Rendering High Detail Models from Displacement Maps

Martin Volovár*
Supervised by: Peter Drahoš†

Institute of Applied Informatics
Faculty of Informatics and Information Technologies
Slovak Technical University
Bratislava / Slovakia

Abstract

In this paper, we present a method to generate a high resolution mesh from low poly mesh directly on GPU to reduce bandwidth overhead between GPU and CPU. We use known methods such as subdivision surface, displacement mapping and adaptive tessellation to generate more geometry in certain parts where it is necessary. This method is suitable for animation because small numbers of control points are modified. The main aim of this work is effective render a high quality mesh in the real time.

Keywords: Vector displacement map, Adaptive tessellation, Feature adaptive subdivision

1 Introduction

Since the first GPU has been released, GPU performance has been highly increased. Modern high-end GPUs can render around 6 billion triangles per second [11]. Memory bandwidth and an I/O latency has been improved too, but not as much as GPU render speed. What was not limiting factor before, is now a performance bottleneck. Transferring data between CPU and GPU is not a problem if a model geometry is static. In case of a model animation, modifying complex objects on CPU and updating GPU buffers can be impossible for every frame [9]. Motivation is to transfer only small parts of data and calculate model on GPU instead of transferring whole updated model. These parts could be changed position of control vertices or a changed sub-image of a displacement map.

Further motivation is to take advantage of adding detail dynamically in certain parts of model. This allows to change a complexity of a model according to its flatness and a screen space area. There is a similar method named LOD (Level of detail), which uses pre-generated models in different resolutions, but that method requires more memory and there is a problem in a continuity when the resolution is switched.

*xvolovar@stuba.sk
†peter_drahos@stuba.sk

2 Background

This section describes methods to generate high detail mesh from control mesh.

2.1 Subdivision surfaces

Smooth surfaces often occur in the nature. Traditional method, polygon surface, requires many polygons to approximate a smoothness [4]. Geometric modelling of complex models is problematic. In the past, memory for storing complex models was expensive. Using subdivision surfaces it is possible save memory storage.

Subdivision surfaces are a curved-surface representation defined by a control mesh [2]. Subdivision surface smooths initial model using recursive subdivision algorithm [3]. Subdivision level depends on how many subdivision steps are required. Subdivision step has two stages: mesh refinement and vertex placement. Mesh refinement subdivide every face and edge. Vertex placement set vertex position according to subdivision rule. Position is calculated by linear interpolation of neighbour vertices.

Hypothetical surface created after an infinite number of subdivision steps is called limit surface [2]. The limit surface has often C^2 continuity everywhere, except at extraordinary vertices [10]. The most well-known subdivision algorithm for quad meshes is Catmull-Clark.

Adaptive subdivision allows to use different subdivision level on certain parts of mesh. Adaptive subdivision uses flatness test to avoid subdividing flat parts of model [2].

In feature adaptive subdivision method, the limit subdivision surface is described by a collection of bi-cubic B-spline patches [8]. This is advantage because patches can be rendered directly using hardware tessellation. Instead of uniform subdivision, where geometry complexity grows exponentially, using feature adaptive subdivision, complexity is close to linear [6].

2.2 Tessellation

Tessellation is the process of breaking patch into many smaller primitives [11]. Patch is defined as set of control points. Patch type can be line, triangle, quad, B-Spline,

Bézier, etc. Tessellation factor controls fineness of patch. Adjacent edges should have the same tessellation factor because T-vertices could create a crack. Tessellation can convert quads to triangles, but common usage is to add geometric detail. Tessellation is now hardware accelerated.

2.3 Displacement mapping

Displacement mapping is using with tessellation to add high frequency detail with low memory I/O [7]. Instead of creating a smooth surface with subdivision surface, using displacement mapping it is possible make a rough surface.

A displacement map is a special type of texture where the stored values refer to a displacement of a vector. A commonly used type of displacement maps is the scalar displacement map, where each value corresponds to a displacement along the normal vector of a vertex. Scalar displacement map is easy to compute since normal vectors are cached. With vector displacement map it is possible displace vertex to any direction. Vector displacement map requires more memory than scalar because it uses all tangent vectors - t, b, n.

3 Our Contribution

Our solution combines feature adaptive subdivision scheme and displacement mapping with adaptive tessellation. Solution scheme is shown in the Figure 3. We use OpenSubdiv[1] library, because it supports subdivision and tessellation. An input for our program is a displacement map and control mesh. The mesh contains vertex positions, UV coordinates and indices. The input model should have a quad topology because we use a Catmull-Clark subdivision scheme. Catmull-Clark scheme can produce undulating artifacts (Figure 1). Faces should not overlap in texture space because we need 0..1 to 1 mapping between surface and texture space. The output is effective rendering of high resolution model.

3.1 Preprocessing

Input mesh is subdivided by feature adaptive subdivision algorithm. Current implementation of OpenSubdiv produces only bi-cubic patches for feature adaptive refinement[2]. Bi-cubic patches can approximate smooth surface like subdivision scheme [5]. UV coordinates of thhe new vertices are linearly interpolated from control points.

Input displacement map is filtered by Laplace filter (Equation 1) with the aim to identify which parts require a higher tessellation factor. Instead of filtering displacement map it is possible filtering the normal map. The advantage of doing this is that values in Laplace map depends on displacement map strength. In the Figure 2 is example of displacement map[3] and calculated Laplace map.

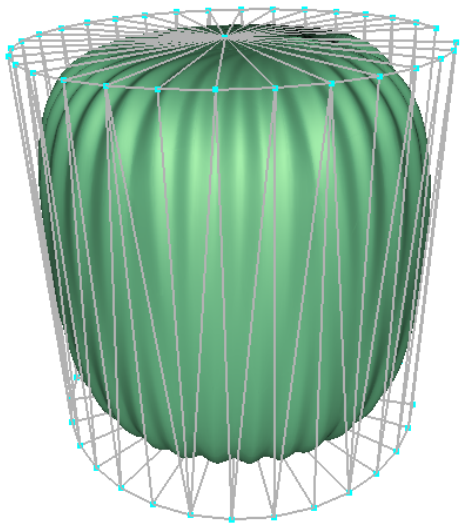

Figure 1: Catmull-Clark subdivision can behave poorly on triangle topology. Wireframe model is control mesh.

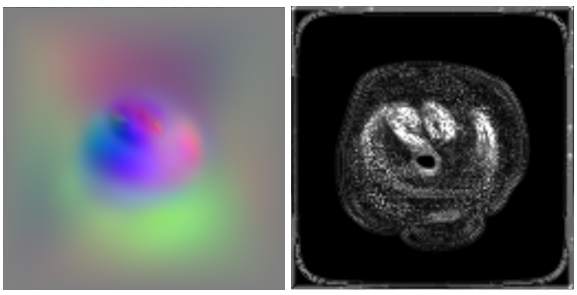

Figure 2: Example of displacement map and calculated Laplace map. The Laplace map is used in adaptive tessellation to affect tessellation fineness.

$$D^2xy = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad (1)$$

We generate a normal map for correct lighting, because application of displacement map can change direction of normal vectors. Normal map is obtained using FBO (Frame buffer Object). When we render the model and use normal map, we apply the normal map shading in Fragment Shader. This can reduce continuity problems when adaptive tessellation is on. Model also looks more detailed when it is low poly. We use two approach of generating the normal map.

First approach is described in Algorithm 1. Model is tessellated with high tessellation factor to get high quality model, so we can use the normal map for different LOD.

[1] http://graphics.pixar.com/opensubdiv/docs/intro.html

[2] http://graphics.pixar.com/opensubdiv/docs/subdivision_surfaces.html

[3] http://content.luxology.com/asset/exref/fd5171e820982995daaf5e15db00955d.png

Next, we apply displacement mapping. Calculating normal vectors is done by using Geometry Shader. This is because Geometry Shader has access to all points in triangle. Normal vector is obtained using cross product of two vectors that lied on triangle. Normal vector is normalized to unit length. Since all calculations are done in object space, normal vector is transformed to tangent space. To get normal map, whole model is rendered in texture space and colors are set respectively to normal vector.

Data: BSpline patches, displacement map
Result: normal map
foreach *patch* **do**
  tessellate with high tessellation factor;
end
foreach *tessellated triangle* **do**
  apply displacement mapping;
  calculate normal vector via cross product;
  transform normal vector to tangent space;
  set position to UV coordinate;
  transform position from UV space to NDC (Normalized Device Coordinates);
  set normal vector as output color;
end

Algorithm 1: Generating normal map from a displacement map.

Problem with this approach is that shading is naturally flat. This is because whole tessellated triangle have the same normal vector. Shading artifact depends on texel size of tessellated triangle. Figure 4 shows shading aliasing, where is normal map with different resolution applied.

Second approach uses linear interpolation of normal vectors of vertices in tessellated triangle. In this approach we do not use Geometry Shader. In the Figure 6 is described how we get the normal vector of vertex P. We use two near points P_A and P_B, where distance from actual point P is e in direction t and b tangent vectors. Like in the Section 3.3 we apply displacement mapping and calculate new positions P', P'_B and P'_A. Normal vector is obtained with cross product of two vectors that lies on triangle defined by vertices P', P'_B and P'_A. Unlike, in the first approach all calculations are in tangent space, so there is no need to transform the normal vector. Parameter e affects blurriness of normal map (Figure 5).

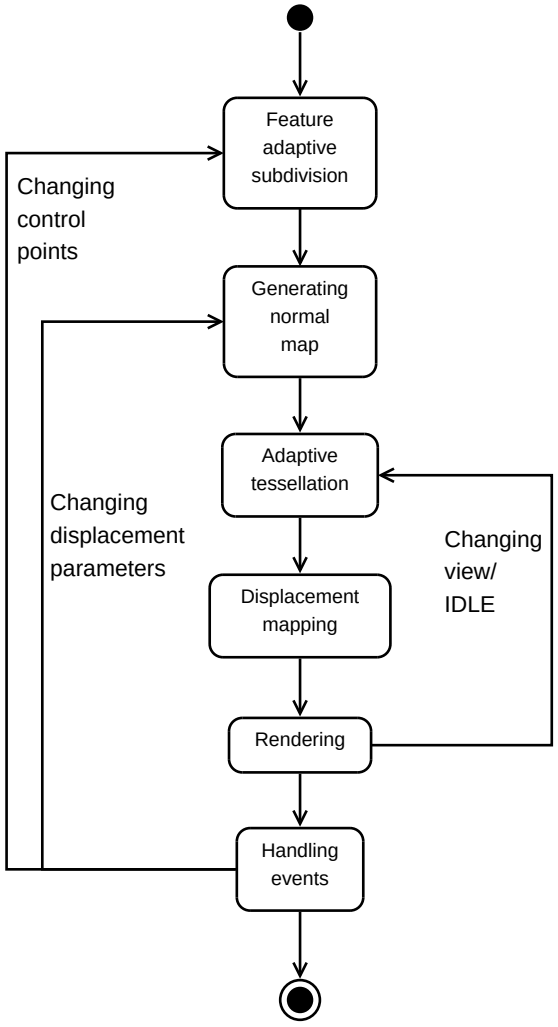

Figure 3: Overview of our method.

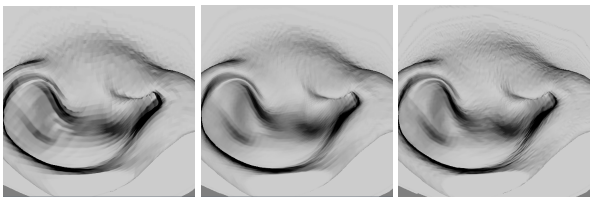

Figure 4: Shading aliasing of first approach of generating normal map depends on displacement map size (from left - 128×128, 256×256, 512×512).

3.2 Tessellation

For patches we use B-Splines, because they can approximate smooth surface. It is important to have the same outer tessellation factors along adjacent edges of patches. Otherwise, cracks can appear. There is also problem that patches can have different size. Two adjacent patches can have the same subdivision level or level differs by one. In the Figure 9 two cases are present, where T-vertices appears.

Tessellation fineness of our solution depends on patch level of detail T_{LOD}, patch flatness T_F and tessellation quality T_Q. Tessellation coefficients T_{LOD} and T_F are used because adaptive tessellation requires them. Initially we measured T_{LOD} as edge length in screen space. There was problem if angle between patch edge and view vector was small. It is shown in the Figure 8. To avoid this artifact T_{LOD} is calculated as l/w in center of edge, where l is dis-

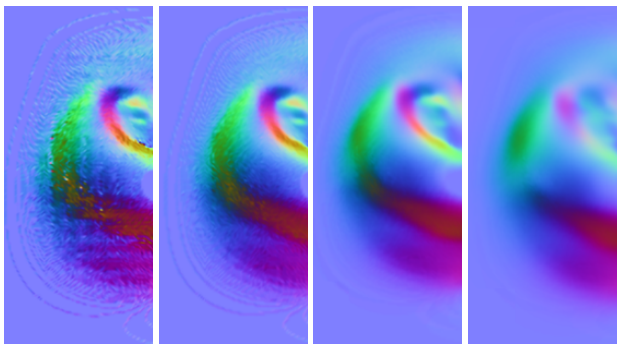

Figure 5: Parameter e (from left - 0.01, 0.02, 0.1, 0.2) affects blurriness of normal map in second approach of generating the normal map.

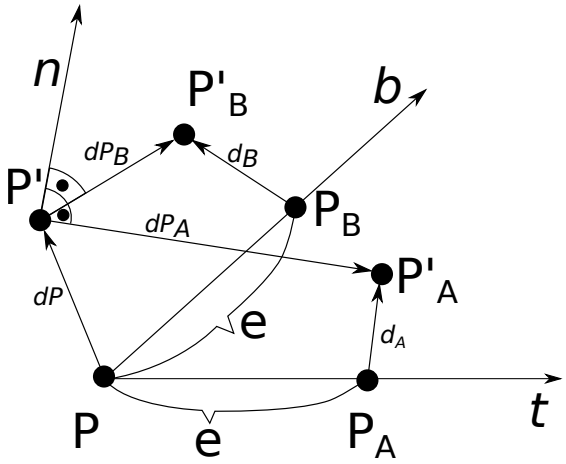

Figure 6: Calculating normal vector n in tangent space adding epsilon value e to P for vertices P_A and P_B on t and b axis. P', P'_A and P'_B are positions after displacement mapping of vertices P, P_A and P_B. Normal vector is obtained using normalized cross product of vectors dP_B and dP_A.

tance of two corner points that lies on the edge. We use 14 tessellation factors 2 inner and 12 outer (Figure 7).

Inner tessellation factor for horizontal and vertical direction is calculated as:

$$T_{in} = T_{LOD} \cdot T_F \cdot T_Q, \tag{2}$$

where:

- T_{LOD} is average of l/w of middle edge points of patch in screen space. l is distance between two corner points that lies on patch edge in object space. l is scale correction, so T_{LOD} depends of patch size.
- T_F is texture value in Laplace map of center patch UV coord.
- T_Q is global value of patch quality. Slow GPU should has this value low.

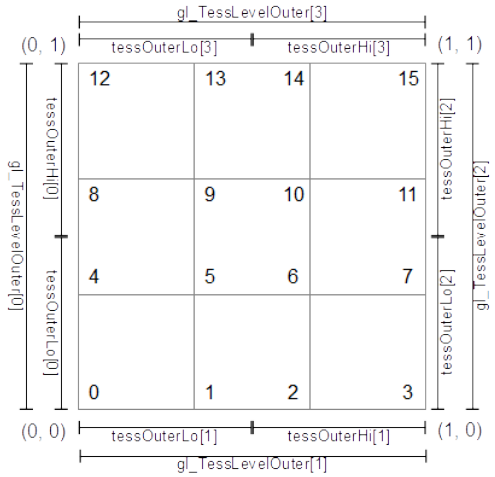

Figure 7: Outer tessellation factors for B-Spline patch used by Opensubdiv to avoid tessellation cracks. *tessOuterLo* and *tessOuterHi* are used in case transition edge (Edge connected with two smaller patch) [1].

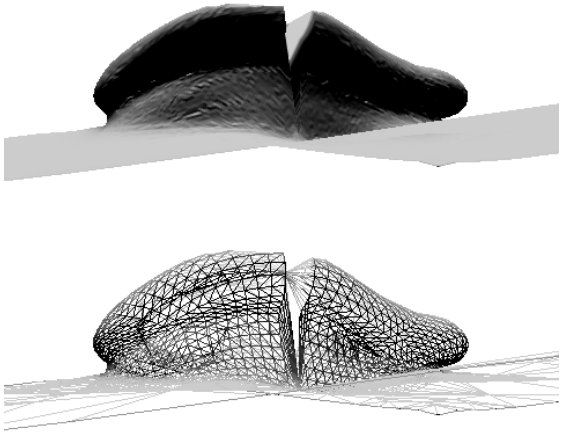

Figure 8: Tessellation artifact caused by using adaptive tessellation, where tessellation factor of edge is calculated as distance of two corner points, lies on this edge in screen space. Distance between corner points is small and edge has small outer tessellation factor.

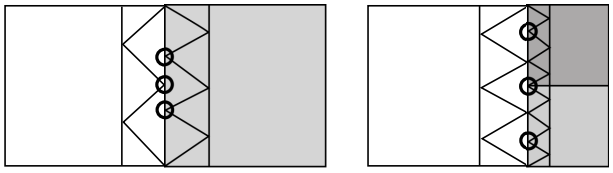

Figure 9: T-vertices occur in edge where patch has a) same subdivision level, but different outer tessellation factor on adjacent edge b) subdivision level that differs by one and $tessOuterLo$, $tessOuterHi$ tessellation factors are not the same as $tessOuter$ of adjacent edge of smaller patch.

T_{in} is rounded to nearest integer value and clamped, so minimal value can be 1. This is because inner tessellation with level less than 1 is undefined.

The easiest solution to avoid T-vertices is to set global outer tessellation factor to a constant value and in case of transition edge to double it. Problem with this solution is that inner tessellation factors change over surface and outer tessellation factor does not have to suit well. Our advanced method is based on simple idea that adjacent vertex has the same UV coordinate. We use the same Equation 2 like on inner tessellation factor, but coefficients T_{LOD} and T_F are calculated differently. For neighbour patches that uses same subdivision level, T_{LOD} is l/w of middle edge point in screen space. Else T_{LOD} is calculated as l/w of center in edge for Lo and Hi segments. If adjacent patch have the same subdivision level then T_F uses UV coordinate of middle point edge. In other case UV coordinate is chosen with weight 1/4 and 3/4 of corner UV coordinates because it is center of edge in smaller patch.

3.3 Displacement mapping

We use a vector displacement map. Tangent vectors - t, b, n are calculated from barycentric patch coordinates and patch parameters. Calculating a new position P' of a vertex in an object space is in the Equation 3.

$$P' = \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} + s \cdot \begin{bmatrix} t_x & b_x & n_x \\ t_y & b_y & n_y \\ t_z & b_z & n_z \end{bmatrix} \begin{bmatrix} d_r - o \\ d_g - o \\ d_b - o \end{bmatrix}, \qquad (3)$$

where:

- P' is new position in object space after the displacement mapping.
- P is an old position in object space before the displacement mapping.
- t, b, n are unit tangent vectors defined in object space.
- s is a strength of displacement.
- d is a vector displacement value in displacement map.
- o is a texture value, where displacement is zero. For unsigned displacement map it should be zero and for signed displacement map it should be set to 0.5.

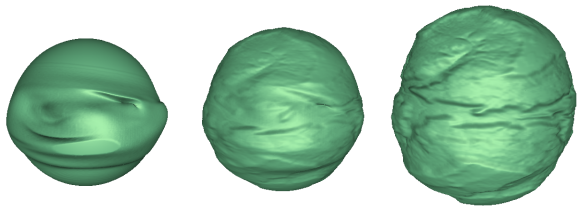

Figure 10: Morphing animation using linear interpolation of two displacement map.

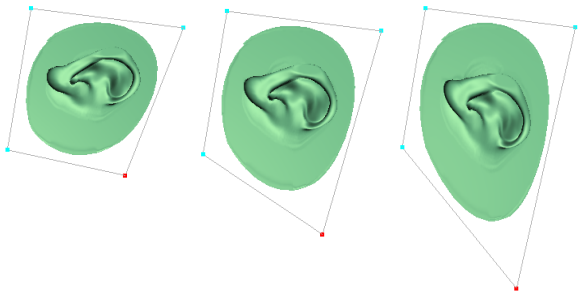

Figure 11: Animation, where control vertex is changing.

4 Results

We can animate model changing displacement parameters or changing control points position. Our solutions can with a small amount of control vertices change shape of model (Figure 11). It is also possible to create morphing animation with linear interpolation of two displacement maps (Figure 10). Our solution uses adaptive tessellation, so generated geometry is view dependent (Figure 12). In our tests we use Nvidia GT 740M GPU and i7-4702MQ CPU.

Generating subdivision surface from control mesh is fast (Table 2) because we use feature adaptive subdivision rather than uniform subdivision. Feature adaptive subdivision is usually faster than uniform because feature adaptive subdivision uses fewer patches than uniform subdivision uses quads (Table 1). This is because bi-cubic patches can better approximate limit surface. However, quad uses only 4 vertices instead of 16 in case of B-Splines. Model complexity of feature adaptive subdivision grows linearly instead of exponentially. CPU time is measured via high resolution timer[4] in beginning and ending of generating function. OpenGL functions can be asynchronous, so GPU time is measured using GL_TIME_ELAPSED query.

[4] https://msdn.microsoft.com/en-us/library/windows/desktop/dn553408

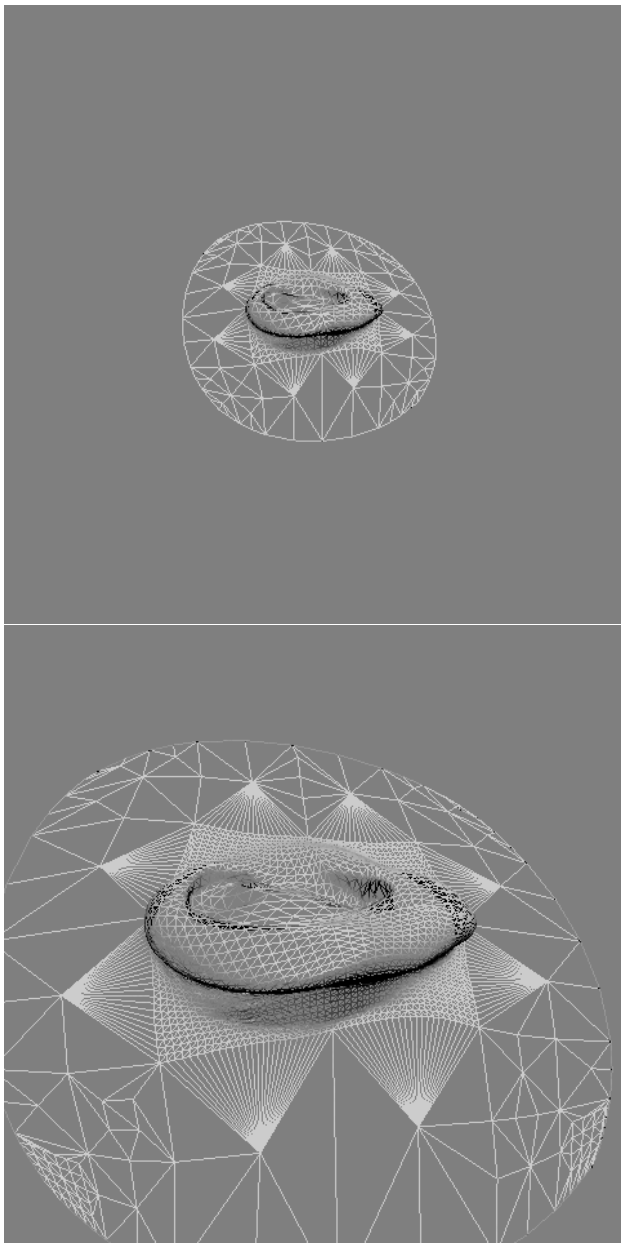

Figure 12: Generated geometry is view dependent.

Table 1: Geometry complexity of Feature Adaptive subdivision and Uniform subdivision.

Control mesh		Subdivision		
		level	Feature Adaptive	Uniform
Faces	Vertices		Patches	Quads
1	4	3	28	256
1	4	6	64	4096
1	4	8	88	65536
406	450	3	442	25984
406	450	6	478	1,663 M
406	450	7	490	6,652 M
4697	450	2	8926	74848
4697	4678	3	11098	0,299 M
4697	4678	5	15442	4,790 M

Table 2: Generating subdivision surface from control mesh.

Control mesh		Subdivision				
		level	Feature Adaptive		Uniform	
Faces	Vertices		Generating time [ms]		Generating time [ms]	
			CPU	GPU	CPU	GPU
1	4	3	1,62	0,018	0,016	0,004
1	4	6	1,71	0,019	0,767	0,001
1	4	8	1,74	0,017	11,42	0,017
406	450	3	1,79	0,003	9,139	0,003
406	450	6	1,80	0,065	649	0,002
406	450	7	1,86	0,064	2755	0,001
4697	450	2	1,73	1,024	231	0,001
4697	4678	3	1,72	3,670	110	0,010
4697	4678	5	2,14	5,865	1966	0,020

We use GPU Evaluator in feature adaptive subdivision, so GPU time grows faster than CPU.

Table 3 shows that second approach of generating the normal map is faster. We assume it is because first approach uses transformation to tangent space. In second approach is one more pipeline stage - Geometry Shader. Time of generating normal map depends mainly on model complexity.

Table 4 shows generating time of Laplace map in different resolutions.

In other test we compare render time between our method with tessellation and drawing raw array of triangles. We also test input memory size of our method and polygon surface method. We would rather compare render time between our method and polygon surface, but all connectivity information is lost. Our test model contains 4 control vertices with position and UV coordinate. Model size in both axis is one unit. Input control of our method mesh takes 0,1 kB. We used 128×128, 256×256 and 512×512 RGB displacement texture. Displacement tex-

Table 3: Normal map generating time.

Triangles	Patches	Texture size	Time of first approach [ms]	Time of second approach [ms]
7048	28	256×256	0,541	0,986
7048	28	512×512	0,618	1,048
7048	28	1024×1024	0,719	1,282
1,037 M	442	256×256	9,256	21,348
1,037 M	442	512×512	9,438	21,277
1,037 M	442	1024×1024	9,954	21,544
29,623 M	11098	256×256	186,671	451,689
29,623 M	11098	512×512	188,061	451,899
29,623 M	11098	1024×1024	191,264	455,060

Table 4: Laplace map generating time.

Texture size	Time [µsec]
256×256	73
512×512	261
1024×1024	970

ture has size of 48, 192 and 768 kB. We capture generated tessellated surface as transform feedback and draw again with polygon surface method. Render speed is measured using GL_TIME_ELAPSED between draw command. In case that generated triangles is more than value around 9000 our method is faster. Maybe, it is because of the memory overload. Number of triangles is obtained via GL_PRIMITIVES_GENERATED query.

In Tables 5, 6 and 7 estimated mesh size is present, if we used polygon surface method. We assume using uint16 index buffer, 32-bit float vertex buffer, 8 vertex attributes (3 - position, 3 - normal vector, 2 - UV coordinate) and triangle grid topology.

5 Conclusions

Our method allows to generate mesh from low poly control mesh. This method has some advantages: automatic generation LOD, animation with changing small number of control points, sculpting surface, faster animating, etc. It is possible that out method uses less memory.

There are some artifacts: aliasing in the normal map, continuity in adaptive tessellation, problematic UV mapping between closed surface and texture space. Future work can try to avoid these artifacts.

Table 5: Generated geometry and render time (displacement map 128×128)

Distance	Triangles	Our method render time [µsec]	Triangle array render time [µsec]	Polygon surface mesh estimated size [kB]
3,138	1258	53	19	27,0
1,722	3546	77	59	76,2
1,0	9834	158	187	211,3
0,513	11858	230	263	254,8

Table 6: Generated geometry and render time (displacement map 256×256)

Distance	Triangles	Our method render time [µsec]	Triangle array render time [µsec]	Polygon surface mesh estimated size [kB]
3,138	1202	54	20	25,8
1,722	3312	81	58	71,2
1,0	9386	163	185	201,7
0,513	11412	233	261	245,2

6 Acknowledgments

This work was supported by the Grant VEGA 1/0625/14.

References

[1] Osd tessellation shader interface. `http://graphics.pixar.com/opensubdiv/docs/osd_shader_interface.html`. Accessed: 2016-02-10.

[2] Michael Bunnell. Adaptive tessellation of subdivision surfaces with displacement mapping. *GPU Gems*, 2:109–122, 2005.

[3] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-aided design*, 10(6):350–355, 1978.

[4] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice (2Nd Ed.)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[5] Charles Loop and Scott Schaefer. Approximating catmull-clark subdivision surfaces with bicubic

Table 7: Generated geometry and render time (displacement map 512×512)

Distance	Triangles	Our method render time [µsec]	Triangle array render time [µsec]	Polygon surface mesh estimated size [kB]
3,138	1246	55	19	26,8
1,722	3462	84	60	74,4
1,0	9363	181	188	201,2
0,513	11464	254	264	246,3

patches. *ACM Transactions on Graphics (TOG)*, 27(1):8, 2008.

[6] Matthias Nießner. *Rendering subdivision surfaces using hardware tessellation*. Verlag Dr. Hut, 2013.

[7] Matthias Nießner and Charles Loop. Analytic displacement mapping using hardware tessellation. *ACM Transactions on Graphics (TOG)*, 32(3):26, 2013.

[8] Matthias Nießner, Charles Loop, Mark Meyer, and Tony Derose. Feature-adaptive gpu rendering of catmull-clark subdivision surfaces. *ACM Transactions on Graphics (TOG)*, 31(1):6, 2012.

[9] Henry Schäfer, Benjamin Keinert, Matthias Nießner, and Marc Stamminger. Local painting and deformation of meshes on the gpu. In *Computer Graphics Forum*, volume 34, pages 26–35. Wiley Online Library, 2015.

[10] P. Schröder, D. Zorin, T. DeRose, DR. Forsey, L. Kobbelt, M. Lounsbery, and J Peters. Subdivision for modeling and animation. *ACM SIGGRAPH Course Notes*, 12, 1998.

[11] Graham Sellers, Richard S. Wright, and Nicholas Haemel. *OpenGL SuperBible: Comprehensive Tutorial and Reference*. Addison-Wesley Professional, 7th edition, 2015.