Computers & Geosciences 41 (2012) 147–155

Contents lists available at SciVerse ScienceDirect

Computers & Geosciences

journal homepage: www.elsevier.com/locate/cageo

Real-time tessellation of terrain on graphics hardware

Oscar Ripolles [a,*], Francisco Ramos [b], Anna Puig-Centelles [b], Miguel Chover [b]

[a] Universitat Politecnica de Valencia, Valencia, Spain
[b] Universitat Jaume I, Castellón, Spain

ARTICLE INFO

Article history:
Received 21 February 2011
Received in revised form
15 July 2011
Accepted 19 August 2011
Available online 10 September 2011

Keywords:
Terrain simulation
Tessellation
Level of detail
Real-time rendering
GPU

ABSTRACT

Synthetic terrain is a key element in many applications, which can lessen the sense of realism if it is not handled correctly. We propose a new technique for visualizing terrain surfaces by tessellating them on the GPU. The presented algorithm introduces a new adaptive tessellation scheme for managing the level of detail of the terrain mesh, avoiding the appearance of *t*-vertices that can produce visually disturbing artifacts. Previous solutions exploited the geometry shader's capabilities to tessellate meshes from scratch. In contrast, we reuse the already calculated data to minimize the operations performed in the shader units. This feature allows us to increase performance through smart refining and coarsening. Finally, we also propose a framework to manage large DEMs as height maps.

1. Introduction

In recent years the research area of procedural modeling has been the focus of much effort. The latest work tries to take advantage of the new graphics hardware technology, making it possible for the geometry to be generated at rendering time in the graphics card itself. Thus, instead of specifying the details of a 3D object, we provide some parameters for a procedure that will create the object.

In the field of computer graphics, tessellation techniques are often used to divide a surface into a set of polygons. Thus, we can tessellate a polygon and convert it into a set of triangles or we can tessellate a curved surface. These approaches are typically used to amplify coarse geometry. Programmable graphics hardware has enabled many surface tessellation approaches to be migrated to the GPU, including isosurface extraction (Buatois et al., 2006), subdivison surfaces (Shiue et al., 2005), NURBS patches (Guthe et al., 2005), and procedural detail (Bokeloh and Wand, 2006; Boubekeur and Schlick, 2005). In this paper we analyze the possibilities offered by GPU-based tessellation techniques for terrain visualization.

For many decades terrain simulation has been the subject of research, and there are many solutions in the literature to its realistic and interactive rendering. Most of these solutions simulate terrain as an unbounded surface that is represented in the synthetic environment as a height map, which is a regularly spaced two-dimensional grid of height coordinates. These grids can later be processed by modeling software or a rendering engine to obtain the 3D surface of the desired terrain.

Some authors have criticized the use of height fields, as these data structures store only one height value for any given (x,y) pair. In specific cases such as caves or complex terrain formations, it may be possible to have more than one height value for each position. We will not consider these complex formations and, thus, the use of a squared height map will still be adequate.

We introduce a new adaptive tessellation scheme for terrain that works completely on the GPU. The main feature of the framework that we are presenting is the possibility of refining or coarsening the mesh while maintaining *coherence*. By coherence we refer to the reuse of information between changes in the level of detail. In this way, the latest approximation extracted is used in the next step, optimizing the tessellation process and improving performance. We also propose a simple framework to manage large terrains using height maps.

The rest of the paper is structured as follows. Section 2 presents the state of the art in terrain simulation. Section 3 thoroughly describes our tessellation technique. Section 4 offers some results on the technique presented and, last, Section 5 presents some conclusions on the techniques developed and outlines future work.

2. Related work

Digital terrain models (DTMs) are usually represented and managed by means of regular or irregular grids. The reader is

* Corresponding author. Tel.: +34 963877000x88442.
E-mail addresses: oripolles@ai2.upv.es (O. Ripolles), francisco.ramos@uji.es (F. Ramos), apuig@uji.es (A. Puig-Centelles), chover@uji.es (M. Chover).

0098-3004/$ - see front matter © 2011 Published by Elsevier Ltd.
doi:10.1016/j.cageo.2011.08.025

referred to recent surveys for a more in-depth review of these methods (Pajarola and Gobbetti, 2007; Rebollo et al., 2004).

2.1. Regular grids

The most common regular structures are quadtrees and binary trees (bintrees). These structures with regular connectivity are suitable for terrain, as the input data usually come as a grid of values. Regular approaches have produced some of the most efficient systems to date (Pajarola and Gobbetti, 2007).

Quadtrees are found in the literature in many papers with CPU-based solutions (Lindstrom et al., 1996; Pajarola, 1998) as well as GPU-based approximations (Schmiade, 2008). This latter approach proposed a tessellation algorithm on the GPU, although the pattern selection process was very complex.

The ROAM method (real-time optimally adapting meshes) (Duchaineau et al., 1997) is a widely known method based on the use of bintrees. They use two priority queues to manage split and merge operations, obtaining high accuracy and performance. As an attempt to improve this solution, in Apu and Gavrilova (2004) the authors eliminated the priority queue for merges to exploit frame-to-frame coherence.

Some authors proposed using bintrees where each node contains, instead of a single triangle, a patch of triangles (Levenberg, 2002; Pomeranz, 2000). Algorithms such as (Cignoni et al., 2003; Schneider and Westermann, 2006) batched the triangular patches to the graphics hardware. The batched dynamic adaptive mesh (BDAM) proposed in Cignoni et al. (2003) used triangle strips to increase performance, although it was based on complex data structures and costly processes that still produced popping artifacts. From a different perspective; Larsen and Christensen (2003) used patches of quads to manage terrain rendering on the GPU. Later Schneider and Westermann (2006) reduced bandwidth requirements by simplifying the mesh and using progressive transmission of geometry. Recently, in Bosch et al. (2009) the authors proposed the use of precalculated triangle patches to develop a GPU-intensive solution.

The projected grid concept offered an alternative way to render displaced surfaces with high efficiency (Johanson, 2004). The idea was to create a grid with vertices that were evenly spaced in post-perspective camera space. This representation provided spatial scalability and a fully GPU-based implementation was described. Schneider et al. (2006) used the projective grid method to render infinite terrain in high detail. They generated the terrain in real time on the GPU by means of fractals. The work in Livny et al. (2008) proposed using ray tracing to guide the sampling of the terrain, this being a technique able to produce both regular and irregular meshes.

As an improvement over binary trees, Losasso and Hoppe (2004) introduced *geometry clipmaps*, caching the terrain in a set of nested regular grids. These grids were stored as vertex buffers, and mipmapping was applied to prevent aliasing. As vertex buffers cannot be modified on the GPU, this approach was later improved by using vertex textures to avoid having to use the CPU to modify the grids (Asirvatham and Hoppe, 2005). This work was also extended to handle spherical terrains (Clasen and Hege, 2006).

Last, it is worth mentioning that bintree hierarchies are also useful for decompressing terrain surfaces on the GPU. In this sense, Lindstrom and Cohen (2010) presented a fast, lossless compression codec for terrains on the GPU and demonstrated its use for interactive visualization.

2.2. Irregular grids

Irregular grids are commonly known as TINs (triangulated irregular networks) and represent surfaces through a polyhedron with triangular faces. These solutions are less constrained triangulations of the terrain and, in general, need fewer triangles, although their algorithms and data structures tend to be more complex.

Hoppe (1998) proposed specializing his view-dependent progressive mesh (VDPM) framework for arbitrary meshes that represent terrain. With more intense GPU exploitation, Dachsbacher and Stamminger (2004) proposed a costly procedural solution that needs three rendering passes to obtain the geometry.

Delaunay triangulation is one of the main techniques used to create the terrain mesh. In computational geometry, a Delaunay triangulation for a set of points is a triangulation where no point is inside the circumcircle (circle that passes through all the vertices of the triangle) of any generated triangle (Delaunay, 1934). This triangulation has been widely used in terrain solutions (De Berg et al., 2008; Rabinovich and Gotsman, 1997). The main problem with Delaunay triangulation is that it relies on smoothing morphing between two triangulations generated in two successive frames, but the triangulations may be very different. As an improvement, Cohen-Or and Levanoni (1996) proposed a solution that involved blending between two levels of Delaunay triangulation without adding more triangles. More recently, Liu et al. (2010) proposed a new technique where points from a DEM are initially given an importance value in order to guide adaptive triangulation in real time. Their proposal allowed smooth morphing and tried to eliminate very small triangles that could produce visual disturbances.

As a conclusion, we can note that techniques based on irregular grids tend to be more complex and less suitable for GPU computations.

3. Our GPU-based tessellation scheme

In this paper we propose a new adaptive tessellation algorithm that works completely on the GPU. Moreover, this algorithm is able to offer view-dependent approximations where more detail is added in areas of interest. Our algorithm will be based on bintrees, creating the hierarchy on the GPU using some specific equations. As mentioned above, there have been other proposals with similar aims, although our scheme is easier to implement, while still robust and efficient.

A successful tessellation algorithm is based on the selection of the most suitable tessellation patterns to amplify the triangles. These patterns affect the algorithm chosen to refine and coarsen the geometry. As our aim is to process the mesh in a *geometry shader*, each triangle is to be processed separately and in parallel. Thus, it will be necessary to develop a technique to alter the geometry of the different triangles without any communication between them. Moreover, the algorithms must ensure that no cracks or holes appear on the surface mesh.

In the remainder of this section, we will address the selection of the patterns and also the algorithms to manage the terrain surface.

3.1. Tessellation patterns

It is possible to find different proposals of tessellation patterns in the literature. Among them, we have selected the seven patterns presented in Schmiade (2008). Fig. 1 presents, on the left side, an initial rectangular triangle whose hypotenuse and catheti (or *legs*) are labeled as H, C_1, and C_2 respectively. Next, the seven tessellation patterns are presented, with the edges of the original triangle that need refinement depicted in red.

These patterns ensure that no *t-vertices* are produced. A *t-vertex* appears after a tessellation step when two edge junctions make a *t-shape* (McConnell, 2006). To clarify the appearance of