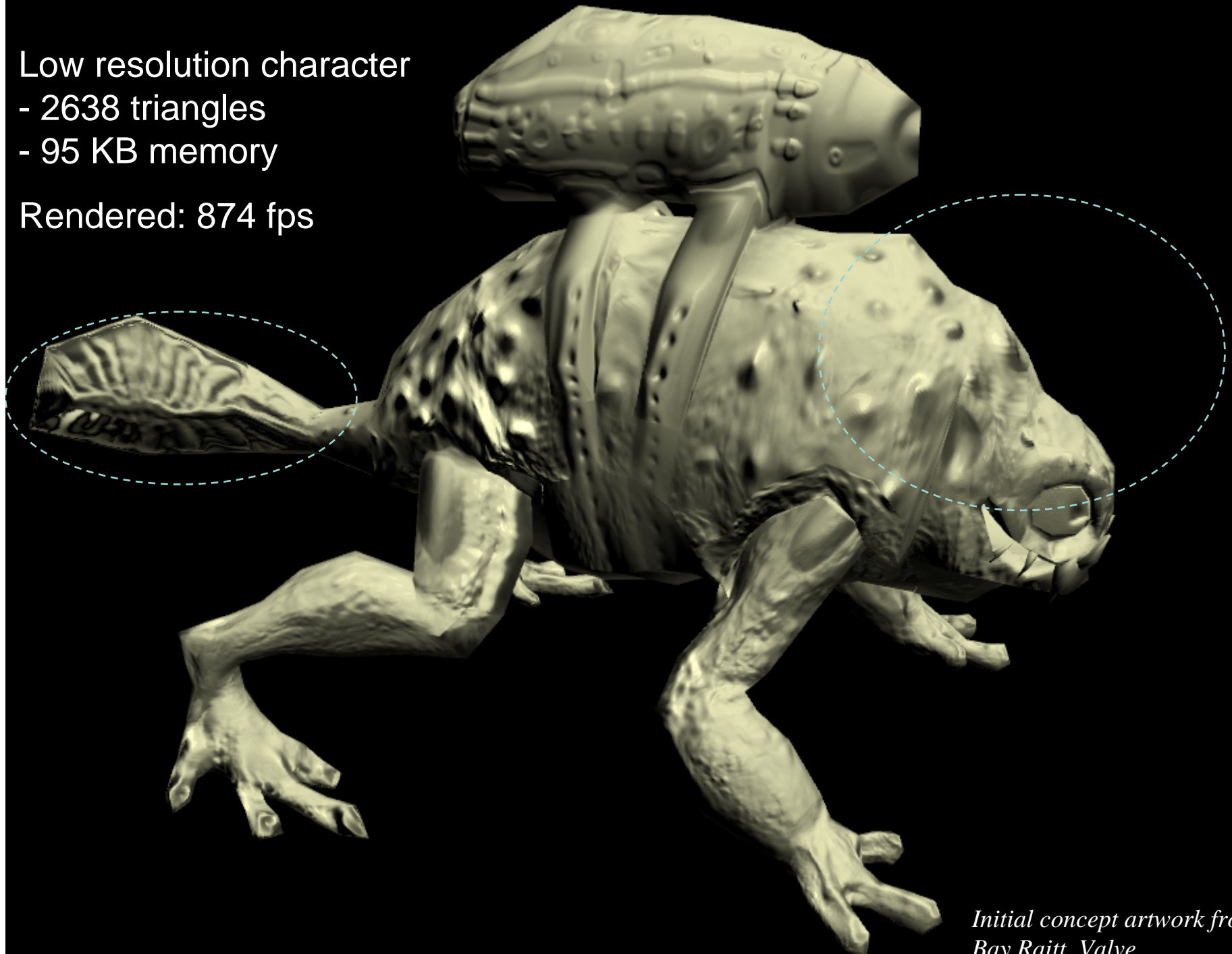
- **Animate on control cage: low polygon model**
 - Allows more complex animation computations
 - Higher quality animation with more animation data
 - Can store animated mesh per-frame for later operations
 - Can re-use animated objects from vertex buffer for shadows, reflections, etc.
- **Tessellate post animation**
 - Generates new vertices and transforms them into screen space
 - Allows more complex pixel shaders
 - Allow higher resolution textures



Low resolution character

- 2638 triangles
- 95 KB memory

Rendered: 874 fps

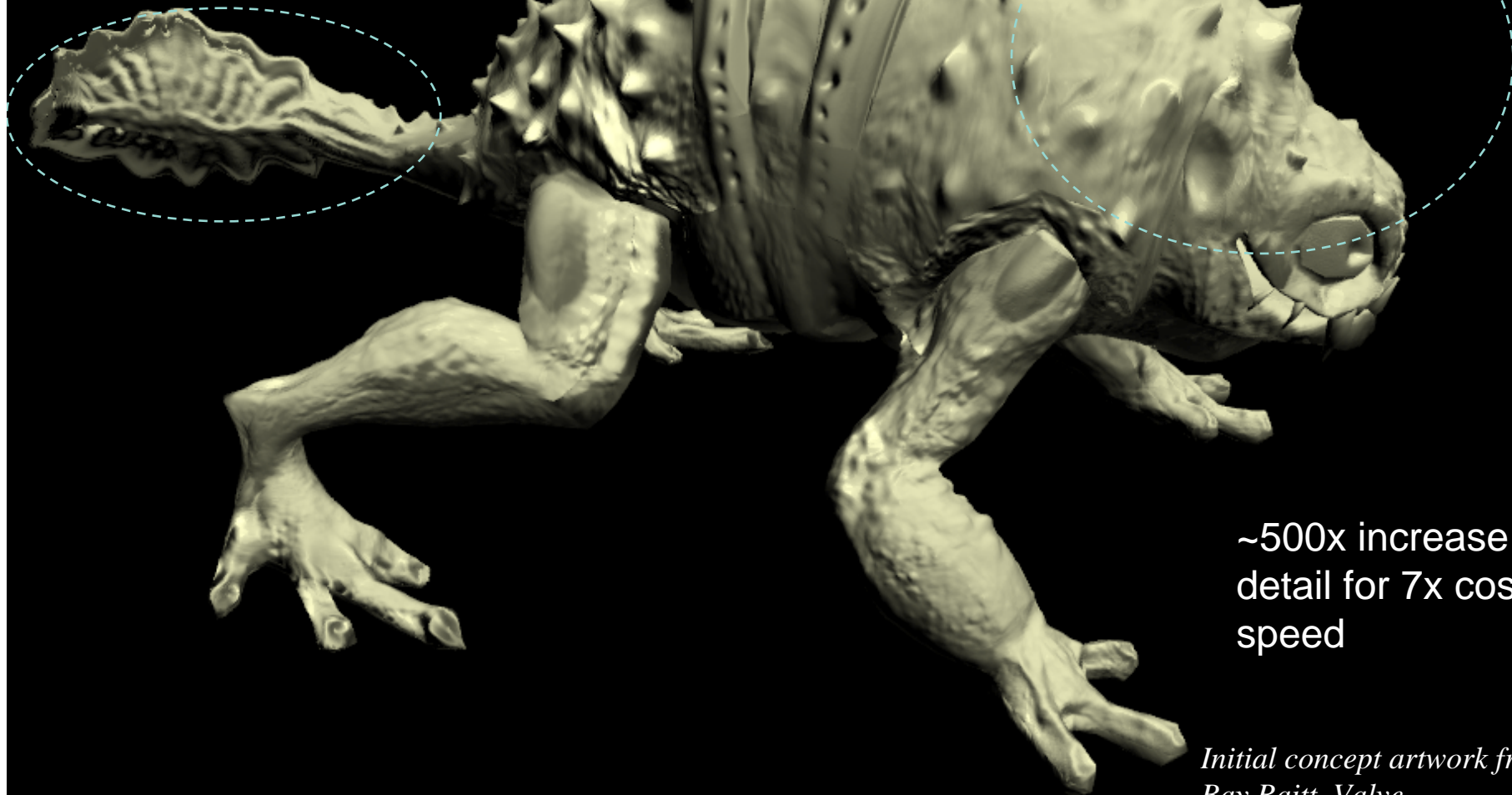


*Initial concept artwork from
Bay Raitt, Valve*

Tessellated character:

- 1, 084,218 triangles
- effective memory: 95KB
- equivalent model memory: 40MB

Rendered at 120 fps



~500x increase in
detail for 7x cost in
speed

*Initial concept artwork from
Bay Raitt, Valve*



Character Creation Pipeline in Film

Use subdivision surfaces

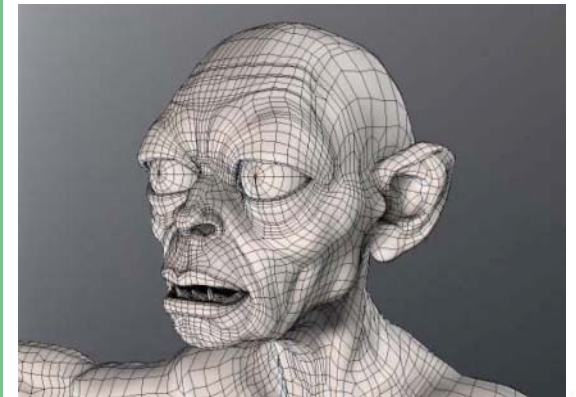
- Much more control over model building for animation and texturing

Rough pipeline:

- Create skeletal framework for creature rigging and animation
- Attach muscles
- Build up into a complete form
- Layer with skin
- Animate
- Shade



Gollum, "The Lord of the Rings",
courtesy of New Line



WWW.GDCONF.COM



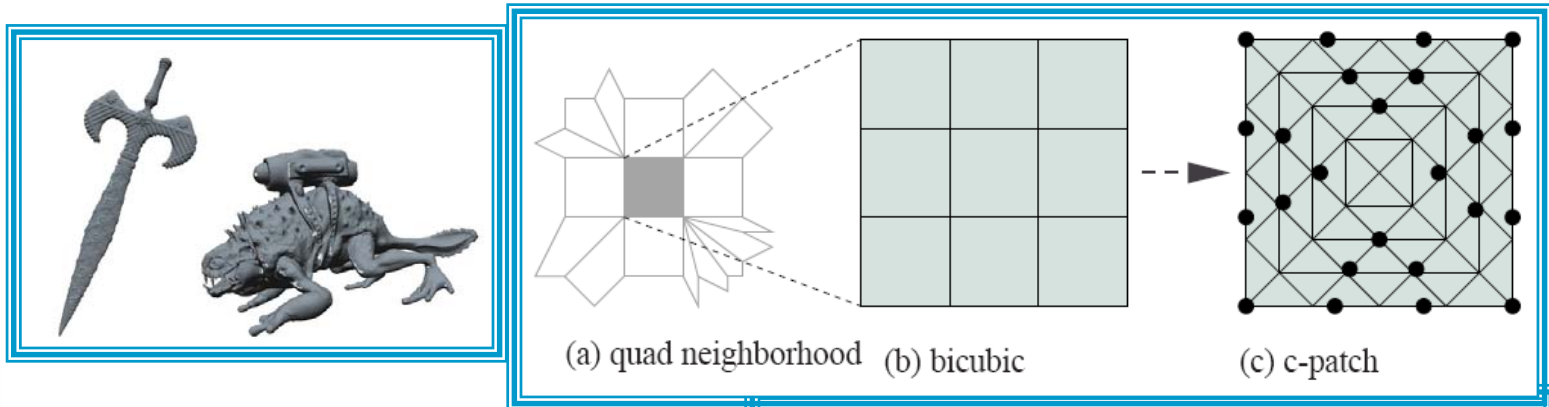
Game Developers
Conference

08

Approximating Catmull-Clark Subdivision Surfaces on GPU

- ④ Convert CC surface to lower order representation
- ④ Directly evaluate some number Bezier patches on GPU approximating the original surface
- ④ Support the same subdivision surfaces as in Maya
 - ④ Very important for art pipelines

Approximating Catmull-Clark Subd Surfaces on GPU



T. L. Ni et al, "**Smooth surfaces from 4-sided facets**", SMI 2008

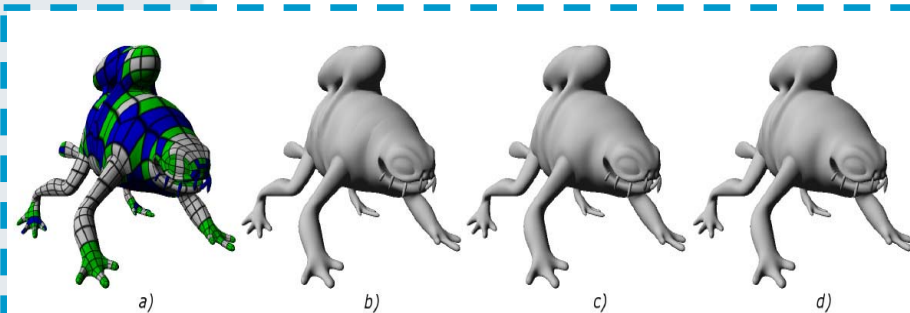


Figure 1: *a)* The patch structure of a Catmull-Clark subdivision surface. The grey patches only contain vertices of valence 4, green have one extraordinary vertex and blue have more than one extraordinary vertex. *b)* Our approximation to the Catmull-Clark subdivision surface using geometry patches and *c)* our final approximation using geometry and tangent patches compared with *d)* the actual Catmull-Clark limit surface.

C. Loop, S. Schaeffer
"**Approximating Catmull-Clark subdivision surfaces with bicubic patches**", MSR Tech Report 2007



Lighting Displaced Models

Tessellation by itself can support a variety of lighting methods

- ⊕ Vertex lighting, normal-mapping, etc

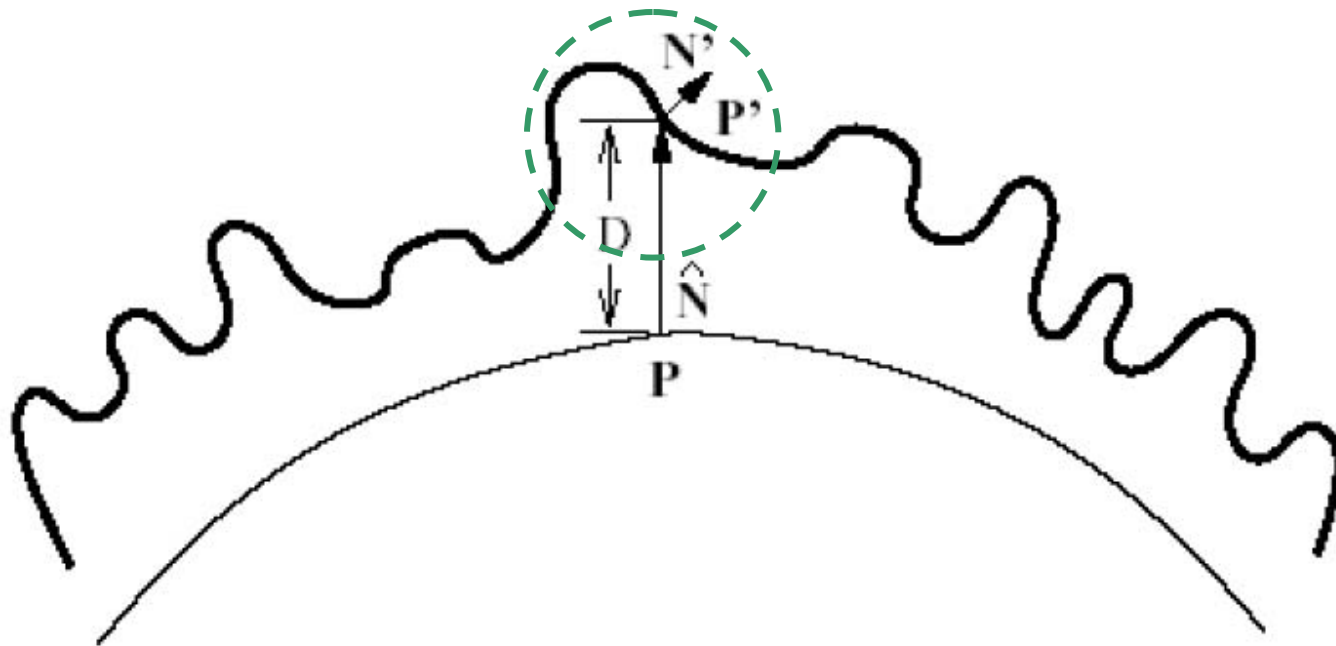
Displacement mapping, on the other hand, is less flexible



Game Developers
Conference 08

Combining Normal Maps and Displacement Maps

- As we displace, we change the actual displayed normal

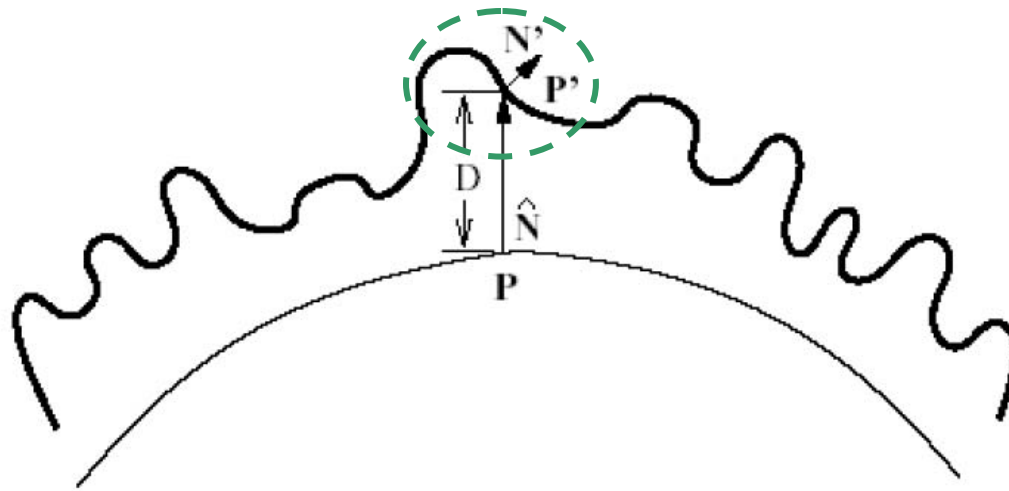


Combining Normal Maps and Displacement Maps

Need to rotate tangent frame

Then could use tangent-space normal maps

But is there a better way?





Lighting Directly from Displacement Map

- ④ Use central differences to approximate the derivative field
 - ④ Compute per-pixel normals based on the per-vertex tangent frames based on current displacement
- ④ Memory savings at the cost of additional ALU ops
 - ④ Instead of storing an additional normal map
- ④ Higher quality resulting lighting
 - ④ Shows off the details that are only encoded in the displacement map
- ④ Support dynamic height fields
 - ④ Great for terrains / destruction
 - ④ Also see Johan Andersson SIGGRAPH 07 talk on terrain rendering for other ideas



GameDeve
Conference

Computing Normal from Displacement Map: Example

```
// calculate new normal based on gradients of height map
float fDeltaU    = 1.0f / vMapSize.x;
float fDeltaV    = 1.0f / vMapSize.y;
float fNeighborU  = DisplacementTapMipped( vUV + float2( fDeltaU, 0 ), sMap, fScale, fBias );
float fNeighborV  = DisplacementTapMipped( vUV + float2( 0, fDeltaV ), sMap, fScale, fBias );
float fNeighborU2 = DisplacementTapMipped( vUV - float2( fDeltaU, 0 ), sMap, fScale, fBias );
float fNeighborV2 = DisplacementTapMipped( vUV - float2( 0, fDeltaV ), sMap, fScale, fBias );

// Change in world-space per texel step. This is object-dependent
float fDistanceU = ( fTangentLength ) * fDeltaU ;
float fDistanceV = ( fBinormLength ) * fDeltaV ;

// average normals four neighboring quads, like so:
//
//      V2
//   U2 X  U
//      V

float3 vUX  = float3( fDistanceU, fNeighborU  - fDisplacement, 0 ) ;
float3 vVX  = float3( 0, fNeighborV  - fDisplacement, -fDistanceV ) ;
float3 vU2X = float3( -fDistanceU, fNeighborU2 - fDisplacement, 0 ) ;
float3 vV2X = float3( 0, fNeighborV2 - fDisplacement, fDistanceV ) ;

return normalize( normalize( cross( vUX, vVX ) ) +
                  normalize( cross( vVX, vU2X ) ) +
                  normalize( cross( vU2X, vV2X ) ) +
                  normalize( cross( vV2X, vUX ) ) );
```



Rendering Terrain: Challenges

- ③ Very large models, continuous spans across space
- ③ Rendered simultaneously very close and far away
 - ③ Necessitates good LOD handling
- ③ Require a lot of data / memory for polygonal representation





Rendering Tessellated Terrain

- ④ Tessellation pipeline handles this well
 - ④ Tessellate flat mesh and displace on the fly
 - ④ Use GPU-based noise or precomputed height map for displacement and shading
- ④ Adaptively subdivide to get detail where needed
- ④ Can perform collision detection directly on the height map

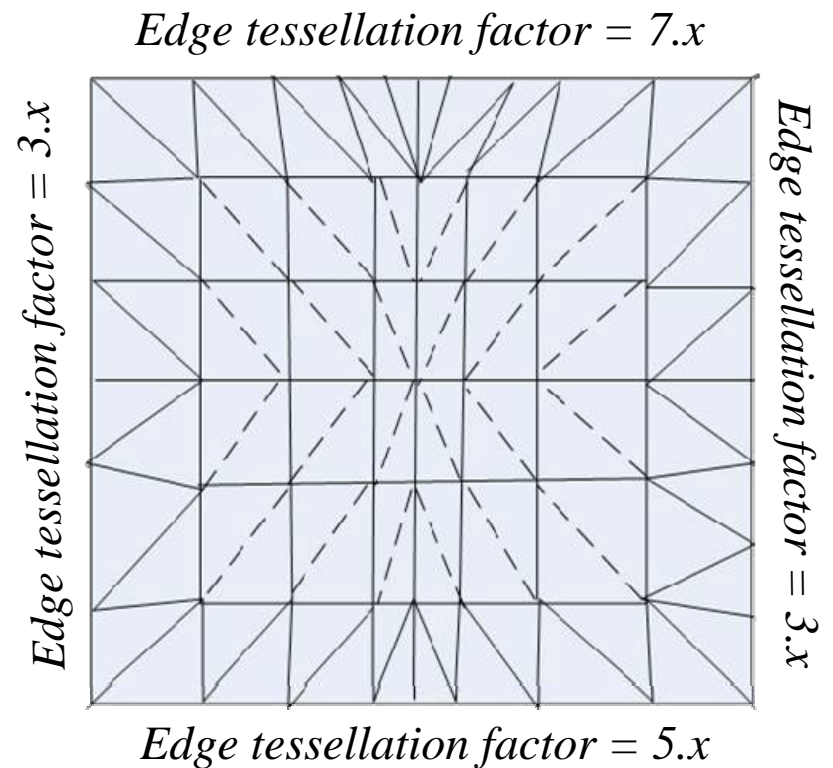
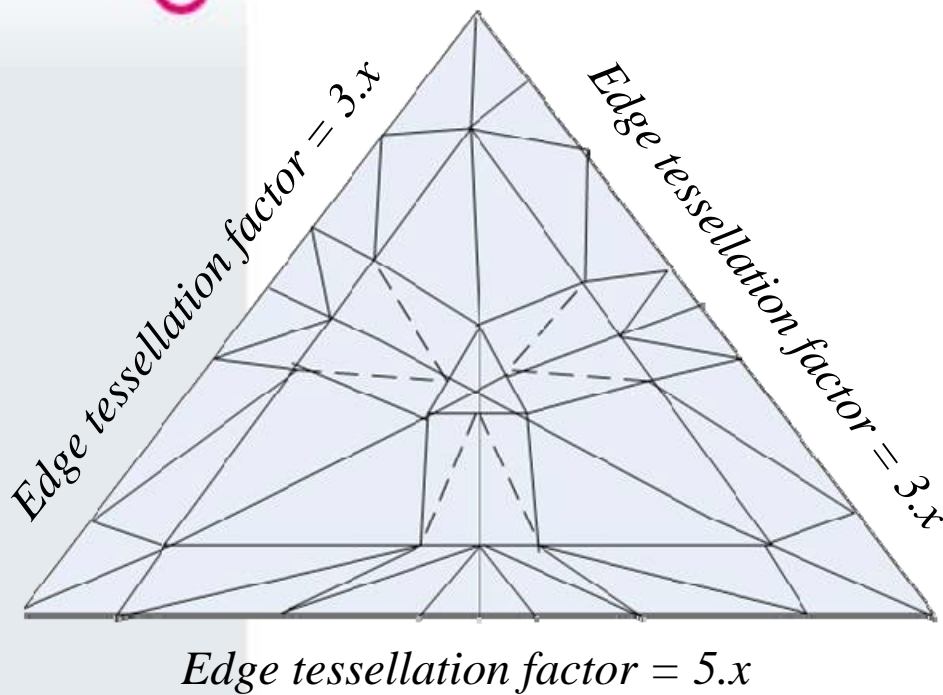


Create Dynamic Terrain

- ⌚ Dynamic terrain becomes very straight-forward
- ⌚ Blend height maps for modification
 - ⌚ Damage, destruction, earthquakes, bullet holes – limitless possibilities!

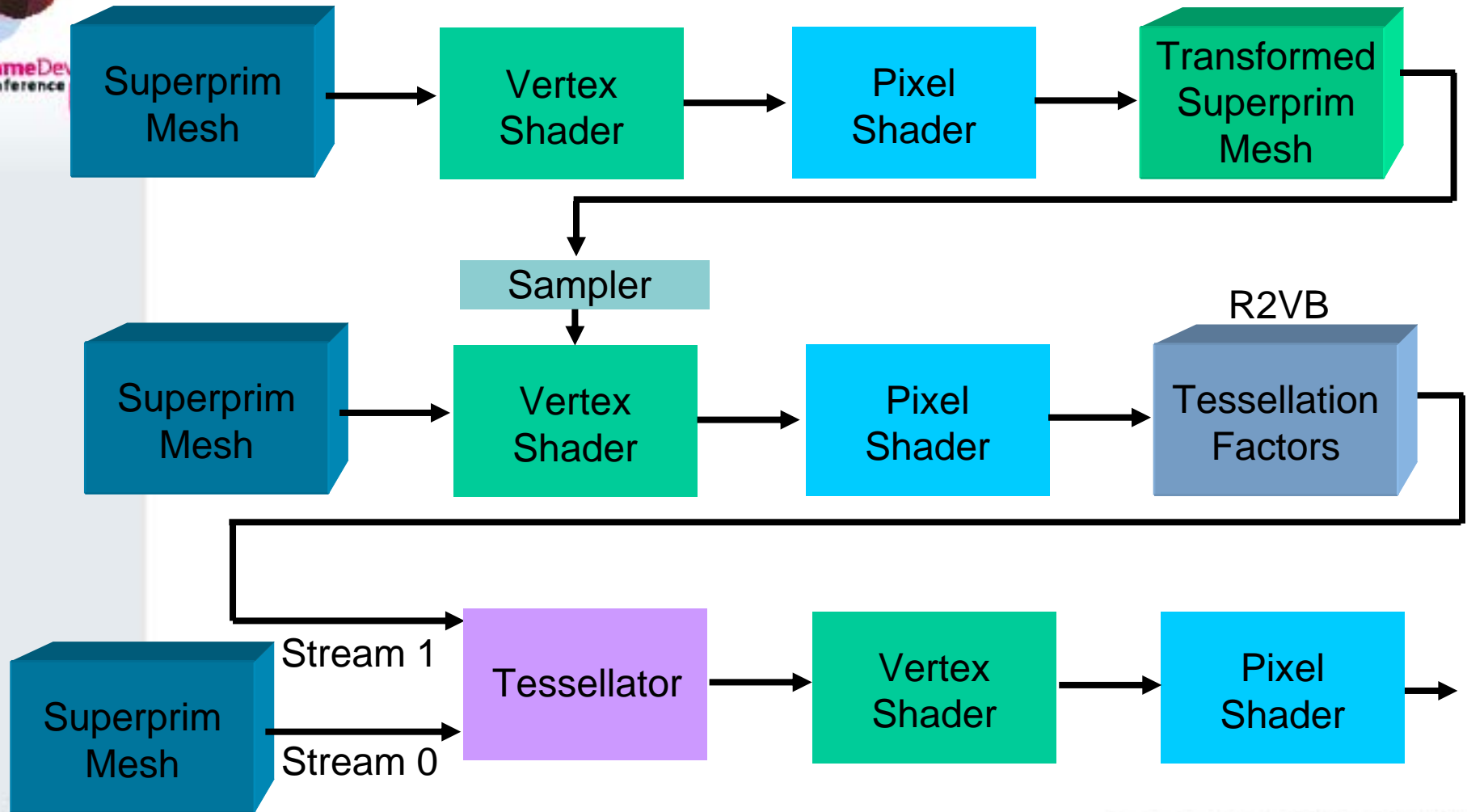
Dynamic Adaptive Tessellation

- ④ Tessellation level is specified per edge





Adaptive Tessellation Flow





Vertex Prepass Code Sample

```
// Compute displaced position
float3 vPositionOS = DisplaceVertex( i.vPositionOS.xyz, i.vTexCoord, vNormalWS );

// Transform displaced vertex position to camera space:
float4 vPositionSS = mul( mV, float4( vPositionOS.xyz, 1 ) );
vPositionSS /= vPositionSS.w;

o.vPositionSS = vPositionSS.xyz;

// Compute output position for rendering into a texture
float2 vTformVertsMapSize = float2( 1024, 16 );

o.fVertexID      = i.vPositionOS.w;
int nVertexID    = floor( i.vPositionOS.w );
int nTextureWidth = vTformVertsMapSize.x;

float2 vPos      = float2( nVertexID % nTextureWidth, nVertexID / nTextureWidth ); // Compute row and column of the position in 2D texture
vPos /= vTformVertsMapSize.xy;
vPos.y = 1.0 - vPos.y;
vPos   = vPos * 2 - 1.0; // Move to [-1; 1] range

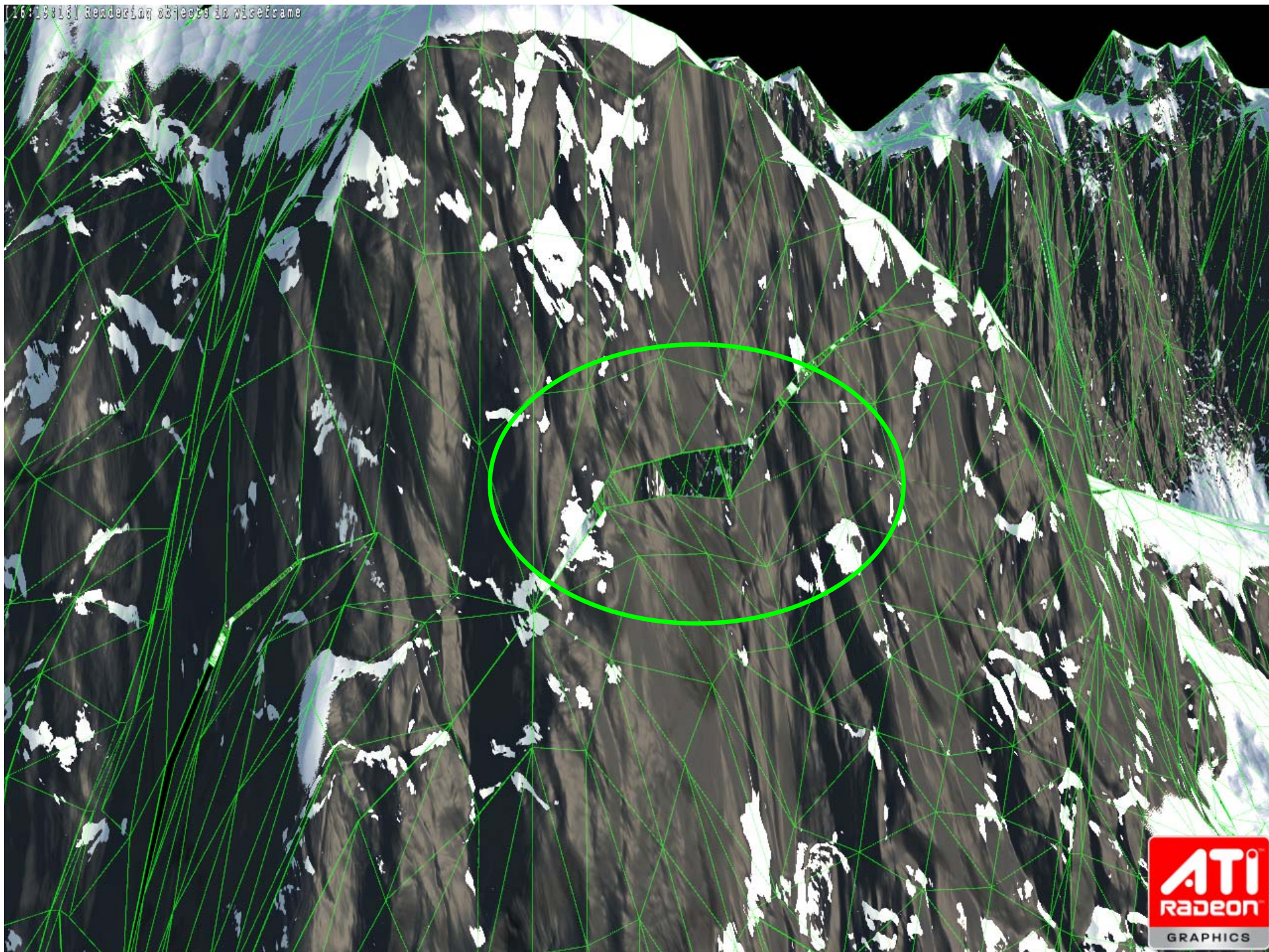
o.vPositionCS = float4( vPos.xy, 0, 1 );
```



Avoiding Cracks with Per-Edge Tessellation Factors

- ⌚ Adjacent edges must have precision-identical tessellation levels
 - ⌚ Otherwise cracks will appear when displacing
- ⌚ Need extra care during tessellation factors computation
 - ⌚ Ensure identical edge direction (using vertex ID)

[16:13:13] Rendering objects in wseframe





Computing Tessellation Factors

```
// Current vertex ID:
int nCurrentVertID = (int)( i.vPositionOS.w );

// Determine the ID of the edge neighbor's vertex (remember that this is done with non-indexed primitives, so
// basically if the current vertex is v0 ,then we have edges: v0->v1, v1->v2, v2->v0

// Manual implementation of MOD works for integer values
int nCurrentVertEdgeID = nCurrentVertID - 3 * floor( (float)nCurrentVertID / (float) 3);

int nEdgeVert0ID = nCurrentVertID;    // this works if current vertex is v0 or v1
int nEdgeVert1ID = nCurrentVertID + 1;

if ( nCurrentVertEdgeID == 0 )
{
    nEdgeVert0ID = nCurrentVertID + 1;
}
else if ( nCurrentVertEdgeID == 1 )
{
    nEdgeVert0ID = nCurrentVertID + 1;
}
else if ( nCurrentVertEdgeID == 2 )    // In case of v2 we need to wrap around to v0
{
    nEdgeVert0ID = nCurrentVertID - 2;
}
```

Compute current edge's end-points' indices



Computing Tessellation Factors

```
// Compute the fetch coordinates to fetch transformed positions for these two vertices to compute their edge statistics:
//

float2 vTformVertsMapSize = float2( nTformedVertsWidth, nTformedVertsHeight );
int    nTextureWidth      = vTformVertsMapSize.x;

// Vertex0: nCurrentVertID, compute row and column of the position in 2D texture:
float2 vVert0Coords      = float2( nCurrentVertID % nTextureWidth, nCurrentVertID / nTextureWidth );
vVert0Coords /= vTformVertsMapSize.xy;

// Vertex1: nEdgeVert0ID, compute row and column of the position in 2D texture:
float2 vVert1Coords      = float2( nEdgeVert0ID % nTextureWidth, nEdgeVert0ID / nTextureWidth );
vVert1Coords /= vTformVertsMapSize.xy;

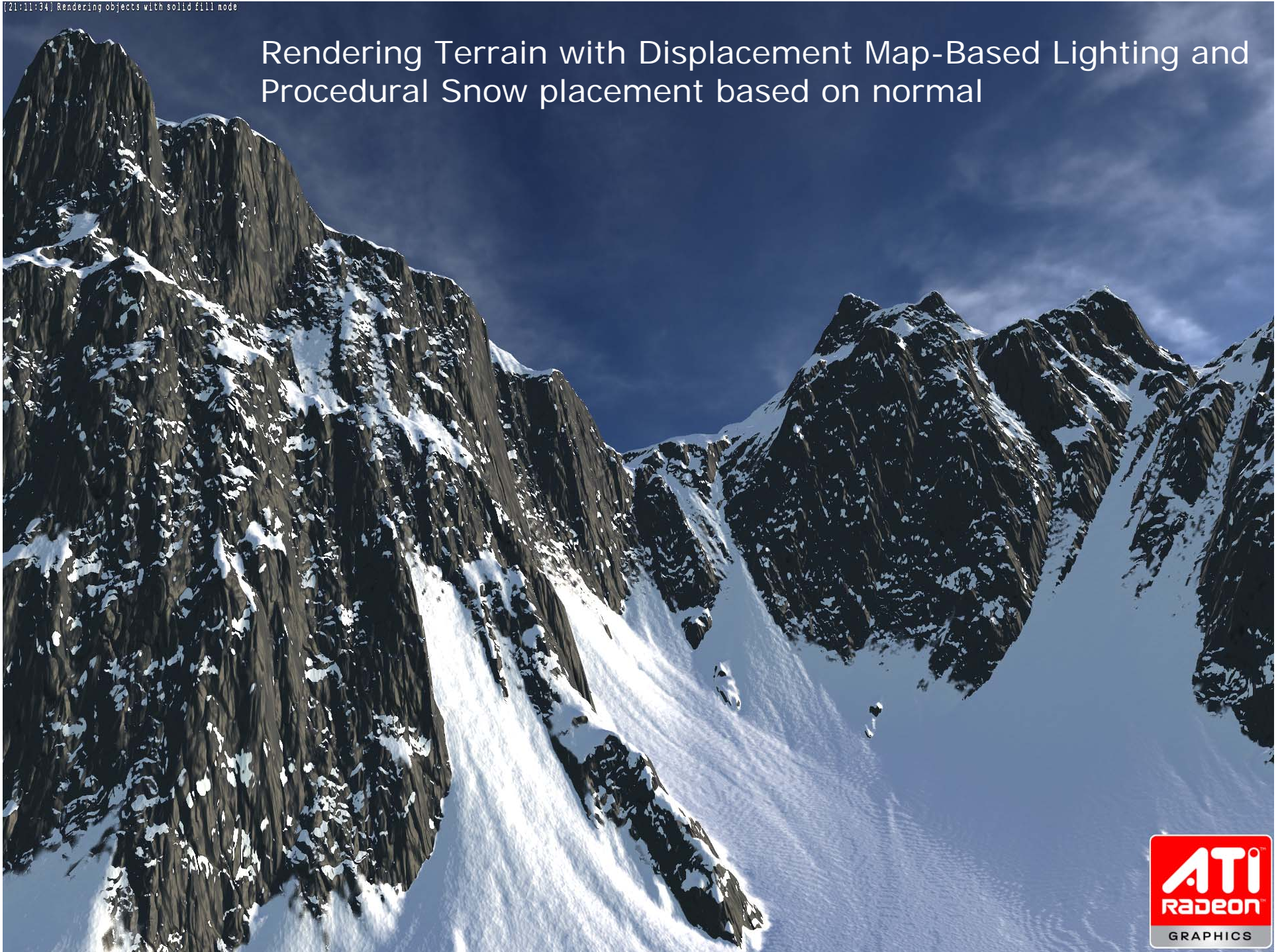
// Fetch transformed positions for these IDs:
float4 vVert0Pos = tex2Dlod( sTformVerts, float4( vVert0Coords, 0, 0 ) );
float4 vVert1Pos = tex2Dlod( sTformVerts, float4( vVert1Coords, 0, 0 ) );

// Swap vertices to make sure that we have the same edge direction regardless of their triangle order (based on vertex ID):
if ( vVert0Pos.w > vVert1Pos.w )
{
    float4 vTmpVert = vVert0Pos;
    vVert0Pos = vVert1Pos;
    vVert1Pos = vTmpVert;
}
```

Fetch the edge's endpoints from transformed vertices texture, then compute tessellation factor with your algorithm and output to PS / render target

[21:11:34] Rendering objects with solid fill mode

Rendering Terrain with Displacement Map-Based Lighting and Procedural Snow placement based on normal



Performance Comparison: High Res versus Full Tessellated Mesh

	Low Resolution with Tessellation	High Resolution, No Tessellation
On-disk model polygon count (pre-tessellation)	840 triangles	1,280,038 triangles
Original model rendering cost	1210 fps (0.83 ms)	
Actual rendered model polygon count	1,008,038 triangles	1,280,038 triangles
VRAM Vertex buffer size	70 KB	31 MB
VRAM Index buffer size	23 KB	14 MB
Rendering time	821.41 fps (1.22 ms)	301 fps (3.32 ms)

Both use the same displacement map (2K x 2K) and identical pixel shaders

Rendering with tessellation is > 6X faster and provides memory savings over 44MB! Subtracting the cost of shading



Performance Analysis: Low Res versus Rendering with Tessellation

Rendering Mode	Num faces:	Far away view		Close-up view	
		ATI Radeon HD 2900 XT	ATI Radeon HD 3750	ATI Radeon HD 2900 XT	ATI Radeon HD 3750
$N_T = 411 \times N_L$					
Original low res mesh (N_L)	4,050 triangles	359 fps	305 fps	275	215 fps
Continuously tessellated mesh (N_T)	1.6 M triangles	143 fps	122 fps	101	85 fps
Adaptively tessellated mesh N_A	Dynamic, $1.6M > N_A > 4K$ triangles	356 fps	253 fps	207	161 fps



Cinematic Rendering Relies on Tessellation for Quality and Control

Currently film and games differ in geometry management

- Cinematic rendering relies on extreme details
- Previously, games couldn't afford this luxury

We are changing this now!

- Both must manage details for stable performance



Geri's Game, Pixar



Cinematic Rendering Relies on Tessellation for Quality and Control – And So Will Games!

Bring tessellation techniques from film in real time rendering scenarios

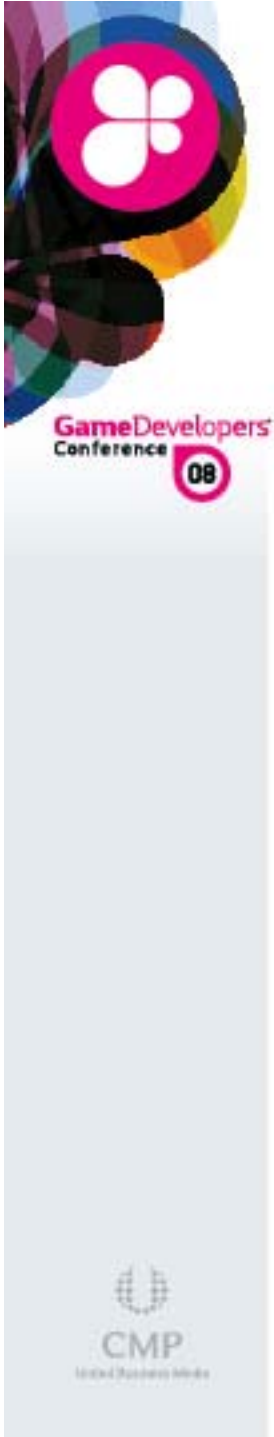
- Fast displacement mapping and animation

Results in significant visual improvements

- Better lighting
- Better silhouettes
- Better details
- Better animation
- Better performance



Geri's Game, Pixar



Acknowledgments

- ⌘ Josh Barczak and AMD Game Computing Applications Group
- ⌘ Nick Thibieroz, Bill Bilodeau, Holger Grün and Richard Huddy



References

- ③ Andersson, J. 2007. Terrain Rendering in Frostbite using Procedural Shader Splatting. SIGGRAPH 2007 Course Notes, Course 28: Advanced Real-Time Rendering in 3D Graphics and Games. San Diego, CA
- ③ Loop, C., and Schaeffer, S. 2007. Approximating Catmull-Clark subdivision surfaces with bicubic patches. Tech. rep., Microsoft Research, MSR-TR-2007-44
<ftp://ftp.research.microsoft.com/pub/tr/TR-2007-44.pdf>
- ③ T. L. Ni, Y. Yeo, A. Myles, V. Goel and J. Peters. 2008. Smooth Surfaces from 4-sided Facets. Shape Modeling International 2008
http://www.cise.ufl.edu/submit/files/file_ecb56556e79a4768187235473d37d700.pdf



Questions?

Thank You!



58
United Nations Media

WWW.GDCONF.COM