Free Adaptive Tessellation Strategy of Bézier Surfaces

Raquel Concheiro[1], Margarita Amor[1], Montserrat Bóo[2] and Emilio J. Padrón[1]
[1]*Dept. Electrónica e Sistemas, Universidade da Coruña, A Coruña, Spain*
[2]*Dept. Electrónica e Computación, Universidade de Santiago de Compostela, Santiago de Compostela, Spain*

Keywords: Bézier Surfaces, Adaptive Tessellation, GPU, Real-Time Rendering.

Abstract: Rendering of Bézier surfaces is currently performed by tessellating the model on the GPU and rendering the highly detailed triangle mesh. Whereas non-adaptive strategies apply the same tessellation pattern to the whole surface resulting in a uniform tessellation of the patch, adaptive approaches make it possible to reduce the number of triangles generated without a loss of quality. However, the most usual approaches to adaptive tessellation have little flexibility and do redundant computations and memory accesses, as each sample is independently evaluated in the Domain Shader of the DirectX11 pipeline. In this paper an adaptive tessellation technique based on the exploitation of the spatial coherence (ESC) data within each surface is presented. The GPU implementation of this technique is simple and efficient and, as consequence, the tessellation of complex models can be performed in real-time. The analysis of the GPU performance and limitations for different adaptive degree of the tessellation performed suggest innovations in future graphics card generations for supporting a larger degree of adaptivity without a penalty.

1 INTRODUCTION

Bézier surfaces are one of the most useful primitives employed for high quality modeling in CAD/CAM tools and graphics software. The excellent mathematical and algorithmic properties (Rogers, 2001), combined with successful industrial applications, have contributed to the popularity of the Bézier representation.

As graphics cards have been traditionally optimized for the processing of triangles, the parametric models are usually tessellated into triangle meshes for the rendering process. There are two main approaches for the tessellation in terms of the regularity of the resulting triangle mesh: non-adaptive and adaptive. With a non-adaptive strategy, the level of tessellation is selected on the basis of the desired resolution and the surface is regularly sampled according to it. This could produce an excessive amount of triangles without increasing the quality of the final mesh, in addition to not generate enough triangles in highly complex areas. Furthermore, overtessellation increases the surface evaluation and rasterization process with a significant performance penalty. Nevertheless, an adaptive strategy adapts the tessellation pattern depending on some features of the surface. This reduces the number of triangles to be rendered as it generates less triangles in some areas according to a specific quality/speed criteria.

Most of the existing proposals for tessellating parametric surfaces on the GPU are based on applying non-adaptive strategies to Bézier patches, either selecting a tessellation level for each surface patch (Guthe et al., 2005) or for a set of patches (Dyken et al., 2009; Concheiro et al., 2010). Thus, the patches are uniformly subdivided once the resolution has been chosen, with only some specific modifications in the edges to prevent holes or cracks between neighboring patches. Other GPU tessellation proposals (Eisenacher et al., 2009; Schwarz and Stamminger, 2009) follow a GPGPU strategy (General-Purpose Computation on GPU) using CUDA (NVIDIA, 2008) and also pursue adaptive tessellation at patch level.

The scenario changed with the presentation of a new GPU tessellation engine in the pipeline of DirectX11. Three new stages (Hull Shader, Tessellator and Domain Shader) were introduced to support programmable tessellation, and novel approaches to the tessellation of parametric surfaces were presented exploiting the new pipeline. The new tessellation unit included (Ni and Castaño, 2009) offers a high performance solution, but with reduced flexibility in its current implementation, as it applies either a fixed or a semi-regular tessellation pattern. Specifically, the DX11 tessellation unit generates a triangle mesh from six independent tessellation factors, one for each domain edge and two for the internal axes of the patch. Once these factors are set, the edges and the inside of the patch are uniformly tessellated in the parametric domain. In (Yeo et al., 2012) an implementation of a tight estimator of the variance between the screen projection of the exact surface and its triangulation is proposed using the GPU tessellation engine. The new tessellation unit also supports regular fractional tessellation, and some works, such as (Munkberg et al., 2008; Amresh and Fünfzig, 2010), add a non-uniform, fractional tessellation to achieve a more uniform screen-space triangle area. Nevertheless, this scheme does not provide enough support for free adaptive tessellation, and the independent processing of primitives requires special care by application developers to prevent cracks. A modification of the DX11 hardware structure is proposed in DiagSplit (Fisher et al., 2009) to allow a higher adaptability, though still keeping a uniform strategy per surface patch. DiagSplit performs a non-uniform tessellation along an edge by applying a recursive process: first, the edge is partitioned at its parametric midpoint, and then seven factors are used, one for each edge of the two subpatches. This proposal, however, is far to get an adaptive tessellation inside the patch. Another approach detailed in (Concheiro et al., 2011) proposes an adaptive tessellation with a simpler scheme where DirectX 11 capabilities are not needed. As this proposal is based on a simpler pipeline without the three stages introduced by DirectX 11, the adaptive tessellation is performed on the Geometry Shader exploiting the generation of several primitives for each invocation of the shader.

Broadly speaking, the tessellation of surfaces in the DX11 pipeline is known for the lack of flexibility of the sampling schemes in the tessellation unit, as well as for the independent evaluation of each sample in the Domain Shader. Therefore, the Domain Shader is computed once for each vertex besides the corresponding invocations of Hull Shader and Tessellation. As the amount of shader invocations is increased, the power consumption is also increased, being the energy consumption a very significant factor in the design of new GPU pipelines, specifically on handheld GPUs since they are supplied by batteries.

In this paper a different GPU approach is presented, based on the exploitation of spatial coherence (ESC) of data within each surface with the goal of achieving a fully adaptive and flexible tessellation. ESC pipeline is based on the utilization of a surface as the input primitive of an adaptive tessellation shader, focusing on having a unique and more computationally complex stage that performs the tessellation and evaluation of a surface. This approach needs fewer shader invocations and allows an optimal computation of the evaluation of all the samples in a surface, minimizing memory accesses as well. All this also results in a considerable reduced power consumption.

The main purpose of this work is to analyze a novel solution to improve the tessellation capabilities of the current rendering pipeline, rather than coding an optimized implementation. With this aim we suggest several modifications in the pipeline to get a coarser grain parallelism based on surfaces instead of the usual sample-based approach. Anyway, as it is shown in the results section, good results in terms of quality and performance have been achieved by the implementation on current GPUs of this method we have done to test its feasibility.

This paper is organized as follows: In Section 2.1 an introduction to the Bézier representation is presented. In Section 2 the pipeline proposal is introduced, and in Section 3 the implemented adaptive strategy is described. Finally, Section 4 shows the experimental results and Section 5 highlights the main conclusions of this work.

2 ESC PIPELINE FOR BÉZIER SURFACES

ESC, a new pipeline for an efficient adaptive tessellation of Bézier surfaces, is introduced in this section. First, a brief introduction to the Bézier representation is presented. For reasons of clarity, Bézier curves are first introduced and then the description is extended to the Bézier surfaces. A more detailed review can be found in (Piegl and Tiller, 1997; Rogers, 2001). Next, the proposed pipeline is described, whereas the details of the tessellation algorithm are detailed in the following section.

2.1 Bézier Surfaces

A Bézier curve is specified by giving a set of coordinate positions, called control points, which indicate the general shape of the curve. Mathematically, a parametric n degree Bézier curve is defined by:

$$P(t) = \sum_{i=0}^{n} B_i J_{n,i}(t), \;\; 0 \leq t \leq 1 \tag{1}$$

where B_i are the control points and $J_{n,i}$ is the *i-th* degree-n Bernstein basis function defined by:

$$J_{n,i}(t) = \binom{n}{i}(1-t)^{(n-i)}t^i \tag{2}$$

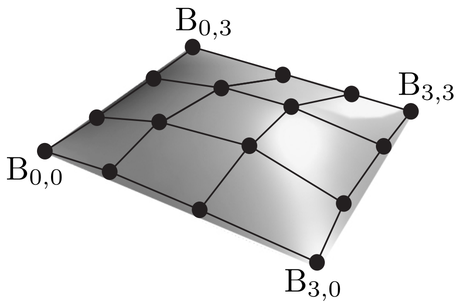


Figure 1: Bicubic Bézier surface ($n = m = 3$).

where n is the degree of the Bernstein basis function. These functions decide the extent to which a control point controls the surface at a particular parametric value t. Note that the first and last control points are coincident with the end points of the curve, that is, $P(0) = B_0$ and $P(1) = B_n$.

The equation for a Bézier curve can be also expressed in matrix form:

$$P(t) = [T][N][G] \tag{3}$$

where $[T] = [t^n \;\; t^{n-1} \dots \; t^1 \;\; t^0]$, the geometry of the curve is represented as $[G]^T = [B_0 \;\; B_1 \dots B_n]$, and the $[N]$ matrix is defined by:

$$\begin{bmatrix} \binom{n}{0}\binom{n}{n}(-1)^n & \binom{n}{1}\binom{n-1}{n-1}(-1)^{n-1} & \dots & \binom{n}{n}\binom{n-n}{n-n}(-1)^0 \\ \dots & \dots & \dots & \dots \\ \binom{n}{0}\binom{n}{1}(-1)^1 & \binom{n}{1}\binom{n-1}{0}(-1)^0 & \dots & 0 \\ \binom{n}{0}\binom{n}{0}(-1)^0 & 0 & \dots & 0 \end{bmatrix}$$

For example, for $n = 3$ the matrix form is:

$$P(t) = [T][N][G] =$$
$$[t^3 \;\; t^2 \;\; t^1 \;\; 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{bmatrix} \tag{4}$$

Likewise, the shape of a (n,m)-degree Bézier surface is controlled by a set of control points through the equation:

$$Q(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_{i,j} J_{n,i}(u) J_{m,j}(v), \quad 0 \le u,v \le 1 \tag{5}$$

where $J_{n,i}(u)$ and $J_{m,j}(v)$ are the Bernstein basis functions in the u and v parametric directions and $B_{i,j}$ are the vertices of a polygonal control net. Again, the number of control points in the u and v directions are $n+1$ and $m+1$ respectively. As an example, Figure 1 shows a bi-cubic Bézier surface, $n = m = 3$. In matrix form, a Bézier surface is given by:

$$Q(u,v) = [U][N][B][M]^T[V] \tag{6}$$

For the specific case of a bi-cubic Bézier surface, the matrix form is given by:

$$Q(u,v) = [u^3 \;\; u^2 \;\; u \;\; 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$
$$\begin{bmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix} \tag{7}$$

2.2 ESC Pipeline

Figure 2 schematically depicts the structure of this proposal, whose goal is to reduce the number of triangles of the final mesh while keeping the quality of the resulting image. The proposed solution is based on a 3-step programmable pipeline: first, a fixed tessellation pattern is computed to guide the adaptive procedure for the patch; next, the new vertices obtained from the first step are conditionally inserted by applying a set of heuristics consisting of tests local to the triangle; finally, a specific scheme is employed to reconstruct the mesh by processing the inserted vertices.

The first step carries out a non recursive procedure based on the utilization of a fixed tessellation pattern to guide the tessellation. The level of resolution of each object depends on the camera position and determines the refinement degree of the surface. Once the level of resolution is selected, only the positions of the uniform tessellation pattern are evaluated for their conditional insertion.

The tessellation patterns are employed to guide the adaptive tessellation in such a way that the new vertices can only be inserted in the candidate positions specified by the patterns. The second step decides which candidate vertices are really inserted as a result of a tessellation test (*Local Test* in Figure 2).

The evaluation of all samples of each surface takes advantage of the constant result of $[N][B][M]^T$ for every point on the surface and that the different control points $[B]$ are accessed only once, transforming the Equation 6 in

$$Q(u,v) = [U][NBM][V] \quad 0 \le u,v \le 1 \tag{8}$$

with $[NBM] = [N][B][M]^T$, whereas the current DX11 tessellation proposal accesses to $[B]$ and computes $[NBM]$ for each sample on the surface.

The last step of the adaptive tessellation algorithms is the reconstruction of the mesh from the set of vertices finally inserted; i.e. once the new vertices

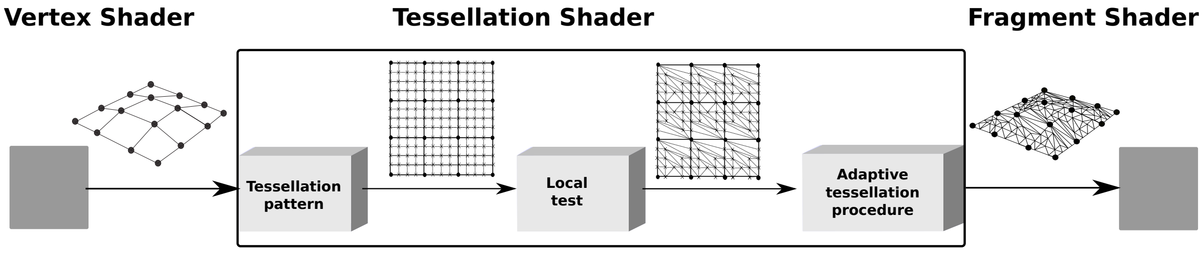


Figure 2: Structure of the ESC tessellation proposal.

have been conditionally added, all the vertices (old and new) have to be organized and connected to reconstruct the final mesh assuring no cracks or holes in the result (*Tessellation Procedure* in Figure 2). The meshing scheme is simple and a triangle strip is directly generated.

Regarding the test selected in this work to guide the adaptive tessellation (Local test), a distance-based heuristic was chosen. The objective of an adaptive tessellation is to generate detailed structures only on those areas where a high resolution is required while keeping a coarse mesh in those areas where a higher tessellation results only in a slight increment in the quality of the final image. In (Concheiro et al., 2011) a set of different tests local to the edges are presented. Among them, the *Distance test* was selected for the analysis and algorithms comparison performed in this work.

This test analyzes the distance between the control mesh and the Bézier surface. Specifically, the distance between a sampling point on the control mesh and the corresponding point on the Bézier surface is evaluated. If the distance is small enough, the mesh is considered a good approximation of the surface, so no vertex is inserted. On the contrary, if the distance is larger than a threshold a new vertex is introduced as this will mean an increase in the quality of the final image. The test is given by:

$$distance = [|V_S - V_B| > \mu] \tag{9}$$

where V_B are the coordinates of the candidate vertex V_B on the Bézier surface, V_S are the coordinates of the corresponding sampling point on the control mesh and μ is a quality threshold selected on the basis of the quality/timing requirements of the application. The *Distance test* achieves good results in terms of quality of the final mesh and has a reduced complexity and, in this sense, is a good candidate to guide the adaptive tessellation.

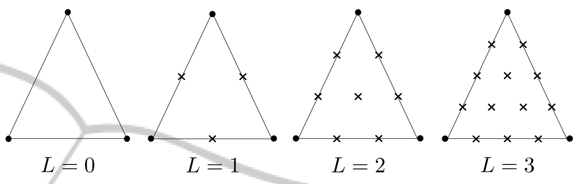


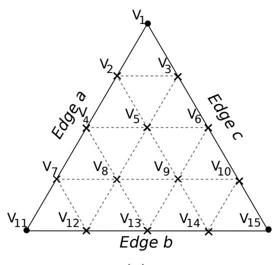
Figure 3: Triangle patterns for $L = 0, 1, 2, 3$.

3 ADAPTIVE TESSELLATION STRATEGY

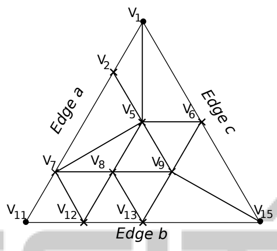
Although the versatility of ESC pipeline makes possible the use of different tessellation algorithms, the strategy chosen in this work to perform an adaptive tessellation of the input surfaces is described in this section. The procedure starts by approximating the surface with a coarse mesh and performing the adaptive tessellation of each coarse triangle. Specifically, the control point mesh is partitioned into $N_u \times N_v$ cells of size $\frac{1}{N_u} \times \frac{1}{N_v}$ where two adjoining triangles are generated per cell. The implementation employs $N_u = N_v = 3$ cells so that eighteen coarse triangles are generated per Bézier patch. This number of triangles is directly determined by the use of Bicubic surfaces.

A level of resolution L is assigned to the patch and all triangles of the patch would be subdivided by employing a unique uniform tessellation pattern. Examples of tessellation patterns for different values of L are depicted in Figure 3. The tests are only applied in the candidate positions located in the original edges of the coarse triangle. Finally, if a vertex is inserted in the edge, the insertion is also performed along the row in all the candidate positions inside the triangle.

An example of the vertex insertion procedure is depicted in Figure 4. Figure 4a shows an example of tessellation pattern for the adaptive proposals, respectively. A single level of resolution $L = 3$ was selected but three different tessellation level $L = 1$, $L = 2$ and $L = 3$ can be applied to each edge. Figure 4b depicts the result of tessellation once the insertion decisions have been performed for the adaptive algorithms, respectively. The test was applied on the original triangle edges. As will be detailed in this section, this



(a)



(b)

Figure 4: Adaptive tessellation (a) Adaptive pattern and (b) Adaptive tessellation.

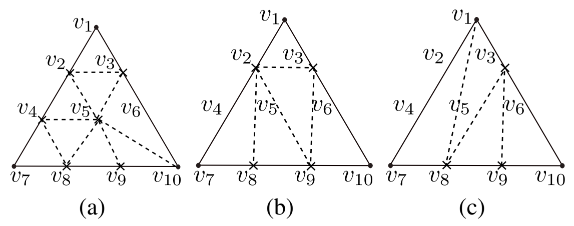


Figure 5: Examples of semi adaptive tessellations (a) No empty rows (b) Empty row, upper row with no missing vertex (c) Empty row, upper row with a missing vertex.

different test application procedure reduces the complexity not only of the test phase but also of the meshing scheme employed to reconstruct the final triangle mesh.

Once the insertion decisions have been taken a meshing procedure to connect the vertices is performed. Triangles are generated by connecting vertices in two consecutive rows of vertices, larger triangles connecting vertices in non-consecutive rows have to be generated in those locations where vertices are not inserted.

Once the subdivision tests are performed, the resulting inserted vertices are organized into a set of lists and the efficient management of this information permits the reconstruction of the final mesh in a direct way. A triangle strip is defined by a sorted list of vertices:

$$TS = \{v_1, v_2, \cdots, v_{Nt}\}$$

where each triangle is defined by three sequential vertices, with an overlap of two vertices between two consecutive triangles. As an example the $i-th$ triangle is defined by:

$$\triangle v_i v_{i+1} v_{i+2}$$

while the $(i+1)-th$ triangle is defined by:

$$\triangle v_{i+1} v_{i+2} v_{i+3}$$

The basic idea of the representation and reconstruction algorithm is the organization of the resulting triangles in rows and their representation as triangles strips. The regular triangle strip structure is broken only in those positions where no vertices are inserted. This happens either in a position in the edge of the coarse triangle or in a full row of vertices if both vertices in the extreme positions are missing. For the sake of clarity, we shall start by analyzing the case when only one of the two extreme vertices of the row are missing.

The methodology we propose is based on the utilization of the *TS* structure that would be generated with a non-adaptive tessellation as basis for the representation. This TS list is updated with the utilization of a *Virtual Vertex* (VV) for the substitution of missing vertices. More specifically, each non-inserted vertex on the edge of the triangle is substituted, in the TS representation by the closest vertex located in the same edge. A simple set of rules for updating the TS structure when missing vertices appear are applied:

1. Missing vertex. If there is a non-inserted vertex in a triangle edge, the VV is the vertex located on the same edge and in the following row of vertices.
2. Group of missing vertices. If there is a group of adjacent non-inserted vertices on a triangle edge, each non-inserted vertex is replaced in the TS representation by the nearest vertex on the edge. In case of equidistant vertices, the vertex in the lower row is selected.
3. Replicated vertices in the TS list. Once the missing vertices are replaced by virtual vertices, replicated vertices can appear in the TS list. These replicated vertices have to be eliminated; i.e., v_i v_i is substituted by v_i.
4. Vertices from alternating rows. In the non-adaptive partitioning, consecutive vertices in the TS list belong to two rows of vertices. For the adaptive proposal, when this regular structure is broken a modification has to be performed. Let us analyze a TS list with three consecutive vertices, v_i v_j v_k, where v_j and v_k come from the same row of vertices. In this case the TS list is updated by including a replicated version of v_i in between. As result the list of vertices becomes v_i v_j v_i v_k.

An example of application is depicted in Figure 5a. A non-adaptive tessellation would generate three rows of triangles, each one to be represented with at TS list. The TS lists for the non-adaptive tessellation are:

$$\begin{aligned} TS_1 &= \{v_2\ v_1\ v_3\} \\ TS_2 &= \{v_4\ v_2\ v_5\ v_3\ v_6\} \\ TS_3 &= \{v_7\ v_4\ v_8\ v_5\ v_9\ v_6\ v_{10}\} \end{aligned} \quad (10)$$

As a result of the adaptive tessellation, the vertex v_6 is not inserted. The TS lists are updated by the following steps:

- The missing vertex v_6 is replaced by a virtual vertex $VV = v_{10}$. This is the closest vertex located in the same triangle edge and in the following row of vertices. As a result the TS_2 and TS_3 lists are updated.
- The utilization of v_{10} as virtual vertex in the TS_3 list generates a replicated vertex. According to rule 3, the list $TS_3 = \{v_7\ v_4\ v_8\ v_5\ v_9\ v_{10}\ v_{10}\}$ becomes $TS_3 = \{v_7\ v_4\ v_8\ v_5\ v_9\ v_{10}\}$.
- With respect to the alternating rows property and according to rule 4, the list $TS_3 = \{v_7\ v_4\ v_8\ v_5\ v_9\ v_{10}\}$ becomes $TS_3 = \{v_7\ v_4\ v_8\ v_5\ v_9\ v_5\ v_{10}\}$.

Finally the triangle strips for this example are:

$$\begin{aligned} TS_1 &= \{v_2\ v_1\ v_3\} \\ TS_2 &= \{v_4\ v_2\ v_5\ v_3\ v_{10}\} \\ TS_3 &= \{v_7\ v_4\ v_8\ v_5\ v_9\ v_5\ v_{10}\} \end{aligned}$$

and the following triangles are generated:

$$\begin{aligned} &TS_1 \rightarrow \triangle v_2v_1v_3 \\ &TS_2 \rightarrow \triangle v_4v_2v_5\ \triangle v_2v_5v_3\ \triangle v_5v_3v_{10} \\ &TS_3 \rightarrow \triangle v_7v_4v_8\ \triangle v_4v_8v_5\ \triangle v_8v_5v_9\ \triangle v_9v_5v_{10} \end{aligned}$$

The methodology has to be extended to include those situations in which both vertices in the extreme positions of a row are missing. As the insertion decisions are applied to the interior vertices in the same row, this implies that a complete row of vertices is missing. In this case fewer triangle strips are generated and a number of modifications have to be made to the method explained above. The first step is to update the TS lists of the non-adaptive case by identifying the two TS lists affected and eliminating the first one. After this the second list is updated by substituting the missing vertices by other vertices in the closest non-empty row of vertices located above. More specifically the substitution has to be performed by adhering to the following rules:

1. Upper row with no missing vertices. If the row above is complete, the vertices are directly employed to substitute the eliminated vertices in the TS list. As the number of vacancies is larger than the number of vertices, the vertices have to be replicated. To obtain a satisfactory tessellation and to prevent the generation of large triangles, the pattern of substitution/replication has to be uniform.
2. Upper row with a missing vertex. If the row above has a missing vertex, this location will be occupied by a VV. The row of vertices is also employed to update the TS list under construction, but a number of considerations have to be taken into account for the VV. The vacancies are again covered by the vertices in the row, but the VV vertex can be employed only once; i.e., the VV cannot be replicated.

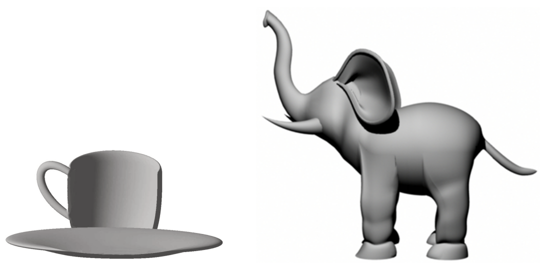

(a) (b)

Figure 6: Models employed: (a) *Teacup* and (b) *Elephant*.

Examples of application are depicted in Figures 5b and 5c. In these examples the row of vertices $\{v_4, v_5, v_6\}$ has not been inserted. The TS lists generated by the uniform tessellation (see Equation 10) have to be updated. In this case lists TS_2 and TS_3 are affected by the missing row and the list TS_2 is eliminated. List $TS_3 = \{v_7\ v_4\ v_8\ v_5\ v_9\ v_6\ v_{10}\}$ has to be updated by identifying the missing vertices (v_4, v_5 and v_6) and replacing them with the list of vertices above. Specifically, and for the example of Figure 5b, the vertices to be employed are v_2 and v_3. The number of vacancies is larger, so the first two vacancies are covered with vertex v_2 and the last one with vertex v_3. The list becomes $TS_3 = \{v_7\ v_2\ v_8\ v_2\ v_9\ v_3\ v_{10}\}$.

Figure 5c shows an example where the row of vertices to be employed has a missing vertex (v_2). Following the methodology explained above, this vertex is substituted by a virtual vertex, in this case by v_1. This virtual vertex is employed only once while vertex v_3 is employed for the other two vacancies. So the updated list is $TS_3 = \{v_7\ v_1\ v_8\ v_3\ v_9\ v_3\ v_{10}\}$.

4 EXPERIMENTAL RESULTS

In this section, the results of the evaluation of ESC pipeline are presented and analyzed. The adaptive tessellation proposal is compared with the tessellation implemented by the DirectX11 tessellation unit. Specifically, we have used the code *SimpleBezier11* included in the DirectX11 SDK to design an adaptive solution (*AdptTess*) based on a distance test (as presented in Section 2.2) computed in the Hull Shader. ESC proposal was coded in the Geometry Shader as this makes it possible to implement a free tessellation algorithm, even though it has the important constraint of limiting the maximum number of new primitives to be generated per input primitive to 1024 32-bit elements.

Our algorithm was implemented with Microsoft's HLSL DirectX11, and the tests were run on an Intel Core 2 2.4 GHz with 2 GB of RAM and three different GPUs: AMD/ATI Radeon 5870 (*ATI 5870*), GPU with 1600 processing elements distributed in 20 SIMD processors, each one having 16 cores with 5-way VLIW support; AMD/ATI Radeon 6970 (*ATI 6970*), with 1536 processing elements distributed in 24 *SIMD* processors, each one with 16 cores with 4-way VLIW support; and Nvidia Geforce GTX 580 (*Nvidia 580*), GPU based on the Fermi architecture that has 4 clusters, with 4 stream multiprocessor (SM) per cluster and 32 stream processors per SM for a total of $4 \times 4 \times 32 = 512$ physical processing elements.

Two models were employed to evaluate the tessellation (see Figure 6): *Teacup* and *Elephant*. These models were used to build two test scenes, *Teacups* and *Elephants*, that consist of replicated versions of the models: 30, 100 and 10 models, respectively. To check the performance of the implemented methods a walk-through animation with the same movement of the camera for the three scenes was performed. The final images have a screen resolution of 1280×1024 pixels.

The next subsections focus on the analysis of the experimental results in two different key points: the analysis of the quality in the final image and the performance in terms of frames per second.

4.1 Performance in Terms of Quality

As a starting point of the analysis of the results from the tests, Table 1 presents the number of generated primitives for each method. In this table the maximum resolution employed is $L_{max} = 3$ (16×18 triangles for each input surface). The second row indicates the number of input Bézier surfaces per scene. The third row shows the number of triangles generated when a non-adaptive tessellation is applied. The rest of the rows show the average number of triangles generated by the adaptive proposal. The *Distance test* was applied with two different thresholds on the basis of a quality criteria: high or medium degree of tessellation. The percentage of triangles obtained for each case with regard to a non-adaptive strategy is shown in parenthesis.

Table 1: Number of triangles generated (in thousands, $L_{max} = 3$) and number of input surfaces.

		Teacups	*Elephants*
Input Data		2600	8110
Non Adaptive		731.32	2280.94
Adaptive	**High**	(93.62%) 684.67	(85.10%) 1941.03
	Med	(51.96%) 379.98	(50.68%) 1155.70

In our experiments, high quality meshes with no cracks or holes are obtained. Obviously, the application of the adaptive proposal gives rise to a reduction in the number of primitives generated, where the decrement can be controlled by the adequate selection of the quality threshold applied. As an example of the tessellation obtained, Figure 7 shows the result of applying three different tessellation procedures to the *teacup* model. Figure 7a depicts a non-adaptive tessellation, with all the patches being uniformly subdivided up to a resolution level determined by the point of view, with a maximum refinement of $L_{max} = 3$. Figure 7b shows the tessellation obtained by ESC, generating about a 50% of the triangles created with the non-adaptive approach. Finally, the output obtained by using *AdptTess*, configured to produce a similar number of triangles than ESC, is depicted in Figure 7c.

As can be observed, significantly fewer primitives are generated in the flat areas with the adaptive approximations, especially noticeable in the output from ESC (see the detail being zoomed on the right of each adaptive solution). In these flat areas the coarse mesh is a good enough approximation to the Bézier surface, so introducing additional primitives does not result in a higher quality of the image for the quality threshold selected. Furthermore, as can be observed in the zoomed detail on the left of the adaptive results, the tessellation produced by ESC achieves a better approximation of the Bézier surface, as geometric aspects are taken into account (distance-based heuristic) in addition to the point of view information. Figure 8 shows a shaded version of the *teacup* model tessellated by both adaptive approaches.

4.2 Performance in Terms of fps

Performance in terms of fps is another important aspect to be analyzed. Figure 9 shows the frames per second (fps) with two different GPUs for an adaptive tessellation with a medium and high degree, as

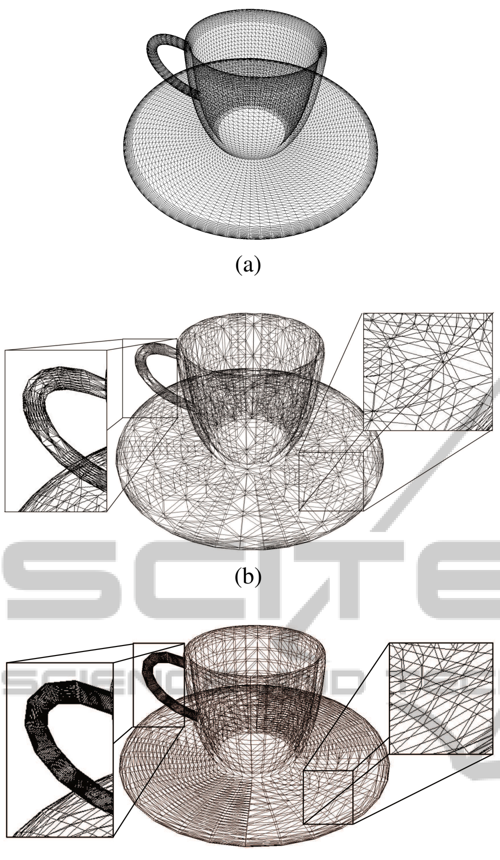


Figure 7: Examples of tessellation: (a) Non adaptive (b) adaptive with *ESC pipeline* and (c) adaptive with *AdptTess*.

shown in Table 1. The column labeled as *AdptTess* display the performance obtained by the *Simple-Bezier11* method implemented using the tessellation unit. The tessellation factors have been selected to generate a number of triangles similar to the obtained

Figure 9 shows that good performance results in terms of frame rate (fps) were obtained, and a real-time adaptive tessellation was achieved even for a high number of triangles. Thus, for instance, the high quality result of the *Elephants* scene is achieved with a frame rate in our adaptive approach of 63.53 fps on the *ATI 6970*, and 61.42 fps on the *ATI 5870*.

Broadly speaking, the frame rate achieved by ESC is similar to *AdptTess*, and it is important to remark that a software approximation of the algorithmic proposal that ESC introduces has being used. It should be pointed out that ESC demonstrates better computational exploitation than the tessellator-based alternatives, since one computing core (shader) is used for each input primitive, instead of the one core per output primitive ratio of the DirectX11 tessellator-based proposals. As a result, an important feature of our

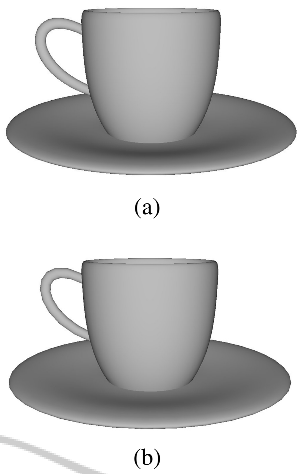


Figure 8: Shading comparison of the two adaptive approaches: (a) *ESC pipeline* and (b) *AdptTess*.

approach is the exploitation of the spatial coherence of data, as shared common computations within the

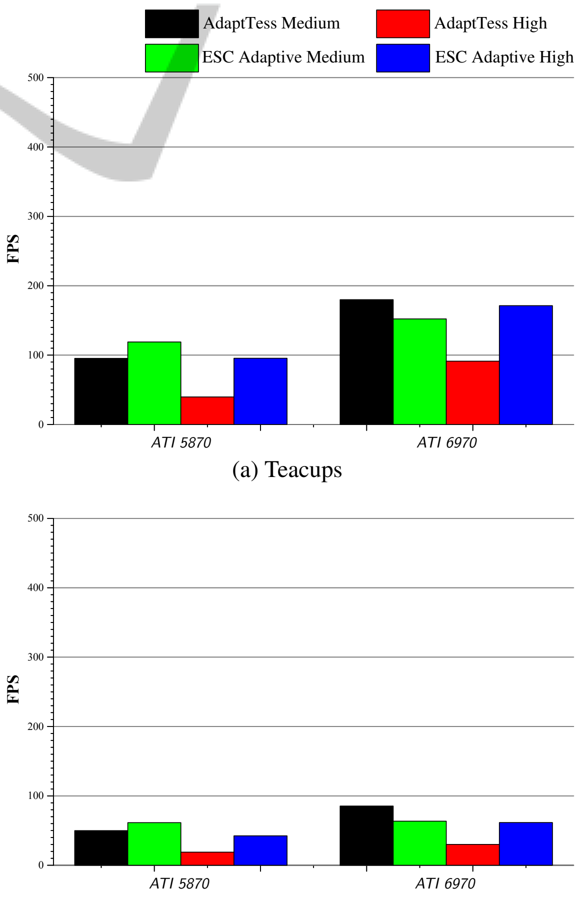


Figure 9: Processing Speed in Frames per Second (L_{max} = 3) (a) *Teacups* (b) *Elephants*.

same patch are computed only once and reused when needed. This results in a better performance when the number of primitives increases. Specifically, the results shown in Figure 9 say that, for the highest level of tessellation, ESC is up to $2.25x$ faster on the *ATI 5870* and up to $2.05x$ faster on the *ATI 6970*.

In summary, considering the good results obtained, the flexibility of the adaptive proposals, the exploitation of the locality and the prevention of redundancy computations, our proposal is a good candidates to be integrated as a specific tessellation unit in future graphics cards, as nowadays the existing tessellation units included in current GPUs do not offer the desirable adaptability.

5 CONCLUSIONS

This paper presents a proposal for the adaptive tessellation of Bézier surfaces on the GPU based on the exploitation of the spatial coherence. The proposal do not require the precomputation of any refinement pattern, and the new vertices coordinates are computed on-the-fly with a non recursive strategy. This permits the exploitation of the vector computation capabilities of current GPUs. The proposal uses a section of the parametric map as input primitive.

This tessellation scheme reduces the divergence in order to achieve an optimum utilization of the computational resources of the GPU; however, a remarkable degree of adaptivity has been introduced. Hence, this proposal processes considerably fewer triangles than a non adaptive proposal.

The adaptive proposal is based on three main strategies: the utilization of a fixed tessellation pattern to guide the procedure, the utilization of a local test to guide the tessellation decisions and an efficient meshing procedure to reconstruct the resulting mesh. Our proposal permits the application of multiple levels of resolution to a Bézier surface and exploits the locality of the surface and, consequently, reducing the number of shader invocations and, as a consequence, the power consumption.

In addition to the good quality and performance results, the flexibility of the adaptive proposal and the simplicity of the computations involved could encourage the inclusion of more flexible tessellation units in future graphics cards.

REFERENCES

Amresh, A. and Fünfzig, C. (2010). Semi-uniform, 2-Different Tessellation of Triangular Parametric Surfaces. In *Proceedings of the 6th International Conference on Advances in Visual Computing (ISVC'10)*.

Concheiro, R., Amor, M., and Bóo, M. (2010). Synthesis of bézier surfaces. In *GRAPP'10: International Conference on Computer Graphics Theory and Applications*, pages 110–115.

Concheiro, R., Amor, M., Bóo, M., and Doggett, M. (2011). Dynamic and adaptive tessellation of bezier surfaces. In *GRAPP'11: International Conference on Computer Graphics Theory and Applications*, pages 100–105.

Dyken, C., M., R., and Seland, J. (2009). Semi-uniform Adaptive Patch Tessellation. *Computer Graphics Forum*, 28(8):2255–2263.

Eisenacher, C., Meyer, Q., and Loop, C. (2009). Real-time View-dependent Rendering of Parametric Surfaces. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, pages 137–143.

Fisher, M., Fatahalian, K., Boulos, S., Akeley, K., Mark, W. R., and Hanrahan, P. (2009). DiagSplit: Parallel, Crack-free, Adaptive Tessellation for Micropolygon Rendering. *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH Asia 2009*, 28(5).

Guthe, M., Balázs, A., and Klein, R. (2005). GPU-Based Trimming and Tessellation of NURBS and T-Spline Surfaces. *ACM Trans. Graph.*, 24(3):1016–1023.

Munkberg, J., Hasselgren, J., and Akenine-Möller, T. (2008). Non-uniform Fractional Tessellation. In *Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*.

Ni, T. and Castaño, I. (2009). Efficient Substitues for Subdivision Surfaces. Exhibition Tech. SIGGRAPH'09 Course Notes, 2009.

NVIDIA (2008). *NVIDIA CUDA Compute Unified Device Architecture. Programming Guide*.

Piegl, L. and Tiller, W. (1997). *The NURBS Book*. Springer.

Rogers, D. F. (2001). *An Introduction to NURBS with Historical Perspective*. Morgan Kaufmann.

Schwarz, M. and Stamminger, M. (2009). Fast GPU-based Adaptive Tessellation with CUDA. *Computer Graphics Forum*, 28(2):365–374.

Yeo, Y. I., Bin, L., and Peters, J. (2012). Efficient pixel-accurate rendering of curved surfaces. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics - i3D 2012*, pages 165–174.