White Paper

PN-Patches

March 2011
WP-03016-001_v01

Document Change History

Version	Date	Responsible	Reason for Change
01	April 7, 2010	KD, SM	Initial release

Go to sdkfeedback@nvidia.com to provide feedback on PN Patches.

PN-Patches

Abstract

Tessellation is one of the key features of DX11 API. It allows creating geometry on-chip without necessity to store it anywhere in video memory. GPU receives a single coarse patch, subdivides it to tiny polygons, displaces them to get rich detail, renders them, and then forgets about them. Subdivision and displacing is guided by two new shader stages – Hull Shader, and Domain Shader.

This sample shows typical tessellation implementation based on a monster model from Metro 2033 title. The model was generously provided to us by game developer - 4A games. Coarse model contains about 6000 polygons and is a mixture of quads and triangles. We subdivide them to tiny polygons using PN-Patches subdivision scheme. Then we displace those tiny polygons using displacement map to get the look of detailed model (modeled in z-brush) containing about 3 million polygons. This original detailed model is also available in the sample so that one could switch between tessellated coarse and detailed versions to see that there is virtually no visible difference. Tessellated version, looking the same, provides much better perf compared to detailed version. When one zooms out, LOD of tessellated version is automatically decreased providing even higher perf advantage (up to 50x depending on GPU) with the same look.

Sample also provides many debug visualization regimes (like wireframe, color coded normals, etc.) so that one could get better feeling of underlying technology.

Kirill Dmitriev
NVIDIA Corporation

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050
www.nvidia.com

How It Works

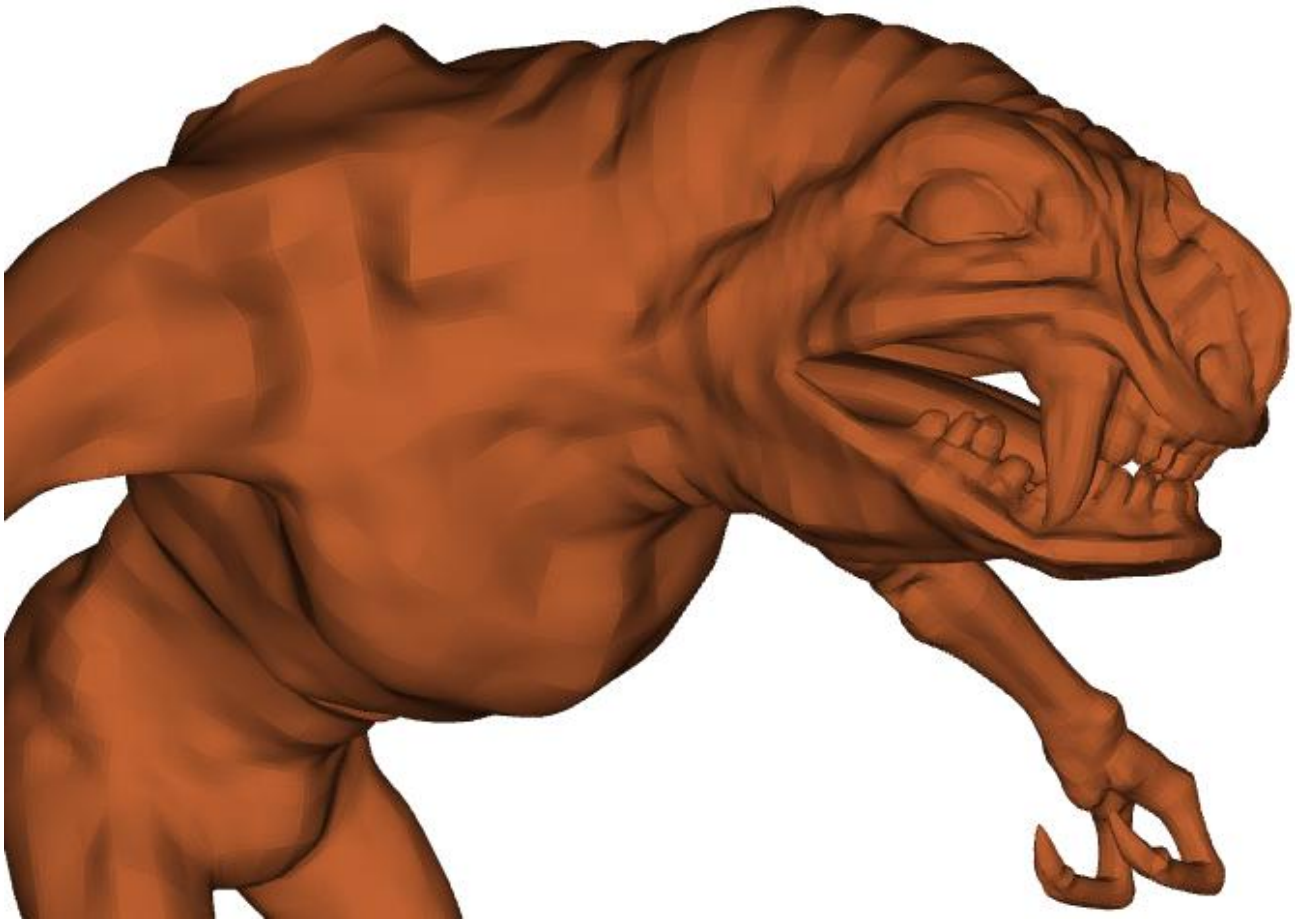
Subdivision scheme

Since model consists of both triangles and of quads, we needed subdivision scheme that can be applied to both triangles and quads. We subdivide triangles according to PN-triangles scheme that was proposed in [1]. For quads we are using PN-Quads scheme proposed in [2]. Those two schemes are very similar and they blend together very well – triangle and quad sharing an edge will have water-tight curved boundary between them after tessellation is applied.

Since PN-Triangle and PN-Quad schemes are very similar, not to address them separately, we refer them by the common name PN-Patch subdivision. We are aware of the fact that this name has been used before for different subdivision scheme, but we use this name here nevertheless because it is so convenient.

The following images give general idea of how the sample works. Initial model HW sees is coarse model, looking like this:

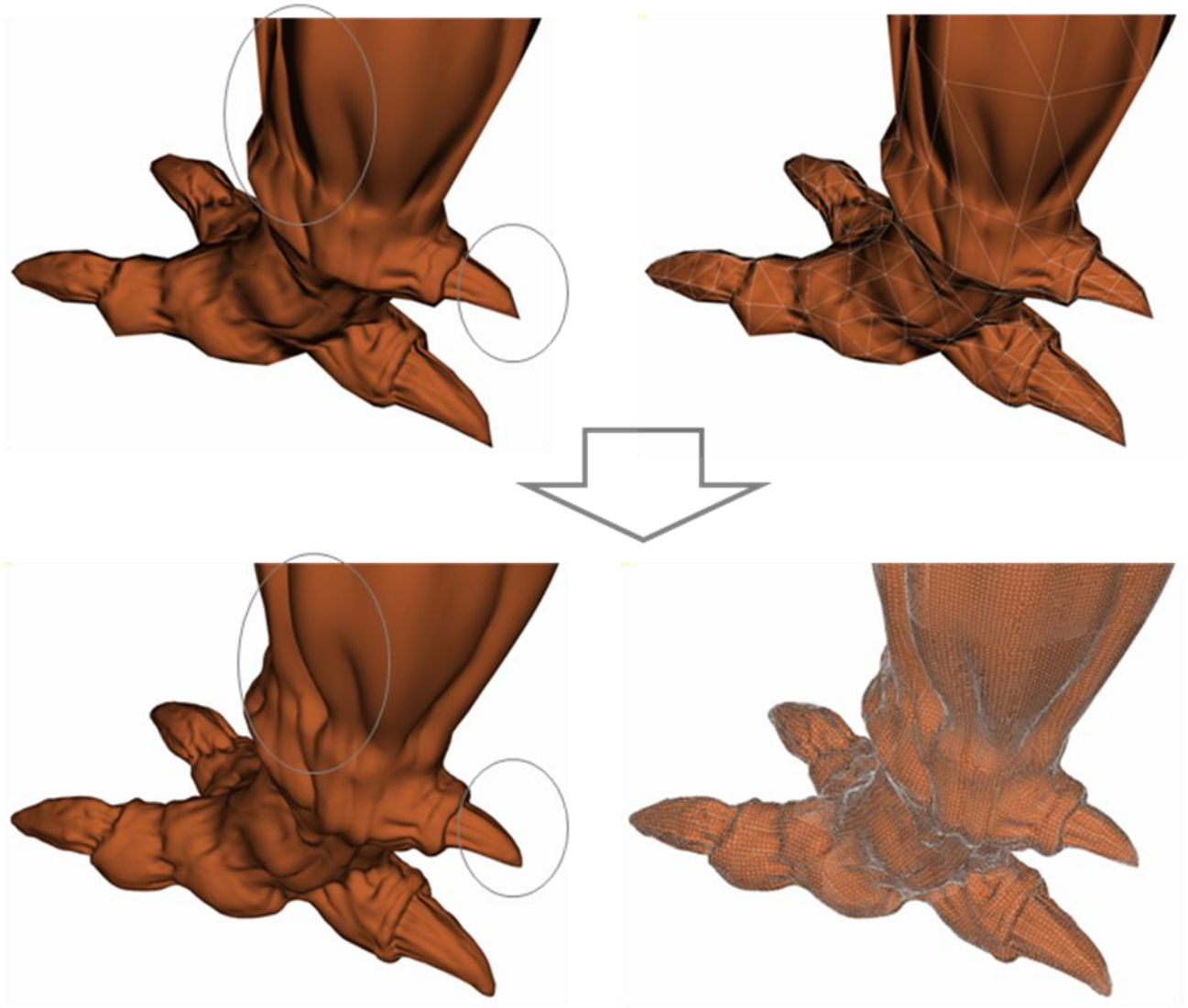
HW tessellates each patch of this model to tiny polygons and applies certain 'smoothing' (according to [1] and [2]) to each patch. If we only applied this smoothing and would not be applying displacement and normal map, we would be getting the following image:



After a patch is smoothed out like that, we let HW applied displacement and normal map, so that the final model looks like this:

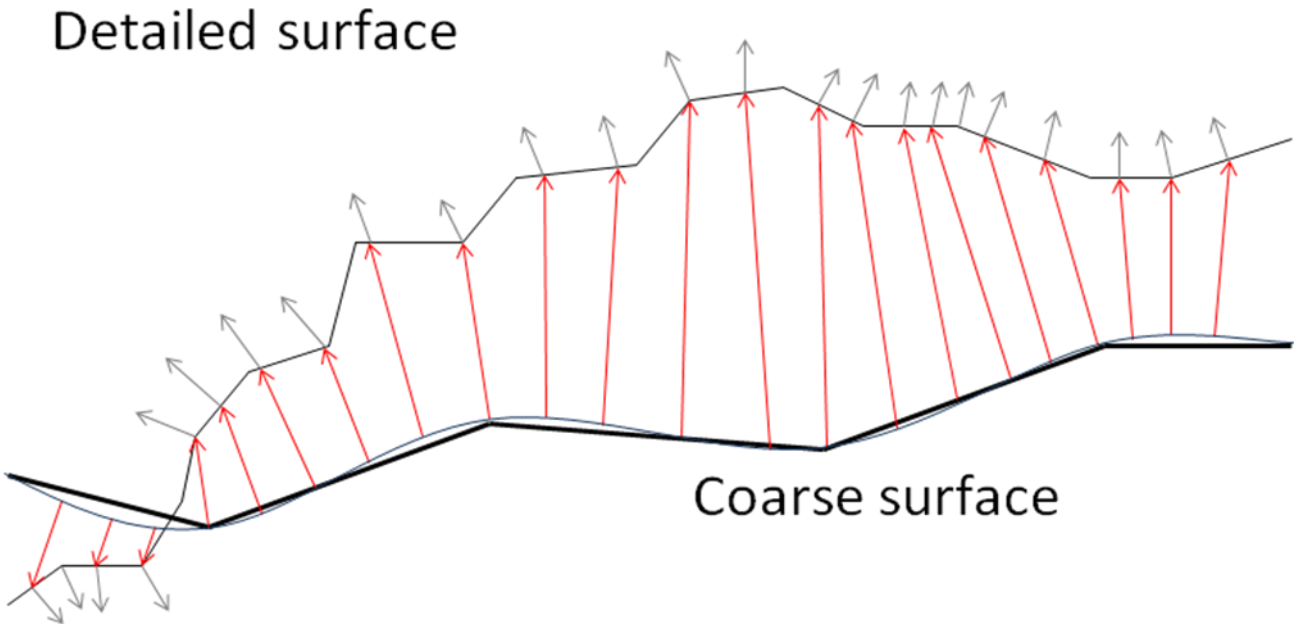
Displacement mapping is mainly needed to get the right silhouettes of monster parts, while normal map is needed to get the right shading. The below images

show that for example normal mapping alone is not enough to get the right look. Images at the top are generated using only normal map, while images at the bottom are generated with both displacement and normal map.



Displacement and normal map

Displacement at particular point on the surface of coarse mesh is distance between corresponding point on infinitely subdivided coarse mesh (also called limit surface) and corresponding surface of detailed mesh taken along the smooth normal to coarse mesh.



On the image above blue line is the limit surface. Red vectors are co-directed with smooth normals to coarse surface and their length is displacement. This displacement is stored in displacement map. Smooth normal need not be stored because it is known at render-time (it is simply an interpolated shading normal). Black vectors are smooth normals to detailed surface in the points where red vectors intersect it. Those normals are stored in normal map.

Both displacement map and normal map have been precompiled and stored in .dds files. They are bound to surface tangent space, so this model can be actually skinned and animated while still using the same normal and displacement maps.

Since the displacement is computed in respect to limit surface of particular subdivision scheme, displacement map applied for PN-Patch subdivision differs from displacement map that could be applied for, say, Catmull-Clark subdivision. At the time of developing this sample we could not find a tool that could generate displacement map in respect to PN-Patch subdivision, so we had to develop our own tool for that. This sample only includes displacement and normal maps that are output from the tool. The tool itself will probably be released separately in the near future. If you need it urgently, please contact us for that.

LOD computation

For LOD computation we use the following formula:

TESS_FACT = LEN * NP * Q / DIST

Where LEN is edge length in world space, NP is proportional to number of pixels on the screen, DIST is distance between observer and edge center and Q is global quality constant that is tweaked according to desired quality and performance. The goal of this formula is to achieve constant triangle size (in pixel) independently of distance between surface and observer and independently of current resolution.

LEN is edge length in world space. This factor in formula causes large polygons to be subdivided more densely than small polygons. So if initial model was subdivided non-uniformly in world space, this factor forces all triangles to become more or less of the same size.

DIST is distance between observer and edge center. The purpose of this factor is to make triangle size constant in screen space rather than in world space. After all, it does not make sense to have triangles smaller than one pixel because this level of detail can't be shown on monitor anyway. On the other hand, too large polygons lead to drop in quality. That's why Q factor is needed.

After LEN and DIST are in the formula, you get triangles of approximately constant size in screen space. This size may be too large or too small for you. To change this size, Q factor is introduced. By tweaking it, you may achieve desired size of triangles. We recommend having triangles with edges of approximately 4 pixels in length.

Now when you have achieved desired size of triangle, what happens when you change image resolution? Your triangles will now have different size in pixels than you originally intended. To fix that, NP factor is introduced. Set your NP to something like NP_NEW/NP_OLD, where NP_OLD is the number of pixels on the screen you had when your triangle size was correct, and NP_NEW is number of pixels in the new video regime. Then your triangle size in pixels will be staying approximately constant no matter what video mode you set.

Culling in HS

As was noted before, tessellation provides automatic seamless LOD for geometry. When surface is far away, its tessellation factor gets very small and performance rises quite a lot without noticeable drop in quality of image. The downside of that is that when geometry actually gets close, it is tessellated to very fine detail and performance suffers.

To fix that, our sample uses culling in HS. When the surface gets close to the camera, only small number of patches are visible – most of the patches end up being invisible due to frustum culling. Our HS determines this situation by analyzing corners of each patch. If all corners of particular patch are outside of

current frustum, zero tessellation factor is output from HS. This lets HW to kill this patch without tessellating it.

This makes performance real-time both at close-ups, and when object is far away.

References

[1] Alex Vlachos, Jorg Peters, Chas Boyd, and Jason L. Mitchell. Curved PN triangles. In 2001, Symposium on Interactive 3D Graphics, Bi-Annual Conference Series, pages 159–166. ACM Press, 2001.

[2] Jorg Peters, "PN-Quads", 2008