A new model for 3D graphical rendering

Alessandro Martinelli
University of Pavia
Faculty of Engineering
Strada Ferrata Pavia PV(27100),Italy
AlexMartinelli@jumpy.it

ABSTRACT

One of the most important tasks of a traditional 3D Rendering engine is the projection on the image plane of geometrical structures (such as triangles or lines). This operation takes place in the middle of the rendering pipeline, between the vertex shader and the fragment shader: its aim is just that of creating fragment data from vertex data. The solution of the projection problem is necessarily bound to the solution of a great number of systems of equations, where the complexity of the equations is in general related to the properties of the geometrical structures. To make this process fast, the most adopted solution is that of using linear models, so that the systems become linear and the module gets the simplest implementation. Unfortunately, linear models have some limitations: the solution is to use approximation, but to get good models they are necessary a lot of linear structures, in particular a lot of triangles; modern 3D Rendering Engines may automate the process of converting non linear models in triangles, but this does not reduce the occupation of memory and doesn't eliminate linear approximation. In this article I consider a non linear model (the Lembo model) for geometrical structures in a 3D rendering engine: firstly I show the properties of the model; then I show an efficient algorithm to solve the projection problem directly on the model equations.

Keywords

Approximation Models, 3D Models, Rendering Algorithms, Non-linear Graphics, Real-Time Graphics

1. INTRODUCTION

I want to show the advantages in using a quadratic triangle model to work with the traditional polygonal model directly in the graphics hardware. Graphics is often concerned with approximation: if you want to represent a function $y = f(t)$, you may consider the linear approximation, or you may consider high order polynomial approximations. In 3D graphics they are used surfaces, and also surfaces may be approximated with many approximation techniques, but there still remains the problem of rendering; rendering is generally concerned with the solutions of systems of equations, and this may take an enormous computational cost.The GPU hardware works with triangles; recently, subdivision meshes have been introduced in the graphical hardware:but they make it possible only the refinement of triangular meshes (see [Bou01a], [Shi00a]).

Short Communications proceedings ISBN 80-86943-05-4 WSCG' 2006, January 30-February 3, 2006 Plzen, Czech Republic. Copyright UNION Agency - Science Press

In general the approximation of a curved surface is concerned with the concept of LOD (Level of Detail), and with some algorithm to decide the triangles to be used to approximate the surface (see for example [Bru00a]).

Only in non-Real Time Graphics, such as for ray-tracing, some techniques are used to render non-linear elements directly without a triangulation process (see [Mar00a]). Our aim is to introduce non linear rendering techniques also in common real-time rendering pipelines.

Scan Line Rendering Techniques have been introduced with this aim. However, they have been thought to work with generical non linear models (for example [Lan00a],[Sch00a],[SAdDC12] e [Sed01a]). This Techniques are known to be slow and less amenable to hardware acceleration. In this article I show that it is possible to construct a fast scan line rendering algorithm for a new model which may be easily introduced in the accelerating parallel hardware. Such a model is a form of Quadratic Bezier Triangle (see [Bru00a]), with some other informations which make it very close to PN Triangles (see [Vla00a] or [Bou00a]) or Steiner Patches (see [Bre00a], [Sed00a]).

2. THE QUADRATIC FIXED-DOMAIN REFERENCED PN TRIANGLE (OR LEMBO) MODEL

The model I have worked on is a quadratic bezier triangle defined on a standard dominion (the triangle with vertices $(0,0)$, $(1,0)$ $(0,1)$) with six reference construction points (the points $(0,0)$, $(1,0)$, $(0,1)$, $(\frac{1}{2},0)$, $(0,\frac{1}{2})$, $(\frac{1}{2},\frac{1}{2})$), and a separate Normal function on the same domain. I use to call it also Lembo, an Italian word which may be translated as 'strip' or 'patch'.

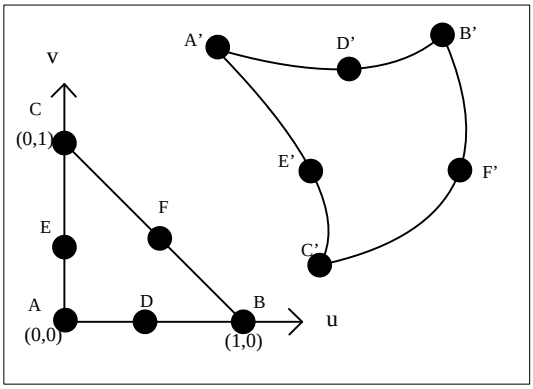

Figure.1 A lembo representation: there are the parametric function (top), the standard dominion with its reference points (bottom), the lembo image and the modeling points.

Such a model is no really new, but it may be considered as the compositions of different ideas from other models. Let us consider Steiner Patches and PN Curved Triangles.

Steiner Patches and Quadratic Bezier Triangles

A Steiner Patch is a Quadratic Bezier Triangles, with a fixed domain expressed in this way

$$\begin{cases} x(u,v) = a_x u^2 + b_x v^2 + c_x uv + d_x u + e_x v + f_x \\ y(u,v) = a_y u^2 + b_y v^2 + c_y uv + d_y u + e_y v + f_y \\ z(u,v) = a_z u^2 + b_z v^2 + c_z uv + d_z u + e_z v + f_z \\ u \geq 0 \\ v \geq 0 \\ u + v \leq 1 \end{cases}$$

The model is similar to Quadratic Bezier Triangles by Bruijns (see [Bru00a]), considering the projection onto the u-v plane of their domain (with the equation $w = 1 - u - v$).

A lot of work have been done to study the properties of similar models, for example in [Bar00a], [Bre00a] and [Sed00a].

Curved PN Triangles

A curved PN triangle (see [Vla00a]) uses two different functions to model the geometry and normals of a patch. The Geometry of the PN Triangle is defined by a cubic patch b

$$\begin{aligned} b &: \mathbb{R}^2 \to \mathbb{R}^3 \\ b(u,v) &= \sum_{i+j+k=3} b_{ijk} \frac{3!}{i!j!k!} u^i v^j w^k \\ u,v,w &\geq 0 \\ u+v+w &= 1 \end{aligned}$$

The geometry function does not create C^1 continuity, but it is possible to simulate it with a normals function. The normal component of a curved PN Triangle is a quadratic function of the normal data, defined as

$$\begin{aligned} n &: \mathbb{R}^2 \to \mathbb{R}^3 \\ n(u,v) &= \sum_{i+j+k=2} n_{ijk} u^i v^j w^k \\ u,v,w &\geq 0 \\ u+v+w &= 1 \end{aligned}$$

The Lembo Model and the Mesh Refinement Techinques

The model I have consider is similar to Curved PN Triangles, but to model geometry I use a quadratic patch instead of a cubic one. The domain of the patch is fixed and it is in the same form of Steiner Patches.

$$\begin{aligned} b &: \mathbb{R}^2 \to \mathbb{R}^3 \\ n &: \mathbb{R}^2 \to \mathbb{R}^3 \\ b(u,v) &= \sum_{i\leq 2, j\leq 2} b_{ij} u^i v^j \\ n(u,v) &= \sum_{i\leq 2, j\leq 2} n_{ij} u^i v^j \\ u,v &\geq 0 \\ u+v &\leq 1 \end{aligned}$$

It's possible to construct Lembos meshes, using the six reference points for construction, or it's possible, as for PN Triangles, to use the Lembo

Model to refine a triangular mesh instead of creating a lembos mesh from scratch. For the geometry function it's necessary to find the edges middle points; this is possible for example using the Near Least Square Acceleration proposed in [Bar00a]; such a method produces both the control points and the control normals on the edges. An edge of a quadratic Bezier triangle between two points P_1 and P_2 with normals N_1 and N_2 may be expressed with a parameter t

$$\mathrm{f} = \mathrm{a}t^2 + \mathrm{b}t + \mathrm{c}$$

The function f has to interpolate the ending points of the edge, and the tangents in the ending points should be tangent to the normal described in the ending points. This conduces to a four equations system. It is not possible to find an exact solution, but it is possible to find f so that the integral

$$\int_0^1 \|\mathrm{f"}\|^2 dt$$

has the minimum value. This is the algorithm (see [Bar00a] for details).

$$\begin{aligned} T_1 &= N_2 - N_1(N1 \cdot N2) \\ T_2 &= -N_1 - N_2(N1 \cdot N2) \\ \alpha &= \frac{T_1 \cdot T_2}{T_1 \cdot T_1} \\ \beta &= \frac{P \cdot T_1}{T_1 \cdot T_2} \\ \alpha' &= \frac{T_1 \cdot T_2}{T_2 \cdot T_2} \\ \beta' &= \frac{P \cdot T_2}{T_1 \cdot T_2} \\ f(t) &= (\frac{\alpha'\beta' T_2 - \alpha\beta T_1}{2})t^2 \\ &(P + \frac{\alpha\beta T_1 - \alpha'\beta' T_2}{2})t + P_1 \end{aligned}$$

where T_1 and T_2 are the tangent required in P_1 and P_2 and $P = P_2 - P_1$.

The points and the normals are evaluated using only information on the edge ending points, so the algorithm will give the same values for two patches with a common edge.

The construction of the normal function may be done also in the similar way used for PN Triangles; another possibility is to consider the geometry of two lembos with a common edge and to evaluate the average of the two normals in the middle point; this approach has the drawback to construct a lembo using data from other lembos.

Once the six points and six normals for the lembo have been found, it is possible to construct the Lembo function from a simple linear transformation, making the values of the lembo in the reference points $\{A(0,0), B(1,0), C(0,1), D(\frac{1}{2},0), E(0,\frac{1}{2}), F(\frac{1}{2},\frac{1}{2})\}$ be equal to the value of the constructing points $\{A', B', C', D', E', F'\}$ in the Point-Normal Space (for every point it's given the position (x', y', z') and a normal vector (n'_x, n'_y, n'_z)).

$$\begin{cases} A'_x = fx \\ B'_x = ax + dx + fx \\ C'_x = bx + ex + fx \\ D'_x = \frac{1}{4}a_x + \frac{1}{2}d_x + f_x \\ E'_x = \frac{1}{4}b_x + \frac{1}{2}e_x + f_x \\ F'_x = \frac{1}{4}a_x + \frac{1}{4}b_x + \frac{1}{4}c_x + \frac{1}{2}d_x + \frac{1}{2}e_x + f_x \end{cases}$$

$$\begin{cases} f_x = A'_x \\ e_x = 4(E'_x - A'_x) - (C'_x - A'_x) \\ d_x = 4(D'_x - A'_x) - (B'_x - A'_x) \\ b_x = C'_x - A'_x - e_x \\ a_x = B'_x - A'_x - d_x \\ c_x = 4F'_x - a_x - b_x - 2d_x - 2e_x - 4A'_x \end{cases}$$

where I have considered only the x component of the geometry function. It is the same also for the normal function, they are both quadratic.

3. RENDERING WITH TESSELLATION

One possibility for rendering is tessellation. Such technique is fast, in particular because it may be introduced directly in the actual hardware. Recently some GPUs have introduced a module to manage the subdivision meshes, and this allow the programmers to use common meshes in the CPU and get the mesh refinement only on the GPU, and this allow better performance (see [Bou01a],[Shi00a]). The subdivision techniques for a quadratic triangular patch have been discussed for example by Bruijns in [Bru00a], and they divide in fixed step techniques and variable step ones.

4. SCAN LINE RENDERING

Scan Line Rendering is a rendering technique that implies the solution of systems of non linear equations. A non linear model (in general a cubic patch) is intersected with a plane in the R^3 space. Generally they are made some assumptions: the screen plane is the x-y plane, with the z axis orthogonal to the screen; every projection transformation on the model points have already been evaluated. So a parametric model is intersected with the plane $y = y_j$, for a set of y_j which

are in the interval between the minimum and the maximum value for y on the surface. The steps of a scan line algorithm are:

```
; Scan Line Rendering Procedure

proc render(Patch)

y_j, y_min, y_max

[y_min, y_max]=evalMinMax(Patch)

for y_j = y_min; y_j ≤ y_max; y_j ++

    ; Study the equation y_j = y(u,v)

    renderEquation(y_j);

endproc
```

In general a scan line algorithm requires the use of a numerical method, or more then one, and the stability properties of the method depend on the model of the patch rendered. The use of a numerical method and the managing of numerical errors to guarantee stability make this algorithms generally slow.

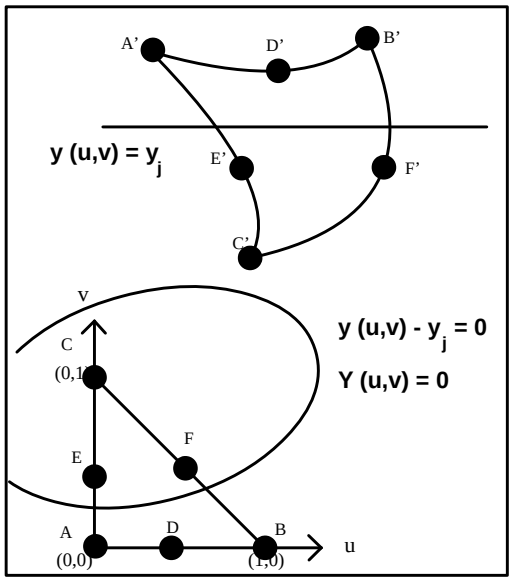

Figure.2 $The Y(u,v) = 0$ **curve**

5. A VERY EFFICIENT SCAN LINE ALGORITHM FOR LEMBOS

I'm going to show an efficient scan line algorithm for lembos. In particular this algorithm doesn't require numerical techinques, doesn't suffer stabilities problem and all is described in terms of solutions of second degree equations.

The $Y(u,v) = 0$ curve

The $Y(u,v) = 0$ curve has the form

$$Y(u,v) = a_y u^2 + b_y v^2 + c_y uv + d_y u + e_y v + f_y - y_j = 0$$

The equation represent a conic in the (u,v) space. In particular they have to be studied the arcs of this curve which are inside the domain. Along these arcs they are constructed points for the next rendering steps.

Finding maximum and minimum values for y

The maximum and minimum values for y are easy to find. They may be one of the following alternatives:

(1) One of the Vertices A',B',C' of the Lembo
(2) One of the maximum or minimum points for the edges, which are arcs of parabola. They may be one of the Vertices (already considered) or one of this values:
  (a) $u = -\frac{d_y}{2a_y}, v = 0$
  (b) $u = 0, v = -\frac{e_y}{2b_y}$
  (c) $u = -\frac{-2b_y + c_y + d_y - e_y}{2(a_y + b_y - c_y)}, v = 1 - u$
(3) The solution of the linear system of equations with the two partial derivatives functions equal to zero $2a_y u + c_y v + d_y = 0$ and $2b_y v + c_y u + e_y = 0$

The extreme method: linearization of the $Y(u,v) = 0$ curve

The idea is to work with extreme linearization: given o point of the $Y(u,v) = 0$ curve, it has to be found another point so that along the segment between the two points the maximum value of $\|Y\|$ (so the maximum error of the approximation of the curve piece with the segment) is exactly a predefined value k. This is very simple, because the $Y(u,v)$ function along the segment behaves as a parabola. So, the maximum error is easily found in the middle point. All the parts of the equation $Y(u,v) = 0$ inside the lembo domain are approximated with an array of segments, with every segment having its maximum error of approximation in the middle point and that error being exactly the one established. In general I suggest a pixel-oriented error choice, such as $k = \frac{1}{2} pixel$.

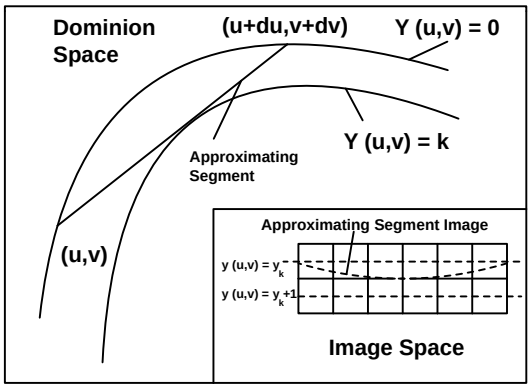

Figure.3 The segmentation of the $Y(u,v)=0$ equation

Once the segments have been found, it is possible to consider the function $x = x(u,v)$. It is very simple to proceed with the same kind of approximation, trying to create a sub-segmentation, where every subsegment has both the property to have an approximatively constant value for y, with an error not higher that the one established, and to have an x with an approximatively linear trend, with also the error of approximation controlled and not higher than the established k. So the generation of fragments along the subsegment may be done with a simple linear equation for the u and v components, which may be evaluated step by step with simple increments. In the next sub sections I'm going to show the steps of this algorithm: the managing of the intervals, the construction of the segment and the construction and rendering of sub-segments.

Managing Intervals

The first problem is to find the pieces of the Y=(u,v) curve which are inside the domain. This may be done constructing intervals inside the domain polygon. The extreme points of this intervals have to be the intersection of the curve with the domain edges, so they may be

(1) The solutions of the equations $a_y u^2 + d_y u + f_y - y_j = 0, v = 0$
(2) The solutions of the equations $u = 0, b_y v^2 + e_y v + f_y - y_j = 0$
(3) The solutions of the equations $(a_y + b_y - c_y)u^2 + (d_y - e_y + c_y - 2b_y)u + b_y + e_y + f_y - y_j = 0, v = 1 - u$

This points may be used has beginning and ending point for the next step, which is the segmentation one.

The segmentation problem

An approximation of the $Y(u,v)=0$ equation can be evaluated step by step with segmentation. If $P_0(u_0,v_0)$ is a point of the curve (inside the domain), to construct the segment I found a point $P_1(u_0+du, v_0+dv)$, where du and dv satisfy these conditions

$$\|Y(u_0 + \frac{1}{2}du, v_0 + \frac{1}{2}dv)\| = k$$
$$Y(u_0 + du, v_0 + dv) = 0$$

so in the middle-Point of the segment it is required the maximum error k, while the point P_1 has to stay on the curve.

The equations may be written in this way

$$\frac{1}{2}\frac{\partial^2 Y(u_0,v_0)}{\partial u^2}du^2 + \frac{1}{2}\frac{\partial^2 Y(u_0,v_0)}{\partial v^2}dv^2 + \frac{\partial^2 Y(u_0,v_0)}{\partial u \partial v}dudv + \frac{\partial Y(u_0,v_0)}{\partial u}du + \frac{\partial Y(u_0,v_0)}{\partial v}dv = 0$$
$$\frac{1}{2}\frac{\partial^2 Y(u_0,v_0)}{\partial u^2}(\frac{du}{2})^2 + \frac{1}{2}\frac{\partial^2 Y(u_0,v_0)}{\partial v^2}(\frac{dv}{2})^2 + \frac{\partial^2 Y(u_0,v_0)}{\partial u \partial v}\frac{du}{2}\frac{dv}{2} + \frac{\partial Y(u_0,v_0)}{\partial u}\frac{du}{2} + \frac{\partial Y(u_0,v_0)}{\partial v}\frac{dv}{2} = \pm k$$

Simplifying these equations I obtain the following

$$\frac{\partial Y(u_1,v_1)}{\partial u}du + \frac{\partial Y(u_1,v_1)}{\partial v}dv = \pm 4k$$
$$\frac{\partial^2 Y(u_1,v_1)}{\partial u^2}du^2 + \frac{\partial^2 Y(u_1,v_1)}{\partial v^2}dv^2 + 2\frac{\partial^2 Y(u_1,v_1)}{\partial u \partial v}dudv = \mp 8k$$

which is a second degree system of equations. The choice for the sign of $k\pm$ depends on the curvature of the curve and may be done before the segmentation phase.

The algorithm produces segments step by step. When P_1 has been found, the solution for P_2 (and so for the following points) is very easy to find. In fact, one of the two solutions of the second degree system for (du, dv) is the one which brings back to the point P_0. So it's possible to find the second solution given the first.

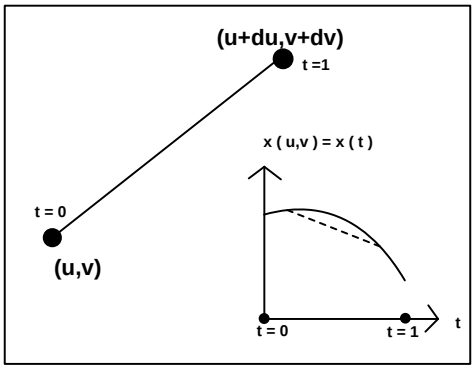

Figure.4 The sub-segmentation with the $x(u, v)$ function along the segment with approximately constant y.

The sub-segmentation

When a segment has been evaluated, it may be expressed in this form

$$\begin{aligned} t &\in [0,1] \\ u(t) &= bt_u t + ct_u \\ v(t) &= bt_v t + ct_v \\ x(t) &= at_x t^2 + bt_x t + ct_x \\ z(t) &= at_z t^2 + bt_z t + ct_z \\ etc.(&if\, there\, are\, other\, functions \\ &as\, normal\, functions) \end{aligned}$$

The x function may be segmented in a way similar to the one used for y. A step dt is evaluated so that the linear approximation of x between t_i and $t_i + dt$ has k as maximum value; the error function is always a parabola, so the maximum error is in the middle point.

$$\begin{aligned} \|\frac{x(t_i) + x(t_i + dt)}{2} - x(t_i + \frac{1}{2})\| &= k \\ dt &= \pm\sqrt{\frac{\pm 4k}{at_x}} \end{aligned}$$

dt is a fixed step. The sub-segmentation process is truly fast. Once a subsegment has been found, it can be considered as linear both for y and x, then rasterization is truly simple.

6. PERFORMANCES

The evaluation of performances is a bit difficult, because it's necessary to consider many factors. Of course the rendering of a lembo is slower than the rendering of a triangle, but if we approximate a lembo with a lot of triangle the rendering time becomes similar or overcomes the one for the lembo with the rendering . In general the rendering of single lembo is 2 or 3 times more slow than the one of a single triangle, but 2 or 3 are not so much if we consider the possibility to use dedicated hardware.

Another reason which make difficult an evaluation is that actually there is not an hardware for lembos, but a comparison should be done exactly on the hardware level, because there is no reason to perform it via software.

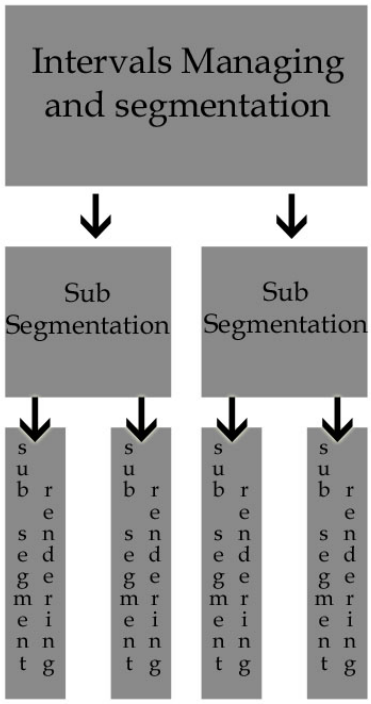

Figure.5 A possible parallel architecture which implements the extreme method for lembos

7. CONCLUSIONS

The Lembo Model makes it possible to get a better approximation reducing the number of elements used (lembos or triangles) and so reducing the number of vertexes processed to obtain high quality renderers; the most evident consequence is the reduction of memory occupied in graphical applications.

The extreme method shown in section 5 is a very fast method for lembos rendering. It's a scanline method, but it doesn't require numerical approximation if it is possible to have a fast and accurate instrument to solve equations of the second degree. Moreover, the algorithm may be divided in more steps, and these steps are more amenable for hardware acceleration than over methods for scan line rendering, in particular the production

of subsegments from segments and the rasterization of the subsegments; they are also amenable for a parallel architecture, since the production of subsegments and the rendering of different subsegments may be done in parallel.

8. REFERENCES

[Bar00a] T. Barrera, A. Hast, E. Bengtsson, Surface Construction with Near Least Square Acceleration based on Vertex Normals on Triangular Meshes, SIGRAD (2002)

[Bou00a] T. Boubekeur, Patrick Reuler, Christophe Schlick, Scalar Tagged PN Triangles, Eurographics 2005

[Bou01a] T.Boubekeur, Christophe Schlick, Generic Mesh Refinement on GPU, Graphics Hardware 2005, ACM 2005

[Bre00a] D. E. Breen, Creation and smooth-shading of Steiner Patches Tessellations, proceedings of 1986 ACM Fall Joint Computer Conference, 1986

[Bru00a] J. Bruijns, Quadratic Bezier Triangles As Drawing Primitives,1998 Workshop on Graphics hardware Lisbon Portugal, ACM 1998

[Com00a] V.Comincioli,Analisi Numerica: Metodi, Modelli, Applicazioni, Mc Graw-Hill, 1995

[Lan00a] J.M.Lane,L.C.Carpenter,T,Whitted, J.F.Blinn, Scanline Methods of Displaying Parametrically Defined Surfaces, J.D.Foley Editor, ACM 1980

[Mar00a] William Martin - Elaine Cohen - Russell Fish - Peter Shirley, Practical Ray Tracing of Trimmed NURBS Surfaces, http://www.cs.utah.edu/vissim/papers/raynurbs/raynurbs.html 2000-08-02

[Qua00a] A Quarteroni, F. Saleri, Introduzione al calcolo Scientifico, Springer 2002

[Sch00a] D. Schweitzer, E.S.Cobb, Scanline Rendering of Parametric Surface, ACM 1982

[Sed00a] T.W.Sederberg, David C. Anderson, Ray Tracing of Steiner Patches, ACM 1984

[Sed01a] T.W.Sederberg,A.K.Zundel, Scan Line Display of Algebraic Surfaces, Computer Graphics, Volume 23, Number 3, July 1989

[Shi00a] L.Shiue,I.Jones,J.Peters, A Real Time GPU Subdivision Kernel, ACM ToG, 2005

[Vla00a] A. Vlachos, J. Peters, C. Boyd, J. L. Mitchell, Curved PN Triangles, ACM Press, 2001

[Wil00a] D.F.Wiley, H.R.Childs, B.F.Gregorski, B.Hamann, K.I.Joy, Contouring Curved Quadratic Elements, Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization (2003)

[Whi00a] T.Whitted, A ScanLine Algorithm for Computer Display of Curved Surfaces, Proc. 5th Conf Computer Graphics and Interactive Techinques, Atlanta, 1978

[Wri00a] Richard S.Wright Jr.,Benjamin Lipchak, OpenGL SuperBible, ThirdEdition, SAMS, 2004

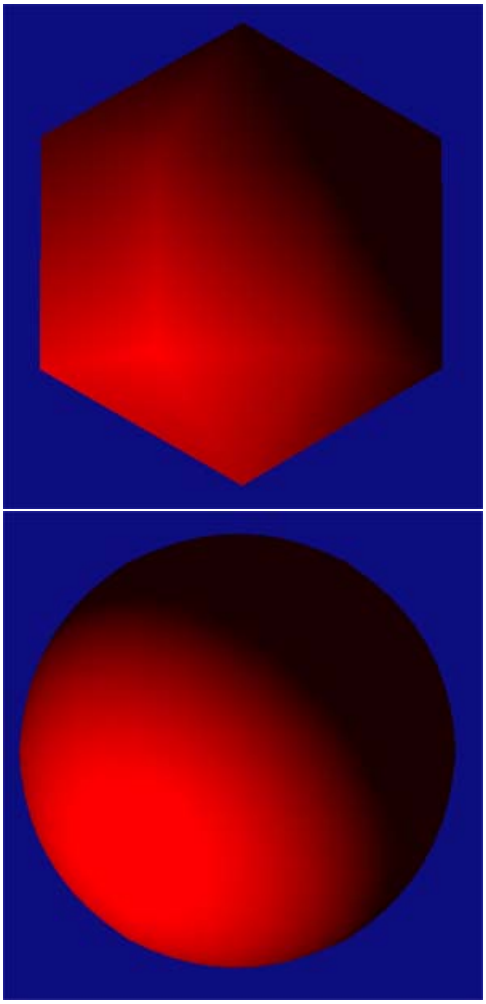

Figure.6 Two spheres, the first constructed with 24 triangles, the second with 24 lembos.

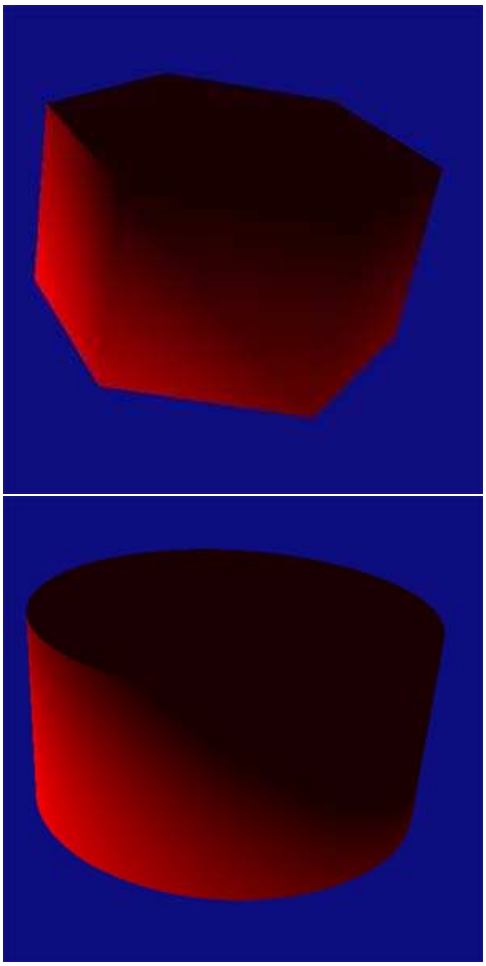

Figure.7 Two cylinders, the first constructed with 24 triangles, the second with 24 lembos.

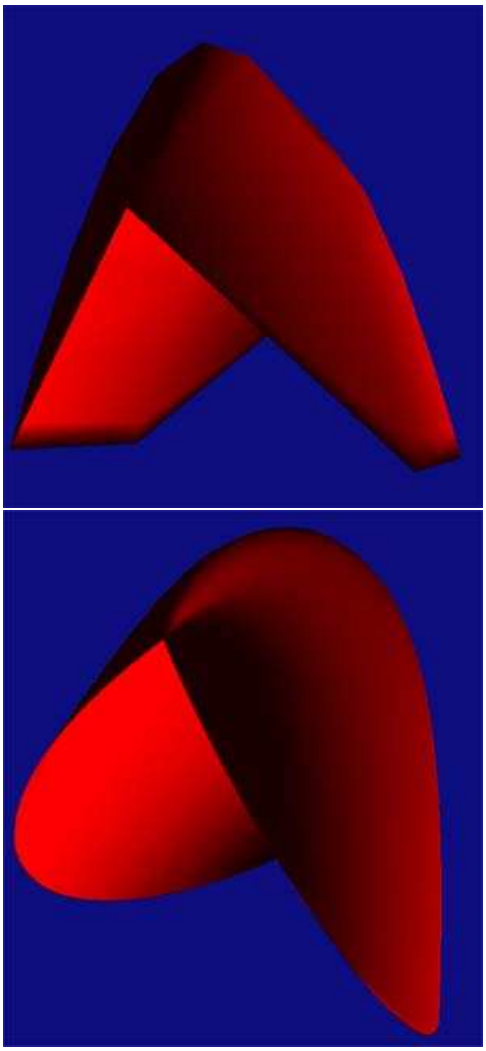

Figure.8 Two images of the same surface, the first constructed with 96 triangles, the second with 96 lembos.