A New Method for Rasterization of Parametric Curves and Its Application to Fast Bezier Patch Shading.

Sergey V.Chirikov, Timour T.Paltashev, Pavel A. Balabko.
State Education Center of Nuclear Energy Ministry, Aerodromnaya str., d. 4, St. Petersburg,197348, Russia
E. Mail: sergchi@arcadia.spb.ru, tpaltash@S3graphics.com, pashab@graph.runnet.ru

ABSTRACT.

This paper presents a unified rasterization method to render parametric curves and patches. The algorithms are extremely simple and fast. No multiplication and division are required. The proposed method combines strong sides of parametric and nonparametric representation of curves and patches. A class of curves named as L-curves is described in here. There is a simple correspondence between their parametric and nonparametric forms. It gives a possibility to divide patch rendering process into two tasks: presentation of a patch in L-form and rasterization of obtained L-curves. L-curves can represent both conic arc and Bezier curves and it is possible to use them for rasterization of B-splines and conic splines. Also L-curves can be used for interactive graphical design directly (font construction, for example). Hardware-oriented L-curve generation algorithms may be used for nonlinear color interpolation for Bezier patch shading. Final quality of this shading technique is comparable with the Phong shading method.

Key words: curve incremental drawing, Bresenham's algorithm, Bezier curve, Bezier patch, B-spline, conic interpolator, raster graphics.

1. INTRODUCTION.

Problems of curve approximation with the set of adjoined pixels arose in the middle of 60's and got an efficient solution in works of Bresenham [1] and Pittway [2]. Fundamental algorithms proposed by them have demanded to represent 2**D** curves as nonparametric equation $F(x,y)=0$, where $F(x,y)$ should be linear, quadratic or cubic form.

Let us consider strong and weak points of fundamental algorithms [1,2] of curve generation. Evidently their main merit consists of using an integer arithmetic and minimizing the number of addition operations. It is required one addition for vector, three for quadratic form and five for cubic form to get the location of one pixel. No multiplication and division are required.

The weak sides of those algorithms include the following points. Algorithms [1,2] are intended to rasterize only 2**D** curves, which should be represented in nonparametric form. Those algorithms do not allow curve generation in 6**D** space X,Y,Z,R,G,B, that is necessary for realistic picture rendering.

In computer geometric design kits the curves and surfaces are usually defined geometrically. Control points and vectors are used to control the shape of the curve. Bezier curves and B-splines can be defined on this way, for example. If we omit evident cases of vector and circle we can see that setup processing of nonparametric representation of a nontrivial curve may demand more time clocks than generation of the curve with methods noted above. Initializing the algorithm [2] can be more expensive than carrying out its main part. The algorithms [2] are formal application of Bresenham's vectors approach for quadratic and cubic forms. Those simple algorithms may be used only for rasterization smooth curves. If the curve has a zone with a high curvature, the algorithms lose stability and begin to

generate wrong pixel coordinates. The vectors and circles are just curves with a constant curvature and so the algorithms [1,3] work always correctly to generate them.

The surface nonlinear shading method proposed in this paper is based on L-curve generator implementation. It's computation cost is comparable with Gouraud's shading method and L-curve generation can be completely implemented in hardware. This method also may be implemented to increase the quality of Phong's shading method by means of nonlinear normal interpolation in 6**D** space (X , Y , Z , n_x , n_y , n_z).

2. BASIC TOOLS FOR RASTERIZATION ALGORITHMS DESIGN.

2.1. Method of vector generation in multidimensional space.

Geometrically a vector may be defined by its start and end points. Let us assume that the start point located in the origin of coordinate, so this vector will be defined by its end point coordinates $X,Y,Z, \ldots$.

Bresenham's algorithm [1] implements the increments of current vector coordinates on each iteration. The major coordinate gets an increment in each iteration while the other coordinates get it conditionally only in some iterations. A multidimensional vector may be represented by a set of 2D vectors (X,R), (Y,R), (Z,R), ... where $R = max(|X|, |Y|, |Z|, ...)$. Evidently mD vector generation may be done by running simultaneously of m 1D Bresenham generators, which work in parallel (increments in 'R'- directions are master ones). During R iterations each 1D vector generator generates increments of pixel coordinates along axes X,Y,Z. The error of pixel approximation of a mD vector does not exceed the error of classic algorithm [1] for a 2D vector. It follows from the structure of the generation algorithm. The initialization of the 1D vector generator is made according to known rules [1]. Further the 1D vector generator will be called a linear interpolator (**LI**).

Let's assume that the **LI** executes one iteration when its input gets a control signal. During R iterations an input parameter t is varying from 0 to 1 and can be represented by the following equation: $t=i/R$, where i - index number of current iteration. While the input parameter t increases from 0 to 1 the output of each **LI** generates unit increments. After being summed the outputs are equal to the current pixel coordinates $X,Y,Z, \ldots$ of the m**D** vector.
Analytically this process is described by the following equation:

$$\mathbf{A}(t) = \mathbf{A}_0 + \mathbf{A}_{01} * t; \quad t \in [0,1], \qquad \textbf{(1)}$$

where $\mathbf{A}_{01} = \mathbf{A}_1 - \mathbf{A}_0$; $\mathbf{A}_0$ - beginning point; $\mathbf{A}_1$ - ending point of the vector.

The pixel rate of hardware-implemented vector generator will not be effected by space dimension but the hardware cost would be multiplied by dimension m (number of LI).

2.2. The conic arc generation method.

Arc incurved in triangle $\mathbf{A}_0\mathbf{A}_1\mathbf{A}_2$ and shown in Fig.1 can be represented by the following equation

$$\mathbf{A}(\tau) = \mathbf{A}_0 + \mathbf{A}_{01} * X(\tau)/R + \mathbf{A}_{12} * (1 - Y(\tau)/R), \qquad \textbf{(2)}$$

where $X(\tau)$, $Y(\tau)$ - symmetrical curve incurved in the square with the following end conditions:

$$X(\tau_b) = Y(\tau_e) = X'(\tau_e) = Y'(\tau_b) = 0; \quad Y(\tau_b) = X(\tau_e) = R, \qquad \textbf{(3)}$$

where τ - $[\tau_b, \tau_e]$ - parameter; τ_b, τ_e - start and end values of parameter τ.

Curve $X(\tau)$, $Y(\tau)$ may have some analytical representations or may not have it at all. If $X(\tau)$, $Y(\tau)$ are quadratic functions we can perform our counting with the ending conditions (3) so we can derive a nonparametric form of the curve $X(\tau)$,$Y(\tau)$

$$n(X^2+Y^2) - 2mXY + 2mR(X + Y) - (2m + n)R^2 = 0. \quad \textbf{(4)}$$

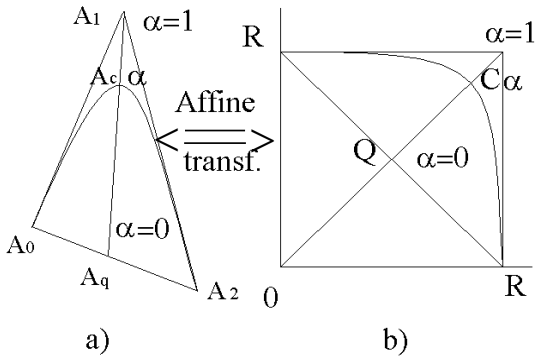

Figure 1: The conic arc incurved in triangle and correspondence between its parametric a) **A**(t) and nonparametric b) $F(x,y) = 0$ definition methods.

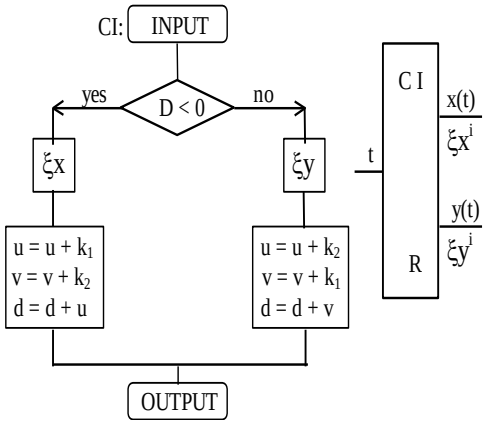

Figure 2: Conic interpolator a) and its functional scheme b).

After finding the invariants of the quadratic form (4) we can comclude that a fraction m/n is an eccentricity of conic (4). In geometry it is known that three points ratio on a straight line is invariant under affine transformation. Equation (2) defines an affine transformation of the 2**D** curve $X(\tau)$, $Y(\tau)$ to multidimensional space (m**D**), Fig.1. The point **C** of the intersection of the curve $X(\tau)$, $Y(\tau)$ with the diagonal of the square divides this diagonal on some proportion. By the affine transformation (2) the point **C** of arc (4) is transformed into the point $\mathbf{A}_c$ of the arc (2) but the three point ratio had remained invariant, albeit the eccentricities of conics (2) and (4) are different.

Based on previous consideration we can conclude the eccentricity m/n of the conic (4) is defined by the location of the intersection point of the curve incurved in the triangle with the median of this triangle. Location of the point can be defined by the factor $\alpha \in [0,1]$. After considering symmetrical property of conic (4), we got

$$m/n = \frac{(\alpha^2 + 2\alpha - 1)}{(1 - \alpha)^2}, \quad \textbf{(5)}$$

where $\mathbf{A}_c = \mathbf{A}_q(1 - \alpha) + \mathbf{A}_1\alpha$; $\mathbf{A}_q = \frac{1}{2}(\mathbf{A}_0 + \mathbf{A}_1)$.

We have deduced that incurved in the triangle conic arc is defined by four points $\mathbf{A}_0$, $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{A}_c$ or, on the other hand, by triangle vertices and the factor α which defines the location of point $\mathbf{A}_c$. After we have stated the geometrical representation of conic arc, let us consider a conic arc rasterization method.

First let's note that generation of the 2**D** conic (4) can be implemented with some already existing algorithms [2]. The algorithm shown in Fig. 2a has been developed on the base of the Bresenham's approach. Further this algorithm will be called as the conic interpolator (**CI**) algorithm. To initialize **CI** it's necessary to get initial values for the following variables: $u = 4n + 2m$; $v = u - 4R(n + m)$; $k_1 = 4n$; $k_2 = 4m$; $d = (n + m)(1 - 2R)$. To rasterize conic (4)

the **CI** must execute 2*R* iterations for any type of the conic arc: elliptic ($\alpha < ½$), parabolic ($\alpha = ½$) or hyperbolic ($\alpha > ½$). We have to note that our proposed **CI** algorithm is always stable versus Pittway's algorithm [2]. The stability of the **CI** is a result of exclusion of diagonal increments. It admits increments only in *X* or *Y* axes directions, and makes it possible to increase curve inflection within one raster unit up to $\pi/2$.

Let's see how the **CI** algorithm works. The functional scheme is represented in Fig.2b. The parametric function $X(\tau),Y(\tau)$ can have several analytical representations. While the curve generation is running the current value of τ can be unknown but on the other hand we can count a current number of the iteration *i*. Let us show that the current coordinate of (4) can be found by using its known index *i*. To reach this aim the following nominations should be introduced:

$$x(\tau) = X(\tau) / R;\ \ y(\tau) = 1 - Y(\tau) / R;$$
$$t(\tau) = (x(\tau) + y(\tau)) / 2; \qquad \textbf{(6)}$$

It is necessary to note that t=i/2R is varying linear with i increasing. Taking that ($m \neq n$) let us deduce parametric representation of $x(t),y(t)$ by quadratic form (4) and Equ (6)

$$x(t) = t - \sigma + sign(\sigma)\sqrt{\sigma(\sigma + 2t(1-t))}; \qquad \textbf{(7)}$$
$$y(t) = t + \sigma - sign(\sigma)\sqrt{\sigma(\sigma + 2t(1-t))}, \qquad \textbf{(8)}$$

where $Z = m/n;\ \sigma = ½(1+Z) / (1-Z);\ \ x(t), y(t), t \in [0,1]$.

We have shown that the **CI** has one input and two outputs. When the **CI** input varies from 0 to 1 signal t, the output signals $x(t), y(t)$ vary from 0 to 1. Each iteration the **CI** generates increments ξ_x, ξ_y and so we can deduce that

$$X_{i+1} = X_i + \xi_x^{[i]};\ \ Y_{i+1} = Y_i - \xi_y^{[i]};\quad \xi_x^{[i]}, \xi_y^{[i]} \in \{0,1\}. \qquad \textbf{(9)}$$

Equ (2) can be represented in another form

$$d\mathbf{A}_i = \mathbf{A}_{01}\,\xi_x^{[i]}/2R + \mathbf{A}_{12}\,\xi_y^{[i]}/2R, \qquad \textbf{(10)}$$

where $d\mathbf{A}_i$ - increment which current coordinate of curve (2) gets after execution of *i*-th iteration of the **CI** algorithm.

Based on (9) we can conclude that for arc rendering (2) it is necessary to perform the parallel generation of two vectors $\mathbf{A}_{01}$ and $\mathbf{A}_{12}$. Every current coordinate of the vector must get a fixed increment after the receiving corresponding signals from **CI** output $\xi_x^{[i]}$, $\xi_y^{[i]}$. Each **LI** will only execute one iteration if its inputs get nonzero signals.

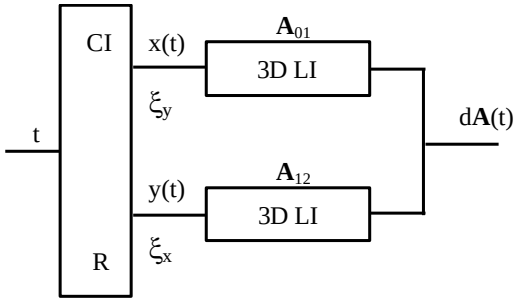

Figure 3 : Functional scheme of conic arc generator.

The functional scheme of the conic arc generator realized by described above algorithm is presented in Fig.3.

As we can see the conic arc generator can be designed as a two-level computing unit, the upper level of which consists of a control (host) **CI**, and lower level (slave) is made up of two multidimensional vector generators (*m***D LI**), which execute generation of vectors $\mathbf{A}_{01}$ and $\mathbf{A}_{12}$. It is necessary to note that **CI** and **LI**s can run in parallel if they are implemented in hardware. **CI** just kicks off **LI**s iterations and their dynamic cycles are not interacting. From previous reasoning we can conclude that the rate of such an arc generator depends only from the **CI** rate. It is equivalent to Bresenham's circle generator [3] by number of addition operations.

Increments at the both *m***D LI** outputs should be summed. Each of the rasterizing vectors $\mathbf{A}_{01}$ and $\mathbf{A}_{12}$ are generated with errors less than 0.5, and after its current coordinates are summed, the sampling error may double. To diminish the error to its initial level it is necessary to execute arithmetic right shift of the current pixel coordinate. Therefore it is necessary to double the value of R, which has been used while start initialization of conic and linear interpolators and was found as $R = max\ (\|\mathbf{A}_1-\mathbf{A}_0\|,\|\mathbf{A}_2-\mathbf{A}_1\|)$, where $\|\mathbf{A}_i-\mathbf{A}_j\| = max\ (\ |X_i-X_j|, |Y_i-Y_j|, |Z_i-Z_j|, \dots\ ),\ i,j \in \{\ 0, 1, 2\ \}$.

So we have shown that the hardware realized conic arc generator would have twice slower rate versus circle generator [3]. The scheme shown in Fig.3 may be analytically described as

$$\mathbf{A}(t) = \mathbf{A}_0 + \mathbf{A}_{01}\ x(t) + \mathbf{A}_{12}\ y(t). \qquad \textbf{(11)}$$

2.3. Recursive computing structure and class of L-curves.

Let us prove that computing structures similar to conic arc generator, Fig.4, are able to rasterize more complex curves incurved in polygon. It may be considered that after normalizing the parameters of interpolators their inputs receive and their outputs send signals, which vary from 0 to 1. From this follows that scheme shown in Fig.4, analytically is defined as

$$\mathbf{A}(t) = \mathbf{A}_0 + \mathbf{A}_{01}\ \Psi_1(t) + \mathbf{A}_{12}\ \Psi_2(t) + \mathbf{A}_{23}\ \Psi_3(t) + \mathbf{A}_{34}\ \Psi_4(t), \qquad \textbf{(12)}$$

where $\Psi_1(t) = x_2\,(\,x_1(t)\,);\quad \Psi_2(t) = y_2\,(\,x_1(t)\,);$

$\Psi_3(t) = x_3\,(\,y_1(t)\,);\quad \Psi_4(t) = y_3\,(\,y_1(t)\,);$

$\Psi_1(t),\ \Psi_2(t),\ \Psi_3(t),\ \Psi_4(t),\ t \in [0,1].$

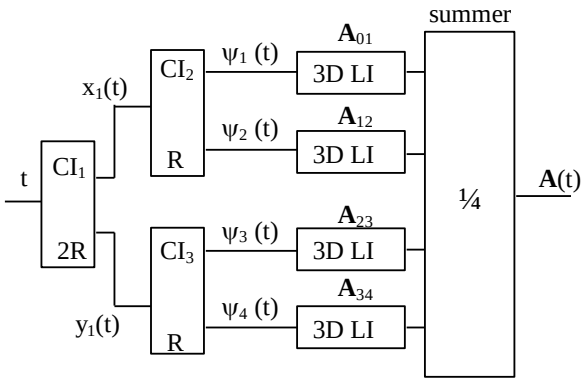

Figure 4 : Functional scheme of L-curve generator.

These curves have no established denomination or names and in [3,4] they are called as L-curves (from "lecalo" -- curved mask tool used for drawing works).

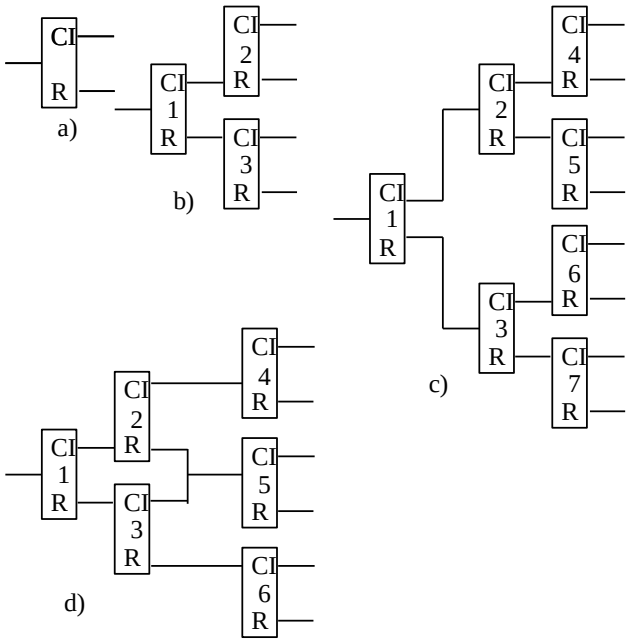

Figure 5: Examples of control (host) schemes of serial version of L-curve generator.

So we propose in this paper a method of unified designs and computing for curve rasterization. The proposed approach consists of following procedures: after computing structure scheme consisting of linear and conic interpolators being composed, the parametric function corresponding to it must be analyzed. This approach differs from the traditional ones based on designing of analytically defined curves.

Let us estimate the rate of the scheme shown in Fig.4. If the considered generator is realized in hardware **CI**s and **LI**s will work in parallel. Four iterations must be executed to find the location of one pixel (4*R* iterations on whole curve generation). Each vector generator executes one iteration and the rasterized curve gets an increment with an error up to 2 discrete raster units. To diminish this error to 0.5 it is necessary to shift right on two bits the pixel coordinates after addition of **Li**s coordinates.

So it follows that it is necessary to increase R which was defined as $R = max(\|\mathbf{A}_1 - \mathbf{A}_0\|,\|\mathbf{A}_2 - \mathbf{A}_1\|,\|\mathbf{A}_3 - \mathbf{A}_2\|,\|\mathbf{A}_3 - \mathbf{A}_2\|)$ in factor 4. Thus, to rasterize the curve (12) it is necessary to execute 16*R* iterations. So the pixel rate of the scheme in Fig.4 will be 8 times less than the rate of a circle generator [3]. It should be noted that only two **CI**s and one **LI** are initialized for each iteration of the generator. So it is evident that with complexity of L-curve the generator scheme is increasing and the number of interpolators being in passive state is increasing too. This is a disadvantage of the proposed approach.

The hardware cost of the proposed generators may be defined after summing the costs of all interpolators. The number of interpolators (**LI**,**CI)** is known (for each version of generators Fig.5, they are shown on the scheme). For example, the 3**D** conic arc generator (Fig.3) consists of 1 **CI** and 6 **LI**s. The 3**D** L-curve generator (Fig.4) consists of 3 **CI**s and 12 **LI**s, and so further on the analogy, Fig.5. a,b,c. The regular recursive structure of the considered schemes consisting of linear and conic interpolators is feasible for the hardware implementation.

The form of an L-curve is defined by the location of polygon vertices and by three eccentricities (or by three alpha-factors), which define working modes of **CI**s. The illustration of eccentricities influence on curve form is shown in Fig.6.

It is evident from this example that if the curve is hyperbolic ($\alpha \to 1.0$) it takes side with open polygon, and on the contrary, if curve is elliptic ($\alpha \to 0.0$), it takes sideways of open polygon.

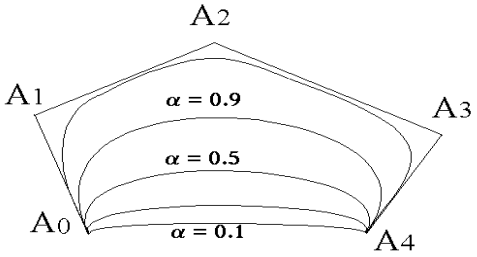

Figure 6: Illustration of eccentricies ($\alpha = \alpha_1 = ... = \alpha_9 = 0.1,...,0.9$)

L-curves may be used in geometric modeling in the same way as Bezier curves are used. It should be noted that L-curves admit more free-form flexibility than Bezier curves. Proposing a new class of curves (L-curves) is an essential achievement of this paper, but the universal usage of L-curves in graphical modeling is still matter of the future.

3. APPLICATION OF THE L-CURVE GENERATOR TO COMMON USED CURVES RASTERIZATION.

3.1. L-curve generator application to conic spline rasterization.

It is known that conic spline is a special case of rational B-spline (of degree 2), and its segment may be represented as follows

$$\boldsymbol{C}_i(t) = \frac{\sum_{k=-1}^{k=1} h_{i+k}\, B_k(t)\, \mathbf{V}_{i+k}}{\sum_{k=-1}^{k=1} h_{i+k}\, B_k(t)}, \qquad \textbf{(13)}$$

where $< \mathbf{V}_i >$ - vertices of spline;

$< h_k >$ - array of node factors;

$B_k(t)$ - basic function of B-spline;

$$B_{-1}(t) = \tfrac{1}{2}(1+t)^2; \quad B_0(t) = \tfrac{1}{2}(-2t^2 + 2t + 1); \; B_1(t) = \tfrac{1}{2} t^2. \qquad \textbf{(14)}$$

To simplify further analysis let us represent the segment (13) in the form of a rational Bezier curve

$$\boldsymbol{C}(t) = \frac{\alpha_0 \mathbf{K}_0 (1-t)^2 + 2\alpha_1 \mathbf{K}_1 (1-t)t + \alpha_2 \mathbf{K}_2 t^2}{\alpha_0 (1-t)^2 + 2\alpha_1 (1-t)t + \alpha_2 t^2}, \qquad \textbf{(15)}$$

where

$$\mathbf{K}_0 = \frac{\mathbf{V}_{i-1} h_{i-1} + \mathbf{V}_i h_i}{h_i + h_{i-1}}; \quad \textbf{(16)} \qquad \mathbf{K}_1 = \mathbf{V}_i; \quad \textbf{(17)} \qquad \mathbf{K}_2 = \frac{\mathbf{V}_{i+1} h_{i+1} + \mathbf{V}_i h_i}{h_i + h_{i+1}}; \quad \textbf{(18)}$$

$$\alpha_0 = \frac{h_i + h_{i-1}}{h_{i-1} + 6h_i + h_{i+1}}; \quad \textbf{(19)} \qquad \alpha_1 = \frac{2h_i}{h_{i-1} + 6h_i + h_{i+1}}; \quad \textbf{(20)} \qquad \alpha_2 = \frac{h_i + h_{i+1}}{h_{i-1} + 6h_i + h_{i+1}} \qquad \textbf{(21)}$$

To use conic arc generator, the conic segment (21) should be represented in the form

$$\boldsymbol{C}(t) = \mathbf{K}_0 + \mathbf{K}_{01}\, x(t) + \mathbf{K}_{12}\, y(t), \text{ where } \mathbf{K}_{01} = \mathbf{K}_1 - \mathbf{K}_0; \quad \mathbf{K}_{12} = \mathbf{K}_2 - \mathbf{K}_1; \qquad \textbf{(22)}$$

$$x(t) = \frac{2\alpha_1 + (\alpha_2^2 - \alpha_0\,\alpha_1)\,t^2}{\alpha_0(1-t)^2 + 2\alpha_1(1-t)t + \alpha_2 t^2}; \quad \textbf{(23)} \qquad y(t) = \frac{\alpha_2\,t^2}{\alpha_0(1-t)^2 + 2\alpha_1(1-t)t + \alpha_2\,t^2}; \quad \textbf{(24)}$$

It should be noted that in spite of equivalence of equ.(22) and (11), parameters t in those formulas are not the same (compare (7), (8) and (23), (24)). Nevertheless the nonparametric forms of equ (23), (24) and (7), (8) must be equivalent. Let's find t(x) {from (23)} and t(y) {from (24)} and taking into consideration t(x)≡t(y) find the nonparametric representation of the curve x(t), y(t) :

$$x^2 + y^2 + 2xy\,\frac{2\alpha_1^2 - \alpha_0\alpha_2}{\alpha_0\alpha_2} + \frac{4\alpha_1^2}{\alpha_0\alpha_2}\,y = 0\,. \quad \textbf{(25)}$$

After comparing (25) and (4) and taking into account (6) and (5), it is evident that equ. (25) and (4) define the same conic arc. Eccentricity of this arc may be found as follows:

$$\frac{m}{n} = \frac{2\alpha_1^2 - \alpha_0\alpha_2}{\alpha_0\alpha_2}\,. \quad \textbf{(26)}$$

We have found a method for representation of conic spline segment defined by control parameters { $\mathbf{V}_{i-1}$, $\mathbf{V}_i$, $\mathbf{V}_{i+1}$, h_{i-1}, h_i, h_{i+1} } in form of conic arc with control parameters { K_0, K_1, K_2, m/n }. So the conic segment may be rasterized by the conic arc generator proposed in this paper.

3.2. The method of Bezier curves rasterization.

First it should be noted that Bezier polynomials of degree 2 are parabolas and can be generated by the conic arc generator. **CI** eccentricity must be set as $m/n = 1$ (α=1/2). Output functions of the **CI** have a simple notation:

$$x(t) = 2t - t^2; \quad y(t) = t^2; \quad x(t), y(t), t \in [0,1]. \quad \textbf{(27)}$$

From (27) follows if all **CI**s on scheme (Fig.4) are parabolically initialized ($\alpha_0 = \alpha_1 = \alpha_2 = 1/2$) the L-curve (12) is a polynomial of degree 4 (this is true only for the scheme shown on Fig.4). Taking into consideration that the Bezier polynomial may be also represented as a power polynomial of degree 4, we have got equality

$$P(t) = \sum_{k=0}^{4} (-1)^k\,C_4^k\,\mathbf{P}_k\,(1-t)^{4-k}\,t^k\,\} = \sum_{k=0}^{4} (-1)^k\,C_4^k\,t^k \sum_{i=0}^{k} (-1)^k\,C_k^i\,\mathbf{P}_i\,, \quad \textbf{(28)}$$

where $\mathbf{P}_i$ - the control point of Bezier curve $i \in \{0..m\}$, m=4 - degree of considered polynomial. Let us equate the coefficients of polynomials (28) and (12). We have got a linear equation system. After solving the system we would be able to find a correspondence between control points of the L-curve < $\mathbf{A}_k$ > and the Bezier curve < $\mathbf{P}_k$ >. Transformation of Bezier polynomial control points into L-curve control points may be executed with matrix transformation:

$$(\,\mathbf{A}_{01}, \mathbf{A}_{12}, \mathbf{A}_{23}, \mathbf{A}_{34}\,)^T = G_{43} * (\,\mathbf{P}_{01}, \mathbf{P}_{12}, \mathbf{P}_{23}\,)^T; \quad \textbf{(29)}$$

$$(\mathbf{A}_{01}, \mathbf{A}_{12}, \mathbf{A}_{23}, \mathbf{A}_{34})^{T} = G_{44} * (\mathbf{P}_{01}, \mathbf{P}_{12}, \mathbf{P}_{23}, \mathbf{P}_{34})^{T}, \quad \textbf{(30)}$$

where $\mathbf{A}_{ij} = \mathbf{A}_j - \mathbf{A}_i$ - side of L-curve control polygon, $\mathbf{P}_{ij} = \mathbf{P}_j - \mathbf{P}_i$ -- side of Bezier curve control polygon, " ^{T} " - operation of vector transposing, "*" - operation of multiplication of matrix and vector, G_{43} - matrix, corresponding to Bezier polynomial of degree 3 (cubic polynomial), G_{44} - matrix, corresponding to Bezier polynomial of degree 4.

The matrices for transformation of Bezier curves of degree 3 and 4, which may be rasterized with scheme on Fig.4 are following:

$$G_{43} = \frac{1}{4}\begin{pmatrix} 3 & 0 & 0 \\ 2 & 2 & -1 \\ -1 & 2 & 2 \\ 0 & 0 & 3 \end{pmatrix} \qquad G_{44} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & -1 & 0 \\ 0 & -1 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Matrices corresponding to Bezier polynomials up to the 8th degree are given in [4]. It is natural that curves from 5th to 8th degree should be generated with the scheme of the next level (7 **CI**s 8*m*D **LI**s). Only the schemes, which have a recursive binary tree structure, are considered in this paper, Fig.5. a,b,c.

From equ. (29), (30) it follows that the polygons defining the equivalent L-curve ($\mathbf{A}_0$, $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{A}_3$, $\mathbf{A}_4$) and Bezier curve ($\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$) are different. It has been shown above that the rate of L-curve generator is defined by the structure of generator scheme and by the value of R, which is found from the known set of open polygon sides. Let us find R_a from the set of points $< \mathbf{A}_k >$ and R_p from set $< \mathbf{P}_k >$, where notation **P** and **A** means the control points of polynomials, represented in Bezier form and L-form correspondingly. After ratio $r = R_a / R_p$ being found, it is possible to estimate a degeneration of Bezier curve generation rate with its degree *m* growth. Let $< \mathbf{p}_k >$ be control vectors of Bezier curve,

$$\mathbf{a}_k = \sum_{i=0}^{m} G_{ki}\, \mathbf{p}_i \text{ -}$$

control vectors of equivalent L-curve, G_{ki} - element of matrix G. Taking into consideration L-curve and Bezier curve are equivalent, let us get transformation, which allow us to find a set of L-curve control vectors a_k from a known set of Bezier curve control vectors

$$\mathbf{a}_k = G\, \boldsymbol{p}_k, \quad \textbf{(31)}$$

where $\mathbf{a}_k$, p_k - vectors, G - matrix. From (17) follows an estimation

$$r = (R_a / R_p) \le \operatorname{MAX}_k \sum_i | G_{ki} |$$

Matrices G are calculated according to the method considered above. On the base of those matrices table 1 is compiled.

Table 1.

m	2	3	4	5	6	7	8
r	1	1.25	3	3.6	18.6	49	447

It is evident from table 1 that the generation of more than 5 degree Bezier curves is not efficient. It should be noted that the rate degeneration has not influenced a stability of generator. Bezier curves would be rasterized by the proposed algorithm quickly or slowly.

4. APPLICATION OF L-CURVE GENERATOR FOR BEZIER PATCHS SHADING.

4.1. Shading of rectangular Bezier patch.

Analytically Bezier patch may be defined with formula

$$\mathbf{P}_{m,n}(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} \mathbf{P}_{i,j}\, C_n^{j}\, C_m^{k}\, (1-v)^{n-j}\, v^{j}\, (1-u)^{m-i}\, u^{i}, \quad \textbf{(43)}$$

where - control vertex coordinates $\{X_{ij}, Y_{ij}, Z_{ij},...\}$ of patch in 6**D** space (*X, Y, Z, R, G, B*) $u,v \in [0,1]$. Let us represent (1) as

$$\mathbf{P}(u,v) = \sum_{i=0}^{m} \mathbf{A}_i(v)^i \, C_m^i \, (1-u)^{m-i} \, u^i, \qquad \textbf{(44)}$$

where

$$\mathbf{A}_i(v) = \sum_{j=0}^{n} \mathbf{P}_{i,j} \, C_n^j \, (1-u)^{n-j} \, u^j, \quad i \in \{0..m\}. \qquad \textbf{(45)}$$

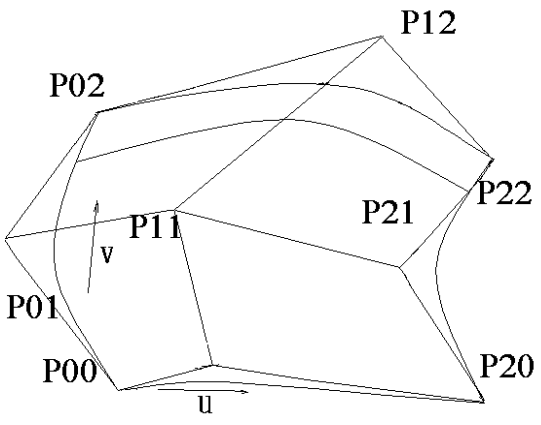

Figure 7 : Geometrical definition of rectangular Bezier patch.

For each value of the parameter $v \in [0,1]$ formula (45) defines m+1 control points of curves family (44) location. For case m=2 this statement is illustrated in Fig.6. From equivalence (43) and (44) follows that the Bezier patch shading problem may be represented as a task of Bezier curve family (44) rasterization.

In paper [5] it has been shown that the Bezier curve rasterization may be executed by hardware L-curve generator. Formula (45) defines m+1 Bezier curves, current points of which are the control points of curve (44). For calculation of those points the L-curve rasterization algorithm may be also used. Taking into consideration that the control points of curve (45) are calculated much more seldom than the points (pixels) of rasterized curve (44) (hundreds and thousands times more seldom for some patch sizes), it is evident that for the calculation of curve (44) control points it is more convenient to use a curve generator realized in software. On the contrary, curves (44) family rasterization are desirable to execute with L-curve generator realized in hardware. The procedure of patch (43) shading may be represented with the following pseudocode:

```
for (i=0; i<=m; i++)InitSoftBezier(P,i,Rv);
    for (j=0; j<Rv; j++) {
      for (i=0; i<=m; i++) {
        A[i]=SoftBezier(P,m,n,Rv,i,j);
      }
      InitHardBezier(A,m,Rv);
    }
```

In this procedure the following notations have been used: *m,n* - define quantity of Bezier patch control points polynomial degree. InitSoftBezier - procedure that makes the initial setup of software-realized algorithm for generation of m+1 curves (45), and calculates the quantity of algorithm iterations Rv necessary for each curve (45) generation. SoftBezier - procedure that executes one iteration for each of m+1 curves (45). This procedure is a software realized L-curve generator, working in Bezier curves generation mode. InitHardBezier - procedure that initialize hardware realized L-curve generator and initiates Bezier curve (44) rasterization.

SUMMARY.

An approach on the base of recursive computer schemes analysis to design curve rasterization algorithms has been described in this paper. Proposed schemes consist of conic and linear interpolators and allow to synthesize algorithm versions for sets of common curves used in graphic design. Rasterized curves are represented with a set of adjoined pixels. The proposed algorithm schemes are feasible for hardware (on chip) realization and allow to

generate non-elementary parametric curves with high velocity on raster units. The rate of L-curve rasterization is comparable with rate of Bresenham's algorithm for circles.

The class of L-curves proposed in this paper includes Bezier curves also. One more possible application of L-curve generator is the rasterization of B-splines, conic splines and Bezier patches.

The proposed method for surface nonlinear shading based on the L-curve generator implementation, has computation costs comparable with Gouraud's shading method and may be implemented in hardware. This method also may be implemented to increase the quality of Phong's shading method by means of nonlinear normal interpolation in 6**D** space (X , Y , Z , n_x , n_y , n_z).

REFERENCES.

[1] Bresenham J.E. Algorithm for Computer Control of Digital Plotter. IBM System Journal. - 1965. - V. 4(1).- P.25-30.

[2] Pitteway M. Algorithm for drawing ellipses or hyperbolae with a digital plotter. Computer Journal. - 1966. - V. 10(3). - 282-289.

[3] Bresenham J.E. A Linear Algorithm for Incremental Digital Display of Circular Arcs. Communication of the ACM. - 1977. - V. 20(2). - P. 100-106.

[4] Chirikov S.V. Algorithms and structures for L-curves rasterization on raster units. Tech.D. Diss. thezises. 17 march 1992. (Sankt-Peterburg, Russia).

[5] Chirikov S.V., Paltashev T.T. An Unifying Approach in Hardware-Oriented Rasterization of Curves and Shaded Surfaces. In Proc: 9th Eurographics Workshop on Graphics Hardware. SINTEF, Oslo, Norway, September 12-13, 1994, pp.124-138.