Adaptive terrain rendering with smooth subdivision surfaces on the GPU

Bachelor thesis

Felix Heide

Computer Graphics and Multimedia Systems Group
Faculty 12: Electrical Engineering and Computer Science
University of Siegen

Supervisor: Prof. Dr. Andreas Kolb
Second supervisor: Dr.-Ing. Christof Rezk Salama
Advisor: Dipl. Math. Martin Lambers

6th July 2009

Eidesstaatliche Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht. Ich bin mir bewusst, dass eine unwahre Erklärung rechtliche Folgen haben kann.

Ort, Datum Vorname, Name

Abstract

This bachelor thesis presents two new methods for effective parallel terrain rendering.

First, a new parallel adaptive terrain rendering method on the GPU, named "Parallel adaptive refinement" (PAR), is presented. The method generates view-dependent meshes consisting entirely of right-isosceles triangles. In a top-down refinement, a coarse input mesh is refined directly in the rendering pipeline until a freely adjustable upper bound on the screen-space error is no longer exceeded. The algorithm produces guaranteed error bounds and does not imply any prerequisites on the error metric or the terrain elevation data.

Second, a new adaptive interpolatory smooth subdivision scheme, named "4pt-hermite scheme", is developed. The positive feature of subdivision surfaces, namely that they naturally support infinite continuous levels-of-details due to their recursive structure and deliver a visually appealing smooth surface, is exploited for terrain rendering on the GPU. The scheme has a local triangular support and can thus be applied fully parallel. It does not produce degenerate or near-degenerate triangles.

The two methods can be used as standalone algorithms. Plugging them together results in a fast CPU-independent terrain rendering algorithm which provides the visualization of terrain elevation data at a freely to define rate of accuracy while still enabling an infinite continuous range of levels-of-detail combined with a visually appealing surface.

Contents

1 Introduction

In recent years a lot of research has been done on the interactive visualization of massive terrain data. This is because of many emerging uses of terrain rendering algorithms, ranging from aerospace industry and scientific applications to education and entertainment purposes. These applications call for the detailed rendering of terrain data at interactive frame rates. With the rapidly improving remote sensor technology, the resolution and hence size of terrain data sets increases enormously. So, the challenging task in the field of terrain rendering is to display as much detail as possible of a visible section of a large data set while still enabling interactivity. A central concept to address this task (beneath exploiting all available new hardware capabilities) is to efficiently calculate the coarsest geometry representing the terrain that satisfies a perceptual image quality - the idea behind the level-of-detail (LOD) methods. Smooth subdivision surfaces naturally support infinite continuous-level-of-detail rendering (CLOD) because of their recursive structure. Additionally, the visual appeal of the terrain is improved for free, using that technique in the context of terrain rendering. So, plugging in smooth subdivision surfaces at the end of existing terrain rendering algorithms seems to be a natural conclusion.

At the beginning of this thesis, in Sec. 2, a requirements analysis is presented. Starting with a definition of the problem addressed in terrain rendering, the focal requirements on terrain rendering algorithms are examined. This requirements analysis represents the starting point for the design of the new methods and delivers criteria for evaluating existing algorithms.

Sec. 3 presents a new parallel adaptive terrain rendering method on the GPU, named "Parallel adaptive refinement" (PAR). The method generates view-dependent meshes consisting entirely of right-isosceles triangles. The requirements analysis is exploited to define features of the new method. After the concepts of the new approach were presented in detail, problems of the method are discussed and a convergence analysis is done.

Sec. 4 begins with an analysis on which existing smooth surface subdivision methods can be optimally integrated into the parallel processing environment on the GPU. Based on that analysis a new interpolatory smooth subdivision scheme, named "4pt-hermite smooth subdivision", is developed. After the concepts were described in detail, the integration into the PAR method and a convergence analysis are presented.

Sec. 5 considers the practical results of the complete terrain rendering algorithm presented in this thesis - consisting of the PAR algorithm followed by the 4pt-hermite subdivision. Advisable parameter values and performance measurements with a sample implementation are presented.

Finally, in each conclusion at the end of the descriptions for both methods, it is evaluated to what extent the respective requirements are fulfilled. In Sec. 6, an overall conclusion and the presentation of future work, which could follow this thesis, round off this thesis.

2 Top requirements on terrain rendering algorithms

Starting with a definition of the problem addressed by terrain rendering, this section considers the central requirements on a new terrain rendering method. This is necessary before beginning with the development of a new method because of two reasons: First, the examined top requirements defined in this section guide the design process. These indicate what the optimal goal for a terrain rendering algorithm is. Second, the requirements can also be understood as criteria on which a terrain rendering method can be evaluated. Problems in existing or new developed algorithms are marked and hence it is clearly shown where additional work should be invested.

A starting point for the requirements analysis could be a survey on the research in terrain rendering in the recent years. A survey on selected terrain rendering algorithms is presented by Schmiade in [5]. An overview of categorized papers in this subject from the early 1990s up to date can be found on `http://www.vterrain.org/` [6]. As summarizing these algorithms would not let enough space left to describe the new methods in this thesis, only the results of my survey are presented in the form of this requirements analysis.

2.1 Motivation and goals

When considering focal requirements, a central question is: What is the problem addressed by terrain rendering ?

Terrain rendering algorithms shall solve the problem of synthesizing an exact image of a given terrain data set from an arbitrary point of view (POV) with minimal memory access and minimal calculation time - that means to display as much detail as possible at a given time for rendering.

This defines the central features of an optimal terrain rendering method:

1. Accuracy: To visualize the terrain data as exact as possible can be regarded as the most central task for an optimal terrain rendering algorithm.

2. Optimal mesh topology: The topology of the generated mesh has great impact on exactness, visual appeal and speed of a terrain rendering.

3. Parallel approach: To enhance performance terrain rendering should be approached as a parallel computing problem.

4. Small memory footprint: Only the fastest available memory should be accessed for exclusively the data which is needed.

These four general design criteria are presented more concretely in the following sections.

2.2 Accuracy

In a terrain rendering algorithm an exact image of the input data, given in the form of a heightmap, has to be generated. The exactness should be measured in dimensions of the synthesized image - called the screen-space error. The screen-space error is defined as the distance between the middle of a first pixel on the rendered image and the position on this image where the first pixel should be if the exact terrain surface defined by the heightmap would be projected on the image plane.

As it depends on the application how exact the image of the rendered terrain image is required to be, the allowed screen-space error should be freely adjustable. The accuracy of a mesh generated by a terrain rendering algorithm can be measured by the mesh's upper bound on its screen-space error, the method can solidly guarantee. An optimal terrain rendering method can guarantee an upper bound towards 0.

A lack of accuracy does not only influence the quality of the approximation of the heightmap elevation data - it also might cause visual artifacts. Meshes generated from the same section of a terrain surface with a large upper bound on the screen-space error are allowed to differ from the heightmap data and thus from each other to a great extent. Therefore, viewing the same terrain section from a changing POV can reveal these differences as so-called "popping". This emphasizes the claim for a guaranteed upper screen-space error bound near to zero.

Lindstrom and Pascucci in [3] as well as others simplify the calculation of the screen-space error with heuristics of the projection transformation. To formulate founded statements about the quality of a rendered image, this error should be measured exactly using the model, viewing and projection transformations.

2.3 Optimal mesh topology

Considering the topology of the mesh generated by an optimal terrain rendering algorithm, several goals can be formulated.

The generated mesh should not contain any gaps. These can result from T-junctions or missing primitives. First, gaps cause rendering artifacts, as objects behind the gap or the background color is drawn in the section of the gap. Second, in the rendered image the information of the exact terrain elevation data set is completely lost at a gap. So, gaps in a terrain surface influence the exactness and the visual appeal of the displayed image to a great extent.

The mesh outputted by an optimal terrain rendering method should not contain degenerate or near-degenerate triangles. A degenerate triangle is a triangle whose three points are collinear. A near-degenerate triangle is then a triangle whose three points are nearly collinear. The area of a degenerate triangle is zero, as its height equals zero. This non-existing area often causes rendering artifacts: The rasterizer does not produce any fragments for degen-

erate triangles. Near-degenerate triangles result in small clusters of fragments. But as the rasterizer can have only a limited resolution, it is possible that these clusters can be associated with several adjacent degenerate triangles - hence causing flickering artifacts at such triangles.

2.4 Parallel approach

As the minimization of the calculation time is a primary goal for an optimal terrain rendering algorithm, it is necessary to distribute the calculations over as many processing units as possible. The full acceleration of a parallel approach to a specific problem is only unleashed, if the calculation on the processing units are completely independent of each other. A processing unit waiting for the result of another processing unit, will produce a latency period for the complete solution of the problem.

For that reason, an optimal terrain rendering algorithm should be computable fully parallel. Current graphics hardware delivers with the appearance of the stream-processor the environment for the fastest parallel floating point calculations available also on consumer-hardware. Therefore, the graphics card can be regarded as the optimal runtime environment for a fast terrain rendering algorithm (this statement takes only effect for the hardware available till the publishing date of this thesis). This conclusion has a strong impact on the concept of a fast terrain rendering as the flexibility of the runtime environment affects the freedom of design to a great extent.

2.5 Small memory footprint

Because of the high costs of low-latency memory (graphics memory or system memory) and the rapidly improving remote sensor technology, a majority of terrain data sets do not fit into system memory. This becomes obvious with considering current satellite technology, which can record various data of the earth at a resolution of 1-2 cm (ESAs GOCE Mission [4]). Data sets of similar sizes are not manageable as a single portion. In addition, it is not likely that in the future, fast system memory will be cheap enough to host the steadily exploding amounts of data recorded with continuously improving sensors.

That is why a hierarchical data management should be integrated (or possible to be integrated) in an optimal terrain rendering algorithm. The domain in which the height information is stored in the heightmap is 2D. Therefore, well-researched quadtree-like 2D data management structures are a suitable choice.
Assuming such a quadtree of heightmaps (see Fig. 1), a terrain rendering algorithm which handles only one heightmap texture in a fast memory location can be easily extended to use hierarchical data management:

Before generating the mesh on the GPU, in a first step on the CPU it can be calculated which knots of the quadtree are needed to render the section of the terrain surface visible from the current POV with the desired resolution. These nodes are a layer of the red subtree in Fig. 1 (for the best resolution, the

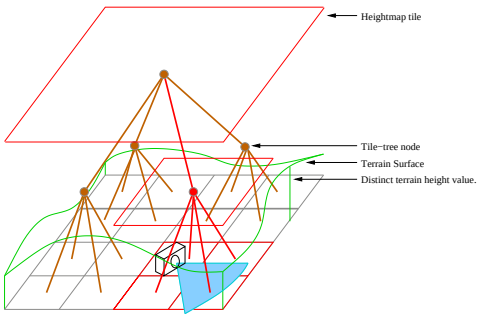

Figure 1: A heightmap quadtree. Each level of the quadtree delivers a doubled resolution of the data.

leaves of the subtree). These textures can then be put in a texture array and transferred to the graphics memory, where the terrain rendering algorithm can access the data. The algorithm can be run for each texture in the texture array independently. Special attention has to be given to the borders of a texture, in order to guarantee gap-free mesh transitions between the meshes generated for each texture. One solution would be to subdivide to the maximal refinement level at the borders.

In consequence, the algorithms presented in this thesis do not make any further considerations about hierarchical data. They are designed using one heightmap, which can be stored entirely in the graphics memory.

Beneath the hierarchical data approach, to enhance performance, there are two criteria under which memory usage has to be carefully considered in terrain rendering:

- Memory transfers: Memory transfers should be done entirely within the fastest accessable memory from the processing unit executing the terrain rendering algorithm - this is the graphics memory. Any transfer over busses will result in a latency and should therefore be avoided.

- Memory accesses: Only the graphics memory should be used for memory accesses. As little as possible height values should be retrieved from memory. This is because even a texture lookup in the fast graphics memory is expensive when interactivity is aimed for. Especially duplicate memory accesses for the same data should be eliminated.

3 Parallel adaptive refinement

This section presents a new parallel adaptive terrain rendering algorithm running entirely on the GPU, named PAR for "Parallel adaptive refinement". In a top-down refinement, a coarse input mesh is subdivided by inserting new vertices with elevation retrieved from a heightmap. The refinement is done recursively directly in the rendering pipeline using the geometry shader stage. It is repeated until a freely adjustable upper bound on the screen-space error between the exact terrain surface and the generated mesh is no longer exceeded. The algorithm produces guaranteed screen-space error bounds and does not imply any prerequisites on the error metric or the heightmap.

At first, the requirements analysis of the preceding section is exploited to define the motivation and goals for the new terrain rendering method. After that, the concepts of the new approach are outlined. Especially the central ideas in the tesselation of the mesh are depicted in detail. In a convergence analysis it will be investigated if the algorithm terminates. Finally, it is described what benefits for the new method result from culling techniques and which problems occur with the new approach.

3.1 Motivation and goals

In the preceding requirements analysis, the requirements accuracy, optimal mesh topology, parallel approach and small memory footprint for an optimal terrain rendering algorithm were presented. The consideration of these requirements as design criteria clearly marks the three central features the new method should possess:

Features aimed for in the new method

First, because of the desired exactness of the rendered image of a terrain, the method should guarantee an upper bound on the screen-space error. This bound should be freely adjustable to any value greater or equal to zero.

Second, the mesh generated by the new method should fulfill the following qualities. It should contain only right-isosceles triangles, as then no degenerate triangles can be generated. This property is discussed in detail in the description of the tesselation stage of the new method. The generated mesh should also contain no cracks.

Third, the new terrain rendering algorithm should run entirely on the GPU. It should be executed fully parallel on the stream-processors on current graphics hardware. The memory used by the algorithm - to store the heightmap, temporary data and the generated mesh - should be located entirely on the graphics board.

Finally, no restrictions on the input data should be made to achieve the features above.

3.2 Concept

An algorithm with the features outlined above shall be designed. A starting point for the design is existing research on terrain rendering algorithms, which achieve a large part of these features:

Ulrich presents in [1] a triangle-based recursive refinement algorithm for a CPU runtime environment. A coarse input mesh is subdivided until the screen-space error between the mesh and the terrain elevation data measured at the triangles' edges falls below a user-defined threshold. The input mesh is right-isosceles and one recursion step the algorithm produces exclusively right-isosceles triangles from right-isosceles input triangles. Consequently, the meshes generated by Ulrich's algorithm consist only of right-isosceles triangles. To achieve this continuous right-isosceles mesh topology, Ulrich like Duchaineau et al. in [2] (and many other publications on terrain rendering) use a tree-like data structure to communicate splits recursively over sequences of adjacent triangles.

Ulrich's concept delivers a good entry point for designing a new terrain rendering algorithm as it possesses the demanded accuracy and mesh topology features. It is not parallel, runs on the CPU and uses exclusively system memory. The Nvidia SDK 10.5 Example "Transform Feedback Fractal" [8] demonstrates the use of the geometry shader stage to do a parallel subdivision of an input mesh directly in the graphics pipeline. Through the use of VBOs as a buffer storage for the generated mesh and Transform Feedback - a technique which allows to store primitives generated by the geometry shader stage in VBOs - this subdivision can be applied recursively. Schmiade [5] adopts this method to transfer a restricted version of Ulrich's concept on the GPU. The central problem pronounced in Schmiade's work is that a purely right-isosceles triangulation is dropped causing near-degenerate triangles.

The concept of the new PAR method utilizes the positive features of Ulrich's and Schmiade's methods as a starting point. The following paragraph presents an overview of the core concept:

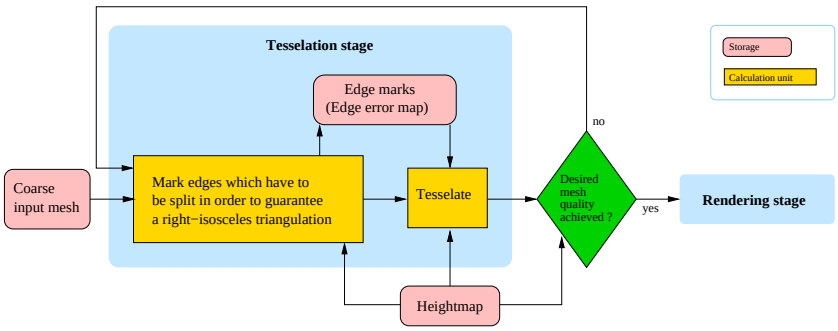

Figure 2: The core concept of the new method.

The algorithm is initiated on the GPU with the tesselation of a coarse input

mesh (see Fig. 2). A triangle of this mesh is subdivided into smaller right-isosceles triangles directly in the tesselation stage of the graphics pipeline if at least one of its edges exceeds a specified screen-space error threshold. The elevation of the inserted vertices is retrieved from the heightmap, thus creating a mesh which approximates the input data more exact. These new vertices are inserted at the triangle edges, resulting in an edge-based algorithm. The tesselation is applied repeatedly until every edge in the mesh falls under the screen-space error threshold and the desired mesh quality is reached. Finally, the mesh is rendered.

To tesselate a coarse input mesh exclusively with right-isosceles triangles fully parallel on the GPU, the tesselation stage is split into two parts: The idea is to force additional edge-splits in order to ensure a right-isosceles triangulation. In a first step, for all edges which shall be forced to be split later, marks are generated (see also Fig. 2). These marks are stored in a storage location freely accessible from every tesselation unit - a texture (referred to as the "edge error map" later on). In a second step, the mesh is tesselated. An edge is tesselated if it exceeds a specified screen-space error threshold or was marked in the previous step. As the edges necessary for a right-isosceles triangulation have been marked, only right-isosceles triangles are output.

In the following detailed description on the concept of the new method, only new inserted vertices and not normals are discussed. As an arbitrary normal sampling algorithm could be integrated into the presented solution, this is a completely orthogonal topic.

3.2.1 Error metric

The goal of the presented algorithm is to guarantee an user-defined accuracy of the generated mesh representing the terrain surface. Consequently, many parts of the calculations performed by the algorithm are dependent on accuracy. Therefore, the description of the new method begins with the error metric - which defines the measurement of this accuracy.

One of the outlined requirements was that the accuracy of the mesh generated by the new algorithm should be measured in dimensions of the synthesized terrain surface image - the screen-space error. The exact accuracy of a given mesh can only be retrieved if for every point lying on the mesh surface, its screen-space error would be calculated. This would mean an infinite amount of points. Therefore, the accuracy can only be estimated with a limited number of sampling locations. It is open at what exact sampling locations in the mesh this screen-space error should be measured:

In the new PAR method, the screen-space error is measured at the edges of the primitives in the mesh. This has the advantage that adjacent primitives receive the same screen-space error on their sharing edges. This property can be utilized to design an edge-based tesselation. The shared screen-space error value on the edges results in sharing tesselations, which guarantees a continuous mesh topology.

Considering a single edge whose screen-space error should be measured, the error calculations could be done for an arbitrary number of sampling locations, only limited by the hardware resources. With the number of disjunct sampling locations, the calculation time increases as well as the quality of the result. Therefore, this number is a compromise between performance and an exact accuracy measurement. In practice, Ulrich's and Duchaineau's works [1], [2] show that one error sampling location per edge delivers already pleasing results. Analogous to their methods, the presented algorithm in this thesis calculates the screen-space error of an edge at the locations where new vertices are inserted eventually in a tesselation step. This ensures that these new vertices are inserted exactly at the positions where the measured screen-space error in the synthesized image exceeds a given threshold.

The PAR method inserts one new vertex at the midpoint of an edge with elevation retrieved from the heightmap (which will be justified in the following Sec. 3.2.2). The distance δ between this new point and the middle of the edge is measured in object-space and then projected to screen-space. Be $h : \mathbb{R}^2 \mapsto \mathbb{R}$ a function that maps a vertex position in $\mathbb{R}^2$ to its corresponding height value from the heightmap, $ssc : \mathbb{R}^3 \mapsto \mathbb{R}^2$ a function that maps from object-space to screen-space and a, b the end vertices of an edge. Then the screen-space error $\delta_{sse}(a, b)$ for the edge $\overline{ab}$ is calculated as follows:

$$\begin{aligned} \delta(a,b) &= \left\| \frac{a+b}{2} - \begin{pmatrix} \frac{a_x+b_x}{2} \\ \frac{a_y+b_y}{2} \\ h\left(\frac{a_x+b_x}{2}, \frac{a_y+b_y}{2}\right) \end{pmatrix} \right\| = \left| \frac{a_z+b_z}{2} - h\left(\frac{a_x+b_x}{2}, \frac{a_y+b_y}{2}\right) \right| \\ \delta_{sse}(a,b) &= ssc\left(\delta(a,b)\right) \end{aligned} \tag{1}$$

3.2.2 Tesselation

The problem which shall be solved by the tesselation stage in the presented algorithm is the following: A given coarse mesh shall be subdivided into a new mesh with smaller primitives - containing new vertices with elevation retrieved from the heightmap - which thus approximates the heightmap to a higher extent. Because a continuous regular mesh topology shall be achieved, there are restrictions on a single subdivision step. The generated mesh is required to consist entirely of right-isosceles triangles and not to contain any gaps.

Reasons for a right-isosceles triangulation

Presenting the overview of the new method's concept, it was glossed over the justification why to demand a right-isosceles triangulation. There are several reasons for meshes consisting entirely of this type of triangle. The decisive ones for the algorithm presented in this thesis are:

- Non-degenerate triangles: Right-isosceles triangles are not degenerate and can only become near-degenerate, if the length of at least one edge is smaller than an $\epsilon \to 0$ with $\epsilon > 0$ (then all triangle points can be assumed to be nearly collinear). Consequently, a subdivision of a right-isosceles triangle only with right-isosceles triangles cannot introduce degenerate triangles.

- Lineary decreasing area: The area of right-isosceles triangles decreases linear with the length of their edges. Thus the speed and convergence of a recursive mesh refinement method are assumed to benefit in comparison to arbitrary shaped triangles.

- Compositing: Adjacent right-isosceles triangles can be easily composed to quads or rectangles, which are a common input primitive type for many algorithms which could be applied to the mesh after the execution of the presented algorithm.

Edge-based split operator

This paragraph describes the split operation, which is executed in a single refinement pass on the inputted mesh primitives. This split operator is edge-based. A given triangle is tesselated by splitting its edges using the screen-space error at an edge to decide whether it should be split or not. The measurement at the edges with the presented error metric had the advantage that sharing edges of adjacent triangles receive the same tesselation, thus ensuring a continuous tesselation.

It is open how many vertices with elevation retrieved from the heightmap should be inserted at which locations on an edge. The PAR algorithm inserts one new vertex at the midpoint of an edge. Inserting the new point at the middle and not elsewhere is a prerequisite for a right-isosceles tesselation. The central reason why only one vertex is inserted is simplicity. There are $2^3 = 8$ possible tesselations for a triangle which can be split at maximum one time at its edges. Already for two possible splits this number increases to $3^3 = 27$. The considerations ensuring a right-isosceles triangulation later in this thesis as well as the tesselation itself would get more complex and require more hardware resources with a greater number of inserted vertices.

Finally, the split operator can be formally defined: An edge $\overline{ab}$ is split into the two edges $\overline{am}$, $\overline{mb}$ where $m = \left(\frac{a_x+b_x}{2}, \frac{a_y+b_y}{2}, h\left(\frac{a_x+b_x}{2}, \frac{a_y+b_y}{2}\right)\right)$, if

$$\delta_{sse}(a, b) > \tau \quad \text{where} \quad \tau \in \mathbb{R} \quad \text{and} \quad \tau \geq 0 \tag{2}$$

and τ is an user-specified screen-space error threshold in pixels.

Ensuring a right-isosceles triangulation

To ensure a triangulation purely based on right-isosceles triangles, all possible triangulations with the edge-based split operator have to be considered.

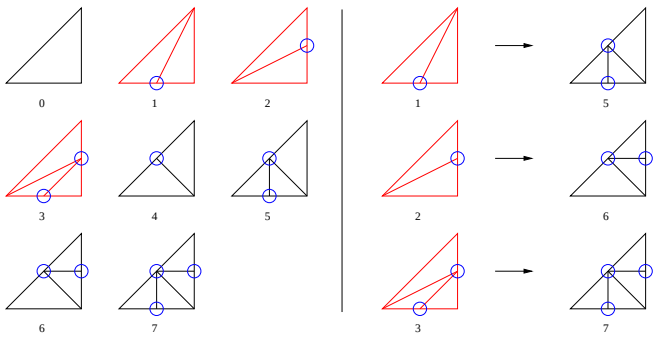

Figure 3: Left: all possible tesselations with the edge-based split operator. The blue circles mark the edges which have to be split. Right: Problem cases can be handled by forcing a split of the hypotenuse.

Fig. 3 depicts that in the cases 1, 2 and 3 the given triangle cannot be split into right-isosceles triangles. Without splitting additional edges only oblique triangles can be inserted. The idea is now to force an additional split of the hypotenuse in the problem cases. As illustrated in the right of Fig. 3, the three problem cases can then be handled as the regular cases 5, 6 and 7. Before applying the actual tesselation to an input mesh, the whole mesh - that means all triangles in parallel - is scanned for the problem cases. Every hypotenuse which has to be additionally split, is marked for a later split in the edge error map. The edge error map is described in detail in the following Sec. 3.2.3.

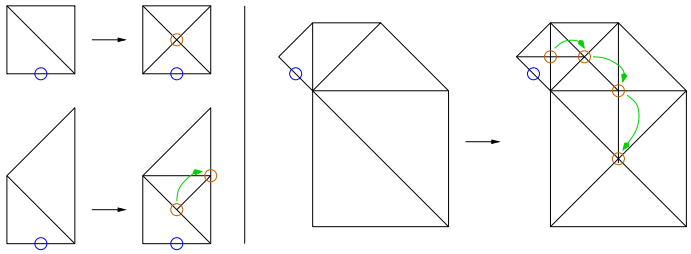

Figure 4: Forced splits can require further splits. These further splits are indicated by green arrows. Blue circles: Splits due to an exceeded screen-space error. Brown circles: Forced splits.

But one pass of searching for the problem cases 1,2 and 3 is not enough because forced splits can force further splits. Is a marked edge a hypotenuse for two adjacent triangles, this is not the case (see top left of Fig. 4). But if a marked edge is a hypotenuse and a cathetus for two adjacent triangles, the problem cases can be produced again (see bottom left of Fig. 4). As this constellation can appear repeated in a mesh, recursive sequences of further splits can occur. The right side of Fig. 4 illustrates a recursive split sequence

of 4 hypotenuses. For that reason the edge marks have to be successively increased in a sequence of parallel marking passes, adding further forced splits, until no new edge marks are generated. This recursive process terminates, as the number of triangles in the input mesh is limited.

Altogether, the process of ensuring a right-isosceles triangulation can be done directly in the graphics pipeline using the geometry shader stage. If we assume the edge error map, storing the marks, to initially contain no edge marks, the following listing gives a pseudo-code version of the geometry shader program for one pass of generating the edge marks from a given input mesh.

```
void main( Triangle T, EdgeErrorMap eem, float tau )
{
    // Calculate the number of the tesselation case
    int case =
    ( [ delta_sse( T.edge2 ) > tau || eem.isMarkedForSplit( T.edge2) ] * 4 ) |
    ( [ delta_sse( T.edge1 ) > tau || eem.isMarkedForSplit( T.edge1) ] * 2 ) |
    ( [ delta_sse( T.edge0 ) > tau || eem.isMarkedForSplit( T.edge0) ] );

    if( case == 1 || case == 2 || case == 3 )
        eem.markForSplit( T.edge2 );
}
```

This geometry shader program is applied repeatedly to the input mesh until the number of outputted edge marks reaches zero.

Tesselating the mesh

After all necessary forced splits are marked in the edge error map, exclusively the tesselation cases 0, 4, 5, 6 and 7 can occur. Triangles corresponding to these cases can be tesselated with right-isosceles triangles according to Fig. 3.

3.2.3 Edge error map

The problem addressed by the idea of an edge error map is that a bijective map from the set of edges to $\mathbb{R}$ is searched for. For the application in the presented algorithm the codomain needs to be only the set $M = \{0, 1\}$ representing the information if an edge is not marked or marked for a later split operation. The function should be available from within a geometry shader program.

Concept

After defining the given problem, it stands to reason to use a texture for storing all function values. Texture maps can be randomly accessed from within a shader program and deliver a range of $\mathbb{R}$ for the texture values in a 2D domain. The open question is how to encode an edge of a given mesh unambiguously into a 2D texture coordinate value. A property of the meshes generated with heightmap elevation data can be used to solve this. Consider a mesh with one component of its vertices exclusively retrieved from the heighmap function (for example the z-component as no. 1 in Fig. 5). The heightmap is a function. This means every input value in the 2D domain maps to only one height value of the represented terrain surface. Therefore, the projection of the mesh onto

the plane spanned by the unit-vectors of the other two components (respective the xy-plane) is unambiguous - no vertex, edge or primitive covers another one (see no. 2 in Fig. 5). In this projected mesh, an edge can be uniquely identified by an arbitrary 2D point on it except the end points which are shared by more than one edge. The 2 coordinates of a point on the edge can then be used to access the edge error map.

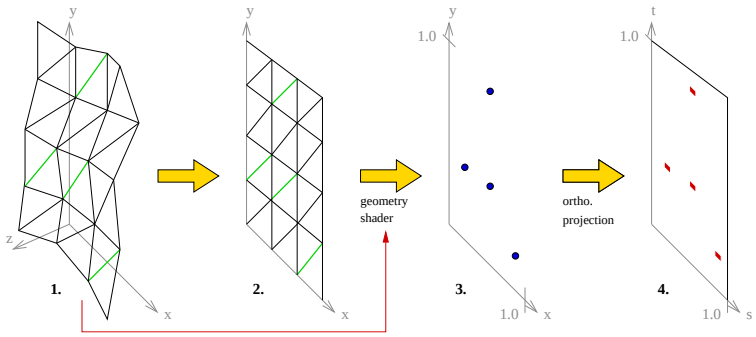

Figure 5: The process of marking an edge. 1: Terrain mesh with edges, which should be marked (green). 2: The projection of 1 onto the xy-plane 3: Points (blue) with coordinates equal to the marked edges midpoint texture coordinates. 4: The orthogonal projected points rendered to a FBO texture.

The presented algorithm uses the midpoint of an edge for its identification. This guarantees that two congruent edges of adjacent triangles whose endpoints are defined in a differing order are identified as the same edge. The projection described above can be done by simply calculating the texture coordinate of the midpoint - this is actually the projection of the midpoint position onto the xy-plane (assuming that the height values are stored in the z-component) with a scaling to the range $[0, 1] \times [0, 1]$. Therefore, an extra projection step from no. 1 to no. 2 in Fig. 5 can be omitted as indicated by the red arrow.

Altogether, an edge's midpoint texture coordinate can finally be used to store and retrieve information from the edge error map for the edge. In the case of the presented algorithm, this information is a flag which is encoded in a color channel of a single texel.

Realization

In the realization of the described idea, retrieving texture information in a shader program can easily be done by the OpenGL texture fetch mechanism. But OpenGL natively supports no method to directly write data to a texture from a geometry shader program. First, GPGPU-languages like OpenCL or Nvidia's CUDA could be used to do this task or second, it can be done directly within the OpenGL rendering pipeline. In the research implementation of the presented algorithm the latter was done, since at the moment of writing this

thesis, it guarantees the highest possible platform independency. For every edge which is chosen to be marked in a geometry shader program, the program generates a 2D point with coordinates equal to the edges midpoint texture coordinates (no. 3 in Fig. 5). The outputted points are then rendered with an orthogonal projection to the according pixels in a texture via the use of a FBO (no. 4 in Fig. 5). To ensure that a generated point will cover exactly one pixel on the texture map, the OpenGL point size has to be set to 1.0, point smoothing and point distance attenuation have to be switched off.

Dimensions of the edge error map

The only open parameters of the presented idea are the dimensions of the edge error map. A lower bound of the texture size is searched for, which still ensures that all values of the bijective function described above can be stored. To figure out this size, the triangle with the smallest possible area has to be considered. In the convergence analysis two properties of the presented algorithm will be shown: First, the minimal length of an edge in texture-coordinate-space is the size of one texel of the heightmap measured in texture-coordinate-space. Second, it can be ensured that every vertex in a mesh generated by the algorithm has texture coordinates lying exactly on a heightmaps texel position.

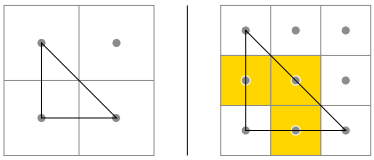

Figure 6: Left: Triangle with the smallest possible area in texture-coordinate-space. The grey points represent the exact texel positions of the heightmap. Right: smallest triangle which can be split again.

Because of these two reasons, the triangle with the smallest possible area has the dimensions in texture-coordinate-space as depicted in the left of Fig. 6. But in the edge error map only edges, which can be further split may need a mark. As the outlined split operator splits edges at their midpoint, the smallest possible triangle which can be split has the doubled dimensions as illustrated on the right of Fig. 6. In this case, every midpoint of an edge can be mapped bijective to a heightmap texel, as indicated by the yellow pixels, whereas in the left case the heightmaps resolution would not be sufficient. For that reason, the heightmaps dimensions are a lower bound for the edge error maps size.

3.2.4 Putting it all together

Until now, the components of one step of the recursive mesh refinement method have been described. This section explains how all parts of the algorithm

collaborate with each other and how recursion is introduced. Fig. 7 gives an overview of the complete functionality, which is executed exclusively on the GPU:

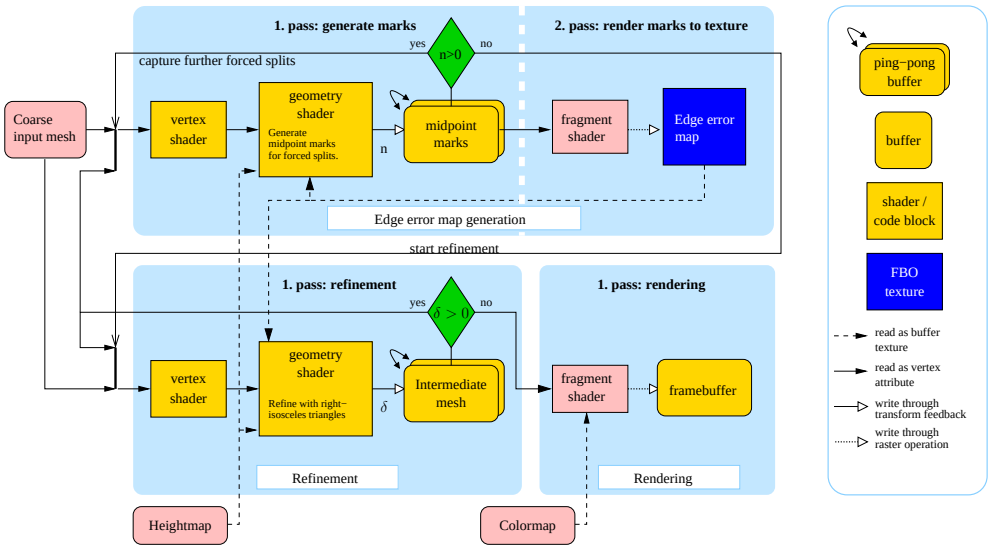

Figure 7: Complete workflow of the presented algorithm.

The algorithm is initiated with a coarse input mesh. After the input of this mesh it can be divided into three units:

1. The edge error map generation: In this unit all edges, which have to be split to ensure an exclusively right-isosceles triangulation, are marked for a later split. These marks are generated within the geometry shader stage in this unit. Describing the generation of these marks it was glossed over a problem with the implementation of this idea which becomes obvious with the overview of the complete algorithm. As depicted in Fig. 7, to generate the edge error map, the according geometry shader program has both, to read and to write from a texture attached to a framebuffer object, which is not possible in OpenGL. Therefore, the marks are temporary buffered and in a second pass rendered to the texture. As a forced split of an edge can force further splits in a recursive sequence, edge marks have to be successively increased until no new edge marks are added (n=0 in Fig. 7). The recursive application of a shader program is done by the use of Transform Feedback [9], which is not further explained as its use in the domain of terrain rendering is well-documented in Schmiade [5]. When no further marks are generated in the geometry shader stage, the edge error map is complete and the refinement of the mesh is started.

2. The refinement of the mesh: With the forced splits from the edge error map, this unit can subdivide the inputted mesh entirely with right-

isosceles triangles. After one refinement step is completed, the difference between the generated primitives in the previous refinement pass and the current pass is measured (named δ in Fig. 7). If no new primitives have been created ($\delta = 0$) in the current pass, the screen-space error of all edges in the mesh falls under the user-specified threshold and the mesh can be rendered. If not, the whole refinement process has to be restarted with the refined mesh - again first creating the new edge error map followed by a further application of the refinement unit. The recursive application of the refinement unit is realized with the use of transform feedback as above.

3. Rendering: After the execution of the second unit, the desired mesh quality is reached and the mesh can be rendered. In the rendering pass, an arbitrary shading model can be applied to the geometry.

The whole process of generating the refined terrain mesh has not to be reinitiated for every rendered frame. A restart of the refinement with the coarse input mesh has to be done only if the POV or the user-defined screen-space error threshold is changed.

Coarse input mesh

After overviewing how all components of the PAR algorithm collaborate, the coarse input mesh is a parameter which has not been defined in the preceding sections. As the terrain elevation data in the heightmap is defined in a 2D-domain, this input mesh should be rectangular and have the heightmaps' aspect ratio. To ensure the generation of right-isosceles triangles, this mesh is further required to be composed of right-isosceles triangles exclusively. Therefore, the mesh does not neccessarily consist of more than 2 triangles. But a higher resolution has the following advantage: The number of refinement passes is decreased by a pre-refined input mesh. For a mesh with very few triangles using the presented shader-based approach is not very effective, as due to the low number of primitives, the parallelism of the method cannot be exploited.

3.3 Convergence analysis

This section analyses if the recursive PAR algorithm terminates.

As defined with the edge-based split operator, the algorithm terminates if every edge in the mesh falls under a given screen-space error threshold. As this threshold can lie arbitrarily near to zero, the algorithm shall subdivide until the mesh's resolution has reached the heightmap's resolution. That means for a screen-space error threshold of exactly zero a maximal refinement of the entire mesh should be done until the heightmaps resolution is reached. In the following paragraphs, it is first presented how the algorithm ensures that only vertices with texture coordinates exactly on heightmap texel midpoints are produced. Based on that the properties of a mesh which has reached

the heightmap's resolution are derived. Finally, it is shown that an arbitrary coarse input mesh fulfills after a limited number of subdivision steps these properties, thus the algorithm terminates.

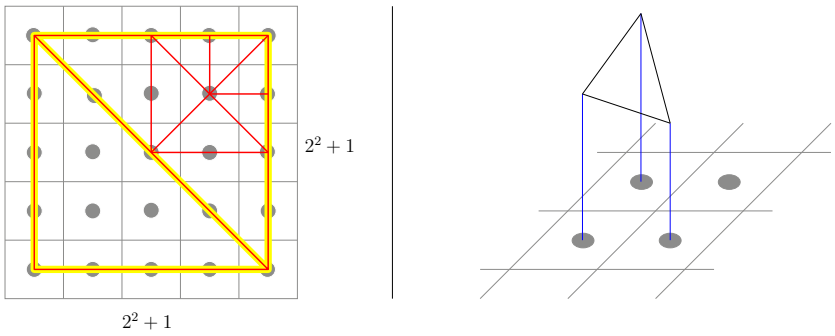

Figure 8: Left: The coarsest possible input mesh (yellow) and a possible tesselation (red) with texture coordinates exactly on texels of the heightmap (the grey points). Right: Maximal subdivided triangle. The blue lines represent the according elevation values.

In order to ensure that only vertices with texture coordinates exactly on heightmap texel midpoints are generated, the idea is to use heightmap sizes of $(2^n + 1) \times (2^n + 1)$ texels where $n \in \mathbb{N}$. Then, the texture coordinates of the input mesh's vertices can be assigned to the their according texels of the heightmap as depicted on the left in Fig. 8. The size of the initial control mesh in texture-coordinate-space is then $2^n \times 2^n$ texelwidth[1]. Therefore, bisecting an arbitrary edge of the coarsest possible control mesh k times with $k \leq n$ produces new vertices lying in texture-coordinate-space exactly on texel midpoints (see Fig. 8). This is because the s,t components of the texture-coordinate-space lengths $l_{s,t}$ of the k times subdivided edges can be expressed by $l_{s,t} = 2^n \cdot (\frac{1}{2})^k \Rightarrow l_{s,t} \in \mathbb{N}$.

It can now be defined when the refined mesh's resolution has reached the heightmap's resolution. That is the case if for every edge the components s and t of the edge's length $l_{s,t}$ in texture-coordinate-space are 1 texelwidth. On the right of Fig. 8, a triangle with three maximal refined edges is depicted. As the heightmap delivers no further information for the line section covered by a maximal refined edge, the algorithm explicitly breaks further subdivisions on an edge whose $l_{s,t}$ become equal to one. Consequently an arbitrary edge of the coarse input mesh will be subdivided at most n times. That is because the edges resulting from its subdivision fulfill $l_{s,t} = 2^n \cdot (\frac{1}{2})^n = 1$. For every new edge inserted for a triangle in the presented tesselation stage (as depicted in Fig. 3) its $l_{s,t}$ is the half of one edges' $l_{s,t}$ of the triangle for which the edge was inserted. That means, the $l_{s,t}$ of every new inserted edge decreases with a single subdivision step.

[1] $texelwidth = \frac{1.0}{2^n+1}$ is the size of one texel of the heightmap in texture-coordinate-space

Altogether it can be concluded: As the coarse input mesh has only a limited number of edges which are subdivided at most a limited number of times (n) and for every new inserted edge this number decreases, the whole algorithm terminates after a limited number of subdivision steps.

After it has been proven, that the algorithm terminates, it can now be formally shown that the presented method does not produce degenerate or near-degenerate triangles. These can be defined as triangles with a length of at least one edge smaller than an $\epsilon \to 0$ with $\epsilon > 0$. Assuming that position coordinates are chosen greater or equal to texture coordinates, the smallest possible length of a right-isosceles triangle's leg is 1 texelwidth. Regarding the value 1 texelwidth as ϵ, the presented algorithm produces no degenerate or near-degenerate triangles, as every edge has a length greater than ϵ.

3.4 Culling

This section describes how the PAR algorithm benefits from culling methods. These techniques are used to determine the visible primitives from a given POV. To enhance performance, the presented algorithm should discard any calculation on primitives which are completely not visible. The following classical culling algorithms can be used:

First, view frustum culling can be applied to every part of the presented algorithm. The calculation of the edge error map, the refinement and the rendering unit are completely disabled for triangles entirely outside the viewing frustum. View frustum culling is done in the presented algorithm by calculating the minimal bounding sphere of a triangle and comparing the sphere's distance to the six clipping planes defined by the view frustum with its radius. These calculations are done in the geometry shader stage. Ericson [7] gives a detailed description on view frustum culling.

Second, occlusion culling can be used to discard calculations for triangles occluded by others. Occlusion culling can only be executed after the complete refinement process - thus only in the rendering of an already refined mesh. An intermediate mesh in the refinement process represents only a coarse sampled approximation of the heightmap elevation data. Therefore, the problem is that triangles of an intermediate mesh occluded by others might in a further refinement step be subdivided into triangles which are then no longer occluded.

3.5 Problems

At the date of writing this thesis the only known problem are under-refined primitives. This is a common problem of many recursive refinement algorithms. Due to the coarse sampling of the terrain elevation data in early steps of the refinement process, only low frequencies in the heightmap are approximated well. Therefore, it is possible that some primitives may be rejected for further subdivision which would be necessary to approximate high frequencies of the terrain data in the domain covered by the primitives. Schmiade gives

in [5] an detailed overview on the problem making a further description of the problem redundant. In practice, a sufficient small screen-space error threshold increases the probability that under refined triangles of a coarse mesh will be refined, thus minimizes the problem. At a screen-space error threshold of 1.0 pixel, no under-refined triangles could be measured with sample terrain data sets and testing environment as described in the results Sec. 5. A more exact calculation of the screen-space error on an edge with more than one sample would make under-refined triangles more unlikely to occur. A measurement of the heightmaps frequencies would be a complete solution. When the resolution of the coarse input is adjusted so that every frequency in the heightmap is represented, under-refined primitives cannot occur.

3.6 Conclusion

A new parallel terrain rendering method on the GPU has been presented. The central features of the method can be overviewed on the basis of the four central requirements on terrain rendering algorithms pronounced in the requirements analysis:

1. Accuracy: The presented algorithm guarantees an upper screen-space error bound τ, where $\tau \geq 0$ can be freely adjusted by the user.

2. Optimal mesh topology: The meshes generated by the algorithm have a consistent continuous topology without degenerate or near-degenerate triangles. They contain no gaps and consist exclusively of right-isosceles triangles.

3. Parallel approach: The presented algorithm is designed parallel and is executed entirely in a parallel GPU runtime environment. The adaptive, view-dependent refinement is done directly in the rendering pipeline.

4. Small memory footprint: Only the fastest available memory - the graphics memory - is used. Solely necessary immediate mesh data and the edge error map as well as the terrain elevation data is stored in this memory.

4 4pt-hermite smooth subdivision

In this section, a new adaptive interpolatory smooth subdivision scheme, named "4pt-hermite smooth subdivision", is presented. The method is completely local on triangle basis and can thus be applied fully parallel directly in the rendering pipeline. Furthermore, no preprocessing or topology data structures are needed. The new subdivision scheme is a generalisation of the classical 4-point subdivision method for curves and the cubic hermite interpolation. It is applied adaptively only to sections of a mesh, where further subdivision adds detail. These details are measured in screen-space. The new scheme does not produce any degenerate or near-degenerate triangles during the adaptive subdivision.

Starting with a definition of the problem addressed by smooth subdivision surfaces, a set of criteria for the classification and comparison of existing smooth subdivision schemes is presented in this section. From these criteria, requirements on the new method are derived. Following, in an overview of existing subdivision schemes it is investigated which existing methods fulfill the pronounced requirements to what extent. Based on these considerations, the concept of the new method is developed and presented in detail. It is presented how the new method can be integrated into the PAR terrain rendering algorithm of the previous section. In a convergence analysis it is investigated if the 4pt-hermite subdivision terminates. Finally, it is shown how a non-degenerate triangulation is be achieved.

4.1 Motivation and goals

In the previous section, a new parallel terrain rendering method on the GPU was outlined, which fulfills the requirements presented at the beginning of this thesis. Consequently, at a first glance the development of a new method addressed to the problem of terrain rendering seems to be superfluous. But in that requirements analysis exclusively the core requirements on a terrain rendering algorithm were outlined. The application of smooth subdivision results in a variety of improvements on the presented PAR method, which were not covered there.

These improvements can easily be named after the problem addressed by smooth subdivision surfaces is defined:

Smooth subdivision schemes shall give a representation for a smooth surface generated from a control mesh. This representation is specified in the form of refinement rules. The smooth surface is then defined as the limit of a recursive refinement of the control mesh with these rules.

Now, two central benefits of using smooth subdivision surfaces in the context of terrain rendering can be pronounced:

Silhouette preservation

The contours of an object represent an essential property for the human visual system to identify the object's shape and position in 3D-space. The application of smooth subdivision schemes to a terrain mesh retains more detail near silhouettes. Therefore, with a higher resolution on object edges, smooth subdivision improves the observed image quality of the rendered terrain.

Infinite continuous levels-of-detail

Through their recursive structure, smooth subdivision surfaces provide an infinite fine resolution of the limit surface in the refinement process. Consequently, smooth subdivision surfaces can be exploited as a representation of the heightmap elevation data with an unlimited resolution. If the mesh generated in a terrain rendering algorithm has reached the heightmap's resolution, smooth subdivision surfaces offer still further levels-of-details. This has the following benefits:

- Close-up POVs: The terrain can be viewed from an arbitrarily close POV while still enabling the construction of a fine mesh. This is an advantage when terrain information other than elevation data (for example color information) is available in a higher resolution as the heightmap or further algorithms should be applied on very small sections of the mesh.
- Large slope changes are smoothened: Large slope changes in a terrain surface mesh - at the edges of cliff areas or summits - can be noticed as sharp creases or darts. Large differences in adjacent elevation values cause these terrain features. As there is no continuous transition between the adjacent height values, the resolution of the heightmap is too low in the affected terrain sections. The application of smooth subdivision surfaces solves this problem by generating a smooth transition between the adjacent height values.

4.2 Criteria for the analysis of smooth subdivision schemes

This section presents criteria for the classification and comparison of smooth subdivision schemes. These are used first, to develop requirements on the new method and second, to structure the variety of existing subdivision schemes. The central criteria are presented in the following listing:

- Interpolation/Approximation: Interpolating schemes have limit surfaces which float through the vertices of the control mesh. In the approximating case, the control mesh vertices do not have to lie on the subdivision surface.
- Continuity: Parametric continuity describes the smoothness of a parameter's change along a curve. In the case of a surface, C^n-continuity with $n \in \mathbb{N}$ defines that n partial derivatives of the surface are continuous.

- Subdivision rules: A subdivision rule describes how an input mesh is related to the refined mesh resulting from a single subdivision step. The split operator - how and where vertices and faces are inserted - as well as the application of the rules are distinguishing features of smooth subdivision schemes. The application can be categorized as follows: In uniform schemes, all sections of an input mesh are subdivided with the same rule, while in non-uniform schemes, various rules for different sections exist. Stationary schemes apply the same set of rules to a mesh in every subdivision step, while non-stationary schemes possess several rules for different subdivision steps.

- Mesh type: The way how vertices are combined to face polygons is a central feature of the mesh, smooth subdivision schemes are applied to and which they output. In practice, nearly exclusively quadrilateral and triangular schemes are distinguished.

- Support: The support of a smooth subdivision surface defines how large the adjacency of vertices is, needed for the application of the scheme to a single primitive.

4.3 Requirements

After the criteria in the preceding section have been presented, the following requirements on a smooth subdivision scheme for application in terrain rendering can be pronounced:

As the scheme shall be applied on meshes generated from terrain elevation data, the subdivision method has to be interpolating. An approximating scheme would alter the original data, thus decreases the accuracy of the rendered terrain scene. To generate the smoothest possible surface, the limit surface should be C^r-continuous with a greatest possible $r \in \mathbb{N}$. In order not to produce any holes in the surface it should at least be C^0-continuous. To make an application of the subdivision method fast and simple to implement, in the best case the scheme should be uniform and stationary.

Beneath these requirements derived directly from the criteria above, the positive features of the PAR terrain rendering method shall be applied and the new smooth subdivision method should be compatible and easy to integrate with this method. This defines the following additional requirements: The new smooth subdivision method should be executable fully parallel in a stream-processor environment on the GPU. Therefore, its support should be as small as possible - in the best case on the vertices of one primitive. As PAR generates meshes consisting of triangles, the scheme should be triangular. The subdivision scheme is demanded to insert non-degenerate triangles (in the best case it inserts exclusively right-isosceles triangles). Finally, to effectively apply the smooth subdivision method, the scheme should be adaptive. Only in mesh regions where a subdivision adds detail, which can be measured by a difference in screen-space, the scheme should be applied.

4.4 Existing smooth subdivision methods

The starting point for the design of a new method, which fulfills a maximal subset of the requirements described above, is a look at the features of existing smooth subdivision methods. Zorin and Schröder in [10] cover many aspects of smooth subdivision theory. As a detailed survey on the vast number of existing schemes would exceed the scope of this work, an overview of the classical interpolating schemes is given:

Scheme	Mesh type	Continuity	Support	Split operator
Butterfly [13]	triangular	C^1	subset of 1-adjacency of an edge	1-to-4 for every triangle
Kobbelt [15]	quadrilateral	C^1	1-adjacency of a quad	1-to-4 for every quad
Ternary subdivision [18]	triangular	C^2	1-adjacency of a triangle	1-to-9 for every triangle
Interpolatory $\sqrt{3}$-Subdivision [16]	triangular	C^1	1-adjacency of a triangle	special $\sqrt{3}$-split
Interpolatory $\sqrt{2}$-Subdivision [17]	quadrilateral	C^1	1-adjacency of a quad	special $\sqrt{2}$-split

Table 1: Interpolating smooth subdivision schemes. All listed schemes are stationary and uniform.

All schemes presented in table 1 are at least C^1-continuous, which guarantees a fair smooth limit surface. But none of the listed subdivision methods has a completely local support on single primitives. This is problematic because of the following reason: To use adjacency information, a topology data structure would be needed. After one primitive is processed in a subdivision, the adjacency information for all adjacent primitives has to be updated, what means that these cannot be processed in the same pass. Therefore, a fully parallel approach is prohibited.

A completely local support on a single triangle solves this problem. In [21] Broubekeur and Schlick pursue this approach to present a GPU-based approximation of existing smooth subdivision schemes. Beneath this work, Nielson [20], Vlachos [22] and Karbacher [14] describe methods for local triangular subdivision methods. These methods have in common that the vertices and normals of a flat triangle are used to construct a curved Bézier patch for this triangle. This approach produces at least C^0-continuous meshes, as adjacent triangles have identical vertices and normals on their sharing edges. It can be applied straightforwardly to a complete mesh. But these methods do not include an adaptive application without degenerate or near-degenerate triangles and exclusively in mesh regions where a subdivision adds detail in screen-space.

Therefore, a new smooth subdivision method is developed, which has features that enable a non-degenerate triangulation while still being fully local on triangle basis. The idea is to derive the new scheme from the classical

subdivision schemes outlined in table 1, to acquire as much of the continuity properties as possible. Ideas of the local triangular schemes shall be applied to enable a completely parallel approach. In a detailed view at the schemes in table 1, the observation can be made that all schemes are directly derived from the 4-point scheme [12] for curves or a modification of it - that means these subdivision schemes can be understood as a generalization of the 4-point scheme for the case of a surface. This scheme is chosen as a starting point for the development of the new 4pt-hermite scheme presented in this thesis. An overview of the 4-point scheme for curves is given in the following paragraph.

4-point subdivison scheme

The classical 4-point subdivision scheme [12] defines for a given set $M^0 = \{p_i^0\}_{i=-2}^{i=n+2}$ of $n+4$ initial control points a C^1-continuous curve G, which interpolates all $p \in M$. The curve G is represented as the limit of the recursive application of the following rule:

$$M^{k+1} = \begin{cases} p_{2i}^{k+1} = p_i^k & -1 \leq i \leq 2^k n + 1 \\ p_{2i+1}^{k+1} = \left(\frac{1}{2} + w\right)\left(p_i^k + p_{i+1}^k\right) - w\left(p_{i-1}^k + p_{i+2}^k\right) & -1 \leq i \leq 2^k n \end{cases}$$
$$G = M^{k \to \infty} \tag{3}$$

The geometric interpretation is depicted in Fig. 9. The new inserted points are the midpoints of a control polygons' edge $\overline{p_i p_{i+1}}$ shifted by a vector $2w\vec{e}$ where $\vec{e}$ is the vector from the midpoint of the secant $\overline{p_{i-1}p_{i+2}}$ to the midpoint of $\overline{p_i p_{i+1}}$. Consequently, $w \in \mathbb{R}$ is a tension parameter, which controls the tightening of the resulting limit curve to the control polygon. Dyn and Levin in [12] introduce the special value $w = \frac{1}{16}$ more as a heuristic value for a visually smooth curve and prove that this value is in a range of w-values which produce a C^1-continuous limit curve.

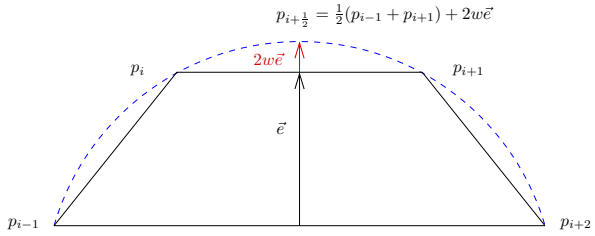

Figure 9: Geometrical interpretation of the interpolatory 4-point subdivision scheme.

4.5 Concept

Based on features of existing smooth subdivision methods overviewed in the preceding section, in this section a new adaptive interpolatory smooth subdi-

vision scheme is designed.

As the new method is demanded to be completely local on a single primitive, the input to the new algorithm is for every mesh triangle exclusively its three points and normals. Consequently, the method does not receive any information about a triangle's adjacent vertices. To apply concepts of smooth subdivision methods which use adjacency information, the idea is to approximate the position of adjacent vertices through the primitive's normals. With these premises a smooth boundary curve is constructed for every triangle edge in the input mesh. This curve is directly derived from the 4-point subdivision scheme for curves. It is proven that exactly the same curve can be constructed with cubic hermite interpolation. In a single subdivision step one new point lying on this curve is chosen for a possible insertion. To achieve a good tradeoff between modeling range, simplicity and a fair limit surface, the new normals are calculated through an adaptive linear interpolation. As altogether the new vertices and normals inserted at an edge can be computed exclusiveley with the edge's point-normals, the new scheme is completely edge-based. To apply the 4pt-hermite subdivision adaptively only to the sections of a surface where further subdivision adds detail, a new point is only inserted if the distance between the edge and the new point exceeds a user-defined screen-space difference threshold.

4.5.1 New inserted vertices

This section describes how the new vertex inserted on an edge in a single subdivision step is calculated. First the 4-point subdivision is used to calculate a smooth curve for every edge of a triangle in the coarse input mesh. This curve interpolates the edges end-vertices. As depicted in Fig. 9 the calculation of a smooth curve for an edge $\overline{p_i p_{i+1}}$ uses the edges' neighboring points p_{i-1} and p_{i+2}. As only the points and normals of that edge are accessible for the calculation, p_{i-1} and p_{i+2} are approximated by the new smooth subdivision method through the normals. The idea behind this approach is that the points lying on a smooth curve and interpolating a given set of control points converge fast with a decreasing distance from a given control point to its tangent. Consequently, the points on the considered tangent, which can be calculated with the according control point's coordinates and normal, can be interpreted as an approximation of points on adjacent control polygon edges. The left of Fig. 10 illustrates an edge $\overline{p_1 p_2}$ with the edge vector $\vec{E} = p_2 - p_1$ and the length $d = \|\vec{E}\| = \|p_2 - p_1\|$. p_0 and p_3 are chosen that they lie on the tangent of the according control point and have a distance of d to that control point. The value d is chosen as the control polygon is supposed to be regularly spaced. This is justified because of two reasons: First, the mesh outputted by the PAR algorithm fulfills this requirement nearly complete as adjacent triangles can differ only by one level of refinement. That is because the introduced mechanism of forced split sequences makes sure that sharing edges of triangles differing by more than one refinement level are split. Second, it will be shown in the convergence analysis of the new smooth subdivision

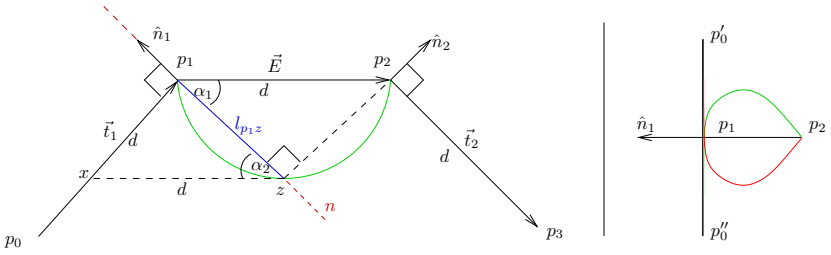

Figure 10: A segment $\overline{p_1p_2}$ of a control polygon for a smooth curve subdivision scheme. The hat on a vector $\hat{v}$ means that $\hat{v}$ is normalized. Left: The construction of adjacent points p_0, p_3. Right: Ambiguity for a normal vector collinear to the edge vector.

method that the adaptive application will not insert triangles in mesh sections where normal variation is very small. Therefore also at the borders of these sections where differing subdivision levels can occur, the normal variation has to be small. Thus intermediate meshes in the refinement process can be expected to have little tangent variation at triangles with differing subdivision levels. Therefore, suggesting an equally spaced control mesh at triangles with large differences in their refinement levels introduces no large errors.

After the points p_0 and p_3 have been defined, they can now be expressed with the end-points and normal vectors of edge $\overline{p_1p_2}$. First, p_0 shall be calculated. The idea is to determine a point x as depicted in Fig. 10. If x is not identical to p_1, the line through p_0 and p_1 is unambiguously defined through x and p_1. The point z is chosen as the intersection of the normal n (red) through p_1 with the Thales' semicircle (green) around the midpoint of $\overline{p_1p_2}$ with the radius $\frac{d}{2}$. Therefore, the triangle $\triangle p_1p_2z$ is right-angled at z. Consequently, the length l_{p_1z} of the triangles edge $\overline{p_1z}$ can be calculated by the length of the projection of $\vec{E}$ on $\hat{n}_1$, so that

$$l_{p_1z} = -\vec{E} \cdot \hat{n}_1 \quad \text{and} \quad z = p_1 - \hat{n}_1 \cdot l_{p_1z} \tag{4}$$

As it is chosen $\alpha_2 = \alpha_1$ as depicted in Fig. 10 these are corresponding angles and the quadrilateral p_1p_2zx is a parallelogram. As parallelograms have parallel opposite sides of equal length, it is:

$$z - x = \vec{E} \Leftrightarrow x = z - \vec{E} \underset{\text{with (4)}}{=} p_1 - \vec{E} + \hat{n}_1 \left(\vec{E} \cdot \hat{n}_1\right) \tag{5}$$

Consequently, the tangent vector $\vec{t}_1 = p_1 - p_0$ and with that p_0 can be calcu-

lated as follows:

$$\begin{aligned} \vec{t_1} &= d\frac{p_1 - x}{\|p_1 - x\|} = \|\vec{E}\|\frac{\vec{E} - \hat{n}_1\left(\vec{E}\cdot\hat{n}_1\right)}{\|\vec{E} - \hat{n}_1\left(\vec{E}\cdot\hat{n}_1\right)\|} \\ p_0 &= p_1 - \vec{t_1} = p_1 - \|\vec{E}\|\frac{\vec{E} - \hat{n}_1\left(\vec{E}\cdot\hat{n}_1\right)}{\|\vec{E} - \hat{n}_1\left(\vec{E}\cdot\hat{n}_1\right)\|} \end{aligned} \tag{6a}$$

Completely analogue the following can be derived for $\vec{t_2} = p_3 - p_2$ and p_3:

$$\vec{t_2} = \|\vec{E}\|\frac{\vec{E} - \hat{n}_2\left(\vec{E}\cdot\hat{n}_2\right)}{\|\vec{E} - \hat{n}_2\left(\vec{E}\cdot\hat{n}_2\right)\|} \quad p_3 = p_2 + \vec{t_2} = p_2 + \|\vec{E}\|\frac{\vec{E} - \hat{n}_2\left(\vec{E}\cdot\hat{n}_2\right)}{\|\vec{E} - \hat{n}_2\left(\vec{E}\cdot\hat{n}_2\right)\|} \tag{6b}$$

It follows that $\vec{t_1}, \vec{t_2}$ and p_0, p_3 cannot be calculated if $\|\vec{E} - \hat{n}_i(\vec{E}\cdot\hat{n}_i)\| = 0 \Leftrightarrow \vec{E} = \vec{0} \vee \hat{n}_i \parallel \vec{E}$ with $i \in \{1, 2\}$. If the respective normal vector is collinear with the edge vector $\vec{E}$, the information about the curvature of the control polygon is ambiguous (see right side of Fig. 10). Considering the point p_1 in $\mathbb{R}^2$ space, two adjacent points $p_0^{'}, p_0^{''}$ and consequently two smooth curves (red and green) could be constructed. In $\mathbb{R}^3$ space, an infinite number of curves can be constructed. As no unambiguous smooth subdivision curve can be generated, in this case it is explicitly set $\vec{t_i} = \vec{0}$ and $p_0 = p_1$, $p_3 = p_2$ which guarantees that the smooth curve will be identical to $\overline{p_1 p_2}$ - that can be derived from the subdivision rules presented in the following sections.

Application of the 4-point subdivision scheme

After all necessary points for a given edge $\overline{p_1 p_2}$ are derived, the 4-point scheme with equation (3) can be applied straightforward to this edge. The new point P_{4pt}^{new} inserted at $\overline{p_1 p_2}$ by one subdivision step in the new scheme is:

$$\begin{aligned} P_{4pt}^{new} &= \left(\frac{1}{2} + w\right)(p_1 + p_2) - w\,(p_0 + p_3) \\ &\underset{\text{with (6)}}{=} \frac{1}{2}(p_1 + p_2) + w\|\vec{E}\|\left(\frac{\vec{E} - \hat{n}_1\left(\vec{E}\cdot\hat{n}_1\right)}{\|\vec{E} - \hat{n}_1\left(\vec{E}\cdot\hat{n}_1\right)\|} - \frac{\vec{E} - \hat{n}_2\left(\vec{E}\cdot\hat{n}_2\right)}{\|\vec{E} - \hat{n}_2\left(\vec{E}\cdot\hat{n}_2\right)\|}\right) \end{aligned} \tag{7}$$

For the same reasons as described in the context of the PAR method (paragraph "Edge-based split operator" in Sec. 3.2.2) the new smooth subdivision method inserts only up to one new vertex at an edge per subdivision step. Therefore, no further applications of the 4-point scheme have to be calculated for equation (7).

Application of cubic Hermite curves

In this paragraph an interesting property of the outlined scheme is presented. The curve generated by the application of the 4-point scheme above is equal to a cubic hermite interpolation with weighted tangent vectors. To show that, in the following paragraph the new subdivision scheme is now derived from cubic hermite interpolation applied to the considered edge with its approximated tangents. Cubic hermite interpolation constructs a smooth curve which interpolates the given positions and derivatives in the endpoints of the curve. As Farin gives in [11] a detailed description of hermite interpolants (and hermite curves can be regarded as a well-researched topic in computer graphics) only the definition used in this work is presented. The cubic hermite curve H interpolating the points p_1, p_2 and tangent vectors $\vec{t}_1, \vec{t}_2$ is given through:

$$\begin{aligned} &H(u) = p_1 h_0(u) + \vec{t}_1 h_1(u) + \vec{t}_2 h_2(u) + p_2 h_3(u) \quad \text{with} \quad u \in [0,1]; u \in \mathbb{R} \quad \text{and} \\ &h_0(u) = 2u^3 - 3u^2 + 1 \quad h_1(u) = u^3 - 2u^2 + u \\ &h_2(u) = u^3 - u^2 \qquad\quad h_3(u) = -2u^3 + 3u^2 \end{aligned} \tag{8}$$

With the value $u = \frac{1}{2}$ it becomes $h_0(\frac{1}{2}) = h_3(\frac{1}{2})$ and $h_1(\frac{1}{2}) = h_2(\frac{1}{2})$. Consequently, in this case both end-points and end-tangents contribute equally to the point $H(\frac{1}{2})$ on the cubic hermite curve. As the subdivision scheme designed in this section shall be symmetric for an inputted edge - that means the scheme should not depend on the order of an edges' point-normals inputted to the scheme - the value $u = \frac{1}{2}$ is chosen for the hermite curve point inserted on an edge in a single subdivision step. For a given edge $\overline{p_1 p_2}$ this point P^{new} can be expressed with equation (6) as follows:

$$\begin{aligned} P^{new} &= H\left(\frac{1}{2}\right) = p_1 h_0\left(\frac{1}{2}\right) + l_t \vec{t}_1 h_1\left(\frac{1}{2}\right) + l_t \vec{t}_2 h_2\left(\frac{1}{2}\right) + p_2 h_3\left(\frac{1}{2}\right) \\ &= \frac{1}{2}(p_1 + p_2) + \frac{1}{8} l_t \vec{t}_1 - \frac{1}{8} l_t \vec{t}_2 \\ &= \frac{1}{2}(p_1 + p_2) + l_t \frac{\|\vec{E}\|}{8} \left(\frac{\vec{E} - \hat{n}_1\left(\vec{E} \cdot \hat{n}_1\right)}{\|\vec{E} - \hat{n}_1\left(\vec{E} \cdot \hat{n}_1\right)\|} - \frac{\vec{E} - \hat{n}_2\left(\vec{E} \cdot \hat{n}_2\right)}{\|\vec{E} - \hat{n}_2\left(\vec{E} \cdot \hat{n}_2\right)\|} \right) \end{aligned} \tag{9}$$

In the equation above, the value $l_t \in [0,1]$ with $l_t \in \mathbb{R}$ is introduced as a weight for the lengths of both end-tangent vectors at the considered edge. The influence of the tangent vectors to the point P^{new} on the curve is thus linear dependent on l_t. Therefore, an increase of l_t makes the curve converge faster to the end-tangents - the curve is "pulled" towards the edges end-tangents. The tangent vectors encode the information on the approximated adjacent control points of an edge. By making l_t freely adjustable, the user or application of the smooth subdivision scheme can decide to what fraction this approximated

data or the original input data should contribute to the curve in a single refinement step. That means $l_t = 1.0$ produces a maximal smooth curve for a given control polygon, $l_t = 0.0$ results in a curve identical to the given control polygon and $l_t = \frac{1}{2}$ would deliver a good tradeoff.

Comparing equation (9) with equation (7), it becomes clear that with $l_t = 8w$ both schemes are equal (with $w = \frac{1}{16}$ it is $l_t = \frac{1}{2}$). Therefore, the presented scheme is named "4pt-hermite scheme". In the following sections equation (9) is meant regarding the 4pt-hermite scheme.

Continuity

The 4pt-hermite scheme generates for triangle edges smooth boundary curves. At sharing edges of adjacent triangles the identical curve is constructed as the vertices and normal vectors are identical there. Therefore, the smooth subdivision scheme can guarantee C^0-continuous limit surfaces. As in completely local schemes, the exact adjacency of a primitive can only be approximated, it is not possible to ensure a higher continuity without assumptions on the input mesh.

The generated smooth boundary curves meet at sharing end-points with tangents on the same tangent plane, because the normals are identical for the curves meeting there. Therefore, at a control mesh vertex, the limit surface is C^1-continuous and in small areas around, it is nearly C^1-continuous. With one subdivision step, the number of control mesh vertices for a following pass is increased. Therefore, the area in which the generated surface is nearly C^1-continuous increases with every subdivision pass. Consequently, the limit surface represents an approximation of a C^1-continuous surface.

4.5.2 New inserted normals

This section describes how the normal of the new point inserted at an edge in a single subdivision step is defined. With the subdivision rules presented in the preceding section a cubically varying geometry has been defined. But the 4pt-hermite scheme, presented in this thesis, does not calculate the exact cubically varying normals. That is because of two reasons: First, the exact normals are expensive to calculate. Second, often interpolating schemes with tangent-continuity produce, just because of the tangent-continuity and interpolation constraint, ripples or creases unwanted in a visually appealing fair limit surface. To weaken this effect, the normal variation over an edge should be as small as possible while still enabling a large modeling range.

Consequently, the central goal for the normal calculation is to find a good tradeoff between the modeling range, simplicity and a fair limit surface. Therefore, the 4pt-hermite scheme approximates the exact normals by an adaptive linear interpolation similar to Vlachos et al. [22] as well as Mao et al. [19].

With the outlined cubically varying smooth subdivision geometry, it is possible to model archs or serpentines at an edge. As depicted in the top of Fig. 11 for edge e_1, a linear interpolation approximates the arch cases of

quadratic or cubic curves, but it ignores the serpentine cases for edge e_2. For these cases in [22] and [19] a quadratic normal interpolation is introduced.

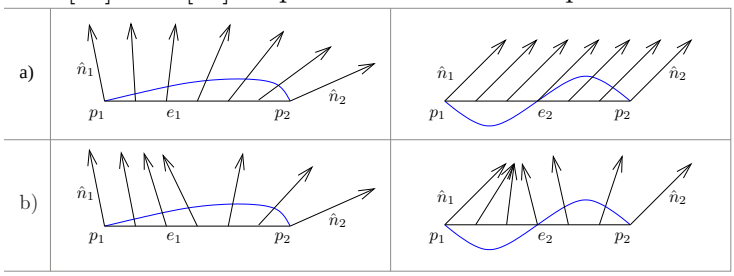

Figure 11: Linear normal interpolation (a) and quadratic normal interpolation (b). The blue curves represent the smooth limit curve.

Therefore, analogous to [19], the 4pt-hermite subdivision handles serpentine cases and quadratic cases seperate. A given edge $\overline{p_1 p_2}$ with the according normals and the edge vector $\vec{E} = p_2 - p_1$ (as depicted in Fig. 11) represents the arch case, if:

$$\begin{aligned} &\left[\left(\hat{n}_1 \cdot \vec{E} \geq 0\right) \wedge \left(\hat{n}_2 \cdot (-\vec{E}) \geq 0\right)\right] \vee \left[\left(\hat{n}_1 \cdot \vec{E} \leq 0\right) \wedge \left(\hat{n}_2 \cdot (-\vec{E}) \leq 0\right)\right] \\ &\Leftrightarrow \left(\hat{n}_1 \cdot \vec{E}\right)\left(\hat{n}_2 \cdot (-\vec{E})\right) \geq 0 \end{aligned} \tag{10}$$

In the following, both, the arch and serpentine case, are described in detail:

Arch case

For the arch case, the new normal vectors are calculated by linear interpolation. For a given edge $\overline{p_1 p_2}$ with the normalized end-normals $\hat{n}_1, \hat{n}_2$ the new arch case normal $\hat{n}^{new}_{arch}$ is calculated from $\vec{n}^{new}_{arch} = (1-l)\hat{n}_1 + l\hat{n}_2$ with $l \in [0,1]$.

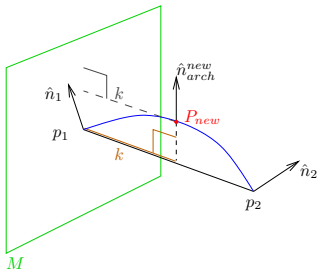

Figure 12: Calculation of the arch case normal vector $\hat{n}^{new}_{curve}$.

The weight l is set to $l = \frac{k}{\|\vec{E}\|}$. As depicted in Fig. 12, k is the length of the perpendicular (dashed) from the new inserted point P^{new} (red) on the plane M (green). $M = \{p \in \mathbb{R}^3 | (p - p_1) \cdot \vec{E} = 0\}$ is the plane through p_1 with

the normal vector $\vec{E}$. Consequently, k can be calculated as the projection $k = (P^{new} - p_1) \cdot \frac{\vec{E}}{\|\vec{E}\|}$. By inserting equation (9), it can be shown that $k \in \left[\frac{1}{4}\|\vec{E}\|, \frac{3}{4}\|\vec{E}\|\right]$:

$$\begin{aligned}
k &= \frac{\vec{E}}{\|\vec{E}\|} \cdot (P^{new} - p_1) \\
&= \frac{1}{2}\frac{\vec{E}^2}{\|\vec{E}\|} + l_t \frac{\|\vec{E}\|}{8\|\vec{E}\|}\vec{E} \underbrace{\left(\frac{\vec{E} - \hat{n}_1\left(\vec{E}\cdot\hat{n}_1\right)}{\|\vec{E} - \hat{n}_1\left(\vec{E}\cdot\hat{n}_1\right)\|} - \frac{\vec{E} - \hat{n}_2\left(\vec{E}\cdot\hat{n}_2\right)}{\|\vec{E} - \hat{n}_2\left(\vec{E}\cdot\hat{n}_2\right)\|} \right)}_{\vec{x}} \\
&= \frac{1}{2}\|\vec{E}\| + l_t \frac{1}{8}\vec{E}\cdot\vec{x} \quad \text{with} \quad \|\vec{x}\| \in [0,2] \\
&= \frac{1}{2}\|\vec{E}\| - l_t \frac{1}{8}\|\vec{E}\|\|\vec{x}\| cos\measuredangle(\vec{x},\vec{E}) \\
&= \frac{1}{2}\|\vec{E}\| - \frac{1}{8}\|\vec{E}\|\omega \quad \text{with} \quad \omega = l_t\|\vec{x}\| cos\measuredangle(\vec{x},\vec{E}) \Rightarrow \omega \in [-2,2] \\
&= \left(\frac{1}{2} - \frac{\omega}{8}\right)\|\vec{E}\| \Rightarrow k \in \left[\frac{1}{4}\|\vec{E}\|, \frac{3}{4}\|\vec{E}\|\right]
\end{aligned} \tag{11}$$

It was demanded that $l \in [0,1]$. As it was shown that $k \in \left[\frac{1}{4}\|\vec{E}\|, \frac{3}{4}\|\vec{E}\|\right]$ it is even $l = \frac{k}{\|E\|} \in \left[\frac{1}{4}, \frac{3}{4}\right]$. Altogether, a closed form of $\hat{n}^{new}_{arch}$ can finally be formulated:

$$\begin{aligned}
\vec{n}^{new}_{arch} &= (1-l)\hat{n}_1 + l\hat{n}_2 = \left(1 - \frac{(P^{new} - p_1)\cdot\frac{\vec{E}}{\|\vec{E}\|}}{\|\vec{E}\|}\right)\hat{n}_1 + \frac{(P^{new} - p_1)\cdot\frac{\vec{E}}{\|\vec{E}\|}}{\|\vec{E}\|}\hat{n}_2 \\
&= \hat{n}_1 + \frac{(P^{new} - p_1)\cdot\vec{E}}{\|\vec{E}\|^2}(\hat{n}_2 - \hat{n}_1) \\
\hat{n}^{new}_{arch} &= \frac{\vec{n}^{new}_{arch}}{\|\vec{n}^{new}_{arch}\|}
\end{aligned} \tag{12}$$

Serpentine case

Analogous to [22] and [19], in the serpentine case, the interpolated normal $\vec{n}^{new}_{arch}$ from equation (12) is reflected across the plane perpendicular to the edge. As it is proven that $l \in \left[\frac{1}{4}, \frac{3}{4}\right]$ this can be regarded as a good approximation for the serpentine case.
For a given edge $\overline{p_1p_2}$, as depicted in Fig. 13, the new serpentine case normal $\vec{n}^{new}_{serp}$ can be calculated with the following equation (13):

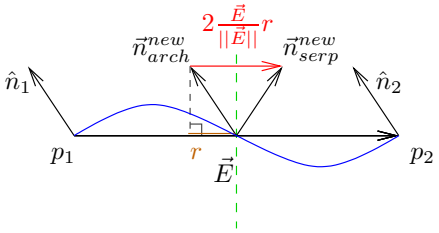

Figure 13: Calculation of the serpentine case normal $\hat{n}^{new}_{serp}$. The green dashed line represents the plane across the interpolated normal $\vec{n}^{new}_{arch}$ is reflected. r is the length of the projection of $\vec{n}^{new}_{arch}$ on $\vec{E}$.

$$\begin{aligned} \vec{n}^{new}_{serp} &= \vec{n}^{new}_{arch} + 2\frac{\vec{E}}{\|\vec{E}\|}r = \vec{n}^{new}_{arch} - 2\frac{\vec{E}}{\|\vec{E}\|}\left(\vec{n}^{new}_{arch} \cdot \frac{\vec{E}}{\|\vec{E}\|}\right) \\ \hat{n}^{new}_{serp} &= \frac{\vec{n}^{new}_{serp}}{\|\vec{n}^{new}_{serp}\|} \end{aligned} \tag{13}$$

Finally, it shall be proven that the serpentine case has only to be handled in the first pass of the subdivision scheme. That is because every edge resulting from subdivision of a serpentine case edge represents then the curve case: It follows from equation (10) that a smooth limit curve for a given edge $\overline{p_1p_2}$ is serpentine if both projections $(\hat{n}_1 \cdot \hat{E})$ and $(\hat{n}_2 \cdot \hat{E})$ have the same sign. The interpolated normal was defined as $\vec{n}^{new}_{arch} = (1-l)\hat{n}_1 + l\hat{n}_2$. Consequently, the projection of the interpolated normal on $\vec{E}$ expressed by $(\vec{n}^{new}_{arch} \cdot \hat{E}) = (1-l)\hat{n}_1 \cdot \hat{E} + l\hat{n}_2 \cdot \hat{E}$ with $(1-l)$, l both positive, has also the same sign. In equation (13), from $\vec{n}^{new}_{arch}$ this projection (the brown line in Fig. 13) is twice substracted (the length of the red vector). Therefore, the sign of the projection on $\vec{E}$ is inverted for $\vec{n}^{new}_{serp}$. Consequently, both edges resulting from a single subdivision of an edge representing the serpentine case, then represent the curve case.

4.5.3 Tesselation

A central goal of the subdivision method presented in this section is adaptivity. Only in mesh regions where further subdivision adds detail, which can be recognized in screen-space, subdivision should be applied. To achieve this, for every edge the distance between the new vertex to insert and the edge is measured in screen-space. This distance represents a measurement of the detail added for the edge in screen-space by a single subdivision step. If it exceeds an user-defined bound, this edge should be subdivided.
With distance between a point and an edge, the length of the perpendicular from the point on the line through both edge points is meant. As depicted in Fig. 14, for a given edge $\overline{p_1p_2}$ and the new point P^{new} this distance δ in object-space and δ_{ssc} in screen-space can be calculated as follows:

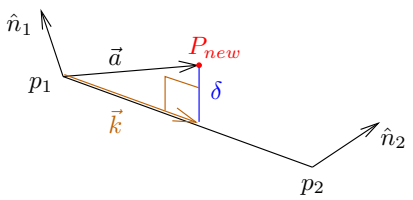

Figure 14: Measurement of the difference δ (blue) between an edge $\overline{p_1 p_2}$ and a new point P^{new} (red).

$$\begin{aligned}
\delta = \|\vec{a} - \vec{k}\| &= \left\| (P^{new} - p_1) - k\frac{\vec{E}}{\|\vec{E}\|} \right\| \quad \text{with} \quad k = (P^{new} - p_1) \cdot \frac{\vec{E}}{\|\vec{E}\|} \\
&= \left\| (P^{new} - p_1) - \frac{\left((P^{new} - p_1) \cdot \vec{E}\right)\vec{E}}{\|\vec{E}\|^2} \right\| \\
\delta_{ssc} &= ssc(\delta)
\end{aligned} \tag{14}$$

where $ssc : \mathbb{R}^3 \mapsto \mathbb{R}^2$ maps from object-space to screen space.

Altogether, the edge-based split operator for the 4pt-hermite scheme can be finally summarized as follows: Be σ the user-defined screenspace difference threshold in pixels, then an edge $\overline{p_1 p_2}$ should be split, if $\delta_{ssc} > \sigma$. The edge is subdivided with the already outlined new point P^{new} and new normal $\hat{n}^{new}_{arch}$ or $\hat{n}^{new}_{serp}$ into the two edges $\overline{p_1 P^{new}}$ and $\overline{P^{new} p_2}$.

Ensuring a non-degenerate triangulation

A central requirement on the smooth subdivision method presented in this section is to produce meshes without degenerate or near-degenerate triangles. In equation (11) it was shown that $k \in \left[\frac{1}{4}\|\vec{E}\|, \frac{3}{4}\|\vec{E}\|\right]$. Therefore, if every edge of an input mesh is subdivided in a single subdivision step, no degenerate or near-degenerate triangles can be generated (unless the inputted mesh already contains triangles of that type). Recapitulating the paragraph "Ensuring a right-isosceles triangulation" in Sec. 3.2.2, the problem occurs when only a subset of a triangle's edges has to be subdivided. This requires the insertion of oblique triangles and resubdividing these oblique triangles again with oblique triangles can produce near-degenerate triangles. To prohibit oblique triangles, the idea was to force additional edge splits which transform the problem cases the cases 1,2,3 (see Fig. 3) to the cases 5,6,7 where no oblique triangles are necessary.

This concept is now applied to make the 4pt-hermite scheme adaptive. The idea is the following: Additional edge splits, introduced to ensure a non-oblique triangulation, are done by inserting an interpolating point on the edge.

All edges, resulting from further subdivisions of that edge, are explicitly forced to lie on the original edge. Therefore, an adjacent triangle, which shares this edge, does not need to know about the split (the green triangle in Fig. 15). Thus a forced split in a triangle can be done local on that triangle. First this approach will be justified, and second, the realization will be presented.

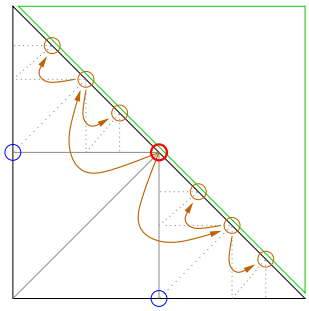

Figure 15: Two different refinement levels of adjacent triangles. In the first refinement level (solid grey) the blue circles represent an exceeded screen-space difference. Thus the black triangle forces a split (thick red circle). In the second refinement level (dotted grey) the brown circles represent an edge flag.

In the convergence analysis of the 4pt-hermite subdivision, it will be shown the following: For an arbitrary edge the distance of the new inserted point to the interpolating midpoint of the edge decreases with every subdivision step. That means the smooth subdivision curve for the edge approaches with every subdivision step more to the edge itself. Consequently for an edge, which does not exceed a user-defined screenspace difference threshold, the Hausdorff distance between the smooth curve for this edge and the edge can be assumed to be sufficiently small enough to justify that the curve can be approximated well by the edge itself. Therefore, when subdividing an edge although it would not be subdivided due to its screen-space difference, it is a good approximation to insert new vertices lying exactly on the edge. As described above, this approximation enables that a forced split can be done local on a triangle. In contrast, an approach using a memory lump for saving forced splits and multiple rendering passes to execute this splits (as introduced by the "edge error map concept") is less parallel and more expensive. Therefore, the local approach as described above delivers a good tradeoff between the performance of the method and the accuracy of the limit surface. That is the central reason why for the 4pt-hermite subivision not the memory-lump/multiple-pass technique was chosen.

In the realization, a forced edge split is done by inserting the according midpoint of the edge. Edges resulting from a forced subdivision are explicitly forced to lie on the original edge through the following idea: A flag is assigned to every edge of a triangle. If the flag is set, the edge is allowed to insert only new points lying on itself in further subdivision steps. This is achieved in the

tesselation of a triangle. Every edge inserted for a flagged edge recieves also the flag (illustrated by the brown arrows and brown circles in Fig. 15). One possible implementation is to encode the 3 edge flags in the color channels of one triangle vertex.

4.5.4 Putting it all together

This section describes how the 4pt-hermite smooth subdivision method can be integrated into the existing realization of the PAR algorithm. Fig. 16 illustrates this integration.

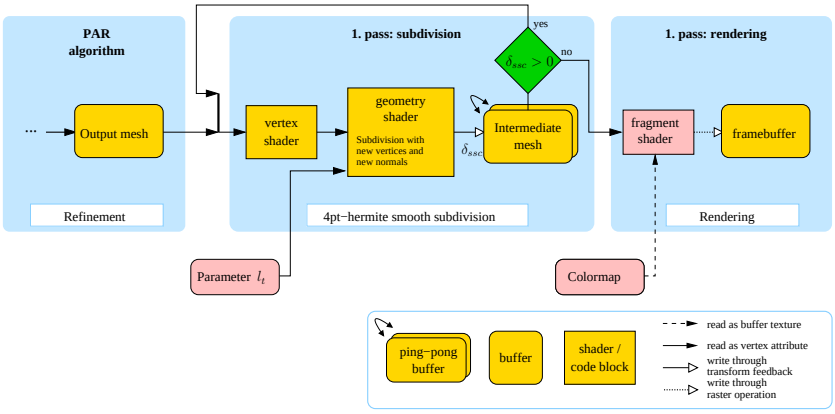

Figure 16: Integration of the 4pt-hermite subdivision into the existing solution.

A smooth subdivision unit is inserted after the refinement of the PAR terrain rendering algorithm is finished and before rendering is done. Thus it is directly feeded with the mesh outputted by the PAR method. For one smooth subdivision step beneath the input mesh and the parameter l_t no further information is necessary. Consequently, it can be done entirely in one geometry shader pass. Only in the first subdivision pass, serpentine edges have to be handled. Analogous to the PAR algorithm, recursive application is introduced by the described Transform Feedback techique [9]. The recursive subdivision is applied until for every edge in the generated mesh it is $\delta_{ssc} \leq \sigma$, thus no further detail would be added by following passes. Finally, the mesh is rendered.

4.6 Convergence analysis

This section analyses the recursive 4pt-hermite subdivision about, if the algorithm terminates after a limited number of passes.

The algorithm terminates if every edge in the mesh falls under the user-defined screen-space difference threshold σ. Consequently, it has to be proven that for an arbitrary edge, the edges produced by the smooth subdivision fall under a $\sigma > 0$ after a limited number of subdivisions. The idea is to first show that the angle between the two edge-normals decreases with every subdivision

step for the inserted edges. Second, from this property it is derived that every $\sigma > 0$ can be can be undershot by the method.

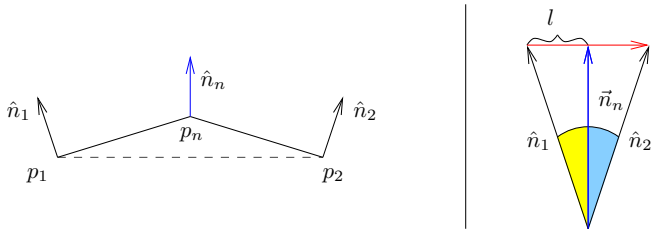

Figure 17: Left: A single subdivision step of an edge $\overline{p_1 p_2}$. Right: Linear normal interpolation.

On the left of Fig. 17 a single subdivision step of an edge $\overline{p_1 p_2}$ into the edges $\overline{p_1 p_n}$, $\overline{p_n p_2}$ is illustrated. As the serpentine case is exclusively handled in the first subdivision pass, only the curve case has to be considered in this convergence analysis. In this case the new normal is linear interpolated by $\vec{n}^n = (1-l)\hat{n}_1 + l\hat{n}_2 = \hat{n}_1 + l(\hat{n}_2 - \hat{n}_1)$. This interpolation is depicted in the right of Fig. 17, where the red vector is $(\hat{n}_2 - \hat{n}_1)$. As it is $l \in \left[\frac{1}{4}, \frac{3}{4}\right]$ the vector $\vec{n}^n$ must split the angle $\measuredangle(\hat{n}_1, \hat{n}_2)$ into two positive angles (light blue and yellow in the right of Fig. 17). Therefore, it is $\measuredangle(\hat{n}_1, \hat{n}_2) > \measuredangle(\hat{n}_1, \hat{n}_n)$ and $\measuredangle(\hat{n}_1, \hat{n}_2) > \measuredangle(\hat{n}_n, \hat{n}_2)$. That means in a single subdivision step the angle between the normals decreases.

With a decreasing angle between normal vectors, it decreases the absolute difference $\gamma = |\measuredangle(\hat{n}_1, \vec{E}) - \measuredangle(\hat{n}_2, \vec{E})|$ between the normal-edge angles at an edge's end points. Therefore, rewriting equation (9) as:

$$
\begin{aligned}
P^{new} &= \frac{1}{2}(p_1 + p_2) \\
&\quad + l_t \frac{\|\vec{E}\|}{8} \underbrace{\left(\frac{\vec{E} - \hat{n}_1 \|\vec{E}\| cos\measuredangle(\hat{n}_1, \vec{E})}{\left\| \vec{E} - \hat{n}_1 \|\vec{E}\| cos\measuredangle(\hat{n}_1, \vec{E}) \right\|} - \frac{\vec{E} - \hat{n}_2 \|\vec{E}\| cos\measuredangle(\hat{n}_2, \vec{E})}{\left\| \vec{E} - \hat{n}_2 \|\vec{E}\| cos\measuredangle(\hat{n}_2, \vec{E}) \right\|} \right)}_{\vec{\epsilon}} \\
&= \frac{1}{2}(p_1 + p_2) + l_t \frac{\|\vec{E}\|}{8} \vec{\epsilon}
\end{aligned}
\tag{15}
$$

shows that $\|\vec{\epsilon}\|$ decreases with γ in every subdivision step. The following equation (16), shows that the first new edge $\overline{p_1 P^{new}}$ resulting from the subdivision of an arbitrary edge $\overline{p_1 p_2}$ has a smaller length as the original edge.

$$
\begin{aligned}
\left\|\overline{p_1 P^{new}}\right\| &= \|P^{new} - p_1\| \\
&= \left\| \frac{1}{2}\vec{E} + l_t \frac{\|\vec{E}\|}{8} \underbrace{\left(\frac{\vec{E} - \hat{n}_1\left(\vec{E}\cdot\hat{n}_1\right)}{\|\vec{E} - \hat{n}_1\left(\vec{E}\cdot\hat{n}_1\right)\|} - \frac{\vec{E} - \hat{n}_2\left(\vec{E}\cdot\hat{n}_2\right)}{\|\vec{E} - \hat{n}_2\left(\vec{E}\cdot\hat{n}_2\right)\|} \right)}_{\vec{x}} \right\| \\
&= \left\| \frac{1}{2}\vec{E} + l_t \frac{\|\vec{E}\|}{8} \cdot \vec{x} \right\| \quad \text{with} \quad \|\vec{x}\| \in [0,2] \\
&\Rightarrow \left\|\overline{p_1 P^{new}}\right\| \in \left[0, \frac{3}{4}\|\vec{E}\|\right] \quad \text{as} \quad \left\| \frac{l_t}{8}\vec{x} \right\| \in \left[0, \frac{1}{4}\right]
\end{aligned}
\tag{16}
$$

It can be shown analogue that also the second new edge $\overline{P^{new} p_2}$ has a smaller length than the original edge. Consequently the edge-length $\|\vec{E}\|$ decreases in a single subdivision step guaranteed. As $\|\vec{\epsilon}\|$ and $\|\vec{E}\|$ decrease in every subdivision step, it follows from (15) that the distance between the new inserted points P_{new} and the interpolating midpoint $\frac{1}{2}(p_1 + p_2)$ decreases in every subdivision step for the new produced edges. Therefore, δ undershoots any value greater zero after a limited number of subdivision steps. Consequently, also after a limited number of subdivision passes the value δ_{ssc} must undershoot an arbitrary value $\sigma > 0$, which had to be shown.

4.7 Conclusion

At the beginning of this section the requirements on smooth subdivision methods in the field of terrain rendering have been presented. Based on that and an overview of existing schemes, a new smooth subdivision scheme has been designed. Guided by the pronouced requirements, the scheme can be summarized by 5 central features:

1. Limit surface properties: The limit surface interpolates the control points. It is guaranteed at least C^0-continuous and approximates in large areas C_1-continuity. With the introduced parameter l_t it is freely adjustable to what extent the limit surface approaches to the control mesh. It was proven that the scheme does not produce degenerate or near-degenerate triangles.

2. Parallel approach: The scheme has a local triangular support and can thus be executed fully parallel on a triangular input mesh.

3. Small memory footprint: In a single subdivision pass, refinement is done without any memory usage. Solely immediate meshes necessary for a recursive application of the scheme is stored in graphics memory.

4. Adaptivity: The presented method is executed adaptively in every recursive subdivision step. Only in mesh regions where a further subdivision adds detail, which can be measured by a difference in screen-space, the scheme is applied.

5. Compatibility with the PAR algorithm: The method can be comfortably integrated into the existing solution via an additional recursive geometry shader pass.

5 Results

This sections describes the results of this thesis. An overview of the practical results of both new methods (as seperate algorithms or in conjunction) is presented on the basis of synthesized images with sample data sets. Following, advisable parameter values and performance measurements with a sample implementation are presented.

First, the results of the PAR terrain rendering algorithm are overviewed. Fig. 18 gives an impression of the overall functionality of the method.

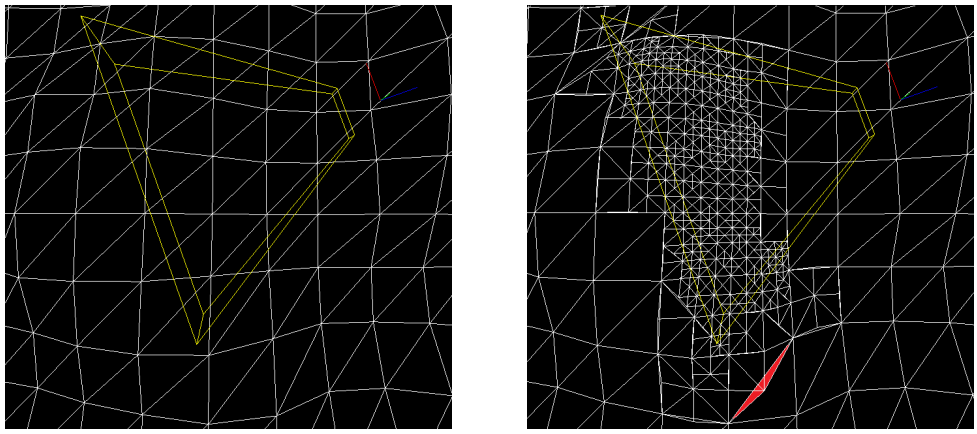

Figure 18: Refinement with the PAR method. The yellow lines represent the viewing frustum. Left: coarse input mesh. Right: refined mesh.

The refinement is done adaptively on the terrain section in the viewing frustum - exclusively on edges which exceed the screen-space error threshold of 20 pixels in this sample. Outside the viewing frustum cracks in the mesh can be noticed, as the method does not communicate forced splits to edges of triangles lying entirely outside the viewing frustum (for example the red colored one). The forced splits ensure that all subdivided triangles in the viewing frustum are allowed to differ only by one refinement level.

Fig. 19 gives an overview of the functionality of the 4pt-hermite subdivision method. On both meshes only 4 subdivision passes lead to a visually appealing smooth surface. The left two images illustrate that only one serpentine capturing pass is necessary. The right two sample images reveal that the limit surface approximates in large areas of the mesh C^1-continuity. In addition with a smooth shading, nearly no C^1-inconsistency can be noticed.

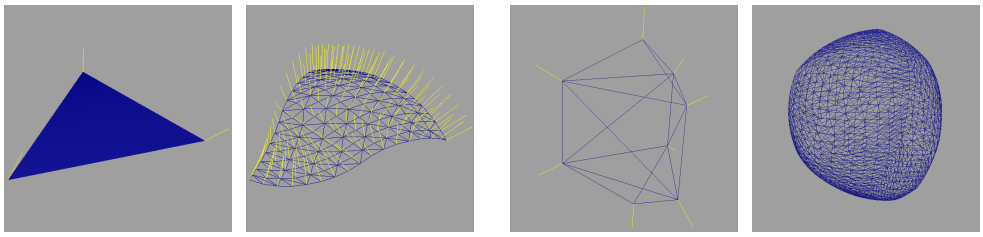

Figure 19: 4 subdivision passes of the 4pt-hermite scheme ($l_t = 0.5$). The yellow lines are the exact normal vectors. Left: Single triangle. Right: Elongated isosceles triangle with pyramids at its ends.

The adaptive application only of the 4pt-hermite smooth subdivision scheme is illustrated in the left of Fig. 20. As forced splits do not have to be communicated over sequences of adjacent triangles, triangles with a sharing edge can have differences larger than one in their refinement levels (see, for example, the red colored one).

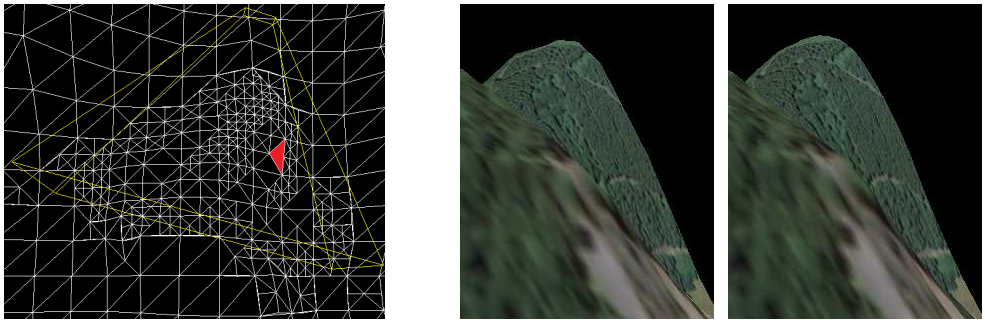

Figure 20: Left: adaptive application of the 4pt-hermite scheme. Right: Application of the PAR algorithm without/with further smooth subdivision ($l_t = 1.0$).

Finally, one profit of using both methods in connection is depicted in the right of Fig. 20. The mesh in the first figure was refined with the PAR algorithm with an allowed screen-space error of 5 pixels. A smooth silhouette is achieved by applying to that mesh the 4pt-hermite smooth subdivision up to a screen-space difference of 0.3 pixels which is depicted in the second image.

Parameters

The freely adjustable parameters of the presented methods are the screen-space thresholds, the coarse input mesh and the parameter l_t. If accuracy is the primary goal, the screen-space error threshold of the PAR algorithm should be chosen as small as available hardware resources allow. In the tests with the research implementation of the method (test environment is described in the next paragraph), values in the range from 0.5 to 1 pixel represented a good

compromise between accuracy and the number of refinement passes, which still enable interactivity. Due to its highly parallel design, the screen-space difference of the 4pt-smooth-subdivision method could be set to values in the range from 0.1 to 0.5 pixels without loosing an interactive framerate. To eliminate under-refined triangles, the resolution of the coarse input mesh should be adjusted according to the frequencies measured in the terrain heightmap. In general, on a very coarse input mesh, using the presented shader-based approach is not very effective as due to the low number of primitives, the parallelism of the method cannot be exploited. Therefore, the coarse mesh should be prerefined - 20×20 turned out to be a compromise in the experimental implementation. Finally, as described in Sec. 4.5 choosing $l_t = \frac{1}{2}$ delivers a good tradeoff between the smoothness and a minimal distance of the limit surface to the control mesh.

Performance measurements

In this paragraph a scale of both presented methods' interactivity and scalability is given, which allows to classify the performance of the algorithms in comparison to other methods. Measurements were done on a PC with an Intel Core 2 Duo E6850 processor, 3 Gb main memory and a Nvidia GeForce GTX 285 graphics card. A sample heightmap of the city of Siegen with a size of 2025×2025 pixels and a quadratic 20×20 coarse input mesh was used. The complete refinement process with a screen-space error of 1.0 pixels, a screen-space difference of 0.3 pixels and $l_t = 0.5$ was restarted for every new frame. In this testing environment, random flights over the scene resulted in framerates not under 30 FPS. Thus the presented method delivers interactivity with current graphics hardware, while not loosing a high accuracy.

To examine the scalability of the research implementation, a method described in Zhou et al. [23] was adopted. Using the NVStrap-driver in RivaTuner, the number of stream-processors available on a Nvidia Geforce 8800 GTX GPU have been successively decreased, while the framerate was measured. As expected due to their parallel design, both methods are scalable to a great extent. A decrease of over $\frac{1}{3}$ (PAR) / nearly $\frac{1}{2}$ (4pt-hermite scheme) of the framerate could be measured at two consecutive halvings of the number of available processors.

6 Conclusion

Two new methods for effective parallel terrain rendering have been presented.

First, at the beginning of this thesis, the requirements on terrain rendering algorithms have been analysed. Based on that, a new parallel adaptive refinement method on the GPU has been developed. The algorithm provides adaptive, view-dependent, continuous levels-of-detail while generating meshes consisting exclusively of right-isosceles triangles. It produces guaranteed error bounds in screen-space.

Second, an overview of existing smooth subdivision schemes has been exploited for the design of a new interpolatory, local triangular, smooth subdivision scheme. The method is adaptive and highly parallel, while it is still guaranteed that no degenerate triangles are produced.

Plugging both methods together (besides they can also be used as seperate algorithms) results in a fast effective parallel terrain rendering algorithm which provides freely adjustable accuracy while enabling an infinite continuous range of levels-of-detail and a visually appealing surface.

In the according conclusion sections, each algorithm has been evaluated, if it meets the pronounced requirements. Finally, problems, advisable parameters and performance measurements have been presented.

Due to its high accuracy, interactivity and CLOD-approach, the complete method is suited for various applications ranging from digital land surveying, demanding primarily a high accuracy, to entertainment applications, which call for an interactive visualization.

Future work

To solve the problem of under-refined triangles, it was suggested to adjust the resolution of the coarse input mesh according to the frequencies measured in the heightmap. Consequently, a very interesting target for future research would be an algorithm which addresses this task.

Considering the design of parallel algorithms in general, a more flexible parallel hardware structure as on current graphics hardware would be desirable. A central challenge for the PAR algorithm and other not fully parallelizable problems is that the parallel processing units cannot exchange information during one execution pass. If a program executed in a single parallel processing unit could stall another unit for several cycles, by using a shared memory lump the communication of information between these units would be possible in one parallel execution pass. It would be interesting to see to what extent such a softening of the parallel hardware architecture would enable a parallel execution of the class of not fully parallelizable problems.

Finally, the effects of inserting more than one vertex at an edge in the PAR terrain rendering algorithm could be investigated. Especially the impact on performance, with expected lesser refinement passes and larger calculation time for all possible tesselations, would be interesting.

References

[1] Ulrich, Thatcher: *Continuous LOD Terrain Meshing Using Adaptive Quadtrees*. http://www.gamasutra.com/features/20000228/ulrich_pfv.htm. Version: 2009.06.15

[2] Duchaineau, Mark ; Wolinsky, Murray ; Sigeti, David E. ; Miller, Mark C. ; Aldrich, Charles ; Mineev-Weinstein, Mark B.: ROAMing Terrain: Real-time Optimally Adapting Meshes. In: *IEEE Visualization '97* (1997)

[3] Lindstrom, P ; Pascucci, V: Visualization of large terrains made easy. In: *IEEE Visualization*, 2001, S. 363–370

[4] contributors: *ESA GOCE mission*. http://www.esa.int/esaMI/GOCE/SEMDU2VHJCF_0.html. Version: 2009.06.15

[5] Schmiade, Timo: *Adaptive GPU-based terrain rendering*, University of Siegen, Diplomarbeit, 2008

[6] contributors: *Virtual Terrain Project*. http://www.vterrain.org/. Version: 2009.06.15

[7] Ericson, Christer: *Minimum bounding circle (sphere) for a triangle (tetrahedron)*. http://realtimecollisiondetection.net/blog/?p=20. Version: 2009.06.15

[8] contributors: *Nvidia SDK 10.5 Example "Transform Feedback Fractal"*. http://developer.download.nvidia.com/SDK/10.5/opengl/samples.html. Version: 2009.06.15

[9] contributors: *OpenGL Transform Feedback Extension*. http://www.opengl.org/registry/specs/EXT/transform_feedback.txt. Version: 2009.06.15

[10] Zorin, Denis ; Schröder, Peter: *Subdivision for Modeling and Animation*. SIGGRAPH 2000 Course Notes, 2000

[11] Farin, Gerald: Curves and Surfaces for Computer Aided Geometric Design, Morgan Kaufmann, 2001, S. 102–106

[12] Dyn, Nira ; Levin, David ; Gregory, John A.: A 4-point interpolatory subdivision scheme for curve design. In: *Computer Aided Geometric Design* 4 (1987), Nr. 4, S. 257–268

[13] Dyn, Nira ; Levin, David ; Gregory, John A.: A Butterfly Subdivision Scheme for Surface Interpolation with Tension Control. In: *ACM Transactions on Graphics* 9 (1990), S. 160–169

[14] Karbacher, Stefan ; Seeger, Stephan ; Häusler, Gerd: A Non-linear Subdivision Scheme for Triangle Meshes. In: *Vision, Modeling and Visualization 2000*, Akademische Verlagsgesellschaft, 2000, S. 163–170

[15] Kobbelt, Leif: Interpolatory Subdivision on Open Quadrilateral Nets with Arbitrary Topology. In: *Computer Graphics Forum*, 1996, S. 409–420

[16] Labsik, Ulf ; Greiner, Günther: Interpolatory $\sqrt{3}$-Subdivision. In: *Computer Graphics Forum* 19 (2000), Nr. 3

[17] Li, Guiqing ; Ma, Weiyin ; Bao, Hujun: Interpolatory $\sqrt{2}$-Subdivision Surfaces. In: *GMP*, 2004, S. 185–194

[18] Ling, Ruotian ; Luo, Xiaonan ; Chen, Ren ; Huang, Wanmin: Interpolatory Ternary Subdivision for Triangular Meshes with Arbitrary Topology. In: *ICAT Workshops*, 2006, S. 5–10

[19] Mao, Zhihong ; Ma, Lizhuang ; Tan, Wuzheng: A Modified Nielson's Side-Vertex Triangular Mesh Interpolation Scheme. In: *ICCSA (1)*, 2005, S. 776–785

[20] Nielson, Gregory: A Transfinite, Visually Continuous, Triangular Interpolant. In: Farin, G. (Hrsg.): *Geometric Modeling: Algorithms and New Trends*, SIAM, 1987, S. 235–246

[21] Boubekeur, Tamy ; Schlick, Christophe: Approximation of subdivision surfaces for interactive applications. In: *SIGGRAPH '07: ACM SIGGRAPH 2007 sketches*, 2007, S. 25

[22] Vlachos, Alex ; Peters, Jörg ; Boyd, Chas ; Mitchell, Jason L.: Curved PN Triangles. In: *In Symposium on Interactive 3D Graphics*, 2001, S. 159–166

[23] Zhou, Kun ; Hou, Qiming ; RuiWang ; Guo, Baining: Real-time KD-tree construction on graphics hardware. In: *ACM Trans. Graph.* 27 (2008), Nr. 5, S. 1–11