Eurographics Italian Chapter Conference (2007)
Raffaele De Amicis and Giuseppe Conti (Editors)

Explicit Adaptive Tessellation based on re-parametrization on Graphics Hardware

Alessandro Martinelli[1]

[1]DIS, Pavia University, via Ferrata 1, Pavia, Italy

Abstract

We propose to use an explicit function for adaptive tessellation of parametric curves and surfaces. This function behaves as a new parametrization from the surface domain (or curve domain) to the domain itself; it is build using information about derivatives and curvature: a fixed tessellation may be re-arranged in an adaptive tessellation, which takes care of those parts of the curve or surface which need to be tessellated more and those which may use a poorer tessellation. We show how to produce and how to use the kernel function with four example: a simple cubic curve, a spline curve, a cubic bezièr triangle and a cubic quadrilateral patch. For every example, we compare the fixed tessellation with the adaptive one: the number of vertexes used is always the same, but the points are re-arranged in a better way. At the end we show how to use commonly known forward differencing methods to evaluate both the explicit parametrization and the curve or surface; we also show how simply this method may be implemented on common graphics cards.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Line and Curve Generation,Display algorithms, Line and Curve generation I.3.5 [Computer Graphics]: Surface Representation, Geometric Algorithms, Splines

1. Introduction

Consider the curve in Figure 1: it is a cubic bezièr curve defined in $[0, T_{max}]$. It has been rendered has a sequence of segments: in the first case (the upper one) we used a fixed tessellation; in the second case the tessellation is also fixed, but the points are re-arranged by an f function. In the rectangles A1 and A2 you can see that the straighter part of the curve is segmented with fewer segments, while the part with more curves is described with more points; the number of points is always the same, but they are used in a better way. This simple principle is useful also for parametric surfaces, where the function f must be defined over all the surface domain.

2. Segmentation And Tessellation

Segmentation and tessellation are two very important processes in graphics, always used when we want to render a curve or a surface which is not described as a set of linear elements. When curves and surfaces are described in a parametric form, they can be generally evaluated with a uniform tessellation, taking advantages of the most common forward differencing techniques, as it has been done in [Mor01]. An alternative to forward differencing, may be Central Differencing, for example [DeL99] shows this alternative on bezièr cubic patches. Differencing schemas are very fast, but the drawback of fixed tessellation is that it uses too much points when the curve (or surface) is very closed to be linear, and not enough points when the geometrical model has an high curvature. To solve this problem, [LP87] proposed to use an adaptive way to chose the step length in forward differencing during rendering when needed.

A more interesting solution is the known technique that can be found for example in [AA00], where the tessellation is produced in a recursive way: if a piece of surface does not approximate the surface well, it is tessellated with more finer triangles to get the necessary precision. This solution requires algorithms which avoid cracks on the surface ([LT96] and [Bri99] illustrate this problem).

Given a parametric curve expressed over the domain $D = \{t/t_{min} \le t \le t_{max}\}$ an explicit adaptive tessellator is a function defined as $f(t) : D \to D$, which has these properties:

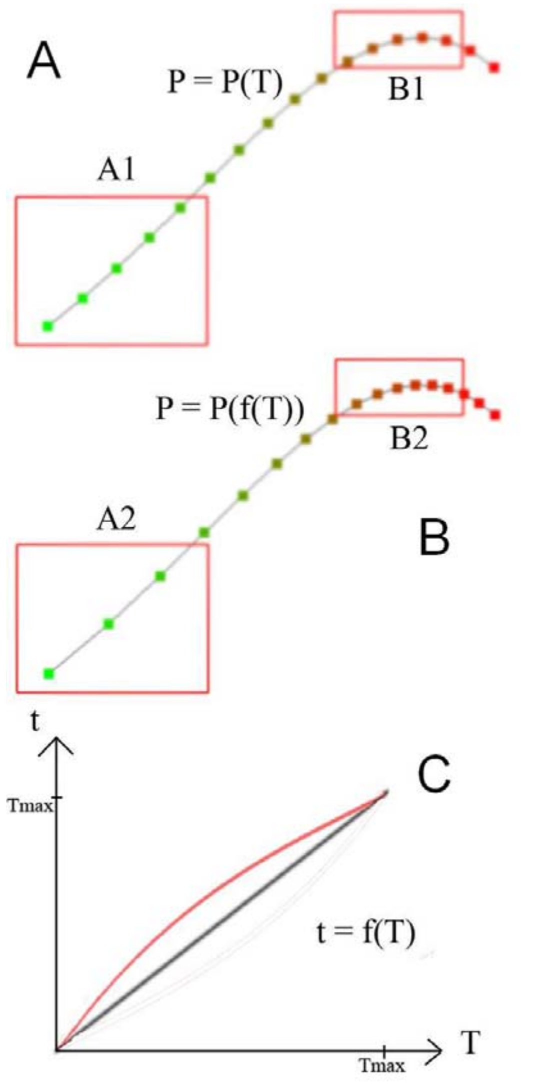

Figure 1: *The first example: we use a function f to construct a re-parametrization of the parametric form of a curve. The parametrization moves the points so that more points are placed where they are more needed. (A) On the top you can see the curve rendered with fixed tessellation, (B) in the middle the curve with explicit adaptive tessellation and (C) in the bottom the the function f(t).*

$$\frac{\partial f}{\partial t} > 0 \,\forall\, t \in D \tag{1}$$

$$f(t_{min}) = t_{min} \tag{2}$$

$$f(t_{max}) = t_{max} \tag{3}$$

Every fixed tessellation over D $\{t_{min}, t_{min}+\Delta t, t_{min}+2\Delta t, ..., t_{max}\}$ can be changed in a new tessellation $\{f(t_{min}), f(t_{min}+\Delta t), f(t_{min}+2\Delta t), ..., f(t_{max})\}$, which is a crescent sequence of points, because of condition 1.

Given a parametric surface expressed over the domain $D = \{(u,v)/u_{min} \le u \le u_{max} \ \ and \ \ v_{min} \le v \le v_{max}\}$ an explicit adaptive tessellator is a set of functions $\{f_u(t): D \to D, f_v(t): D \to D\}$, which have these properties:

$$\frac{\partial f_u}{\partial u} > 0 \,\forall\, (u,v) \in D \tag{4}$$

$$f(u_{min}, 0) = u_{min} \tag{5}$$

$$f(u_{max}, 0) = u_{max} \tag{6}$$

$$\frac{\partial f_v}{\partial v} > 0 \,\forall\, (u,v) \in D \tag{7}$$

$$f(0, v_{min}) = v_{min} \tag{8}$$

$$f(0, v_{max}) = v_{max} \tag{9}$$

$$\frac{\partial f_u}{\partial u}\frac{\partial f_v}{\partial v} - \frac{\partial f_u}{\partial v}\frac{\partial f_v}{\partial u} > 0 \,\forall\, (u,v) \in D \tag{10}$$

which has an effect similar to the case seen before. The conditions on partial derivatives ensure that the vector function $\{f_u, f_v\}$ maps always to different values of (u,v) (so there aren't repeated values of (f_u, f_v)).

In the next subsections we are going to see some important considerations about the construction of the $f(t)$ or $\{f_u, f_v\}$ functions.

2.1. Avoiding cracks between surfaces

A very important task in tessellation is to avoid cracks between adjacent surfaces (see Figure 2). It is important that every surface produces the same points on the common edges: to ensure this, when we want to determine the functions $\{f_u, f_v\}$ for a parametric surface, we firstly find a function $f_i(t)$ for every edge of the patch, and then we reconstruct $\{f_u, f_v\}$ using the information produced in the edges.

For this reason we are going to speak mostly about curves. In section 5 we are going to see how to use informations on edges to construct a patch tessellator.

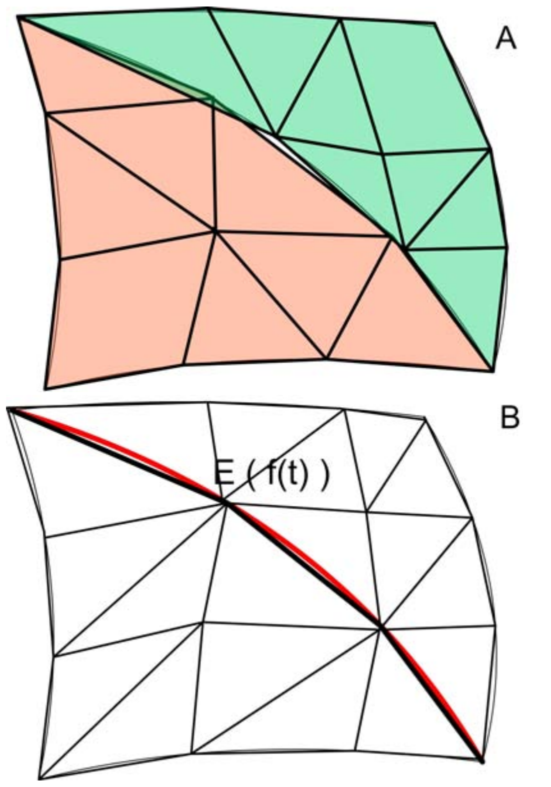

Figure 2: *(A) Two patches are adjacent, but they are tessellated in a bad way, so we have cracks. (B) We construct an f function firstly on the curve defined on the common edge; the tessellator for the patches is constructed in a way that allows the common edge to be tessellated in the same way the curve tesselator do.*

2.2. Defining the f tessellator function for a curve

In this contest we use always the same model for the f(t) tessellator of a curve:

$$f(t) \; = \; at^3 \; + \; bt^2 \; + \; ct \; + \; d \tag{11}$$

that is a cubic in t. t is supposed to be in the interval $[0,1]$; it is very easy to extend this to the general case. We have to satisfy the conditions for the tessellator (formulas 2 e 3).

$$f(0) \; = \; d \; = \; 0 \tag{12}$$

$$f(1) \; = \; a + b + c + d \; = \; 1 \tag{13}$$

So we take $b = 1 - a - c$. The model may be reduced to

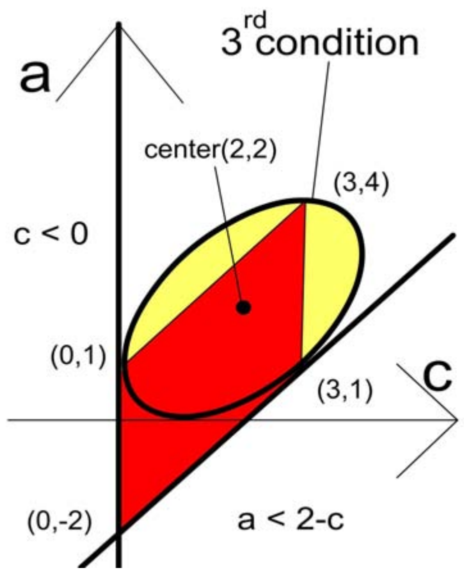

Figure 3: *The Area in red+yellow represent all the values of a and c for which the cubic tesselator satisfy all the tassellator conditions. To make this control easiest, we propose to work with the reduced red area.*

$$f(t) \; = \; at^3 \; + \; (1 \; - \; a \; - \; c)t^2 \; + \; ct \tag{14}$$

The last condition (formula 1) tells that:

$$\frac{\partial f}{\partial t} = 3at^2 + 2(1-a-c)t + c > 0 \; \forall \, t \in [0,1] \tag{15}$$

The solution of this condition is given by:

$$\frac{\partial f(0)}{\partial t} = c > 0 \tag{16}$$

$$\frac{\partial f(1)}{\partial t} = a - c + 2 > 0 \tag{17}$$

$$if \;\; 0 \le \frac{-(1-a-c)}{3a} \le 1 \, , \tag{18}$$

$$\frac{a^2 + c^2 - ac - 2a - 2c + 1}{3a} < 0 \tag{19}$$

The last condition says 'if the second derivative has a zero between 0 and 1, in that value of t the first derivative must be positive', that's we ensure f to be crescent and without oscillations. Figure 3 gives the form of the resulting set of possible solutions. Every time we construct a tessellator with this model, we have to ensure that its parameters a and c are inside that red+yellow set and b to be $b = 1 - a - c$.

2.3. Defining the model parameters

The parameters may be defined with every possible value, but to have a useful automatic instrument, we need an evaluation of how the curve we are drawing is bending.

In general we need a way to decide 'how much the curve bends' starting from the point t_i and ending in the point $t_i + \delta T$.

2.3.1. Error analysis

The first solution we propose is to evaluate the error made by a step dt. If we have a curve P=P(t) and we are in t_i, P may be expressed as

$$P(t_i+\delta t) = P(t_i) + \frac{dP(t_i)}{dt}\delta t + \sum_{j=2}^{\infty} \frac{1}{j!}\frac{d^j P(t_i)}{dt^j}\delta t^j \qquad (20)$$

where δt is $(t - t_i)$.If we consider a step δT, we can construct the line which pass through $P(t_i)$ and $P(t_i + \delta T)$ as

$$P'(t_i+\delta t) = P(t_i) + (P(t_i+\delta T) - P(t_i))\frac{\delta t}{\delta T} \qquad (21)$$

$$P'(t_i+\delta t) = P(t_i) + \frac{dP(t_i)}{dt}\delta t + \sum_{j=2}^{\infty} \frac{1}{j!}\frac{d^j P(t_i)}{dt^j}\delta T^{j-1}\delta t \qquad (22)$$

so the error function in $t = t_i + \delta t$ is

$$E(t_i+\delta t) = P'(t_i+\delta t) - P(t_i+\delta t) \qquad (23)$$

$$E(t_i+\delta t) = \sum_{j=2}^{\infty} \frac{1}{j!}\frac{d^j P(t_i)}{dt^j}(\delta T^{j-1} - \delta t^{j-1})\delta t \qquad (24)$$

This error function is 0 in $\delta t = 0$ and $\delta t = \delta T$. It has three components, that is the error for the x component, for the y component and for the z component. One possible solution is to find the integral of the three absolute values of the three functions, and then sum them. The formula of the integral is

$$I(t_i,\delta T) = \sum_{j=2}^{\infty} \frac{1}{j!}\frac{d^j P(t_i)}{dt^j}\delta T^{j+1}\left(\frac{1}{2} - \frac{1}{j+1}\right) \qquad (25)$$

This way can get good results but it has a little drawback: it is based on the parameters. Consider this example of simple curve:

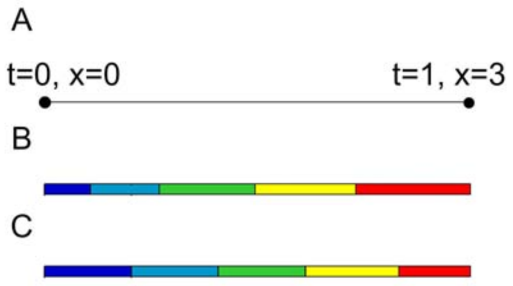

Figure 4: *(A)The line is a straight line, so there are no reasons to use more then one segment to approximate it. (B) On the domain t, it is defined a color map; the mapping between color and t is linear, but the mapping between the x value and t is not linear, so the colors appear to change in a non linear manner. (C) If we approximate t with only one segment, the colors are rendered in a wrong way.*

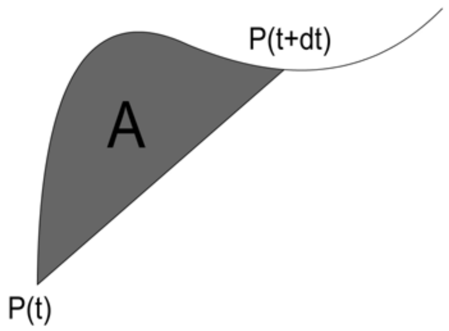

Figure 5: *The Area A is the Area between the curve and the approximating segment.*

$$x = t^2 + t + 1, y = 0, t \in [0,1] \qquad (26)$$

If we take $t_0 = 0$ and $\delta T = 1$ we find the error $\frac{1}{6}$ and this suggest that the curve should be segmented with a more little step δT, but the truth is that we are working with a straight line. The problem is to decide if for us it is important only to consider the geometry of the curve or even values of parameter t: in fact if t is used for other aspects of rendering (such as texturing, time control and so on) an error based on t is important.

2.3.2. Curvature Analisys

The curvature is a perfect measurement instrument if we want to consider only the geometric form of the curve (so without any interest on the curve parameter). In fact it can be shown that the integral of the curvature is a good evaluation of the area between the curve and the approximating segment (Figure 5). If we are considering only the curve in the screen space, the curvature may be written as

$$\kappa(t) = \frac{\frac{dx}{dt}\frac{d^2y}{dt^2} - \frac{dy}{dt}\frac{d^2x}{dt^2}}{\left(\frac{dx}{dt}^2 + \frac{dy}{dt}^2\right)^{\frac{3}{2}}} \quad (27)$$

Unfortunately, given a value t_i and a step δT, it is not so simple (or not very fast) to evaluate the integral of the curvature function in that formula. For the rest of this paper, we are considering only the Error Analisys seen in the subchapter before.

2.3.3. Finding the model parameters

There are a lot of possibilities in order to find values for a and c, given a method to evaluate how the curve bends. For example one may use a lot of samples and evaluate the result with a least squares method.

Our solution is a bit simpler. The first curve we want to work with is a Bezier Curve of third degree, defined over four points P_k:

$$P(t) = P_0(1-t)^3 + P_1 3t(1-t)^2 + P_2 t^2(1-t) + P_3 t^3 \quad (28)$$

Given a value t_* and a step δT, the error due to a linear approximation is (see 24)

$$E(t) = \frac{1}{2}\frac{d^2P(t_*)}{dt^2}(\delta T - \delta t)\delta t + \frac{1}{6}\frac{d^3P(t_*)}{dt^3}(\delta T^2 - \delta t^2)\delta t \quad (29)$$

Now we can chose if we want to treat the error as a three-dimensional error or a screen based (bi-dimensional) value. In the first case we evaluate the sum of the integrals of x, y and z.

$$\mathrm{I}(t,\delta T) = \frac{1}{2}\left(\frac{d^2x(t_*)}{dt^2} + \frac{d^2y(t_*)}{dt^2} + \frac{d^2z(t_*)}{dt^2}\right)\delta T^3\left(\frac{1}{2} - \frac{1}{3}\right) + \frac{1}{6}\left(\frac{d^3x(t_*)}{dt^3} + \frac{d^3y(t_*)}{dt^3} + \frac{d^3z(t_*)}{dt^3}\right)\delta T^4\left(\frac{1}{2} - \frac{1}{4}\right)$$

In the second case we don't consider the z-component. The difference is that in the first case we are constructing a tessellator useful in every contest, while in the second case we are constructing it using screen coordinates, so optimized for a particular point of view, and we have to repeat the operation every time (but the results are better).

When we have this error function, we divide the domain into three parts and get the integral of the error in every parts: $S_1 = I(0,\frac{1}{3}), S_2 = I(\frac{1}{3},\frac{1}{3}), S_3 = I(\frac{1}{3},\frac{1}{3})$. Now we consider this fact: if the error is high, we desire the tessellator to be slow, so that with the same segments it approximates a more little part of the curve; if the error is little, we desire the tessellator to change rapidly, so that every segment is used

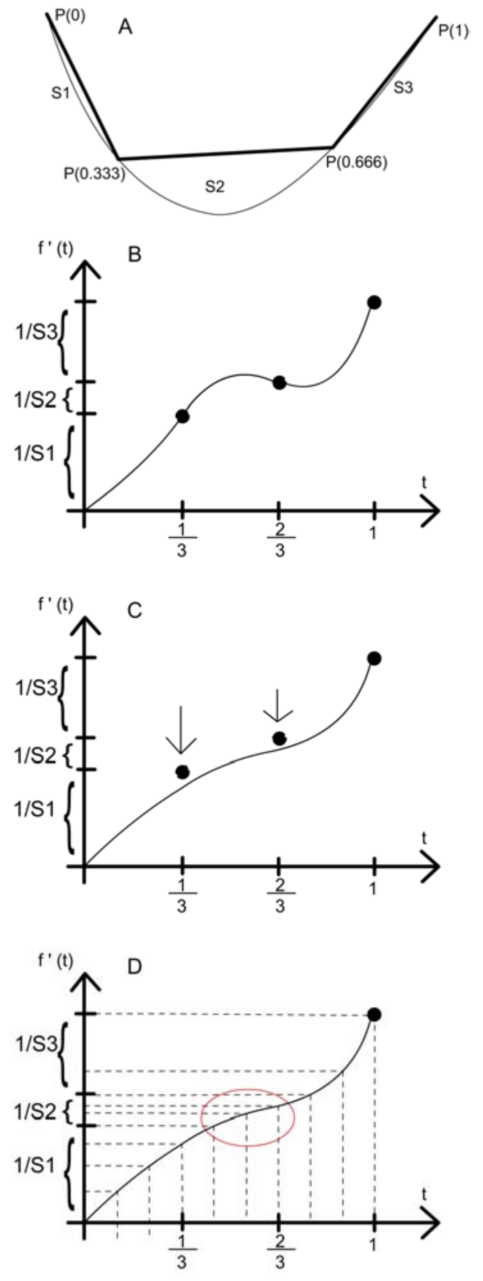

Figure 6: *This figure is an explication of the procedure we suggest to find values a, b and c for a cubic tessellator. (A) We evaluate the errors S_1,S_2 and S_3 that we get if we approximate the curve with three segments with equal step $\delta T = \frac{1}{3}$; observe that S_2 has the biggest value (only in this example, it depends on the curve). (B) We construct a cubic curve which interpolate the points $(0,0),(\frac{1}{3},\frac{1}{S_1}),(\frac{1}{3},\frac{1}{S_1}+\frac{1}{S_2}),(1,\frac{1}{S_1}+\frac{1}{S_2}+\frac{1}{S_3})$ (C) We adapt the curve so that they are satisfied the condition on the tessellator. (D) If we tessellate t with a fixed step, we get that the tessellation points are concentrated near S_2 (as we could considerate from considerations in (A))*

to approximate a big part of the curve. To manage this, we firstly construct a cubic tessellator $f'(t) = At^3 + Bt^2 + Ct$ in this way

$$\begin{cases} f'(\frac{1}{3}) = A\frac{1}{27} + B\frac{1}{9} + C\frac{1}{3} = \frac{1}{S1} \\ f'(\frac{2}{3}) = A\frac{1}{27} + B\frac{1}{9} + C\frac{1}{3} = \frac{1}{S1} + \frac{1}{S2} \\ f'(1) = A + B + C = \frac{1}{S1} + \frac{1}{S2} + \frac{1}{S3} \end{cases} \quad (30)$$

When we have this cubic, we reduce it to a cubic that has value 1 for $t = 1$, in this way

$$(a,b,c) = (A,B,C)\frac{1}{A+B+C} \quad (31)$$

It is not enough. We have to satisfy 16,17 and 18 to ensure that the f function is a crescent function. Figure 3 show the values for a and c which satisfy condition 18: we prefere, for simplicity, to use a subset, the quadrilateral with vertexes (0,-2) (3,1) (3,4) and (0,1); after the application of formula 31, we test the a and c values and if they are not inside that domain, usually we find the (a',c') point of the red domain nearest to (a,c). Of course, b' is always set as $b' = 1 - a' - c'$.

3. Rendering the Curve

When we have an f function, rendering is obvious. A simple pseudo-OpenGl code would be similar to this:

```
int DIVISIONS=20;
float step=1.0f/DIVISIONS;
float t,x,y,z;
glBegin(GL_LINE_STRIP);
    for(int i=0;i<=DIVISIONS;i++){
     t=i*step;
     t =((a*t+b)*t+c);
     x=getX(t);
     y=getY(t);
     z=getZ(t);
     glVertex3f(x,y,z);
    }
glEnd();
```

A faster solution is to consider a forward differencing method. It is obvious, but let as show this: the function $P(f(t))$ may be written as

$$P(f(t)) = P(f_0) + \frac{dP(f_0)}{df}\delta f + \frac{1}{2}\frac{d^2P(f_0)}{df^2}\delta f^2 + \frac{1}{6}\frac{d^3P(f_0)}{df^3}\delta f^3 \quad (32)$$

$$\delta f = f_0 + \frac{df(t_0)}{dt}\delta t + \frac{1}{2}\frac{d^2f(t_0)}{dt^2}\delta t^2 + \frac{1}{6}\frac{d^3f(t_0)}{dt^3}\delta t^3 \quad (33)$$

Combining these two equations bring us to an evaluation system for the curve based on simple sums.

4. Working with a Bezier Spline

We have considered the extension of the case of a single curve to the case of a spline curve (Figure 7). We use a fixed step knots vector, with step equal to one, so $t_i = i$. We construct a simple tassellator for every curve in the spline. Then we associate to every curve Ψ^i a weight which is

$$w_i = \frac{(S1_i + S2_i + S3_i)}{\sum_{j=0}^{N}(S1_j + S2_j + S3_j)} \quad (34)$$

So if a curve is highly curved, it get an high weight, while if a curve is a straight line it can either get a $w_i = 0$. Usually, if $w_i = 0$, we set $w_i = \varepsilon$, with ε very little, because a value 0 can get the curve behave in a wrong manner.

When we have finished, we reconstruct the curve with the new knots vector defined as $t_0 = 0$ and $t_{i+1} = t_i + w_i$. A fixed step tessellation along the set of t values, produces the effect you can see in figure 7B

5. Working with a Bezier Triangle

We have worked with a Cubic Bezier Triangle (Figure 8)which is constructed in this way:

$$b(u,v) = \sum_{i+j+k=3} b_{ijk}\frac{3!}{i!j!k!}u^iv^jw^k, w = 1-u-v, u,v,w \geq 0 \quad (35)$$

As we explained in subsection 2.1, the simplest way to define a tassellator for a Bezier Triangle which doesn't produce cracks is to define the tassellator on the information produced by the tassellators for the edges. We propose as explicit tessellator a cubic function too, that we formalize in this way:

$$f_{u,v}(u,v) = \sum_{i+j\leq 3} d_{ij}u^iv^j \quad u+v \leq 1\ , \quad u,v \geq 0 \quad (36)$$

The domain is always the same. The edges of the surfaces are cubic curves and the edges of the tassellator are cubic tessellator functions for that edges. In fact, for example, if we set $v = 0$, we get

$$b(u,0) = \sum_{i+k=3} b_{i0k}\frac{3!}{i!k!}u^i(1-u)^k, \quad 0 \leq u \leq 1 \quad (37)$$

$$f(u,0) = \sum_{i\leq 3} d_{i0}u^i \quad u \leq 1\ , \quad 0 \leq u \leq 1 \quad (38)$$

Now we must describe the functions $f_u(u,v)$ and $f_v(u,v)$. If $P_0(u,0)$, $P_1(0,v)$,$P_2(u,1-u) = P_2'(1-v,v)$ are respectively the edge tessellator for the edge $v = 0$, for the edge

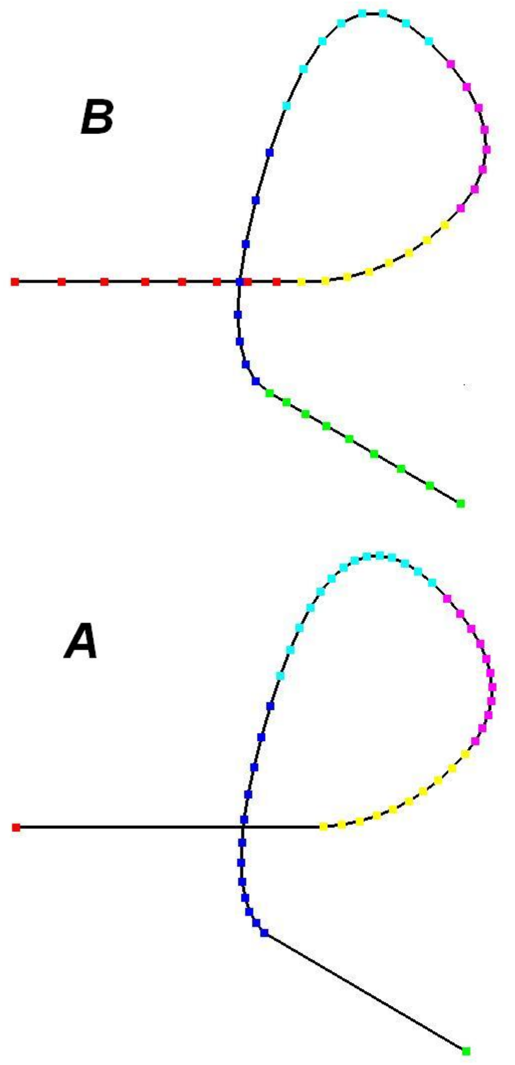

Figure 7: *The same bezier spline: (A) Rendered with a fixed step (B) Rendered with an adaptive tessellator along the set of bezier curve: the first and last curves are straight lines, so they are rendered in a fixed manner.*

$u = 0$ and for the edge $u = 1 - v$. We use the following functions:

$$f_u(u,v) = P_0(u,0) + (P_2(u,1-u) - P_0(u,0))\frac{v}{1-u} \quad (39)$$

$$f_v(u,v) = P_1(0,v) + (P_2'(1-v,v) - P_1(0,v))\frac{u}{1-v} \quad (40)$$

In this way the tessellation is based only on the edges tessellators (it is in fact an interpolation of the edges tessellators). It is easy to demonstrate that if $P_0(u,0)$, $P_1(0,v)$ and $P_2(u,1-u) = P_2'(1-v,v)$ satisfy the tessellation conditions for a curve, $f_u(u,v)$ and $f_v(u,v)$ satisfy the conditions for a patch.

6. Extension to a quadrilateral Patch

The extension to a quadrilateral patch is truly obvious. For example consider this cubic quadratic patch:

$$f_{u,v}(u,v) = \sum_{i\leq 3}\sum_{j\leq 3} d_{ij}u^i v^j \quad u+v \leq 1\ , \quad u,v \geq 0 \quad (41)$$

A possible cubic tessellator is in the form:

$$f_{u,v}(u,v) = \sum_{i\leq 3}\sum_{j\leq 3} d_{ij}u^i v^j \quad u+v \leq 1\ , \quad u,v \geq 0 \quad (42)$$

We must consider the tessellators for the edges, as we have done before. Given $P_0(u,0)$,$P_1(0,v)$,$P_2(u,1)$,$P_3(1,v)$ we construct the two tessellators

$$f_u(u,v) = P_0(u,0) + (P_2(u,1) - P_0(u,0))v \quad (43)$$

$$f_v(u,v) = P_1(0,v) + (P_3(1,v) - P_1(0,v))u \quad (44)$$

That is, they are a linear interpolation of the opposite edges.

7. Explicit Adaptive Tessellation on GPU

It is very easy to introduce explicit adaptive tessellation on GPU: we are considering mostly the work from Boubecker ([BS]). We worked on a common ATI x1600 card ([ATI]), using OpenGL ([JL04]) binded to java code using the JOGL framework ([JOG]). We have defined the rendering procedure into more steps:

- We send a fixed tessellation to the graphic pipeline using a display list. This procedure must be done only once.
- For every patch we send to the graphics hardware all the parameters of the pacth and of the tessellator.
- The graphical pipeline is charged at the Vertex Stage with a vertex program which firstly evaluate the new values for u and v given the ones of the fixed tessellation, then evaluates the x,y and z for the patch.
- For every patch we call the list sent before.

On a common Pentium 4 at 1.66 Ghz of clock, we have found that we can evaluate with a java procedure the tessellators for cubic patches with a frequency of 6.9 10^5 patches per second, and for a cubic curve with a frequency of 1.9 10^6 curves per second. This is the most important added cost of

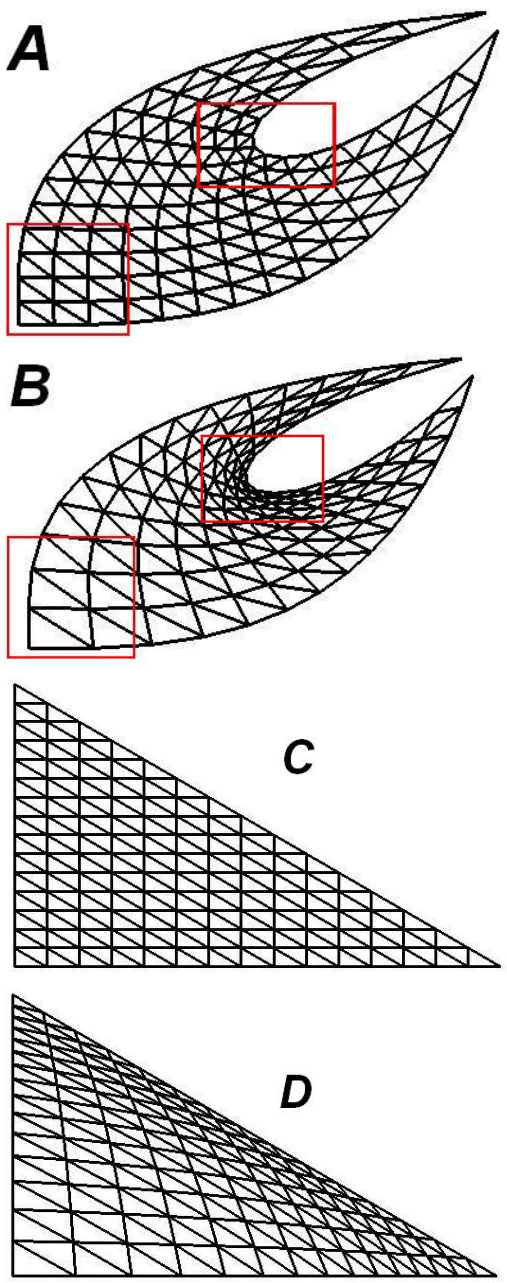

Figure 8: *The same cubic patch: (A) Rendered with a fixed step (B) Rendered with an adaptive tessellator (C) The tessellation of the domain with fixed step (D) The tessellation of the domain with variable step*

the rendering process. The other part of rendering has a cost very closed to the cost of fixed tessellation, if we are using the same number of points, and depends on the tessellation precision.

8. Conclusions

We have proposed a procedure to adapt a fixed tessellation of a parametric geometry to follow better the form of that geometry. This proposal is based on a re-parametrization, which make it suitable for forward differencing schemas and rendering on graphics hardware. The last advantages in graphics cards, with the introduction of geometry shaders (see [Dir]), suggest a possible future use of this strategy all hardware-implemented. Our method can also give a suggestion on how to adapt the number of steps to use for the beginning fixed tessellation: it is easy to build it starting from the error or curvature analysis we described in 2.3, with considerations similar to the ones in 4

References

[AA00] A.J. C., A.J.FIELD: A simple recursive tessellator for adaptive surface triangulation. In *journal of graphics tools, vol 1, no. 3* (2000), pp. 1–9. 1

[ATI] *ATI site:*. www.ati.it. 7

[Bri99] BRIAN S.: Implementing curved surfaces geometry. In *Game Developer, vol. 6 no. 6* (1999), pp. 44–53. 1

[BS] BOUBEKEUR T., SCHLICK C.: Generic mesh refinement on gpu. 7

[DeL99] DELOURA M. A.: An in-depth look at bicubic bèzier surfaces. In *Gamasutra, http:// www.gamasutra.com /features /19991027 / deloura_01.htm* (October 1999). 1

[Dir] *DirectX (for DirectX 10) site:*. www.ati.it. 8

[JL04] JR. R. S. W., LIPCHAK B.: *OpenGL SuperBible*, 3rd ed. Sams Publishing, London, 2004. 7

[JOG] *JOGL (java Bindings for OpenGL) site:*. jogl.dev.java.net. 7

[LP87] LIEN SHEUE-LING M. S., PRATT V.: Adaptive forward differencing for rendering curves and surfaces. In *Computer Graphics (SIGGRAPH '87 Proceedings)* (1987), ACM press, pp. 111–118. 1

[LT96] LINDSTROM KOLLER R. H. F., TURNER: Real-time, continuos level of detail rendering of height fields. In *Computer Graphics (SIGGRAPH '96 Proceedings)* (1996), pp. 109–118. 1

[Mor01] MORETON H.: Watertight tessellation using forward differencing. In *ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware* (2001), ACM press, pp. 25–132. 1