Game Developers Conference® | March 9-13, 2010 | Moscone Center | San Francisco, CA

THINK™ SERVICES
A DIVISION OF UNITED BUSINESS MEDIA LLC

GDC 10

Learn. Network. Inspire.

www.GDConf.com

Game Developers Conference®
March 9-13, 2010
Moscone Center
San Francisco, CA
www.GDConf.com

DiRT2 DirectX 11 Technology

Gareth Thomas, Codemasters

Jon Story, AMD

Agenda

- DirectX Comparison Video
- Porting to DirectX 11
- Tessellation Features
- DirectCompute HDAO
- Shadows using GatherCmp()
- Free Threaded Resource Loading
- Summary

DirectX Comparison Video

Porting to DirectX 11

- Render API already abstracted for multiple platforms:
  - PC, Xbox360, PS3, Wii
- Platform independent APIs for:
  - Buffers, Shaders, Textures, Render States, Drawing
- Rendering layer automatically handled:
  - Vertex Decls, Shader Constants, Multisampled Render Targets

Getting Started

- Dynamically loaded DX11 DLL, so we could live with a single EXE
    - Windows XP, Vista & Win7
- On start up we attempt creation of a DX11 device
- If that fails then we fall back to a DX9 device
- We actually kept our original DX9 device enumeration code – this worked out fine

The Dumb Port ☺

- Wanted to get 2D and static 3D objects on screen
- In place creation and destruction of state objects
  - Very slow ☹ – but got quick visual results
- Large constant buffer updates all over the place
  - Again very slow ☹ – but got things working

The State Manager

- Created a State Manager class
  - Hash map of state objects
- Check all state objects for existence with the manager
  - Create and store as necessary
- Only a handful of different state objects ever created
  - DX11 divides state objects logically
- Only load time state object creation, ~10% performance gain

Immutable Objects

- Static VBs & IBs must be flagged as D3D11_USAGE_IMMUTABLE
- Initially this was missed
  - Caused a massive GPU frontend bottleneck
  - Gave rise to ~30% gain in baseline performance

Tackling the Constant Buffer Problem

- Unreferenced constants & samplers are not optimized away
  - Different to DX9
- Shader source was organised such that multiple shader programs lived in one source file
- Therefore each shader would come with an enormous globals buffer
- This gave rise to _many_ large constant buffer updates
  - Very slow indeed

Constant Buffers – Solution (1)

- Sort constants by frequency of update:
  - Per frame constants
    - Lighting, Fog, HDR Multipliers, etc.
  - Render target constants
    - Width, Height, etc.
  - Camera constants
    - View, Projection, Eye, etc.
- This improved the situation
  - But still lots of constants getting dragged into global buffer

Constant Buffers – Solution (2)

- Wrap shader source & constant declarations with defines:
  - VERTEX_SHADER, PIXEL_SHADER, DOMAIN_SHADER, HULL_SHADER
- Pass appropriate define in when compiling shaders off-line
- ~25% performance gain
- Solution is still not perfect – work in progress on this problem…

Tessellated Animated Crowd – The Problem (1)

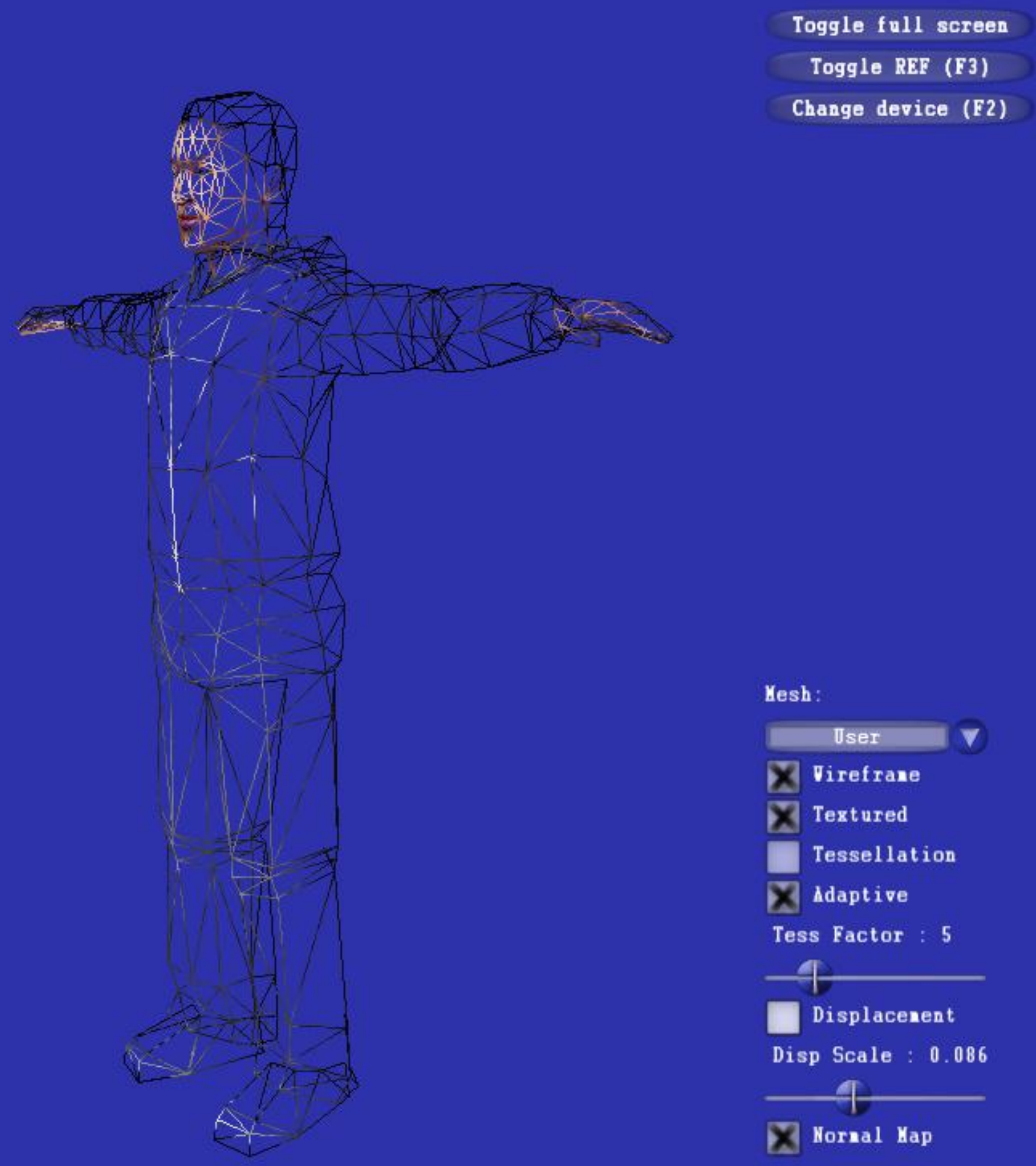
- Crowd meshes skinned on CPU
- Used instancing to render ~100k crowd models
- Difficult to up the fidelity of these meshes without incurring a large cost:
  - CPU skinning cost
  - Memory footprint & bandwidth
- We've wanted to improve quality here for some time...

Tessellated Animated Crowd – The Problem (2)

- Current high LOD mesh uses around 800 triangles
- Silhoutte is pretty angular
- Normal maps used to gain better lighting

Original Mesh

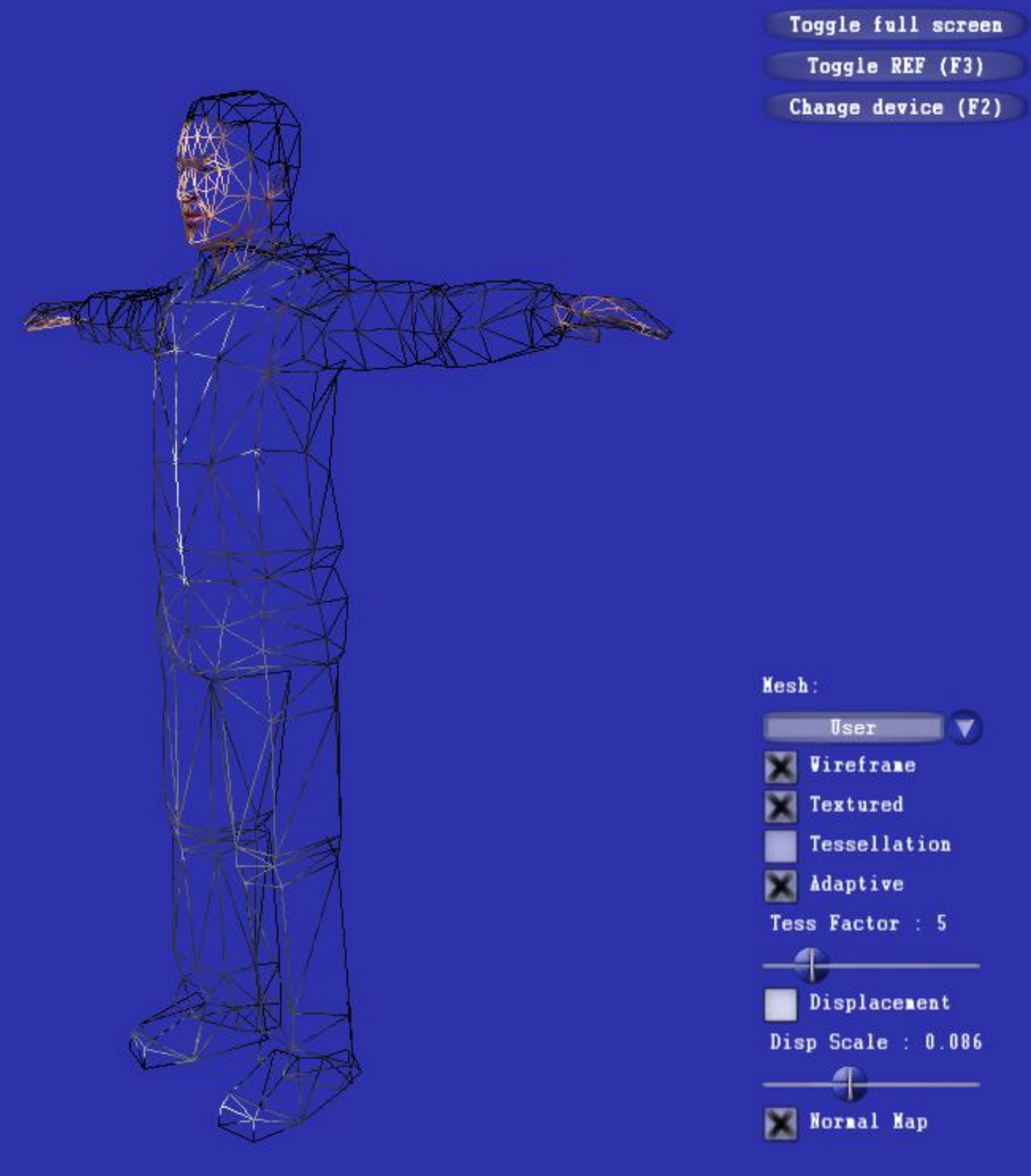
Original Mesh



Tessellated Animated Crowd – The Solution (1)

- Used PN-Triangles technique to smooth out base mesh using the existing data set
  - Curved PN-Triangles by Alex Vlachos, Jörg Peters, Chas Boyd, and Jason L. Mitchell
  - August 09 DirectX SDK Sample: PNTriangles11
- No new artwork required!
- CPU skinning, memory footprint & bandwidth unchanged
- Silhouettes much improved at tessellation factor 5

Original Mesh



PN-Triangles

PN-Triangles

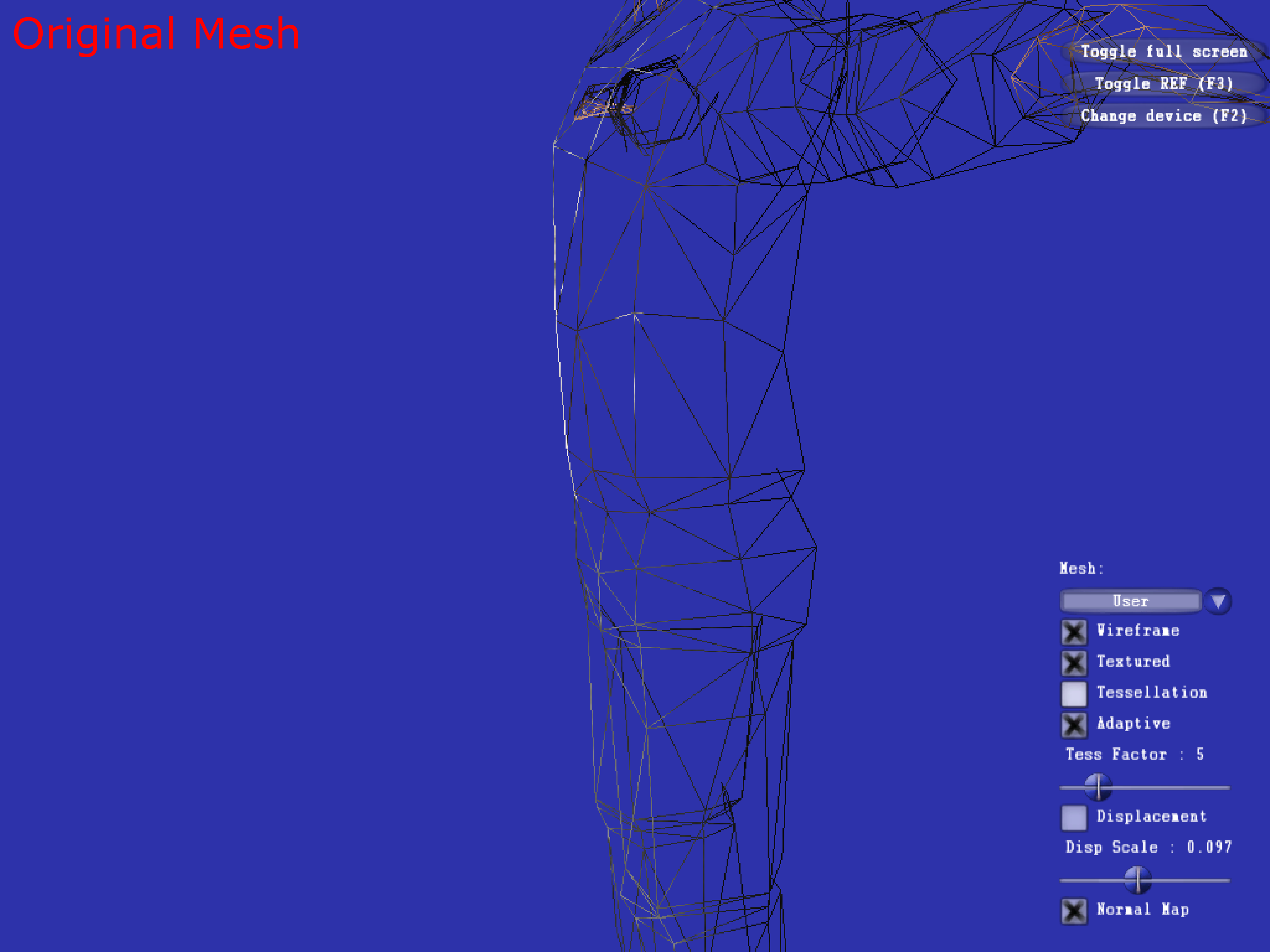
Tessellated Animated Crowd – The Solution (2)

- Used the normal maps as displacement maps
- Some artwork involved to ensure no cracks in the displaced meshes
  - Wrapped texture coords cause this
- Able to bring out nice details in hair and clothing
- Good use of the extra triangles generated

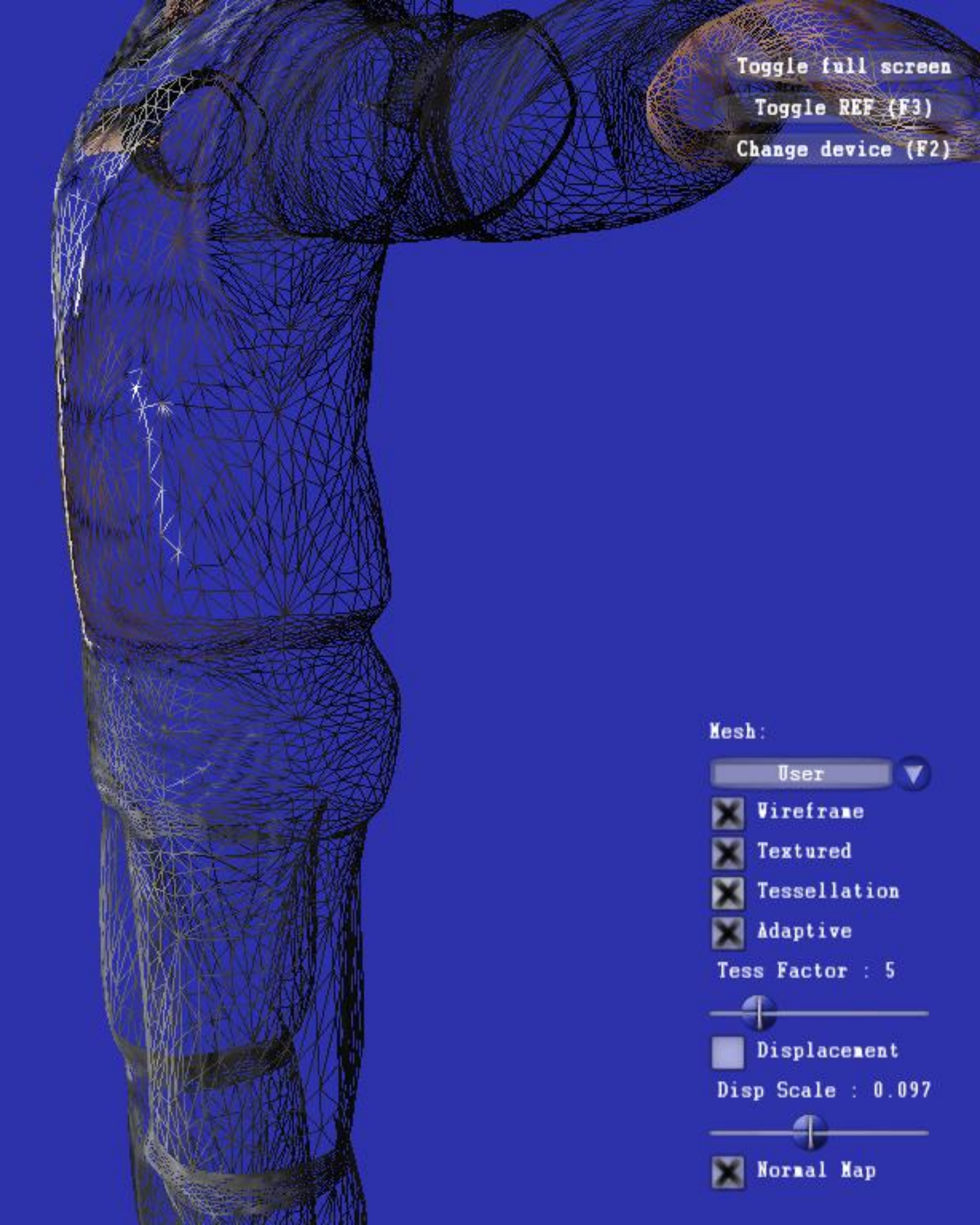
PN-Triangles

PN-Triangles + Displacements

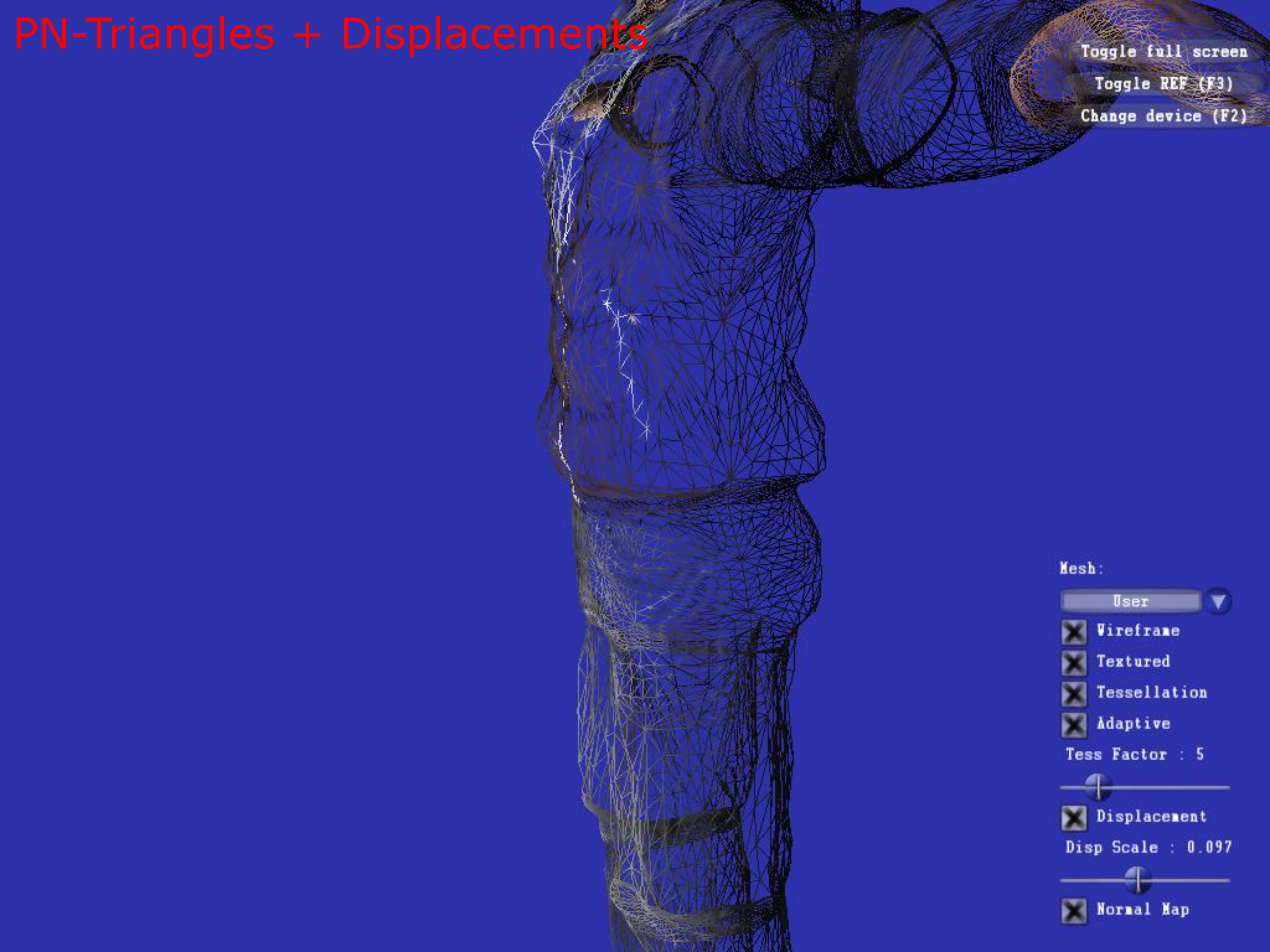
PN-Triangles + Displacements


Original Mesh
Toggle full screen
Toggle REF (F3)
Change device (F2)
Mesh:
User
Wireframe
Textured
Tessellation
Adaptive
Tess Factor : 5
Displacement
Disp Scale : 0.097
Normal Map

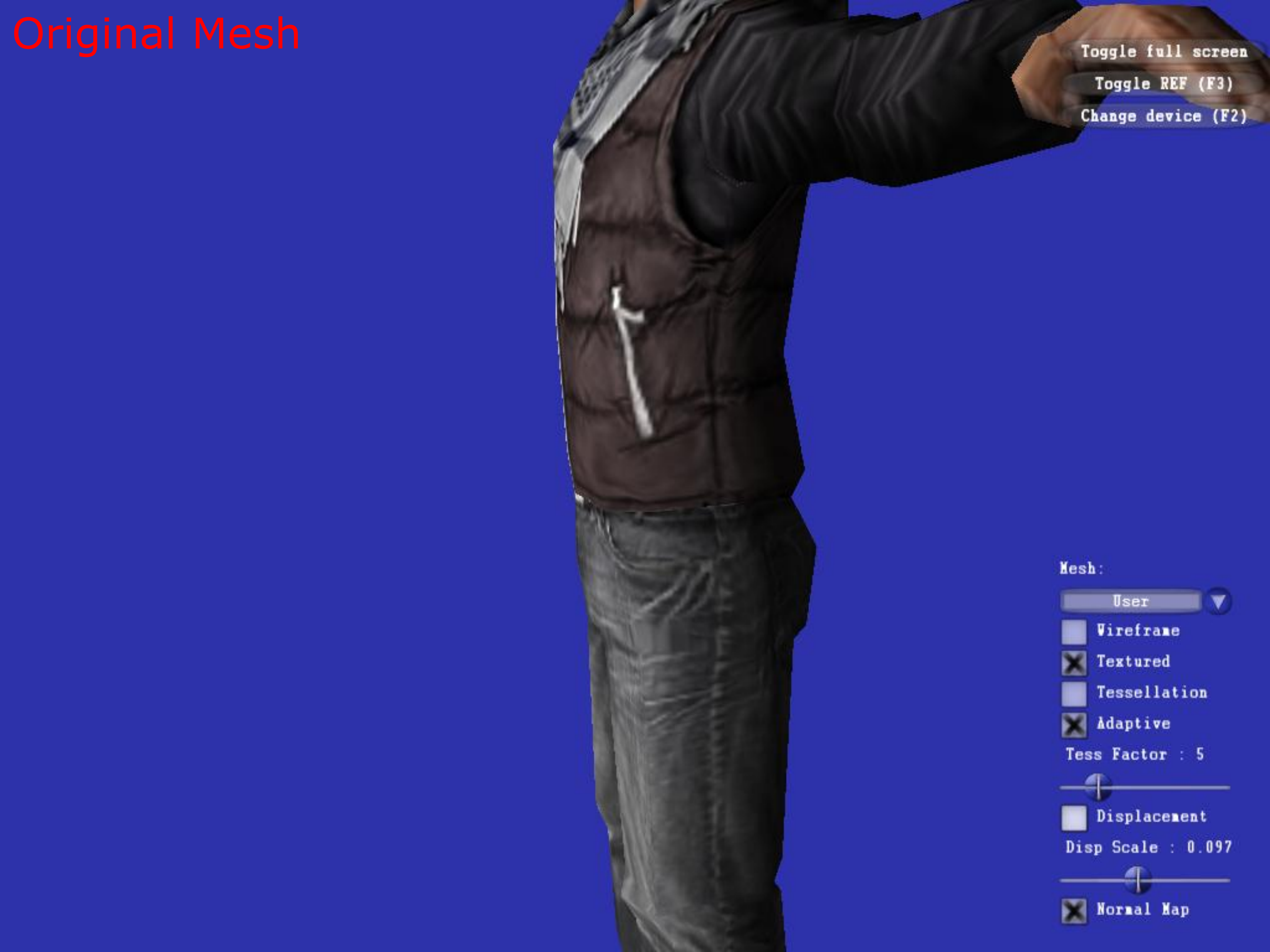
PN-Triangles



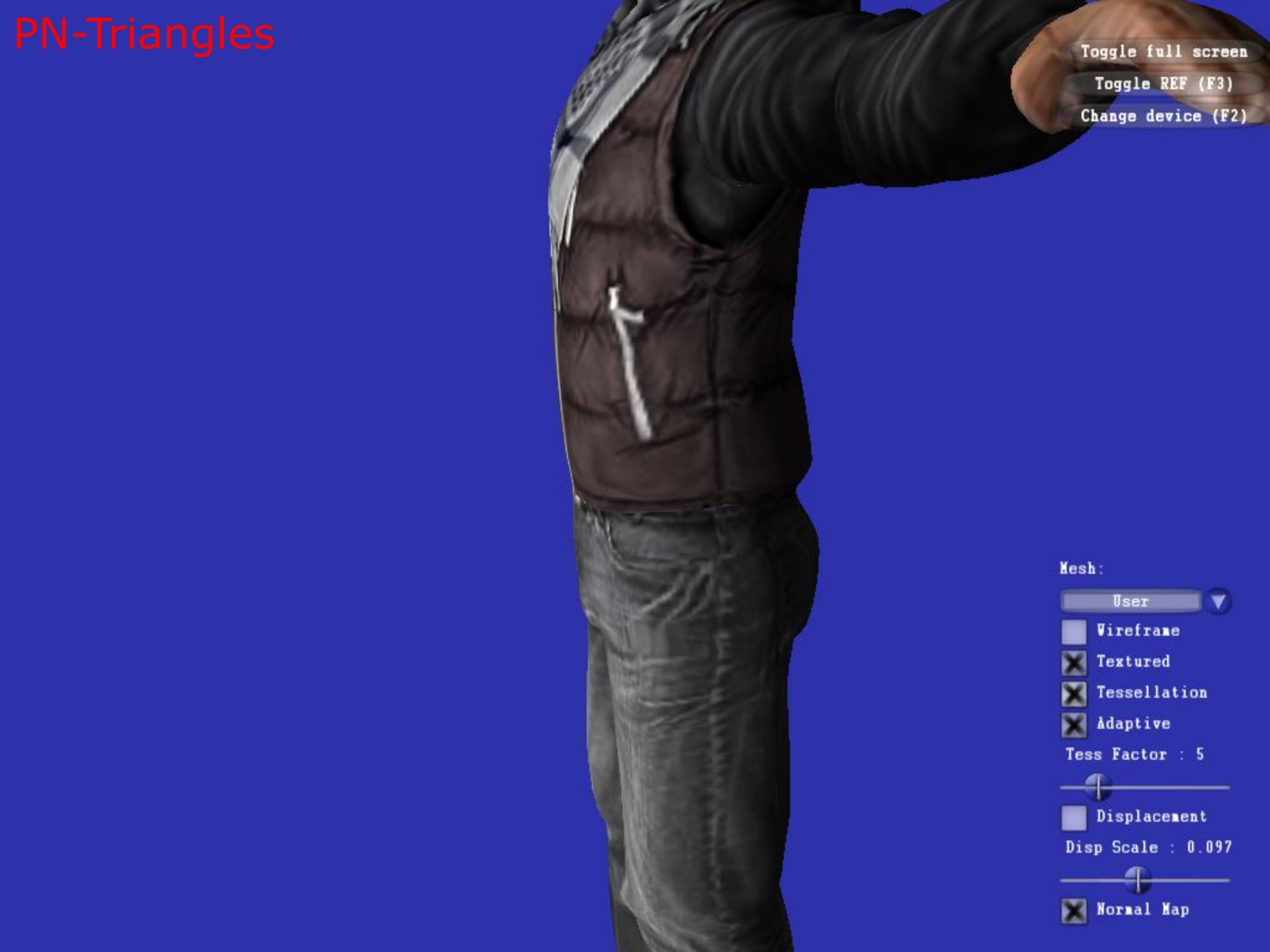
PN-Triangles + Displacements



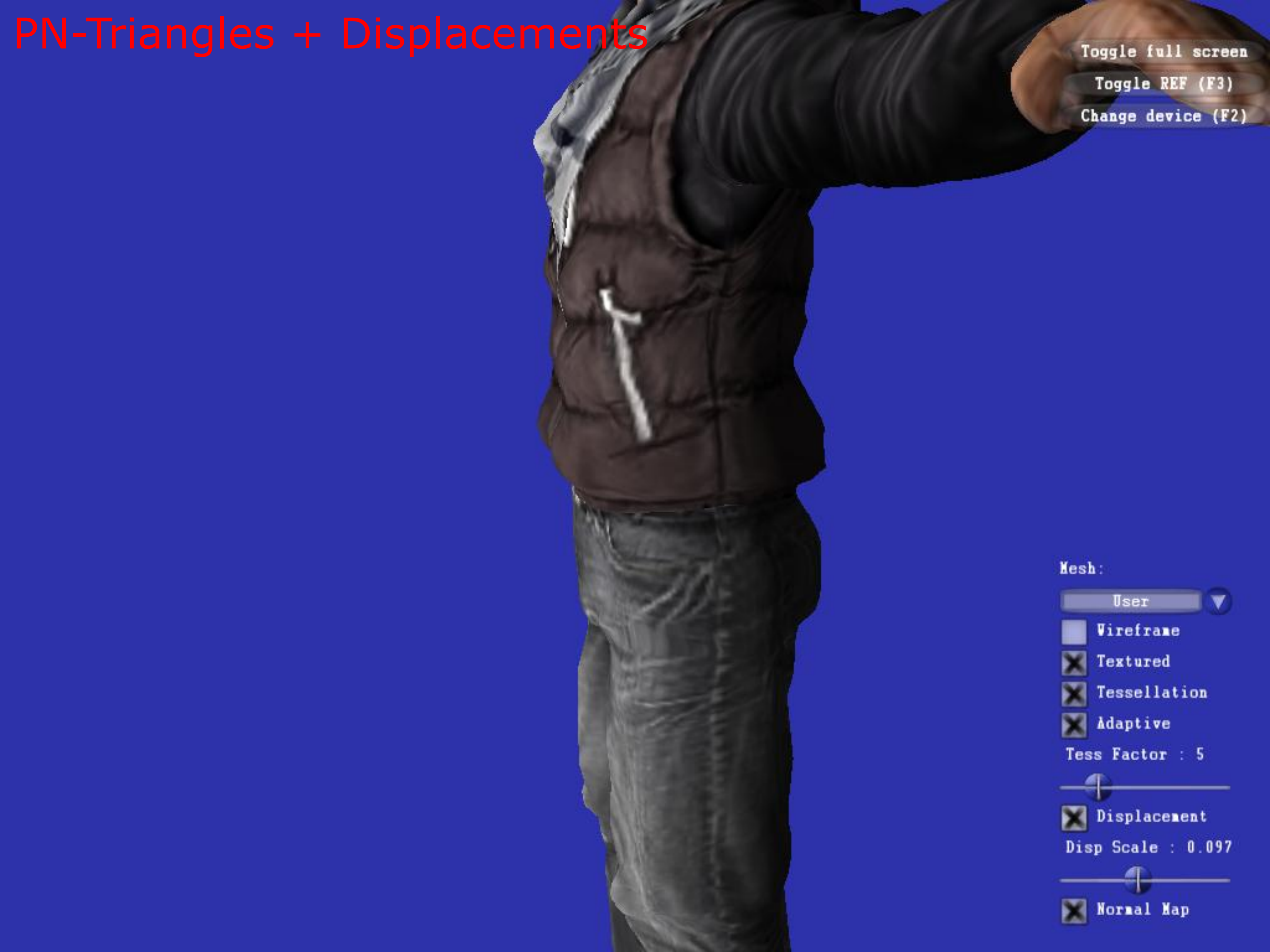
Original Mesh



PN-Triangles



PN-Triangles + Displacements



Tessellated Animated Crowd – The Solution (2)

- For adaptive tessellation factors we used two metrics:

  1) Patch size in screen space, because this worked even for camera zooms
  2) Distance from camera

- We switched tessellation off for meshes rendered below the top LOD

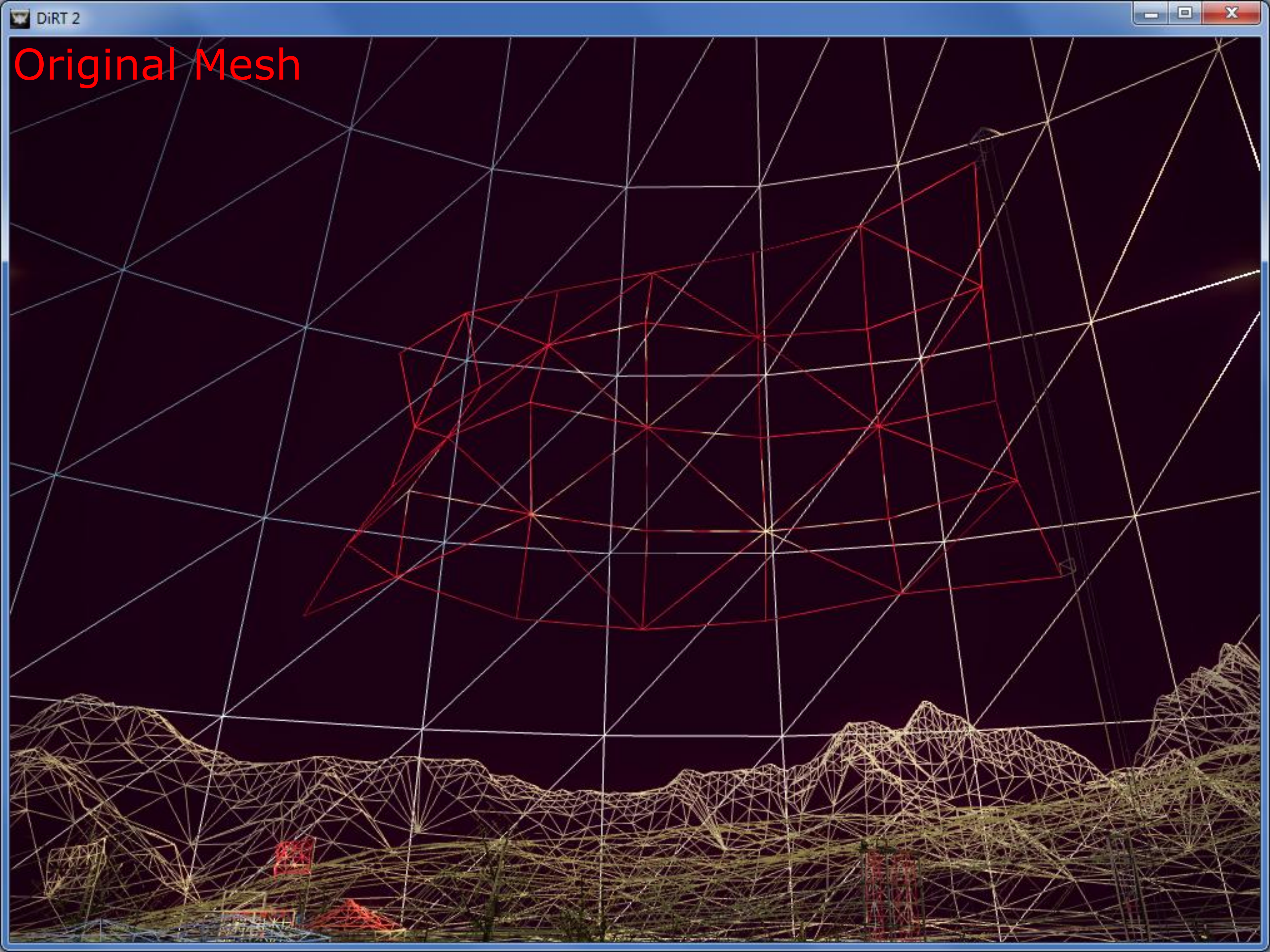
Tessellated Cloth (1)

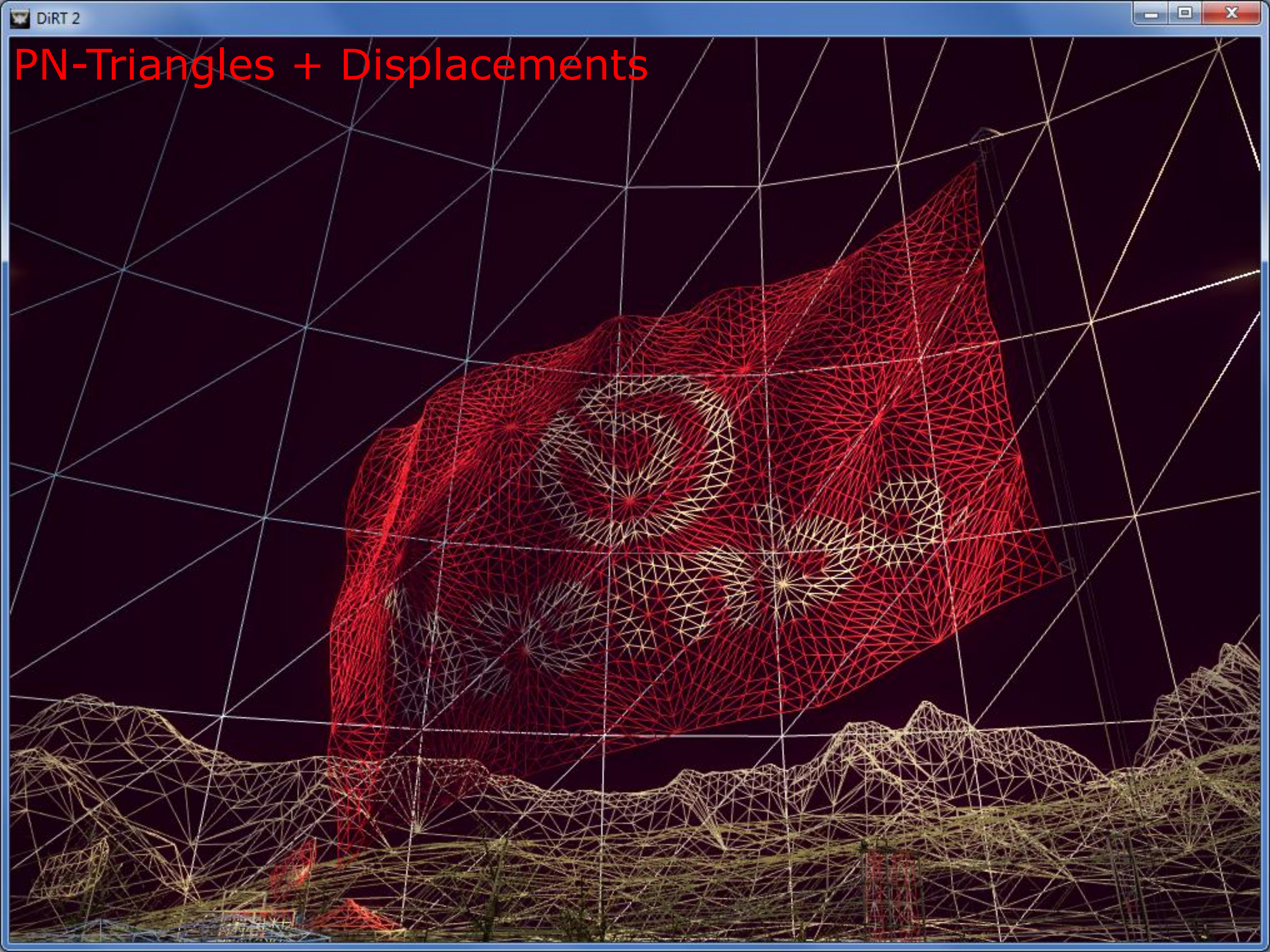
- CPU based physics simulation
  - Lots of flags and border cloth material
  - ~35 vertices in base flag mesh
- Also used a scrolling normal map for high frequency wind ripples
- Used PN-Triangles + Displacement Mapping
- Smoothed out the low detail mesh and added ripple details

Tessellated Cloth (2)

- Adding real geometry also meant improved self shadowing
- We employed the same adaptive tessellation algorithm as used for the crowd


DiRT 2
Original Mesh
brembo


DiRT 2
Original Mesh


DiRT 2
PN-Triangles + Displacements

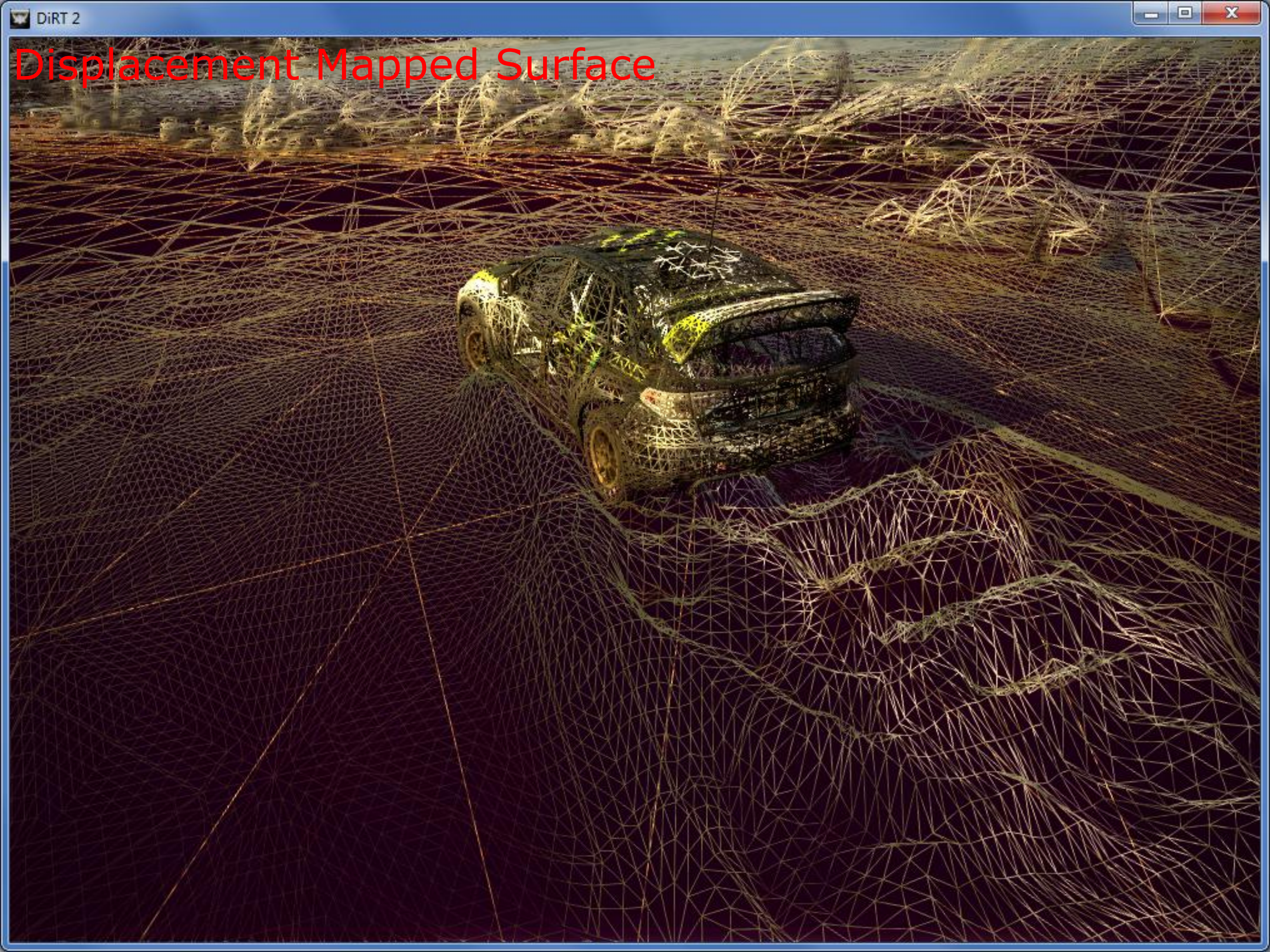

DiRT 2
PN-Triangles + Displacements
brembo

Tessellated Water (1)

- Wake simulated on CPU
- Normals uploaded to a texture
- Combined with scrolling ripple texture
- In DX9 we only rendered 2 triangles and used per-pixel lighting
- Worked quite well – but for many view positions the illusion was broken

Tessellated Water (2)

- In DX11 we used the DS to displace the water surface
  - Sampled from the normals texture
  - Typically a 512 x 512 map
- Resulted in a more physically accurate surface


DiRT 2
Displacement Mapped Surface

Displacement Mapping: OFF

Displacement Mapping: ON

DirectCompute HDAO

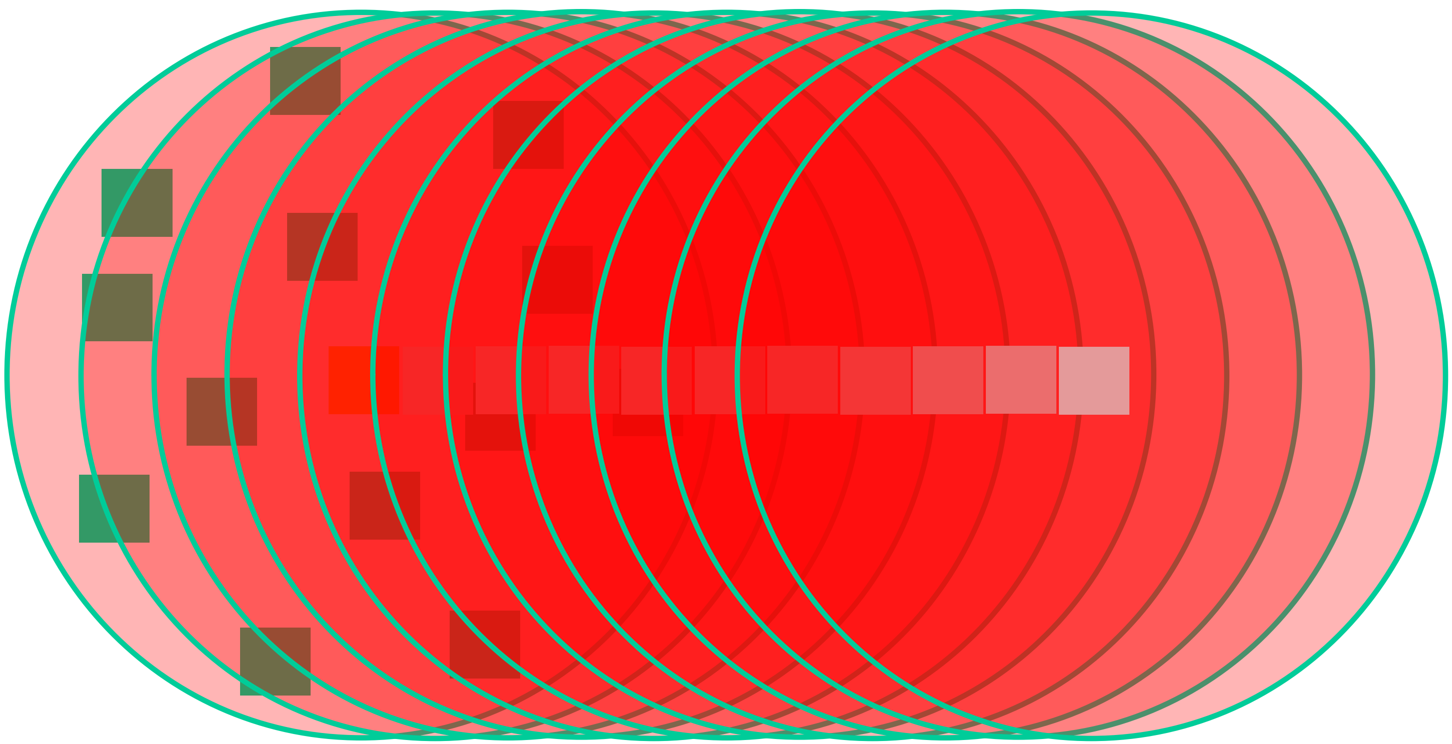
- HDAO adds dynamic high quality AO over and above our pre-baked solution
- HDAO is very texture heavy
- Refer to this link for an explanation on how it works:
  - http://developer.amd.com/gpu_assets/
- We used the CS to accelerate this technique...


HDAO: OFF
TIME 06:14.36
02:44.42 1 Mohammed Ben Sulayem
+00:03.20 2 Kent Kaufman
+00:03.62 3 Will King
+00:04.12 4 Jayde Taylor
000
MPH
R
0001435


HDAO: ON
TIME 06:01.59
02:44.42 1 Mohammed Ben Sulayem
+00:03.20 2 Kent Kaufman
+00:03.62 3 Will King
+00:04.12 4 Jayde Taylor
000
MPH
R
000143

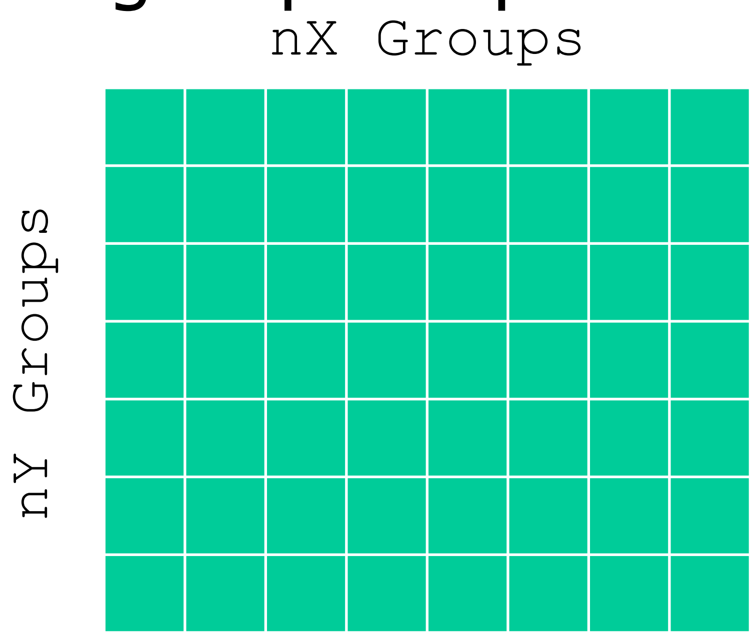
Post Processing the PS Way…

- [illegible] then sampled



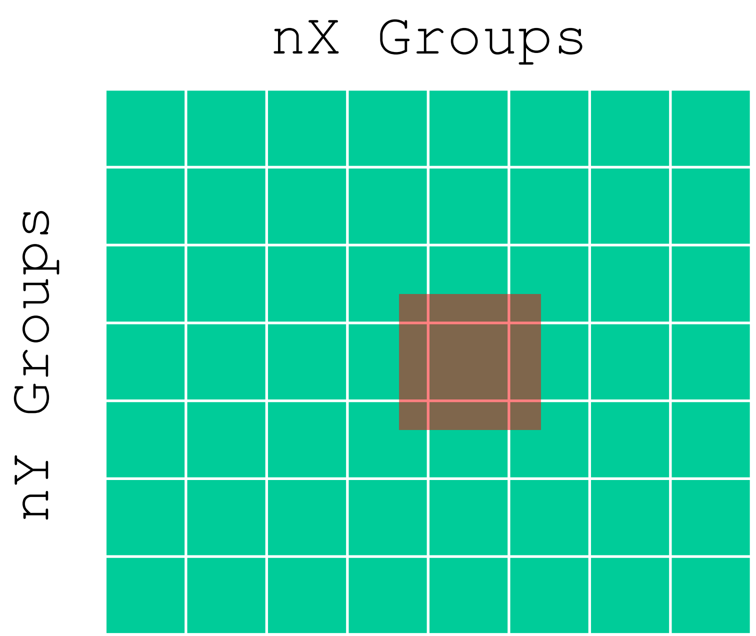
Overlapping Tiles (1)

- Use the LDS to drastically reduce the texture sampling cost
- Divide the screen up in to tiles for thread groups to process



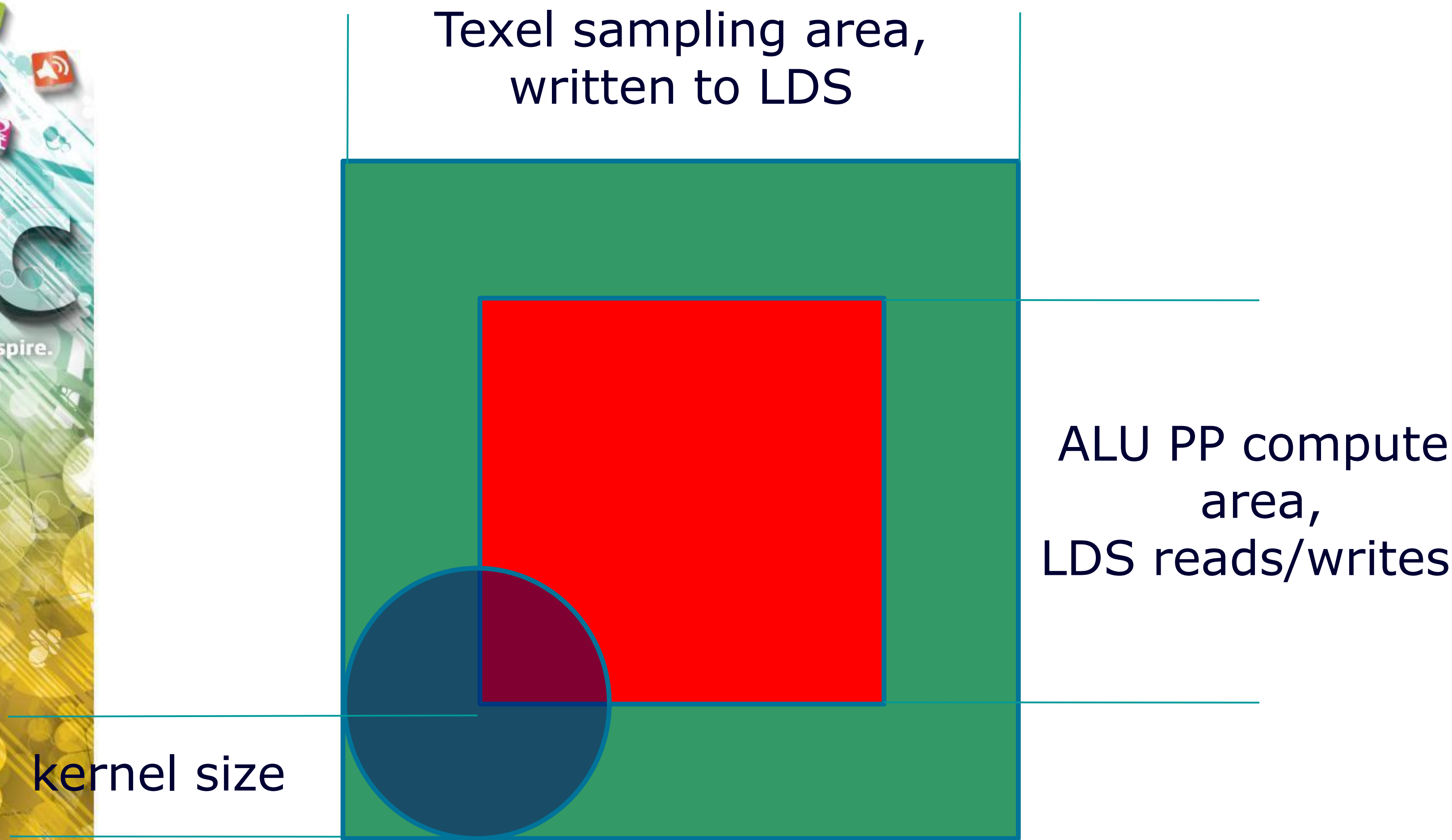
Overlapping Tiles (2)

- Kernel size determines level of overlap




```
// Region stored in LDS
uTexelDim = 56;
uTexelOverlap = 12;
uTexelDimAfterOverlap = uTexelDim - ( 2 * uTexelOverlap );
// Compute thread groups from screen res
iGroupsX = ceil( fScreenWidth / uTexelDimAfterOverlap );
iGroupsY = ceil( fScreenHeight / uTexelDimAfterOverlap );
// Dispatch thread groups
pd3dImmediateContext->Dispatch( iGroupsX, iGroupsY, 1 );
```

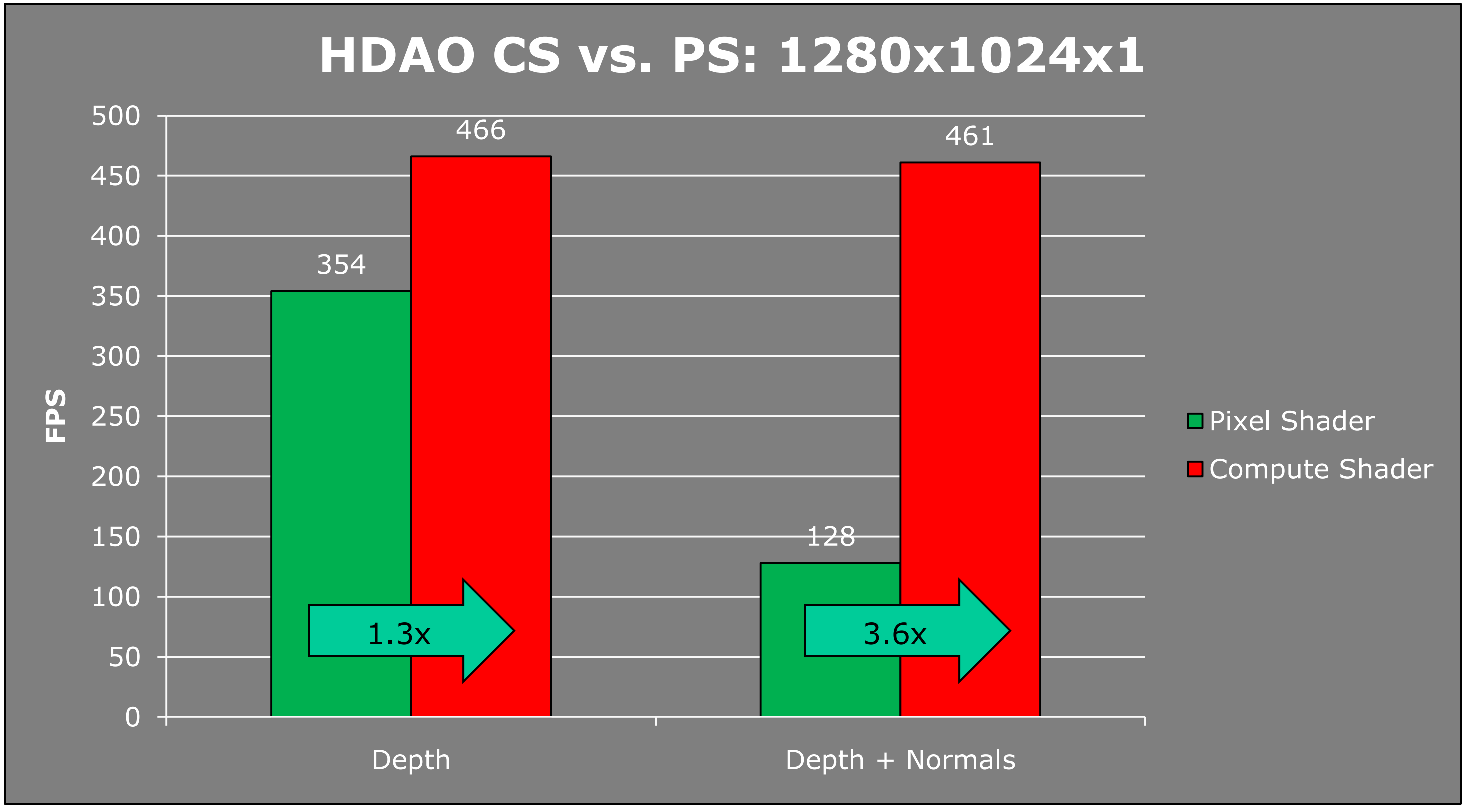
Overlapping Tiles (3)



Overlapping Tiles (4)

```
// Outline code...

// CS result texture
RWTexture2D<float> g_ResultTexture : register( u0 );

// LDS
groupshared float g_LDS[TEXELS_Y][TEXELS_X];

[numthreads( THREADS_X, THREADS_Y, 1 )]
void CS_PPEffect( uint3 Gid : SV_GroupID, uint3 GTid : SV_GroupThreadID )
{
    // Sample texel area based on group thread ID - store in LDS
    g_LDS[GTid.y][GTid.x] = fSample;

    // Enforce barrier to ensure all threads have written their
    // samples to the LDS
    GroupMemoryBarrierWithGroupSync();

    // Perform PP ALU on LDS data and write data out
    g_ResultTexture[u2ScreenPos.xy] = ComputePPEffect();
}
```

HDAO Performance



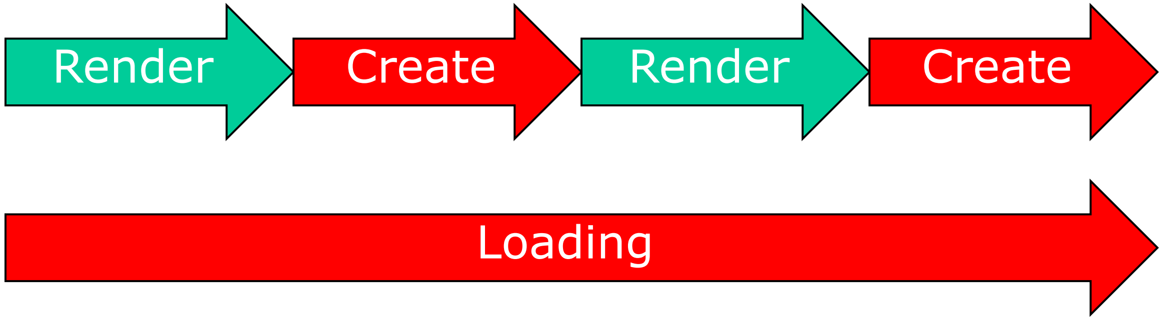
- Windows 7 64-bit, AMD Phenom II 3.0 GHz, 2 GB RAM, ATi HD5870, Catalyst 10.2

Shadows using GatherCmp()

- DX9 renderer implemented several solutions:
  - Fetch 4 (older AMD HW)
  - PCF sampling
- Cascaded shadow maps using D16 surfaces
- DX11 lookups performed using GatherCmp() instruction
  - Simpler to implement
  - Only one solution ☺

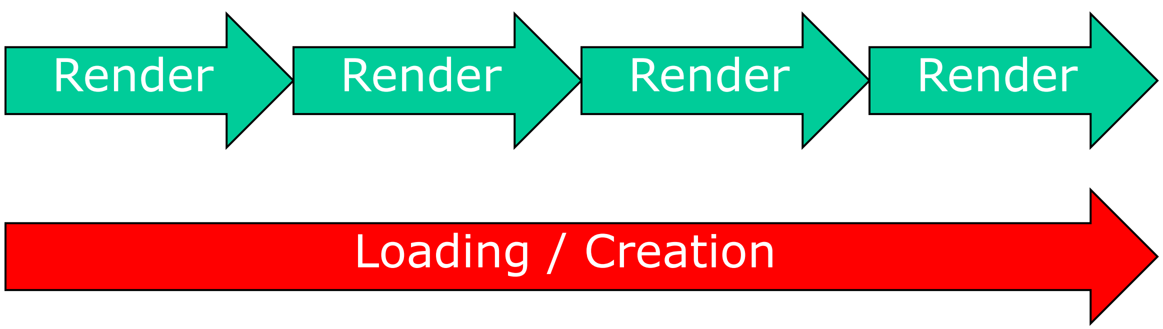
Free Threaded Resource Loading (1)

- DiRT2 uses a background loading thread
- Resources placed in a queue
- In DX9 mode resources are created on the main thread



Free Threaded Resource Loading (2)

- In DX11 mode resources are created on the loading thread
- Simpler and faster implementation
- Noticeably faster loading times, ~50% faster



Summary

- A naive port to DX11 will not be fast
- Tessellation greatly improves image quality and saves memory
- DirectCompute can significantly improve post processing performance
- Use free threaded resource loading to reduce loading times
- DirectX 11 Rocks!

Questions?

jon.story@amd.com

gareth.thomas@codemasters.com