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

An Efficient Algorithm for Adaptive Segmentation and Tessellation with Pixel Precision

Alessandro Martinelli[1]

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

Abstract

We propose a new algorithm to get a representation of a curved surface with the precision of the image pixel. This technique uses some results from Scan-line algorithms, but it considers also the new functionalities from graphics hardware and takes advantages from it. We explain the general method, with principles common to every kind of surface: then we illustrate how these principles can be applied to quadratic and cubic beziér triangles, showing formulas and some algorithm details.

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

Graphics usually involves non-linear patches to describe any kind of surfaces. The most used objects for rendering can be quadratic patches, cubic patches or spline ([Mar99]). The problem is to chose a way to render this surfaces: using graphics hardware we can manage only linear elements; the most common technique is fixed tessellation, which involves fixed step to construct linear elements. Usually we can get a better reconstruction using adaptive tessellation algorithms, but these algorithms are recursive and they are very difficult to introduce in hardware(see [ML98], [Mor01], [LP87], [DeL99], [AA00]). One adaptive solution is the one proposed with scan line rendering, whose principle is to render a surface as it is, without tessellation, generally using most depth linear approximations. The problem of rendering non-linear surfaces for real time rendering on Graphics Hardware was also issued by [HT], but only for rendering of non planar Quads. In this article we consider some results from scan-line rendering algorithms, and we propose a new family of algorithms, which may be implemented also on graphics hardware, and which can render curve surfaces with pixel precision (or with less precision, but faster). We also show the implementation of this algorithm for two classes of curved surfaces: quadratic bezier triangles and cubic bezier triangles.

2. Scan-Line Rendering

The main idea of scan line rendering is to find the intersection pixels of a parametric surface with the scan-line plane, that is $Y = Y_j$, where we use screen coordinates, so Y_j is a screen line. This generally means that we have to know the Y_{max} and the Y_{min} values on the screen of the surface to be rendered, and an y_{step} to determine all the possible Y_j between Y_{max} and Y_{min}. The intersection is of course a curve in the (x,z) space, and it is also a curve in the (u,v) parametrical domain of the surface. The problem is then to find out possible rendering of this curve. The advantages of using this technique are the pixel-level precision of rastering and the perfectly adaptive representation of objects (which requires a computational cost proportional to the number of pixels occupied by the projection of the parametric surface). The most important issue of old scan line algorithms was to find possible ways to make the solution of high order systems of equations faster along the scan-line. Generally we want to consider the single ray $(X = X_k, Y = Y_j)$; for the curve which is the intersection of the plane $Y = Y_j$ with the surface, there are maximum and minimum values for X. Changing the value of Y_j changes slowly the extreme values for X. Moreover intersection points change slowly on the scan-line for the same Y_j, while the X_k changes. All this suggests the use of numerical methods to find new values, starting from previous ones. This methods, may have dif-

ferent properties of stability, sometimes unpleasant, and this is the most important reason to consider better and simpler ways to tessellate the surface with fine triangles.

For example Blinn in [Bli78] proposes the concept of active segment list that we call active intersections list in 2.1: this is a very important concept, because it is one of the advantages in using Scan-Line renderering; he then introduces some concept as Silhouette Trackers, Boundary Trackers and X-Scan which we will refer to in this article. Blinn proposed to follow the curve $Y = Y_j$ with a general Newton Step.

Also Whitted [Whi] and later Schweitzer and Cobb [DS] proposed an algorithm based on Newton's method, but they approximate every surfaces using cubic pathes: these patches are constructed such that the y function is monotonic on the edges and there are no singularities inside the patch; this involves subdivision where singularities are found.

Another solution is the one proposed by Lane-Carpenter [JML], which uses results from subdivision methods ([Cat]) to represent the curve $Y = Y_j$, where the curve is usually approximated with a parametric cubic curve. Unfortunately the proposed subdivision uses successive refinement, and this may not be a good thing for hardware implementation.

In the next subsections we are going to introduce some useful scan-line concepts

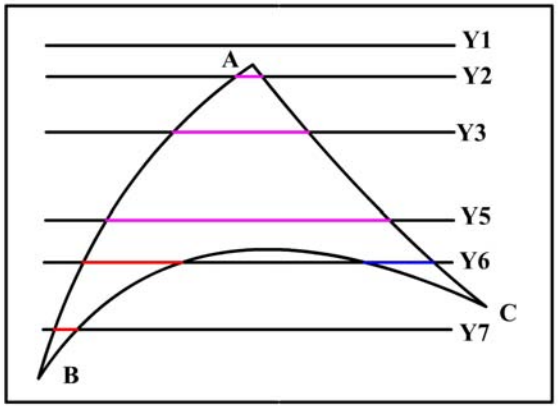

Figure 1: *Active-Intersections list. The figure shows what happens while changing the Y_j value along the surface. In Y_1 there are no fragments; passing to Y_2 a fragment is placed between $\overline{AB}$ to $\overline{AC}$, because the vertex special point A is crossed. There isn't any change in the fragment list till Y_5(so there is always the pink fragment). Crossing the maximum of $\overline{BC}$, the fragment splits in a red fragment from $\overline{AB}$ to $\overline{BC}$ and a blue one from $\overline{BC}$ to $\overline{AC}$. Then crossing C the blue fragment vanishes (Y_6), and crossing B the red fragment vanishes too (Y_7) and the fragments list is empty again.*

2.1. Active-intersections list (Fig. 1)

The problem of classification of what we will call fragments, that is the pieces of the curve $Y = Y_j$ which are inside the domain, may be a bit laborious and costly. To reduce the cost of this operation, we may make an assumption. The fragment order and type doesn't change when we change the y value if they are not crossed some special points, such as vertexes or local maxima or minima. So, it is possible to change the configuration of the Silhouette Trackers and Boundary Trackers list (see next subsection) only when we cross some of these special values. This list may be associated with a list of active-intersections which classifies all the possible intersections of the $Y = Y_j$ with the edges of the domain, and whose values are evaluated at every line.

2.2. Silhouette Trackers and Boundary Trackers(2)

When we consider the $Y = Y_j$, we should define the pieces of this curve which are inside the surface domain and follow them to produce the rendered figure (with Newton's method as Blinn [Bli78] do or with refinement as for Lane-Carpenter [JML]). For every solution we would like to know that a piece of curve begins with a point and ends with another point. Usually we track the surface from the first point to the last point. We construct a curve with a beginning point and an end point in two ways:

- The curve intersects the domain edges. These intersections may be used to limit curve pieces and we will call them Boundary Trackers
- The curve is closed and all inside the domain, so there aren't intersection with the domain. We define at least 2 points on the curve and two curve pieces between those points. We will call such points Silhouette Trackers.

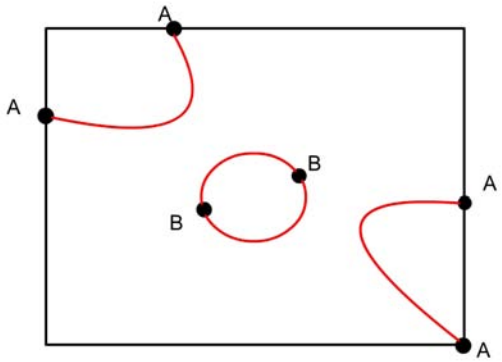

Figure 2: *Trackers:A) Boundary Trackers B) Silhouette Trackers*

2.3. X-Scan

When we have defined a piece of curve, we proceed with X-scan, that's we track x values along the curve. As in Blinn [Bli78] we may use a Newton step, or as in Lane-Carptenter [JML] we can find x values using approximations and subdivions.

2.4. Scan-Line Rendering with Hardware Acceleration: Adaptive Segmentation and Tesselation (Fig.1)

Before introducing our algorithm, we propose an extension of scan-line algorithm for their implementation on graphics hardware. The simplest idea is that it is possible to interpolate the curve which is the intersection of the surface with the $Y = Y_j$ plane using segments. Every segment may be directly communicated to graphics hardware (for example using LINE_STRIP in OpenGL); this makes the rendering scan-line truly fast. Another possibility is to consider scan-line algorithms as a source for adaptive tessellation algorithms: in fact we may consider an interval $Y_j \leq Y \leq Y_{j+1}$ and construct screen-oriented triangle strips instead of lines. In the first case we speak about adaptive segmentation and in the second case we speak about adaptive tessellation.
Considering the capabilities of the hardware, we may now suggest some ideas to make very fast Scan-Line Algorithms. The first problem to be addressed is to transfer the amount of calculation from the CPU to the GPU. Generally, old scan-line algorithms require a lot of complicated mechanisms to simplify the various rendering steps and to reduce computation. It may be for example important to know the maximum and minimum of the X value along the intersection curve, because they represent the borders of the projected figure and because they may create some problems with numerical methods(They may represent singular points).
With the idea to ask the specific hardware to solve part of the computation, we may neglect most of these complicated problems and address a more easy and immediate way to render. In fact now the most important issue is not only to construct a fast algorithm, but an algorithm which may produce in an efficient, rapid way the data to be communicated to graphics hardware, so to lighten the CPU.

3. Our Algorithm

The class of algorithms we are going to propose is based on some principles from Scan-Line rendering, but it tries to resolve the most common problems of scan-line algorithms to produce fast solutions. The aim is the rendering of cubic surfaces with pixel precision. The ideas on which these algorithms structure is based are:

1. Scan-Line Algorithms can achieve the problem of rendering surfaces with pixel precision: every tessellation algorithm can difficulty reach this task, usually using a very big amount of memory, or recursive methods, as it is for adaptive tessellation.

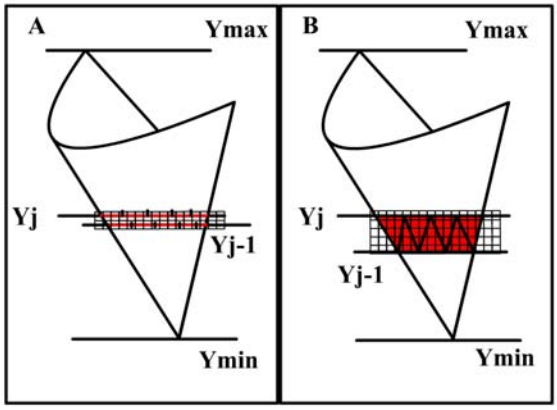

Figure 3: *Scan-Line Segmentation and Tessellation. A) The segmentation produces horizontal lines of pixels with a number of vertexes proportional to the space occupied on the screen by the intersection of the surface with the Y_j plane B) The tessellation produces a Triangles Strip with a number of triangles proportional to the space occupied on the screen by the intersection of the surface with the Y_j plane*

2. Scan-Line Algorithms have the advantage of using local incremental information from scan line to scan line and from x Scan to x Scan; this allows them to be faster than analogous ray-tracing algorithms.
3. Scan-Line Algorithms usually requires approximation and numerical methods. This is not usually a good thing, in particular when implementing algorithm which may work directly on hardware.
4. It could be fine having a pseudo scan-line solution which may be implemented on graphics cards, in particular on the new Geometry Shading Units (see [DIR]).

The algorithm follows these steps:

1. Firstly we find the maximum and minimum values for y.
2. For every y between the maximum and the minimum with step δy, we query the active intersections list, to manage transitions through local maxima, minima or vertexes.
3. From active intersection list, we evaluate values for the Silhouette Trackers and Boundary Trackers
4. The X-Scan tracking along the curve is solved with our new proposal, that's the extreme algorithm, which we are going to see in 3.2. The extreme algorithm is a schema for rendering, and it may be applied with the cooperation of graphics card, using solutions similar to [BS]. We are going to see how we can do it with quadratic and cubic patches.

In the following sub-sections we are considering some important issues and methods which may interest every scan-line algorithm, independently from the specific kind of parametric surface for which they are designed. The rest of the paper proposes some specific cases.

3.1. Fragments and Segments

The most important thing to consider is to distinguish between fragments and segments. The intersection between the scan plane $Y = Y_j$ and the surface is a curve in the parametric domain. If we consider this curve, we may see that there are points in which it exits the domain and points in which it enters the domain: usually we manage this identifying Boundary Trackers; generally every time it enters the domain, it needs to exit somewhere. The important thing is to determine these tracts of the curve. We will call them 'Fragments'. When we have a fragment, we may proceed with segmentation (as for a surface we should proceed with tessellation). Lane-Carpenter ([JML]) propose an algorithm based on recursive subdivision. We propose an algorithm based on immediate segmentation.

3.2. Extreme algorithms

An extreme algorithm has three main phases: ***fragmentation***, ***segmentation*** and ***sub-segmentation***(Fig. 3). The ***fragmentation*** may be performed using an active-intersections list and requires some ways to distinguish between different values of intersection on the domain. The segmentation follows the main principle of extreme approximation (see [Mar]): a piece of a fragment is approximated with a segment so that the first and last points of the segment are on the curve and so that the maximum of the error function $Y_{err} = \|Y(u,v) - Y_j\|$ along the segment is an exact ε value, maybe a pixel oriented one such as $\frac{1}{2}$ pixel. This means that there is only one point along the segment where we get the maximum error, and in every other point the error is lesser. When we have a segment, we can proceed with sub-segmentation, whose aim is exactly to construct linear segments in the parametric domain so that along the sub-segments also X may be considered linear, with a maximum error of approximation equal to the ε value.

3.2.1. Non-Controlled and Controlled Segmentation

The segmentation may be constructed in a Controlled way or in a non controlled way. When we are constructing a segment, we may use solutions from the previous steps to find the next step more efficiently. Unfortunately this means to use an algorithm where the next point is decided regardless possible errors of approximation, which may deviate the segment from the original trajectory (See Fig.4). One solution is to control the value of Y at every step of the tessellation, then to adjust possibly wrong step values so to correct the deviation. This may get an overhead in computation, but it makes the algorithm more stable.

4. Scan-Line Rendering of Quadratic Bezier Triangles

We have worked on quadratic bezier triangles.

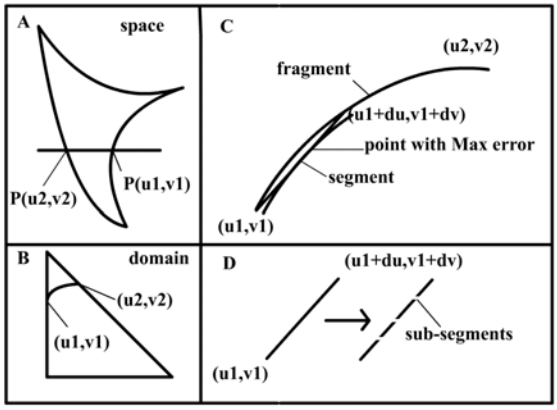

Figure 4: *The four steps of an Extreme Scan-line Algorithm: A) The intersection with the Y_j plane, B) The identification of fragments, C) The segmentation of every fragment, D) The sub-segmentation of every segment.*

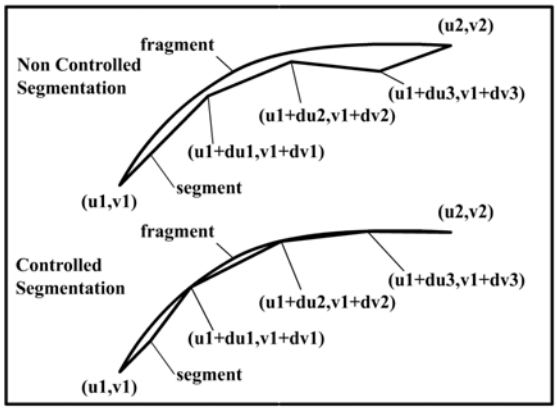

Figure 5: *Non Controlled Segmentation and Controlled Segmentation. In non controlled segmentation the last point of every segment may not touch the fragment because of approximation errors; after more steps, the errors may become greater. In controlled segmentation we verify that the last point is always on the fragment.*

$$P(u,v) = P_A(1-u-v)^2 + P_B u^2 + P_C v^2 + 2P_{AB}(1-u-v)u + 2P_{CA}(1-u-v)v + 2P_{BC}uv$$

Points are considered as 4 coordinates vectors. So, they have four components: this is to manage possible projection operations. Bruijns [Bru] proposes this instrument for modeling and he suggests some good techniques for adaptive rendering of quadratic bezier triangles.

We have built a segmentation algorithm and an adaptive tessellation algorithm both based on the same scan-line algorithm. The quadratic triangles may be produced by common triangular meshes, with methods similar to the ones proposed in [TB] and [VA03].
To make a comparison, we have also constructed a fixed step tessellation algorithm; this algorithm uses a fixed number of division n for every edge of the quadratic triangle, then constructs exactly n_2 triangles on the surface. When a triangle is constructed, the algorithm evaluates an error which is the maximum distance from the point of the triangle and the corresponding point on the surface curve.
Tables 1,2 and 3 show performance results: we have considered an animation of a moving and morphing bezier triangle, so every frame has a different bezier triangle, and the images were 10000. We have evaluated the maximum error of approximation derived by the fixed-step tesselation algorithm, with different values for n. The simulation has been considered for different image resolutions. The maximum error is represented in pixel, so it may be compared with the error of the scan-line algorithms which is set to 0.5 pixels. In the tables we have marked the case in which the error of approximation is less or more 0.5 pixels. The cost of computation is evaluated in terms of frames per second (fps) estimated for the animation. The system has been implemented in OpenGL, so the rendering of lines and triangles is executed on the graphical hardware, but the comparison is based on the rendering system without vertex shader acceleration. It is possible to see that the computational cost is comparable, but we have an efficient and flexible system whose cost depends on the dimension of the surface to be represented. The advantage is also the guarantees to have a pixel precision rendering.

4.1. Find Maxima and Minima

The first task in rendering is to find local and general maxima and minima for the quadratic surface. As we said before, the patch points have four coordinates, so the patch is a $(x(u,v),y(u,v),z(u,v),w(u,v))$; we should consider the transformed patch which is:

$$X(u,v) = \frac{x(u,v)}{w(u,v)} \tag{1}$$

$$Y(u,v) = \frac{y(u,v)}{w(u,v)} \tag{2}$$

$$Z(u,v) = \frac{z(u,v)}{w(u,v)} \tag{3}$$

Maxima and minima may be found as:

- Vertexes values
- Local Maxima and Minima of Y(u,v) constrained to the domain edges
- General Maxima and Minima of the Y(u,v) equation, that's the values of Y int points (u,v) such that:

$$\begin{cases} \frac{\partial Y}{\partial u} = 0 \\ \frac{\partial Y}{\partial v} = 0 \end{cases} \tag{4}$$

The third problem may be simplified with some assumptions. The System becomes:

$$\begin{cases} \frac{\frac{\partial y}{\partial u} w - \frac{\partial w}{\partial u} y}{w^2} = 0 \\ \frac{\frac{\partial y}{\partial v} w - \frac{\partial w}{\partial v} y}{w^2} = 0 \end{cases} \tag{5}$$

Which has the same solutions of

$$\begin{cases} \frac{\partial y}{\partial u} w - \frac{\partial w}{\partial u} y = 0 \\ \frac{\partial y}{\partial v} w - \frac{\partial w}{\partial v} y = 0 \end{cases} \tag{6}$$

Consider the first equation. $\frac{\partial y}{\partial u}$ is

$$\frac{\partial y}{\partial u} = \frac{\frac{\partial w}{\partial u} y}{w} \tag{7}$$

From this we can deduce a condition for $\frac{\partial y}{\partial u}$

$$\| \frac{\partial y}{\partial u} \| \leq \| \frac{max(\frac{\partial w}{\partial u} y)}{min(w)} \| \tag{8}$$

This says that, to get a solution, $\frac{\partial y}{\partial u}$ must be constrained. In particular, if we are working with projection and the patch has not too much different values for z, $\frac{\partial w}{\partial u}$ has a little value. In general we found the only solution of the system:

$$\begin{cases} \frac{\partial y}{\partial u} = 0 \\ \frac{\partial y}{\partial v} = 0 \end{cases} \tag{9}$$

and then the solutions of the system 6 will be very closed to that point.

For Local Maxima and Minima of Y(u,v) constrained to the domain edges, it is possible to construct a similar (but simpler) method.

4.2. Working with Curve Pieces

At the step j, we want to know how the curve $Y(u,v) = Y_j$ behaves. Using an active intersections list, we can reduce the computation of the curve pieces structure only when passing some special points. But how can we manage this structure?

The equation $Y(u,v) = Y_j$ can be simplified to:

$$\frac{y(u,v)}{w(u,v)} = Y_j \tag{10}$$

$$y(u,v) = Y_j w(u,v) \tag{11}$$

$$Y'(u,v) = y(u,v) - Y_j w(u,v) = 0 \tag{12}$$

The last equation is a conic in the domain space. We can find a linear transformation (from (u,v) to (s,t)) such that it can be expressed in one of these forms:

1. $At^2 + Bs^2 + F = 0$
2. $At^2 + Bs + F = 0$
3. $(At + F_1)(B_t + F_2) = 0$

Given this form, we can evaluate Silhouette Trackers and Boundary Trackers. Boundary Trackers are very easy to be found. Silhouette Trackers are necessary only in the first case if A and B has the same sign, because it means that the curve is an ellipse. In this case two Silhouette Trackers may be the solution of equation $At^2 + F = 0$ or $Bs^2 + F = 0$, that's the intersection with the ellipse axis.

4.3. Extreme Algorithm on quadratic Patches

Given a point (u,v) on a curve piece, we want to find a step $(\Delta u, \Delta v)$ such that the point $(u + \Delta u, v + \Delta v)$ is on the curve. This means that $\Delta Y'(u,v,\Delta u,\Delta v) = Y'(u + \Delta u, v + \Delta v) - Y'(u,v)$ must to be zero:

$$\Delta Y'(u,v,\Delta u,\Delta v) = \frac{1}{2}\frac{\partial^2 y}{\partial u^2}\Delta u^2 + \frac{1}{2}\frac{\partial^2 y}{\partial v^2}\Delta v^2 + \tag{13}$$

$$\frac{\partial^2 y}{\partial u \partial v}\Delta u \Delta v + \frac{\partial y}{\partial u}\Delta u + \frac{\partial y}{\partial v}\Delta v = 0 \tag{14}$$

The second condition is based on an extreme logic. The error along the segment $(Y'(u,v), Y'(u + \Delta u, v+, v))$ is a parabolic function, with value zero in $Y'(u,v)$ and $Y'(u + \Delta u, v+, v)$. So the maximum value of the error must be in the middle point. We set this value to be a particular value k, in particular we will chose the maximum error k to be $\frac{1}{2}$.

$$\Delta Y'(u,v,\Delta u,\Delta v) = \frac{1}{8}\frac{\partial^2 y}{\partial u^2}\Delta u^2 + \frac{1}{8}\frac{\partial^2 y}{\partial v^2}\Delta v^2 + \tag{15}$$

$$\frac{1}{4}\frac{\partial^2 y}{\partial u \partial v}\Delta u \Delta v + \frac{1}{2}\frac{\partial y}{\partial u}\Delta u + \frac{1}{2}\frac{\partial y}{\partial v}\Delta v = \pm k \tag{16}$$

The sign of k must be determined considering the curvature of the curve piece.

In this paper we take in consideration the most difficult case (of the three in the previous section), which is

$$Y'(s,t) = At^2 + Bs^2 + F \tag{17}$$

The conditions, which are rearranged to consider the space (s,t), become (simply)

$$A\Delta s^2 + B\Delta t^2 + (2As)\Delta s + (2Bt)\Delta t = 0 \tag{18}$$

$$\frac{1}{4}A\Delta s^2 + \frac{1}{4}B\Delta t^2 + (As)\Delta s + (Bt)\Delta t = \pm k \tag{19}$$

These conditions may be simplified to get

$$A\Delta s^2 + B\Delta t^2 = \mp 4k \tag{20}$$

$$(As)\Delta s + (Bt)\Delta t = \pm 2k \tag{21}$$

from the second condition you can express Δs as

$$\Delta s = \frac{\pm 2k - Bt\Delta t}{As} \tag{22}$$

We find a second degree equation for Δt

$$\Delta t^2(AB(As^2 + Bt^2)) \mp \Delta t(4ABkt) + 4Ak^2 \pm 4kA^2S^2 = 0 \tag{23}$$

If we have already done a step evaluation, one of the two solutions is of course the step which brings us back to the previous point of segmentation. This means that we can evaluate the new step from the previous one:

$$\Delta t_{i+1} = \frac{\pm ABkt}{AB(As^2 + Bt^2)} - \Delta t_i \tag{24}$$

that's, simply

$$\Delta t_{i+1} = \frac{\pm kt}{(-F)} - \Delta t_i \tag{25}$$

We should evaluate Δs in a similar way.

$$\Delta s_{i+1} = \frac{\pm ks}{(-F)} - \Delta t_i \tag{26}$$

The only problem is that this is a non controlled step generation, and it should give some stability problems. It is possible to adjust this algorithm with some controls.

After Segmentation, we should manage X-Scan through the linear segment. On this segment, y is supposed to be constant, because of the first step of the extreme algorithm.

X behaves on the segment as a parabolic function. X can be expressed as

$$u(t) = u_0 + r(u_1 - u_0) \quad (27)$$
$$v(t) = v_0 + r(v_1 - v_0) \quad (28)$$
$$x(t) = x_c + rx_b + r^2 x_a \quad (29)$$

Where (u_0, v_0) and (u_1, V_1) are the extreme points of the segment. We want to approximate the curve with sub-segments, so that the maximum error along sub-segments is defined as $\frac{1}{2}$. We find that this is possible with the fixed step:

$$\delta r = \sqrt{\frac{2}{\|x_a\|}} \quad (30)$$

4.4. Rendering on a GPU

When you have a curve piece it is very easy to automate the extreme algorithm described on graphics hardware. Using vertex program and display lists, it is possible to render easily the curve pieces, in a way similar [BS] do for PN-Triangles; instead, using recent geometry shaders it is easier, because you can implement all the algorithm directly on graphics hardware.

We have implemented the first case. The implementation is based on these ideas:

- We can send to the graphics hardware a fixed tessellation of a curve or a fixed triangle strip for Adaptive tessellation. This is done only once.
- The software manages the active intersections list and the preparation of curve pieces.
- During rendering, we can send the information about the patch (only once for patch) and the specific information of a curve piece (in particular the first and last point and k-error with sign), using local parameters.
- A vertex shader re-uses the reference tessellation or segmentation to evaluate the true surface points.

Using Geometry Shaders, it would be also possible to implement all the algorithm in hardware. We haven't had the possibility to do this, but it is one of the possible future works.

5. Scan-Line Rendering of Cubic Bezier Triangles

A Bezier Cubic Patch is formalized in this way.

$$\begin{aligned} P(u,v) = P_A(1-u-v)^3 + P_B u^3 + \\ P_C v^3 + 3P_{AB}(1-u-v)^2 u + 3P_{BA}(1-u-v)u^2 + \\ 3P_{CA}(1-u-v)v^2 + 3P_{AC}(1-u-v)^2 v + \\ 3P_{BC}u^2 v + 3P_{CB}uv^2 \\ +6P_{ABC}(1-u-v)uv \end{aligned}$$

1024x798	eMax	fps A	fps B	fps C
4x4	68.20	986	382	620
8x8	17.00	871	382	620
12x12	7.60	848	382	620
16x16	4.20	829	382	620
24x24	1.90	779	382	620
40x40	◇ 0.68	◇ 612	◇ 382	◇ 620

Table 1: *The animation rendered on a 1024x798 image. The case where the middle pixel error is obtained is with 40x40 triangles.(eMax: maximum distance from the original. fpsA: frame per second with the fixed tesselation. fpsB: frame per second with scan-line segmentation. fpsC: frame per second with scan-line tesselation.)*

307x239	eMax	fps A	fps B	fps C
4x4	20.40	6075	1520	2500
8x8	5.10	5602	1520	2500
12x12	2.30	4710	1520	2500
16x16	1.30	3773	1520	2500
24x24	◇ 0.57	◇ 2384	◇ 1520	◇ 2500
40x40	0.20	1093	1520	2500

Table 2: *The animation rendered on a 307x239 image. The case where the middle pixel error is obtained is with 24x24 triangles. (see table 1 for header information)*

Of course the extreme algorithm can't have the same results as its application for quadratic patches. In general it is possible to construct quasi-extreme algorithm based on local approximation of the surface with a quadratic patch. So, for example, when we are considering the X-Scan, the cubic curve

$$\begin{aligned} X(t) = At^3 + Bt^2 + Ct + D \\ X(t_i + \delta t) = X(t_i) + (3t_i^2 A + 2t_i B + C)\delta t \\ +(3t_i A + B)\delta t^2 + A\delta t_i^3 \end{aligned}$$

it can be approximated as

$$\begin{aligned} X(t_i + \delta t) = X(t_i) + (3t_i^2 A + 2t_i B + C)\delta t \\ +(3t_i A + B)\delta t^2 \end{aligned}$$

102 x 80	eMax	fps A	fps B	fps C
4x4	6.82	10353	5120	7900
8x8	1.70	7567	5120	7900
12x12	◇ 0.76	◇ 5342	◇ 5120	◇ 7900
16x16	0.42	4085	5120	7900
24x24	0.19	2388	5120	7900
40x40	0.07	1096	5120	7900

Table 3: *The animation rendered on a 102x80 image. The case where the middle pixel error is obtained is with 12x12 triangles. (see table 1 for header information)*

But we evaluate the maximum value of the truncated part $A\delta t_i^3$. Then, instead of using a fixed precision equal to $\frac{1}{2}$, we use a variable precision which is $\frac{1}{2} - |max(A\delta t_i^3)|$. This allows us to work directly on the parabolic function, because we have already taken care of the third degree derivative. Other rendering sub-processes may be derived in the same manner. This principle may be applied generally to every kind of patch, to re-conduce the complex case to the simplest case of quadratic patches.

6. Conclusions and Future Works

We have shown possible ways to re-use some concepts from scan-line rendering, and their application to graphics hardware. We have also proposed a new algorithm; applied to quadratic patches, this algorithm becomes a very fast algorithm, based on simple steps evaluation. This method may be used to make the rendering of curved surfaces fast with a pixel precision rendering and so with adaptive solutions. The performances are good. We hope to get new interesting results working on the implementation of our algorithm directly on geometry shader units: this can be considered one of our future works. We also hope to reach a fast implementation of our algorithm on spline and NURBS surfaces.

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

[Bli78] BLINN J. F.: A scan-line algorithm for displaying parametrically defined surfaces. July 1978. 2, 3

[Bru] BRUIJNS J.: Quadratic bezier triangles as drawing primitives. *Workshop on Graphics hardware Lisbon*, Portougal, 1998. 4

[BS] BOUBEKEUR T., SCHLICK C.: Generic mesh refinement on gpu. *Graphics Hardware*, July 2006, pp. 99–103. 3, 7

[Cat] CATMULL: E.e.computer display of curved surfaces. *Proc. IEEE Conference on Computer Graphics, Pattern Recognition and Data Structures*, Los Angeles, May 1875 2

[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. 3

[DS] D. SCHWEITZER C.: Scanline rendering of parametric surfaces. *IEEE Computer*. 2

[HT] HORMANN K., TARINI M.: A quadrilateral rendering primitive. Graphics Hardware, Florence, 2004, pp. 99–103. 1

[JML] J. M. LANE L. C. CARPENTER T. W. J. F. B.: Scan line methods for displaying parametrically defined surfaces. *Communication of the ACM*, Los Angeles, V. 23, N. 1, January 1980 pp. 23–25. 2, 3, 4

[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

[Mar] MARTINELLI: A new model for 3d graphical rendering. *WSCG 2006 short Communications*, February 2006, pp. 133–140. 4

[Mar99] MARSH D.: *Applied Geometry for Computer Graphics and CAD*. Springer, 1999. 1

[ML98] MOORE R., LOPES J.: Paper templates. In *TEMPLATE'06, 1st International Conference on Template Production* (1998), INSTICC Press. 1

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

[TB] T. BARRERA A. HAST E. B.: Surface construction with near least square acceleration based on vertex normals on triangular meshes. 5

[VA03] VLACHOS A. PETERS J. BOYD C., MITCHELL J. L.: Curved pn triangles. *ACM Press*, 2003 5

[Whi] WHITTED T.: A scanline algorithm for computer display of curved surfaces. 2