◊ .1

Adaptive Sampling of Parametric Curves

Luiz Henrique de Figueiredo
IMPA, Instituto de Matematica Pura e Aplicada
Rio de Janeiro, Brasil
`lhf@visgraf.impa.br`

◊ Introduction ◊

Approximating a parametric curve by a polygonal curve is a practical undertaking, involving a sampling of the parameter domain. A first-principles *uniform* sampling strategy remains the most popular. Unfortunately, it can prove very inefficient if high precision is required.

This gem presents an *adaptive* method for sampling the domain with respect to local curvature. Samples concentration is in proportion to this curvature, resulting in a more efficient approximation—in the limit, a flat curve is approximated by merely two endpoints. Applications of this sampling strategy, including rasterization and arc length parametrization, are also discussed.

◊ Uniform Sampling ◊

Let $\gamma\colon [0,1] \to R^d$ describe a curve lying in d-dimensional space (typically, $d \leq 3$). To approximate γ, choose n equally spaced sample points $0 = t_1 < t_1 < \ldots < t_n = 1$, which define the vertices $v_0, v_1, \ldots, v_n$, where $v_i = \gamma(t_i)$. The challenge is to choose sample points $t_1 \ldots t_n$ which induce a good approximation while keeping n small. Methods of uniform sampling most often choose n by trial and error (trading accuracy for efficiency), though automated heuristic tests are available (Lindgren et al. 1992).

◊ Adaptive Sampling ◊

Ideally, a non-uniform parametric sampling gives rise to a "uniform" vertex precision, leading to increased sampling density in regions of high curvature. The general method described here is based upon the following strategy:

ISBN 0-12-XXXXXX-X

1. choose a criterion for refining samples.
2. evaluate the criterion on the interval.
3. if the curve is almost flat in the interval, then the sample is given by its two extremes;
4. otherwise, divide the interval into two parts and recursively sample the two parts.

This strategy is similar to using the de Casteljau algorithm for Bézier curves, stopping when the control polygon is almost flat. There are a number heuristic refinement criteria applicable to general curves.

Refinement Criteria

The refinement criteria employed here, termed *probing*, chooses an intermediate point m within the interval of consideration $[a, b]$. Next, it tests the three candidate vertices for (approximate) collinearity, that is $\overline{v_a v_m} \parallel \overline{v_m v_b}$ is evaluated with $v_a = \gamma(a)$, $v_m = \gamma(m)$ and $v_b = \gamma(b)$. This flatness (parallel vector) test may take a number of forms:

- the area of the triangle $v_a v_m v_b$ is small;
- the angle $\angle v_a v_m v_a$ is close to $180°$.
- v_m lies near the chord $\overline{v_a v_b}$;
- $|v_a - v_m| + |v_m - v_b|$ is approximately equal to $|v_a - v_b|$;
- the curve's tangents at $\gamma(a)$, $\gamma(m)$ and $\gamma(b)$ are approximately parallel.

(The last is of value when a closed-form expression for γ's derivative is available.)

Although not equivalent in theory, empirical practice shows that these tests are equally effective in locating regions of low curvature. The area criterion is the algorithms' method of choice as it requires no square roots.

Choosing the Interior Point

The intuitive choice for the interior point m is the interval's midpoint, $m = \frac{1}{2}(a + b)$. A subtle form of sampling error, or *aliasing*, arises: the probing strategy considers a curve "flat" throughout the interval $[a \leq t \leq b]$ when a flatness test is satisfied. In fact, the heuristic fails should the curve undulate along the interval under consideration. The sinusoid curve $[t, \sin(t)]$ with t sampled at $\pm i\pi$ is a ready example. Here, every vertex v_i lies along the x-axis. In general, any deterministic probing is vulnerable to aliasing.

Random probing along the interval avoids aliasing sampling due to such symmetries in γ. Since sampling near the interval's midpoint is desirable, a uniform distribution may be biased toward this interior. In this implementation, a uniform distribution on the restricted interval $[0.45, 0.55]$ is employed. Other variations may substitute Gaussian distributions (easily found by summing random variables, by virtue of the central limit theorem) or use multiple probes. Vertices having greatest deviation may then be chosen preferentially or otherwise used in a recursive evaluation of higher order (Chandler 1990). In practice, the binary recursion scheme presented below provides adequate results.

The Algorithm

For the sake of efficiency, arrange the sampling so that the interior vertex used for the flatness test is also the point of subdivision, should recursion take place. This assures that γ is evaluated exactly once at each sample point. The pseudocode for the algorithm is

```
sample(a, b, va, vb):
    m ← random point in [a, b]
    vm ← γ(m)
    if flat(va, vm, vb)
        line(va, vb)
    else
        sample(a, m, va, vm)
        sample(m, b, vm, vb)
```

Invoking the function `sample`$(0, 1, \gamma(0), \gamma(1))$ initiates the procedure on the complete domain $0 \leq t \leq 1$. The function `flat` implements one of the refinement criteria suggested above; `line` is the basic line-drawing operation.

Note that this algorithm generates points in the exact order they occur along the curve, as the recursive implementation performs a depth-first search of the interval "tree". With proper adaptation, the algorithm performs a direct rasterization of the curve, as when `line(va,vb)` describes coincident pixels or the test `flat(va,vm,vb)` evaluates adjacent pixels. Here, (discrete) Γ substitutes for its continuous counterpart γ. The pseudocode appears below.

```
raster(a, b, va, vb):
    if neighbors(va, vb)
        plot(va)
    else
        m ← random point in [u, v]
        vm ← Γ(m)
        raster(a, m, va, vm)
        raster(m, b, vm, vb)
```

Invoking the function `raster`$(0, 1, \Gamma(0), \Gamma(1))$ initiates the procedure on the complete domain $0 \le t \le 1$, though the final pixel $\Gamma(1)$ must be plotted explicitly. Note that this implementation requires neither a tolerance test nor a `line` function; recursion depth is determined by the display resolution.

◊ Applications ◊

Adaptive sampling strategies may be adopted to solve problems in related fields of numerical integration, quadrature of explicit functions, line integration or arc length parametrization. The pseudocode for computing the length of a curve is:

```
length(a, b, va, vb):
    m ← random point in [u, v]
    vm ← γ(m)
    if flat(va, vm, vb)
        return |vb − va|
    else
        return length(a, m, va, vm) + length(m, b, vm, vb)
```

Functions may also be evaluated based upon methods of binary descent. The arc length parametrization of a curve locates the parameter value t having the corresponding vertex v_t at a desired length along a curve. This calculation is of value in digital animation (Guenter and Parent 1990).

Finally, polygonal lines digitized either manually or automatically (e.g., using an image edge detection algorithm) usually have a large number of vertices. Here, the refining criterion may be used to thin (remove) any redundant vertex v_m collinear with bracketing vertices $\overline{v_a v_b}$. The size of the bracketing interval can once again be computed in adaptive fashion using binary recursion. In this guise, an adaptive sample "running in reverse" rederives a method commonly employed in digital cartography (Visvalingam and Whyatt 1990).

◊ Implementation ◊

An implementation of the adaptive sampling method is now presented. Created for use with three-dimensional curves, the code is easily modified to arbitrary dimension, or reworked to provide direct rasterization as in the second pseudocode example.

The data structure `Point` is a `struct` containing the parameter value t and the coordinates x, y, z of $\gamma(t)$. It is used to ensure that γ is never evaluated more than once at any point. Macros providing efficient access to this structure are also provided; these macros also increase program clarity.

The code's core is the recursive sampling function `sample`, which calls two user-defined functions: `gamma`, which computes the coordinates of the point on the curve corresponding to a parameter value, and `line`, which performs the line segment rendering. The function `flat` implements a refinement criterion based on triangle area, using the cross product (the user-supplied tolerance `tol` is used within this code).

The code has been written for simplicity and presentation. A production version passes the functions `gamma`, `line` and `tol` as parameters to the top-level entry point `aspc`, allowing generic operation. In all cases, invoking `aspc`(a, b) begins the adaptive sampling of the curve along the interval $[a, b]$.

```
/* aspc.c -- generic adaptive sampling of parametric curves */

typedef struct point { double t,x,y,z; } Point;

#define T(p)    ((p)->t)
#define X(p)    ((p)->x)
#define Y(p)    ((p)->y)
#define Z(p)    ((p)->z)

extern void gamma(Point* p);            /* user supplied */
extern void line(Point* p, Point* q);   /* user supplied */

static void sample(Point* p, Point* q)
{
 Point rr, *r=&rr;
 double t = 0.45 + 0.1 * (rand()/(double) RAND_MAX);
 T(r) = T(p) + t*(T(q)-T(p));
 gamma(r);
 if (flat(p,q,r)) line(p,q); else { sample(p,r); sample(r,q); }
}

static int flat(Point* p, Point* q, Point* r)
{
 extern double tol;                     /* user supplied */
 double xp = X(p)-X(r); double yp = Y(p)-Y(r); double zp = Z(p)-Z(r);
 double xq = X(q)-X(r); double yq = Y(q)-Y(r); double zq = Z(q)-Z(r);
 double x =  yp*zq-yq*zp;
 double y =  xp*zq-xq*zp;
 double z =  xp*yq-xq*yp;
 return (x*x+y*y+z*z) < tol;            /* |pr x qr|^2 < tol */
}

void aspc(double a, double b)           /* entry point */
{
 Point pp, *p = &pp;
 Point qq, *q = &qq;
 srand(time(0));                        /* randomize */
 T(p)= a; gamma(p); T(q)=b; gamma(q);   /* set up */
 sample(p,q);                           /* sample */
}
```

◊ Bibliography ◊

(Chandler 1990) R. E. Chandler. A recursive technique for rendering parametric curves. *Computers and Graphics*, 14(3/4):477–479, 1990.

(Guenter and Parent 1990) B. Guenter and R. Parent. Computing the arc length of parametric curves. *IEEE Computer Graphics and Applications*, 10(3):72–78, May 1990.

(Lindgren et al. 1992) T. Lindgren, J. Sanchez, and J. Hall. Curve tesselation criteria through sampling. In D. Kirk, editor, *Graphics Gems III*, pages 262–265. Academic Press, Boston, 1992.

(Visvalingam and Whyatt 1990) M. Visvalingam and J. D. Whyatt. The Douglas-Peucker algorithm for line simplification: Re-evaluation through visualization. *Computer Graphics Forum*, 9(3):213–228, September 1990.