Workshop on Virtual Reality Interaction and Physical Simulation VRIPHYS (2010)
J. Bender, K. Erleben, and M. Teschner (Editors)

A Geometry-Shader-Based Adaptive Mesh Refinement Scheme Using Semiuniform Quad/Triangle Patches and Warping

M. Knuth[1] , J. Kohlhammer[1] , A. Kuijper[1,2]

[1]Fraunhofer IGD, Germany
[2]TU Darmstadt, Germany

Abstract

In the field of garment simulation the resolution of the simulation mesh has a direct impact on visual quality. Unfortunately, an increase in mesh resolution introduces a much higher computational cost and potentially causes instability inside the simulation. In addition, it increases the amount of data sent to the renderer for visualisation. Therefore, a GPU-based refinement of the simulated mesh has several advantages, since all additional data is generated immediately before rendering. This allows an increase in visual quality without adding to computational costs for the simulation process or bandwidth necessary for rendering.

In this paper we present a view-dependent, adaptive tessellation method designed for the geometry processing stage of modern GPUs. It uses uniform meshes internally, removing the necessity to store external patches. Since we deal with a local refinement scheme, sudden changes in the mesh structure size on adjacent patches may occur incidentally. To reduce this effect as far as possible, we control the triangle density distribution of the refinement process inside a refined triangle patch.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

Nowadays, the garment industry's design and prototyping process is increasingly performed virtually with the help of computers. In this workflow the garment designers rely on real-time garment simulation in order to get fast feedback on changes they intend to apply. This is a very dynamic process and a challenge for the simulation and visualisation system used. Often a trade off between computational cost and simulation mesh resolution has to be made to achieve a good interactivity. Reducing the simulation mesh resolution additionally has the side effect of reducing the amount of data necessary to update the geometry inside the rendering part of the interactive system.

With the existence of geometry processing stages in modern GPUs an interesting alternative solution for increasing the visual quality of the rendered simulation data exists. A geometry-shader-based tessellation allows the inclusion of a single-pass refinement inside the existing rendering system with a minimum of changes. For this purpose we present a GPU-based method intended to be run *inside* its programmable geometry shader. Our proposed algorithm has three design goals:

i) The algorithm has to fit into an existing system, working with the existing vertex/normal-based triangle sets.

ii) The simulation can lead to suboptimal (thin) triangles which degrade or have large differences within their edge length. Thus, our algorithm has to refine these triangles without adding to much geometry, ideally making the inner triangles more homogeneous.

iii) For a fine refinement control the algorithm has to allow arbitrary odd and even vertex counts on the refined patch borders.

In addition to an edge-based refinement level we change the triangle density over the patch. This feature uses a per vertex interpolation and allows a better view-dependent silhouette refinement when dealing with large triangles with high curvatures.

DOI: 10.2312/PE/vriphys/vriphys10/021-029

Our main contribution is therefore an *adaptive* tessellation method using a combination of *uniform* triangle tessellation with a *semi-uniform* quadrangle tessellation method. In contrast to the inspiring method presented in [DRS09] this combination allows a better handling of thin triangles, since a uniform quad patch can natively handle different refinement levels in the horizontal and vertical directions. This way, our presented solution allows a smooth transition of refinement levels inside the triangle patch (See Figure 1). Additionally, our scheme is not limited to dyadic (power of two) edge refinement. This allows us a linear increase of the refinement level on the edges of the patch, similar to the refinement described in [VPBM01].

Our second contribution allows *control* of the triangle density distribution inside the patch. It improves silhouette outline refinement on large base triangles. The presented method is based on warping the coordinates necessary for patch surface evaluation. Thus it is independent of the refinement process itself and can be used in conjunction with recently introduced GPU hardware tessellation methods.

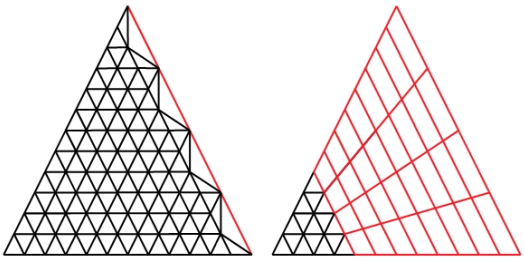

Figure 1: *Hybrid refinement: an adaptive semi uniform refinement scheme can lead to sudden resolution changes on the edge with the lower resolution level (Left). To distribute the resolution changes we combine a quad and a triangle based refinement process (Right). In this particular case the split allows us to solve the resolution change with uniform meshes.*

In the next section we refer to related work centred on GPU-based adaptive local mesh refinement techniques. This is followed by the presentation of our method. We then describe our implementation and the results we obtained using our approach. We conclude with a discussion of these results and possible enhancements of our method.

2. Background

In this section we discuss techniques related to GPU-based mesh refinement. We focus especially on local refinement techniques. Local refinement is well suited for shader-based approaches since it only operates with data provided by a single patch of the control mesh.

A refinement process can be roughly divided into 3 stages. First, a patch setup is performed. The patches describe the surface curvature. They are controlled by the input mesh, which is to be refined. Second, a tessellation logic is required, which divides the basic input mesh into a finer representation, creating interpolation coordinates. The third stage uses these coordinates in conjunction with the patch data to compute the positions of the new vertices. In the newest Rendering APIs of DirectX 11 and OpenGL 4.0 there exist special shaders, which directly resemble these stages (See Figure 2). Both APIs implement a similar mechanism. A description of this process inside OpenGL can be found in the specification of OpenGl 4.0 [www10] The new stages allow a direct hardware tessellation. It resides between the vertex processing unit and the triangle rasteriser. To allow a high flexibility, the stages need a new primitive type. This type is patch-based and allows a definable number of vertex indices per patch. The tessellation logic itself is a configurable fixed-function stage. As an alternative approach, the geometry shader stage allows us to create a tiny tessellation engine, which can be adapted directly to an existing rendering system with only few modifications. In addition this shader type is more common, since it was already introduced in the previous shader model [Bly06]. With the step from DX10 class to DX11 class GPUs the Geometry shader stage was improved. In addition to speed improvements, the output buffer size was increased from 512 output elements to 32k output elements. This alone allows roughly 64 times more complex geometry. Thus, hardware tessellation of a DX11 class GPU is not necessarily superior to a geometry-shader-based approach (Concheiro et al. [RMM10]).

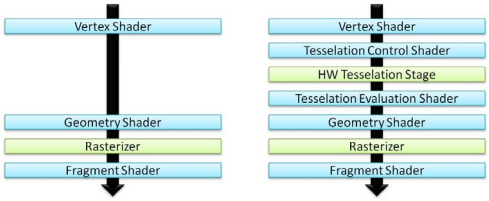

Figure 2: *DX10/OpenGL3.2 (left) versus DX11/OpenGL4.0 (right): With DX11 3 new Shading pipelines are available. Tessellation can now be achieved within the 3 new stages or by the geometry shader. Green blocks mark fixed function stages, while blue blocks are freely programmable.*

In [LSNC09] Loop et al. describe a technique for approximating subdivision surfaces with Gregory patches. Their technique allows the approximation of Catmull-Clark subdivision surfaces directly on the GPU. Additionally they give an overview of the new hardware tessellation unit in DX11 class graphics hardware.

2.1. Triangular Patches

There exist several methods regarding smooth surface patches usable for local triangle refinement. The patches

represent the curvature described by the control or input mesh sent to the refinement process. Unfortunately, positions and normals are the only curvature data source available for patch setup in our situation.

A fast technique allowing a patch setup under these conditions is Phong tessellation [BA08]. It is a technique, which uses a triangular patch of second order in conjunction with Phong-shading to create a smooth surface. To work properly it needs an adequately tessellated base mesh, since it cannot handle infliction points.

Without inflexion limitation, triangular patches of cubic degree can be used to create smooth surfaces. A first approach for using the triangle mesh data directly as control mesh is called Curved PN Triangles and is presented in [VPBM01]. To provide a control for creases by additional vertex data Boubekeur et al. [BRS05] extend this technique. The methods rely on a cubic triangular Bezier patch. Since the centre control point of the Bezier patch is shared by all 3 edges of the patch, geometric continuity of degree one (G1) can only be reached at the vertices of the patch. A technique promising a G1 continuity along patch borders is presented in [FMHF08]. It combines several triangular patches of cubic degree to grant continuity on the edges and blends them by a rational function to build the final surface. Unfortunately, this inspiring technique needs data from neighbouring triangles for parameter computations. There is a representation for such adjacency information available for the GPU's geometry shader, but it replaces the given index set of the base geometry.

2.2. Tessellation and Adaptive Refinement

Early GPU-based refinement approaches were limited to the vertex and fragment shader. The GPUs used did not contain stages for direct mesh manipulation. Thus, it was necessary to encode the new geometry data in textures to be able to process it. A theoretical framework for GPU based refinement is presented by Shiue et al. in [SGP03]. A list of approaches utilising the GPU as a general purpose processing unit for refinement can be found in the work of Boubekeur and Schlick [BS05].

A first approach on tessellation inside consumer graphics hardware was introduced in 2001 by ATI inc.The implemented technique internally uses Curved PN Triangles. The achieved refinement has a uniform topology and allowed a linear increment of the number of edge vertices. A uniform refinement is quick to compute. On the other hand, it does not allow changes of the refinement level within the same geometry without topological inconsistencies.

Nowadays there exist mechanisms, which simplify the process chain for adaptive mesh refinement on the GPU. The data necessary to represent smooth transition topologies between the different refinement levels for adaptive triangle refinement can be pre-computed. These topologies can be stored inside a lookup table sent to the GPU as geometry data. The Adaptive Refinement Kernel (ARK) technique presented by Boubekeur and Schlick in [BS08] uses this technique. Another similar method using adaptive topological patches called Dynamic Mesh Refinement is presented by [LD08]. The patterns describing the necessary topology have to fit to all possible combinations of refinement levels. Thus the number of topological patches can be very high. This problem is reduced in the work presented by Lenz et al. [LCNV09]. A permutation technique is used to reduce the overall number of patterns necessary by permutation of the barycentric coordinates used for describing positions on the patch surface.

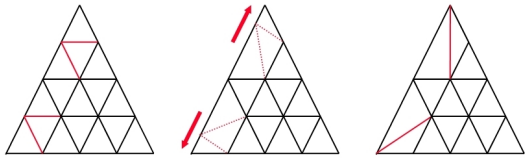

Figure 3: *Semiuniform tessellation of a triangle: We start with a uniform mesh (Left). Vertices of an edge with lower degree are snapped to valid positions (middle) until the desired lower resolution is reached (right).*

These methods rely on large topology catalogues in order to deal with the different combinations of refinement levels on the patch edges. The topological patches collected in these catalogues allow a high flexibility for choosing refinement patterns. Additionally, they can be computed in a preprocessing step. The drawback is that the patterns use a lot of memory space and have to be addressed. Thus, these algorithms usually perform several drawing passes until the final refined geometry is drawn.

An alternative to a catalogue of topological patterns is presented in [DRS09]. The technique is a topologically consistent extension of the work presented in [DRS08]. Their edge-based refinement technique uses a simple mesh topology based on dyadic uniform triangle meshes. First, a uniform mesh is chosen representing the highest refinement level provided on the edges. On edges with lower refinement level the vertices are snapped to lower refinement positions. This grants a watertight, topologically consistent patch. The drawback is that the snapping on the edge can create sudden resolution changes, since only the vertices in the edge are moved. (See Figure 3)

GPU tesselation is not only aimed at creating smoother meshes. It can be used to reduce the necessary bandwith of a 3D application in a system. This is especcially useful for mobile devices which have limited bandwidth resources. In [CYKK09] Chung et al. address this problem and present a shader-based solution for mobile phones. Ad-

ditionally, they discuss the differences between the ARK technique and the HW tesselation system of DX11 in detail. Semiuniform adaptive triangle tessellation is adaptable for

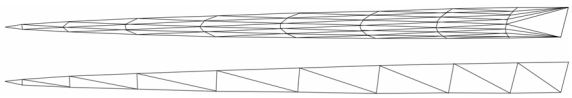

Figure 4: *Comparison between semiuniform triangle tessellation and our approach: Tessellation of a thin tetrahedron: Semi uniform triangle tessellation uses the highest edge refinement level for its internal structure, resulting in a high triangle count. Our scheme uses triangle based tessellation only on the tip and fills the reminder with a quad patch, which can be tessellated with different resolutions for the x and y axis. This results in a much lower polygon count in this example and triangles more adapted to the resolution on the edges.*

geometry shader implementations, too. But the limitation to uniform triangles as a basic refinement strategy leads to unnecessarily high polygon counts when dealing with thin triangles (See Figure 4). Additionally, the snapping algorithm can lead to sudden resolution changes on the patch border. Our presented refinement algorithm uses a quad- and a triangle refinement to circumvent these two problems. The triangle refinement uses a uniform mesh, while the quadrangular mesh is based on a semi adaptive approach.

3. Refinement based on Quad/Triangle Patches and Warping

In this section we present our novel hybrid adaptive refinement method we use inside the geometry processing stage of the GPU. In the GPU shader model of DX10 a new shader stage was introduced [Bly06]. The stage is located between vertex processing and triangle rasterisation. These geometry shaders take mesh primitives as input and create new primitives from them. The shader has limited resources, especially regarding the number of new vertices created from an input primitive. In order to minimize the number of vertices in our generated geometry we use triangle strips. Another limiting factor is the restricted accessibility of lookup data from the shader side. A catalogue of topological patches is not suitable under this restriction. Thus, we generate our topology on the fly. Since we combine different refinement strategies on parts of the patch surface we have to represent relative positions on the patch. We use barycentric coordinates as suggested in [LCNV09]. This allows an easy interoperability of the two different refinement strategies. Additionally, it allows us modify the triangle density on the patch as a post process after refinement and prior to the surface evaluation of the patch. This warping process is described in Section 3.2.

We first focus on the refinement stage.

3.1. Refinement Stage

This step consists of a rotation procedure, a split logic, which divides the input triangle patch into a triangular and a trapezoidal part and two mesh generators.

3.1.1. Rotation

We first rotate the triangle patch (TP) to have the edge with the lowest refinement level on its base. Thus, we always have the same starting condition. This is inspired by the work of Dyken [DRS09] which uses a permutation process to reduces the number of possible refinement states. After rotation, the quad patch is placed in the lower part, while the triangle part is located in the top part of the TP.

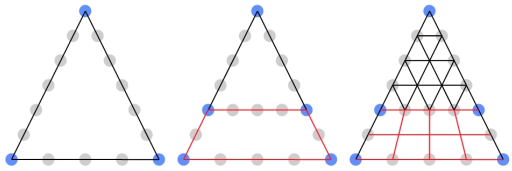

Figure 5: *Overview over our refinement process: Given a triangular patch and three refinement levels on its edges (Left), we perform a split between the two edges with the highest refinement level (Middle). The remaining triangle is filled with a uniform triangle mesh with a resolution equal to the patch's lowest resolution. The gap is filled by a trapezoidal Quad matching the remains of the surrounding edge resolutions (Right).*

3.1.2. Split

After rotation, we perform a split of the TP into a triangular part and a trapezoidal one. (See Figure 5) The triangular part is processed by a triangle mesh generator. The quad part is processed by a rectangular mesh generator. The split of the TP is performed on the two edges with the highest refinement level. This allows us to reduce the maximum refinement level used in the two refinement generators.

We start by defining the refinement level for the triangle mesh generator. Our quad mesh generator is only adaptive for the left and the right side. Thus the resolution of the TP's base is directly routed through the quad patch to the triangle, defining its resolution. The triangle mesh itself always has a uniform topology. Through the initial rotation the left and right edge of the TP have a resolution higher then or equal to its base edge. Thus, the remaining difference on the two edges has to be chosen as target resolution for the left and the right side of the quad mesh generator.

A problem arises in the case of having two equally low resolution edges and a high resolution edge. In this case large fan-like structures are generated. We circumvent this case by rotating the quad part by 90 degrees. This is possible, since

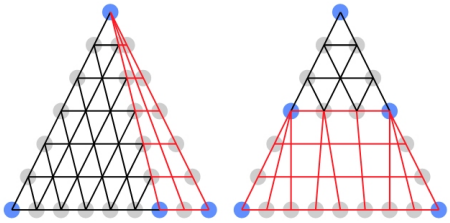

Figure 6: *Special case: Two equal low resolution edges and one high resolution edge can lead to awkward topologies. In this case we rotate the quad patch and place it on the high resolution edge. This allows the adaption of the high resolution edge to the other low resolution edges.*

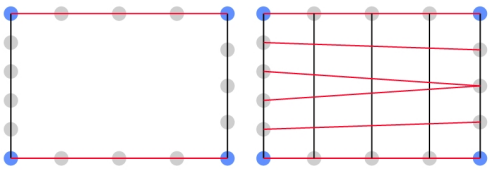

Figure 7: *Overview over our adaptive quad generation scheme: In order to match the different resolutions for a given patch, we create strips from left to right. Utilising a snapping algorithm, the middle strip's right side is pinched to a single vertex in order to match the provided resolution for the right edge. By using linear interpolation for coordinate generation the change of resolution is distributed over the quad.*

both sides of the quad have the same resolution. Now the resolution of the refinement side can be lowered toward the top triangle (See Figure 6)

3.1.3. Mesh Generators

Our approach uses two different types of mesh generators. The first one creates a uniform triangle mesh for a given refinement level. The second generates a rectangular tessellation which is semiuniform and uses a snapping mechanism inspired by the work presented by Dyken [DRS09]. Top and bottom of the rectangle share the same refinement level. In contrast the left and the right side allow different refinement levels (See Figure 7). The vertical resolution represents the maximum of both refinement levels. To create a watertight mesh a snapping process adapts the high resolution mesh to the lower refinement level. In contrast to [DRS09], we perform the vertex snapping only for the vertical axis of generated triangle strips from the quad refinement patch. Performing adaptation only on one axis simplifies the mesh generator. It allows the performance of snap operations in conjunction with linear interpolation for the vertical axis of the generator. The snapping process itself can be moved outside the inner loop of the tessellation procedure this way.

3.2. Barycentric Warping - Improving the View Dependency

The method described above already allows an adaptive tessellation of the input triangle patch. However, the method can only create a uniform distribution of vertices on an edge. Our edge based level of detail (LOD) is derived from a vertex-based LOD scheme. Thus, it is obvious to use the vertex-based data to derive a density distribution for the generated triangles.

To provide a vertex-based density control mechanism we perform a warping on the barycentric coordinates generated by the refinement stage. The warping itself is performed by a quadratic Bezier triangle patch. The triangular patch is a quadratic function:

$$f : R^3 \mapsto R^3, u+v+w \leq 1, u,v,w \geq 0 \quad (1)$$

$$f(u,v,w) = \sum_{i+j+k<2} C_{ijk} * u^i * v^j * w^k \quad (2)$$

$$f(u,v,w) = C_{200}u^2 + C_{020}v^2 + C_{002}w^2 \quad (3)$$

$$+C_{110}uv + C_{011}vw + C_{101}uw \quad (4)$$

It provides 6 control points. Three control points resemble the corners of a triangle (C_{200};C_{020};C_{002}). The remaining three represent points on the triangle edges (C_{110};C_{011};C_{101}).

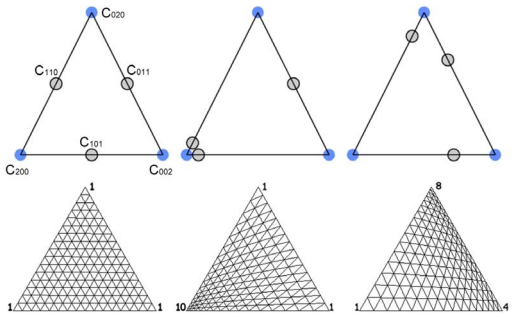

Figure 8: *Coordinate warping of barycentric coordinates generated from the refinement: Using a triangular Bezier patch of degree two allows us to change the density of the generated structures on the patch. We compute a level of detail (LOD) on each vertex. The proportion of the two lod levels of an edge is used to place the corresponding edge control vertex of the patch. Equal levels result in placing the control point on the centre of the edge. The mesh stays in its original state (Left). Increasing the level of a vertex results in moving the control points and the resulting density toward this vertex (middle and right)*

The corner control positions are initialised with their corresponding barycentric positions.

$$\vec{C_{200}} = (1,0,0) \quad (5)$$
$$\vec{C_{020}} = (0,1,0) \quad (6)$$
$$\vec{C_{002}} = (0,0,1) \quad (7)$$

The edge control points represent the relation of the vertex refinement level projected onto their common edge. Additionally, we have a set of level of detail values for the three corner points $(L_{100};L_{010};L_{001})$. We compute the 6 necessary weighting values $(W_{210};W_{120};W_{021};W_{012};W_{201};W_{102})$ as follows:

$$W_{210} = L_{100}/(L_{100}+L_{010}) \quad (8)$$
$$W_{120} = L_{010}/(L_{100}+L_{010}) \quad (9)$$
$$W_{210} = L_{100}/(L_{010}+L_{001}) \quad (10)$$
$$W_{120} = L_{010}/(L_{010}+L_{001}) \quad (11)$$
$$W_{201} = L_{100}/(L_{001}+L_{100}) \quad (12)$$
$$W_{102} = L_{001}/(L_{001}+L_{100}) \quad (13)$$

To compute an edge control point we linearly blend between their endpoints with the weights:

$$\vec{C_{110}} = \vec{C_{200}} * W_{210} + \vec{C_{020}} * W_{120} \quad (14)$$
$$\vec{C_{011}} = \vec{C_{020}} * W_{021} + \vec{C_{002}} * W_{012} \quad (15)$$
$$\vec{C_{101}} = \vec{C_{200}} * W_{201} + \vec{C_{002}} * W_{102} \quad (16)$$

This way the control points stay on the edge, always guaranteeing a straight triangular outline (See Figure 8). The result of this process is a mesh with a varying distribution of triangles (See Figure 9). In contrast to [LCNV09] we not only use barycentric coordinates to represent coordinates on the patch, but we manipulate them prior to their usage to get a better distribution of the generated Triangles.

4. Implementation

We implemented this method utilising the OpenGL Shading Language (GLSL) with the geometry shader extension within an OpenGL based rendering system. The process inside the shader is roughly divided in 3 stages (See Figure 10). In the setup phase we prepare data, which is consistent for the whole triangle patch. For surface smoothing we use the PN triangle patch approach presented by Vlachos in [VPBM01]. Additionally, a quadratic Bezier patch is set up to perform the coordinate warping post process. Initial splitting into the quad and the triangle refinement pattern is performed as well. The refinement stage hosts the two refinement processes and their setup. Finally, the quadratic patch

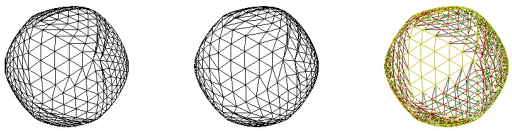

Figure 9: *Coordinate warping used on an icosahedron: The low polygon count of the initial mesh and its high curvature introduces high changes in the LOD level for the vertices. In the left image we see the refinemence without warping. The middle image shows refinement in conjunction with warping. The right image visualises the difference of both images. The result is a smoother transition of the structure size on patch borders.*

is evaluated to compute the final coordinates used in the surface evaluation by the Cubic PNTriangle Bezier patch.

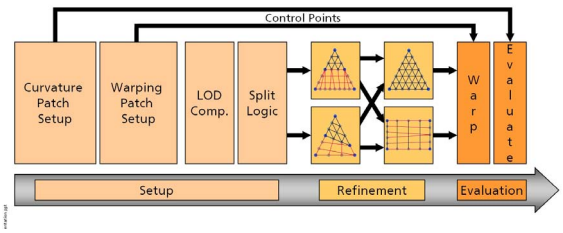

Figure 10: *Overview of our geometry shader: We start with the control coefficient computation for surface and warping patch. This is followed by view dependent refinement computations. Dependent on the refinement data, a split is chosen dividing the patch into a quad and a triangular area. After the refinement the generated coordinates are warped and then evaluated.*

4.1. View Dependency

Our presented refinement method is independent of the LOD computation mechanism used. However, for refinement testing purposes we have to choose one. We decided to combine a simple distance-based refinement mechanism and a silhouette refinement mechanism.

Distance based refinement aims to define different detail levels according to the distance of the object to the observer. Our computation is roughly inspired by the technique presented in Sander et al. [SM05]. The observer is a point in 3D space, which describes the camera position $O \in R^3$. The distance to a mesh vertex V is now simply the length of the vector from point O to point V. In order to get the current LOD for a distance we additionally define a min and a max distance, which define where to use the minimum and the maximum LOD. The distance is then scaled and normalised

to represent a refinement level of one at the max distance and the maximal refinement level at the min distance. Additionally, a logarithmic falloff between both is chosen to take into account the effects of perspective projection. On the other hand, the main intention behind silhouette refinement is to increase the detail on the surface's creasing angle. We use a simple approach which is roughly based on the technique described by Hoppe in [Hop97]. The factor is computed per vertex by the dot product of the normalised distance vector from O to V and the provided surface normal at the position of V.

5. Experimental results

We tested our concept on two different hardware architectures, both capable of performing geometry operations. An NVIDIA GTX 280 was chose as a representative of the shader model 4 league. Additionally, an AMD Radeon 5770 was used as a representative of the shader model 5 league. Besides differences in the computational power, the second GPU has a larger geometry shader output buffer for generated geometry data. This allows us to achieve higher tessellation levels in hardware.

We compared our solution against an implementation of dyadic semiuniform adaptive tessellation and PNTriangles. All tests were performed with geometry shader based implementations to create comparable results.

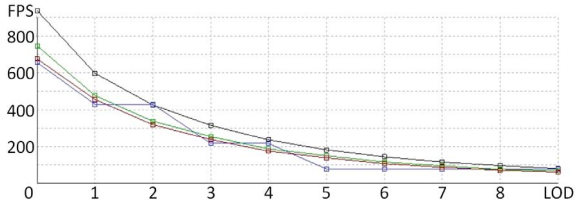

Figure 11: *Comparison: The black line represents a simple uniform refinement scheme like it is used by the PNTriangles scheme. The blue line represents a dyadic adaptive semiuniform tessellation scheme. The green line represents our method without warping applied. The red line shows the same experiment with warping enabled. Y axis: frames per second. X axis: refinement level. Since the dyadic sheme cannot represent all refinement levels it's curve has a stair like structure.*

We have tested and compared different refinement strategies and LOD computation methods. Additionally, we performed a worst-case scenario test for our scheme (See Figure 11). The test was performed on the NVIDIA system. In addition, we have compared the performance of the ATI and the NVIDIA-based systems (See Figure 12). As a reference we used a simple uniform refinement strategy (black line) which is, in practise, a geometry-shader-based implementation of the PNTriangle technique. For our method we tested two candidates. In number one (green line) the coordinate

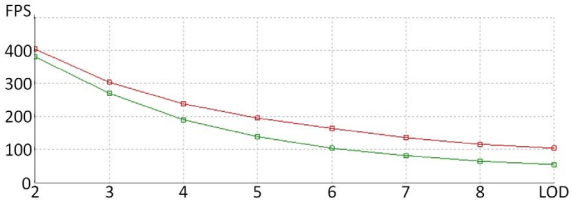

Figure 12: *Comparison of a DX11 class ATI 5770 (red line) to a DX10 class NVIDIA GTX 250 (green line). The ATI performs much better on the geometry shader tessellation; even though it is the less powerful GPU in other disciplines. A Geometry shader program performs much better on DX11 class hardware in comparison to DX10 class hardware. Thus, both GPUs perform similarly at low tessellation levels. At higher tessellation levels they are strongly separated.*

warping was disabled. In number two (red line) it was enabled. Additionally, both create the maximum triangles for a given refinement level using the longest path in the shader. This way a worst case scenario is created. The best case for our method is a uniform grid. Thus, in practise, performance ranges between the values of the red (green) and the black line.

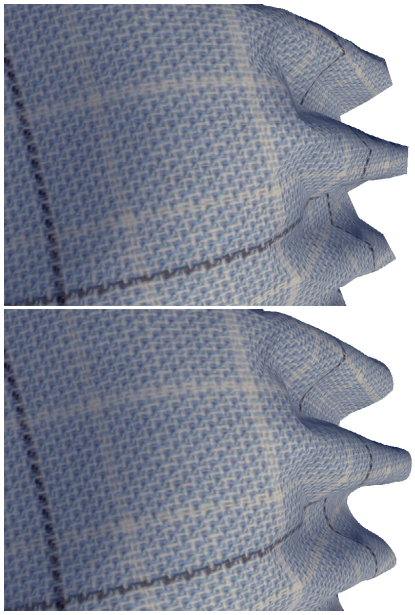

Figure 13: *Overhead view rendering of a tablecloth draped over a torus: In the upper Image we see the direct output of the cloth simulator. Since we want to perform real-time simulation the resolution of the simulation mesh is quite low. Using our tessellation method inside the rendering pipeline smoothes the appearance of the folds of the tablecloth noticeably (lower Image).*

In our tests the warping adds a loss of approximately nine percent of the frames per second to the overall process. As seen in Figure 13, 14 and 15, our proposed scheme grants a much smoother grid density control. Especially in the tablecloth and torus example the sagging of the tablecloth in its centre is clearly noticeable.

6. Conclusion

We have presented a GPU-based approach for adaptive mesh refinement. Our approach makes use of the GPU's geometry shader to perform the necessary geometry processing in real time within an existing rendering pipeline. In contrast to existing work we use a quad and a triangle-shaped refinement strategy to combine on the fly topology computation with LOD transition inside the patch. It allows creating reasonable tessellations on thin triangles by utilizing the quad mesh independency of vertical and horizontal resolution. The non dyadic tessellation scheme allows a finer control of the generated triangles as opposed to a dyadic scheme. This finer resolution step size results in additional smaller transitions of level changes on patch borders. To grant an additional smoothing of this border transition we have introduced a warping mechanism which manipulates the generated barycentric coordinates of the patch. This allows control of the density of generated triangles inside the patch. The method presented is intended to be used for silhouette refinement of a garment's geometry. Especially the creasing angle of folds and curvatures reveal rough triangular structures, which have to be smoothed.

The geometry-shader-based approach has the advantage of allowing full control over the complete refinement process. However, the size of generated refinement meshes is limited, which is not a problematic aspect in our use case. The garment mesh already has a good resolution and our goal is to smooth the simulator's output mesh. DX11 Hardware tessellation, on the other hand, allows a smooth transition of refinement levels. However, its tessellation method seems to introduce two new vertices per refinement step. Our method has a finer granularity in this case, but is limited to fixed LOD transitions. Future work can address this issue. Currently our approach uses a simple split algorithm which could be extended to feature recursive splits. Allowing a much finer control over the generated mesh structure, it would be a nice tradeoff between accuracy and speed. Replacing the uniform triangle mesh generator in our approach with a semiuniform one would increase the number of possible splits of the split logic. This can result in better refinement meshes and is a topic of future research.

References

[BA08] BOUBEKEUR T., ALEXA M.: Phong tessellation. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers* (2008), pp. 1–5.

[Bly06] BLYTHE D.: The direct3d 10 system. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers* (New York, NY, USA, 2006), ACM, pp. 724–734.

[BRS05] BOUBEKEUR T., REUTER P., SCHLICK C.: Scalar tagged pn triangles. In *EUROGRAPHICS 2005 (Short Papers)* (2005), Eurographics.

[BS05] BOUBEKEUR T., SCHLICK C.: Generic mesh refinement on gpu. In *HWWS '05: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (2005), ACM, pp. 99–104.

[BS08] BOUBEKEUR T., SCHLICK C.: A flexible kernel for adaptive mesh refinement on gpu. *Computer Graphics Forum 27*, 1 (2008), 102–114.

[CYKK09] CHUNG K., YU C.-H., KIM D., KIM L.-S.: Technical section: Shader-based tessellation to save memory bandwidth in a mobile multimedia processor. *Comput. Graph. 33*, 5 (2009), 625–637.

[DRS08] DYKEN C., REIMERS M., SELAND J.: Real-time gpu silhouette refinement using adaptively blended bézier patches. *Comput. Graph. Forum 27*, 1 (2008), 1–12.

[DRS09] DYKEN C., REINERS M., SELAND J.: Semi-uniform adaptive patch tessellation. *Computer Graphics Forum 28(8)*, 8 (2009), 2255–2263.

[FMHF08] FÜNFZIG C., MÜLLER K., HANSFORD D., FARIN G.: Png1 triangles for tangent plane continuous surfaces on the gpu. In *GI '08: Proceedings of graphics interface 2008* (2008), pp. 219–226.

[Hop97] HOPPE H.: View-dependent refinement of progressive meshes. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1997), ACM Press/Addison-Wesley Publishing Co., pp. 189–198.

[LCNV09] LENZ R., CAVALCANTE-NETO J. B., VIDAL C. A.: Optimized pattern-based adaptive mesh refinement using gpu. In *SIBGRAPI '09: Proceedings of the 2009 XXII Brazilian Symposium on Computer Graphics and Image Processing* (2009), IEEE Computer Society, pp. 88–95.

[LD08] LORENZ H., DÖLLER J.: Dynamic mesh refinement on gpu using geometry shaders. In *WSCG'2008 Full Papers Conference Proceedings* (2008), pp. 97–104.

[LSNC09] LOOP C., SCHAEFER S., NI T., CASTA NO I.: Approximating subdivision surfaces with gregory patches for hardware tessellation. In *SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers* (New York, NY, USA, 2009), ACM, pp. 1–9.

[RMM10] R.CONCHEIRO, M.AMOR, M.BÓO.: Synthesis of bezier surfaces on the gpu. In *Proceedings of the International Conference on Computer Graphics Theory and Applications (GRAPP 2010)* (2010).

[SGP03] SHIUE L.-J., GOEL V., PETERS J.: Mesh mutation in programmable graphics hardware. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (2003), pp. 15–24.

[SM05] SANDER P. V., MITCHELL J. L.: Progressive buffers: view-dependent geometry and texture lod rendering. In *SGP '05: Proceedings of the third Eurographics symposium on Geometry processing* (Aire-la-Ville, Switzerland, Switzerland, 2005), Eurographics Association, p. 129.

[VPBM01] VLACHOS A., PETERS J., BOYD C., MITCHELL J. L.: Curved pn triangles. In *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics* (2001), pp. 159–166.

[www10] WWW.OPENGL.ORG: Opengl 4.0 specification.

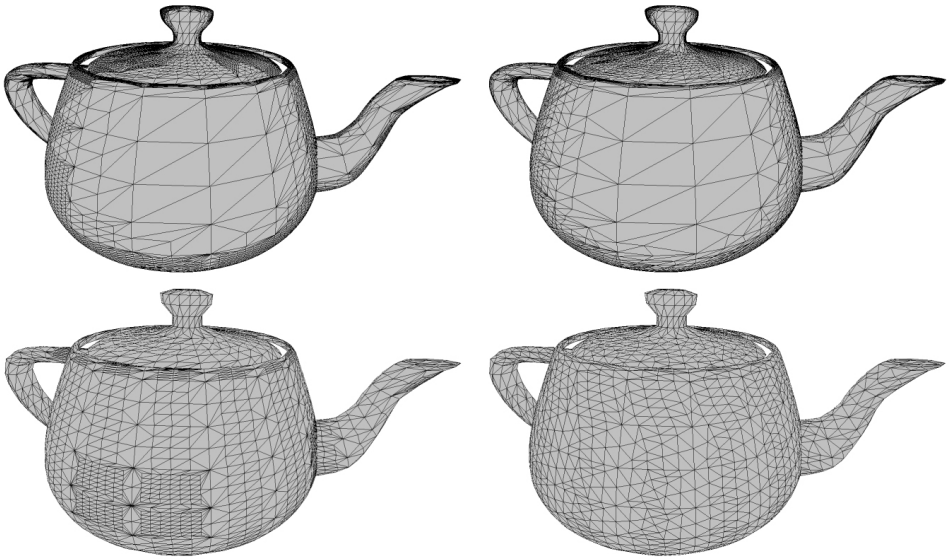

Figure 14: *Teapot example: the left side was created using dyadic adaptive semi uniform tessellation. The right side used our algorithm. Two different metrics for LOD computations were used: In the upper row we used a distance-and-angle-based scheme. In the lower row the LOD is chosen to reach a maximal edge length. The 3D teapot model used has its longest triangle edges on the side surfaces. Thus, a maximum resolution is reached here which is slightly over the trigger level of the next dyadic refinement level. This results in the sudden increase of resolution visible in the bottom left image.*

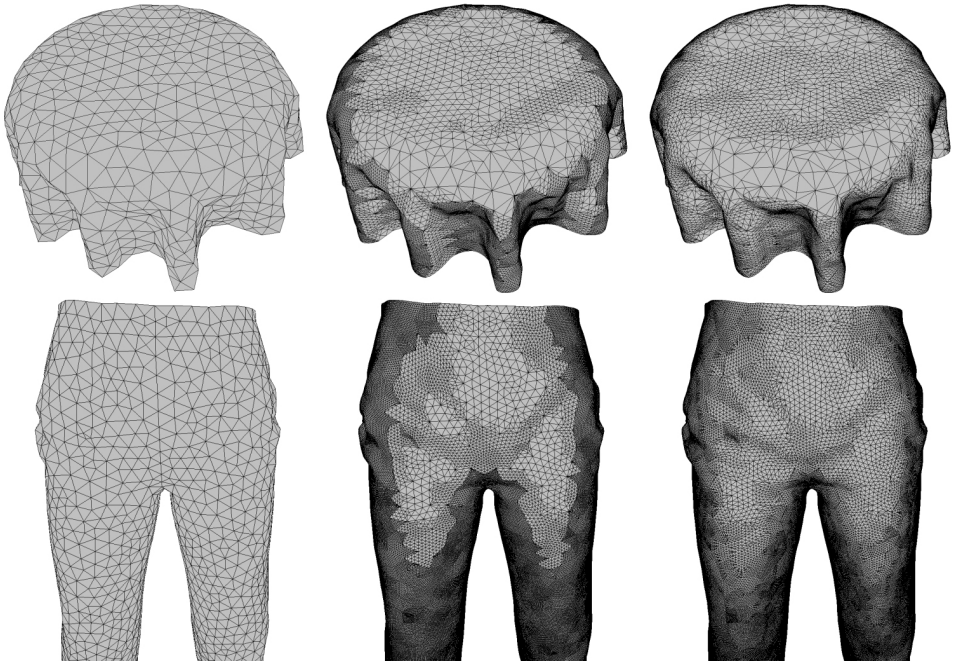

Figure 15: *Cloth example: Tablecloth on a torus (top row) and a pair of trousers (bottom row). The left side shows the original mesh. In the centre column we applied dyadic adaptive semiuniform tessellation. On the right side our algorithm was used. Using a non dyadic scheme a finer granularity of the possible refinement steps is applied. In conjunction with the proposed warping scheme a much smoother appearance of the refined mesh is created.*