Arc-Length Parameterized Spline Curves for Real-Time Simulation

Hongling Wang, Joseph Kearney, and Kendall Atkinson

Abstract. Parametric curves are frequently used in computer animation and virtual environments to control the movements of synthetic objects. In many of these applications it is essential to efficiently relate parameter values to the arc length of the curve. Current approaches to compute arc length or to construct an arc-length parameterized curve are impractical to use in real-time applications. This paper presents a simple and efficient technique to generate approximately arc-length parameterized spline curves that closely match spline curves typically used to model roads in high-fidelity driving simulators.

§1. Introduction

Parametric cubic splines are the curves of choice for many applications of computer graphics. They are widely used in computer animation and virtual environments to define motion paths [6]. As the parameter variable ranges over the interval of definition, the computed position traces a smooth curve in space. Naively, one might compute position using the parameter variable directly. However, it is difficult to regulate the speed of traversal in this way because the parameter variable and curve length are not, in general, linearly related. (For example, see the left graph in Figure 1.) Motion control is simple if object trajectories are parameterized by arc length. To move an object at a constant speed along a path of an arc-length parameterized curve, the controller need only evaluate the parametric function at parameter values separated by the speed times the inter-frame time interval.

Let us assume we have a parametric representation of a cubic spline curve

$$Q(t) = (x(t), y(t), z(t)),$$

Curve and Surface Design: Saint-Malo 2002
Tom Lyche, Marie-Laurence Mazure, and Larry L. Schumaker (eds.), pp. 387–396.

ISBN 0-9728482-0-7.

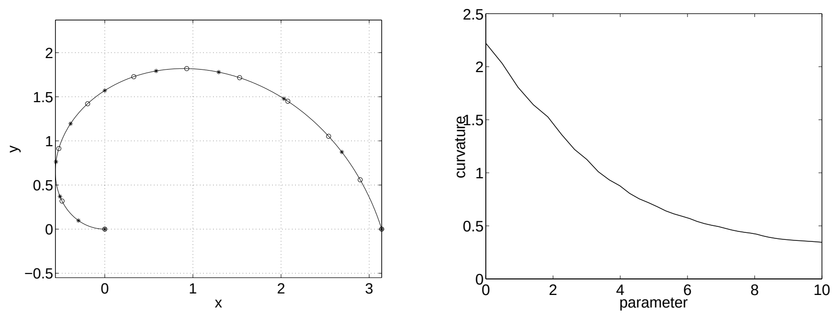

Fig. 1. A spiral curve (left) with the points at constant parameter interval ('*') and constant arc-length interval ('o') and its curvature (right).

where t is from t_0 to t_n, n is the number of spline segments, and $\{t_0, t_1, t_2, \cdots, t_n\}$ are the break points. The arc-length parameterization of a curve can be constructed from any other differentiable parameterization by the following two-step process [2]:

- Compute arc length s as a function of parameter t: $s = A(t)$. Since s is a strictly increasing function of t, there is a one-to-one correspondence between s and t.
- Compute $t = A^{-1}(s)$, the inverse of the arc length function. This function is well defined and monotonically increasing for cubic splines. By substituting $t = A^{-1}(s)$ into $Q(t)$, we get a curve parameterized by arc length s, $P(s) = (x(A^{-1}(s)), y(A^{-1}(s)), z(A^{-1}(s)))$, where $s \in [0, L]$ and L is the total length of the curve.

The arc length is a geometric integration,

$$A(t) = \int_{t_0}^{t} \left((x'(t))^2 + (y'(t))^2 + (z'(t))^2\right)^{1/2} dt, \qquad (1)$$

where for a cubic spline,

$$\begin{cases} x(t) = a_{x,i}(t-t_i)^3 + b_{x,i}(t-t_i)^2 + c_{x,i}(t-t_i) + d_{x,i} \\ y(t) = a_{y,i}(t-t_i)^3 + b_{y,i}(t-t_i)^2 + c_{y,i}(t-t_i) + d_{y,i} \\ z(t) = a_{z,i}(t-t_i)^3 + b_{z,i}(t-t_i)^2 + c_{z,i}(t-t_i) + d_{z,i}, \end{cases}$$

where $t \in [t_i, t_{i+1}]$, $i = 0, 1, 2, ..., n-1$, the values for x, y ,and z are of class C^2 on $[0, L]$. In general, the integral (1) cannot be computed analytically. Therefore, the arc-length parameterization for cubic spline curves cannot be expressed as a combination of elementary functions and must be evaluated numerically.

§2. Previous Work

A number of researchers have developed numerical methods to compute approximate arc-length parameterizations of curves. The integral formula (1) to compute arc length, $A(t)$, can be approximated with conventional numerical integration methods such as Gaussian quadrature or Simpson's rule. In order to provide a means to control the accuracy of the approximation, Guenter et al. [3] introduced a multi-step method called adaptive Gaussian integration. The adaptive Gaussian integration method divides the interval of integration in half and separately integrates each sub-interval. The sum of the integrals computed on the two sub-intervals is compared with the value computed over the entire interval. If the difference between them is less than the desired accuracy, then the procedure returns the sum of the two halves; otherwise it recursively applies the procedure on the sub-intervals.

Cubic spline curves are typically composed of a sequence of cubic segments. To compute the arc-length $A(t)$ of a composite curve $Q(t)$, the arc lengths of the component cubic segments are first computed. A table of parameter t against accumulated arc length at the segment boundaries is then constructed from the segment lengths. To compute arc length, $A(t)$, the index of the table entry on which t lies is calculated. Then the arc length in a small interval inside the segment is computed. The sum of the accumulated arc length and the arc length on the small interval is computed as the final result.

The bisection method is commonly used to find $A^{-1}(s)$. The bisection method begins with a table search. Given an arc length value s_a, the table is searched to locate the starting parameter interval $[t_i, t_{i+1}]$ corresponding to the arc-length interval $[A(t_i), A(t_{i+1})]$ containing s_a. While the bisection method can achieve high accuracy, it converges too slowly for many real-time applications.

In Guenter et al. [3], the inverse arc-length computation is formulated as a Newton-Raphson root finding method. As in the bisection method, the first step is to search the table to find the interval $[t_i, t_{i+1}]$ such that $s_a \in [A(t_i), A(t_{i+1})]$. The parameter value at arc length s_a is computed by finding a root $t \in [t_i, t_{i+1}]$ of the equation $f(t) = s_a - A(t) = 0$. The Newton-Raphson method iteratively generates a sequence of estimates $\{r_k\}$, $k = 1, 2, ...$, where $r_k = r_{k-1} - \frac{f(r_{k-1})}{f'(r_{k-1})}$. On the first iteration, the value r_0 is computed by linearly interpolating t_i and t_{i+1}.

Several problems may arise with this approach. One complication is that r_k may lie outside the interval $[t_i, t_{i+1}]$. If r_k lies off interval, the computation must be terminated. Perhaps the most severe impediment for real-time applications is that the Newton-Raphson method sometimes converges very slowly or even diverges. Hard real-time applications demand that computations meet stringent scheduling requirements. Computations must be completed in a fixed time interval. While the Newton-Raphson method is normally very efficient, the unpredictable occurrences of rare failures can have disastrous consequences in real-time environments.

As an alternative to computationally intensive table look-up and integration techniques, Walter et al. [4] present an approximate closed-form solution to compute arc length directly from the parametric variable. Their method computes a Bézier curve that relates the length of the curve to the parametric variable. They assume that a one-span or a two-span Bézier curve $\overline{L}(t)$ has sufficient flexibility to accurately represent a large range of "arc length vs. parameter" curves. The great advantage of this approach is that $A(t)$ can be very quickly estimated from the cubic curve $\overline{L}(t)$ locally approximating $A(t)$. The method can also be applied in reverse to estimate t from s by approximating the curve $t = A^{-1}(s)$.

The approximation curve $\overline{L}(t)$ introduces sources of error that are difficult to analyze and control. The error inherent in the numerical integration required to estimate the control points of the Bézier curve is exaggerated by misfits between interpolation curve and the true arc-length curve. There are two additional disadvantages of this approach for our application. The first is that to accurately compute arc length for a composite curve, an approximation curve would have to be calculated for every segment of the composite curve. Since the arc lengths at the boundaries of the composite curve are unevenly spaced, a table search would be required to locate the segment on which t resides for the inverse mapping from s to t. The second disadvantage is that two separate sets of approximation curves are computed to serve the forward mapping from t to s and its inverse mapping. Our application requires frequent mapping in both directions. Both the complexity of maintaining separate forward and inverse mapping functions and the potential inconsistencies that could arise from accumulation of errors through alternately evaluating forward and inverse mapping functions are problematic.

The aim of our work is to efficiently model road geometry for real-time driving simulation. The road model is used to control motions of synthetic traffic. In addition to being used for motion guidance, the road model provides a coordinate system in which the relative positions of objects on the roadway are defined. Arc-length parameterized piecewise cubic splines are attractive for road modeling because of their tangent and curvature continuity. However, the methods presented above are too slow and insufficiently robust for real-time simulations.

§3. Computing an Approximate Arc-Length Parameterized Curve

Our method computes the approximation curve in three steps. First, the arc lengths of all the cubic segments in the input spline curve, $Q(t)$, are computed and summed to determine the arc length L of $Q(t)$. The second step is to find $m+1$ points equally spaced along $Q(t)$. The third step is to compute a new spline curve using the equally spaced points as knots. The result is an approximately arc-length parameterized piecewise spline curve divided into m cubic segments.

The arc length of each spline segment on the input curve is

$$l_i = \int_{t_i}^{t_{i+1}} \sqrt{(x'(t))^2 + (y'(t))^2 + (z'(t))^2} dt,$$

where i varies from 0 to $n-1$ and n is the number of spline segments in the original curve. Thus, the arc length of the whole curve is $L = \sum_{i=0}^{n-1} l_i$.

Using the bisection method, we compute $m+1$ equally spaced points on $Q(t)$ located at distances $0, \tilde{l}, 2 \cdot \tilde{l}, ..., m \cdot \tilde{l}$ from the start of the curve, where $\tilde{l} = L/m$ is the length of each segment in the output curve. These points are defined by the parameter values $\tilde{t}_0, \tilde{t}_1, ..., \tilde{t}_m$, which satisfy the following integration,

$$\int_{t_0}^{\tilde{t}_i} \frac{ds}{dt} dt = i \cdot \tilde{l}, \tag{2}$$

where $i = 0, 1, ..., m$, s is arc length, and t is the parameter of the spline functions.

The value of $\tilde{t}_i$ can be computed in two steps. The first step is to find a spline segment indexed by j which satisfies $\sum_{p=0}^{j-1} l_p \leq i \cdot \tilde{l} < \sum_{p=0}^{j} l_p$. This condition ensures that $t_j \leq \tilde{t}_i < t_{j+1}$. Formula (2) is written as,

$$\int_{t_0}^{\tilde{t}_i} \frac{ds}{dt} dt = \int_{t_0}^{t_j} \frac{ds}{dt} d_t + \int_{t_j}^{\tilde{t}_i} \frac{ds}{dt} dt = \sum_{p=0}^{j-1} l_p + \int_{t_j}^{\tilde{t}_i} \frac{ds}{dt} dt = i \cdot \tilde{l}.$$

The second step is to compute $\tilde{t}_i$ such that

$$\int_{t_j}^{\tilde{t}_i} \frac{ds}{dt} dt = i \cdot \tilde{l} - \sum_{p=0}^{j-1} l_p,$$

where $\tilde{t}_i$ is on the cubic spline segment starting with parameter value t_j. The second step is accomplished with the bisection method. We suppose $t_{left} = t_j$ and $t_{right} = t_{j+1}$. The interval $[t_{left}, t_{right}]$ contains the solution $\tilde{t}_i$. This interval is bisected into two subintervals $[t_{left}, t_{middle}]$ and $[t_{middle}, t_{right}]$, where $t_{middle} = (t_{left} + t_{right})/2$. We can calculate the arc length between $[t_j, t_{middle}]$ as $\triangle s = \int_{t_j}^{t_{middle}} \frac{ds}{dt} dt$. The solution lies in the upper subinterval $[t_{middle}, t_{right}]$ if $\triangle s < i \cdot \tilde{l} - \sum_{p=0}^{j-1} l_p$. Otherwise, the solution lies in the lower subinterval $[t_{left}, t_{middle}]$. This bisection process is repeated until a required error tolerance for arc length is achieved.

With the above bisection method, we get $\tilde{t}_0$,$\tilde{t}_1$,...,$\tilde{t}_m$ that divide the original curve into equal arc-length segments. Using the original cubic spline function, we then compute the evenly spaced points $(\tilde{x}_0, \tilde{y}_0, \tilde{z}_0)$, $(\tilde{x}_1, \tilde{y}_1, \tilde{z}_1)$, ..., $(\tilde{x}_m, \tilde{y}_m, \tilde{z}_m)$ at arc-lengths $s_0 = 0$, $s_1 = \tilde{l}$, $s_2 = 2 \cdot \tilde{l}$, ..., $s_m = m \cdot \tilde{l}$. We reparameterize the spline curve by interpolating $[(s_0, \tilde{x}_0)$, $(s_1, \tilde{x}_1)$, ..., $(s_m, \tilde{x}_m)]$, $[(s_0, \tilde{y}_0)$, $(s_1, \tilde{y}_1)$, ..., $(s_m, \tilde{y}_m)]$ and $[(s_0, \tilde{z}_0)$, $(s_1, \tilde{z}_1)$, ..., $(s_m, \tilde{z}_m)]$. In this interpolation, we interpolate x, y and z to arc length s and get the cubic spline functions in formula (3). Our goal is to have the following result for $s \in [0, L]$:

$$\sqrt{(\tilde{x}'(s))^2 + (\tilde{y}'(s))^2 + (\tilde{z}'(s))^2} = 1.0.$$

Therefore, the magnitude of the beginning tangent vector and the magnitude of the ending tangent vector should be 1.0. The new curve is

$$\begin{cases} \tilde{x}(s) = \tilde{a}_{x,i}(s - s_i)^3 + \tilde{b}_{x,i}(s - s_i)^2 + \tilde{c}_{x,i}(s - s_i) + \tilde{d}_{x,i} \\ \tilde{y}(s) = \tilde{a}_{y,i}(s - s_i)^3 + \tilde{b}_{y,i}(s - s_i)^2 + \tilde{c}_{y,i}(s - s_i) + \tilde{d}_{y,i} \\ \tilde{z}(s) = \tilde{a}_{z,i}(s - s_i)^3 + \tilde{b}_{z,i}(s - s_i)^2 + \tilde{c}_{z,i}(s - s_i) + \tilde{d}_{z,i}, \end{cases} \tag{3}$$

where $s \in [s_i, s_{i+1}]$, $i = 0, 1, 2, ..., m - 1$, and the values for $\tilde{x}$, $\tilde{y}$, and $\tilde{z}$ are of class C^2 on $[0, L]$. The tangent vectors of the derived curve at the beginning point and the ending point are set to be equal to the normalized tangent vectors of the original curve at the beginning point and the ending point, respectively.

This is an arc-length parameterized spline curve. It has m equal-length spline segments. Our technique avoids the high cost of arc-length parameterization by generating a new curve that accurately approximates the input curve and is approximately parameterized by arc length. A substantial advantage of this approach is that the heavy burden of computation is pushed into off-line preprocessing steps. The payoff is that on-line computations can be completed very quickly. Moreover, the precision of the on-line computations can be predetermined during the preprocessing stage by selecting the granularity of the approximation curve. The major on-line cost accrued by increasing the accuracy of the approximation is in the number of coefficients of the spline functions that must be stored.

§4. Error Analysis

The derived curve is an arc-length parameterized approximation of the initial spline curve in two senses:

- The shape of the derived curve approximately matches the shape of the input curve. We call the misfit of the derived curve from the input curve the match error.

- The derived curve is approximately arc-length parameterized. We call the deviation from arc-length parametrization the parameterization error.

Match and parameterization errors are related to one another. If the derived curve precisely matches the input curve, then the parameterization error will be minimal. In this case, the principal source of parameterization error will be from errors in estimating the points $\tilde{t}_1$, $\tilde{t}_2$, ..., $\tilde{t}_{m-1}$ that divide the original curve into equal length intervals. A desirable property of our method is that these points are pre-computed, and therefore we can afford to spend considerable computational effort in finding very accurate estimates. The bisection method used to solve equation (2) allows us to estimate $\tilde{t}_i$ with any desired accuracy.

The magnitude of the match error is related to the curvature of the input curve. At the knot points, $\tilde{t}_0$, $\tilde{t}_1$, ..., $\tilde{t}_m$, the derived curve interpolates the input curve. Between two adjacent knot points, the derived curve approximates the input curve with a cubic. For an arc of a circle, the match error decreases precipitously as the number of knot points on the approximation curve increases. For curves with smooth variation curvature, the maximum error tends to be highest in regions of high curvature (as we demonstrate below in the results of experiments with spiral curves.)

§5. Experimental Results

We demonstrate the properties of our arc-length parameterization method by testing the method on a cubic spline constructed by interpolating points sampled from the spiral curve drawn in the left graph in Figure 1. Our choice of a spiral is motivated by our interest in using the technique to model roads for real-time driving simulation. Modern highways are designed according to standards that specify road layout to meet safety and drivability criteria. A highway is composed of a sequence of segments of constant curvature interconnected by transition spirals that smoothly join segments of differing curvature. Roads designed in this fashion have the desirable property that the change in curvature is always smooth. In addition to being an important curve in road modeling, a spiral curve is useful for testing because it contains a continuous range of curvature (as shown in the right graph in Figure 1) and thus permits us to examine the relationship between curvature and errors. Figure 2 shows the cubic arc-length parameterized curves (solid) generated from the spline curve in the left graph in Figure 1 with the number of spline segments $m = 5$ and $m = 10$. The input spline curve (dashed) is overlaid in both cases.

The match error can be measured by the offset of the generated arc-length parameterized curve from the original curve. We can compute this error in the following way. We traverse the arc-length parameterized

number of spline segments	maximum arc-length parameterization error	maximum match error
5	0.09966	0.0422
10	0.018	0.0037
20	0.0028	0.000395
40	0.00045	0.000032

Tab. 1. The relationship between the number of spline segments and the maximum error in the derived arc-length parameterized curve.

curve and the original curve at the same time. For a point on the arc-length parameterized curve computed with the arc-length parameter $s \in [0, L]$, we locate the point on the original curve with the arc length value equal to the corresponding arc-length parameter s. We compute the offset between the corresponding points. Figure 3 shows the match error in the approximation curves for $m = 5$ and $m = 10$. Comparing the match errors in Figure 3 to the curvature displayed in the right graph in Figure 1, we see the magnitude of match error is related to the curvature of the input curve. High curvature regions require fine-grained sampling to capture the turn of the curve. By setting the sampling distance to be small enough to well represent a circle with radius equal to 1/max curvature of the spiral, we can keep the error in a tolerable range. As a rule of thumb, the sampling distance should be at least 1/(4 max curvature).

Figure 4 shows the arc-length parameterization error measured by the formula $\sqrt{(\tilde{x}'(s))^2 + (\tilde{y}'(s))^2 + (\tilde{z}'(s))^2} - 1.0$. From the graphs in Figure 4 we can see the magnitude of arc-length parameterization error is also related to the curvature of the curve. When the curvature becomes smaller, the match error becomes smaller.

As with match error, the quality of arc-length parameterization improves when we increase the number of spline segments in the derived arc-length parameterized curve. Table 1 shows the relationship between the number of spline segments and the maximum error. We observe that the match error decreases about 10 times, and the arc-length parameterization error decreases more than 5 times for each doubling of the number of spline segments in the arc-length parameterized curve.

§6. Conclusion

The technique presented for constructing an approximately arc-length parameterized approximation curve is well adapted for a variety of real-time applications. For example, driving simulators commonly need to compute distance queries at very high frequencies such as finding a point at a particular distance on a curve. This query is important for the behavior code that guides autonomous vehicles on synthetic roadways. Using the method introduced, a point at a particular distance along the curve can

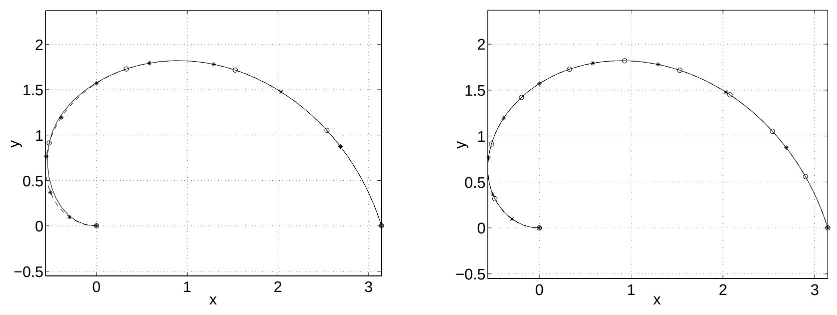

Fig. 2. The arc length parameterized curve (solid with knot points marked by ‘o’) and the original curve (dashed with knot points marked by ‘*’) for $m = 5$ in the left panel and $m = 10$ in the right panel.

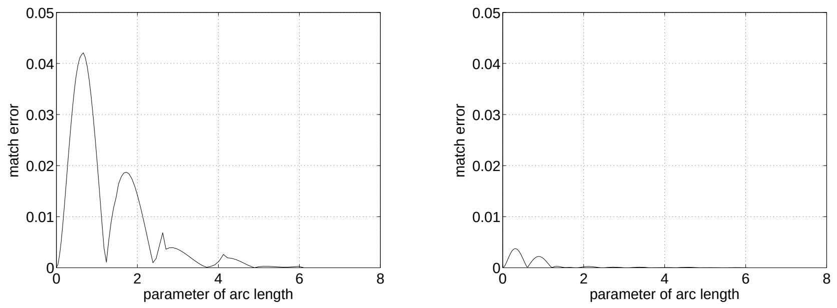

Fig. 3. Match error in the arc-length parameterized curve for $m = 5$ in the left panel and $m = 10$ in the right panel.

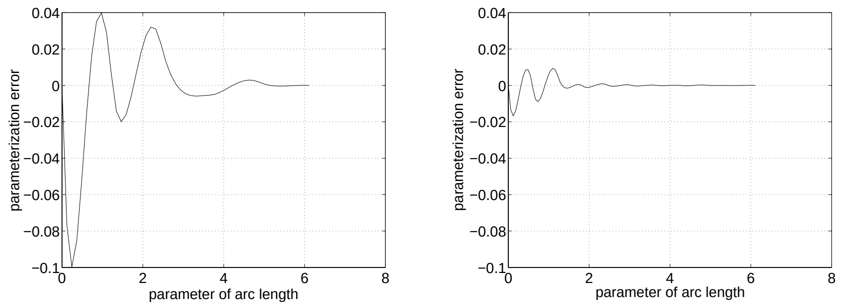

Fig. 4. Parameterization error in the arc-length parameterized curve for $m = 5$ in the left panel and $m = 10$ in the right panel.

be determined with a single evaluation of a cubic. Most techniques for computing arc length from a parameter value or for computing the parameter value at a specific arc length require a table search be performed on a composite curve. Because our segments all have the same length, we can very efficiently map to the correct segment without a table search.

An attractive property of our approach is that the precision of the computations (measured as either match error or parameterization error) is determined solely by computations performed off-line, prior to running a simulation. We can achieve a desired error tolerance by refining the estimate of the equally-spaced sample points on the input curve and by increasing the number of intervals in the derived curve. The only run-time cost accrued is in the space need to store a larger number of coefficients of cubic spline functions.

Acknowledgments. This work was supported in part through National Science Foundation grants INT-9724746, EAI-0130864, and IIS-0002535. Jim Cremer and Pete Willemsen made significant contributions to the development of the Hank simulator.

References

1. Farouki, R. T. and T. Sakkalis, Real rational curves are not 'unit speed', Comput. Aided Geom. Design **8** (1991), 151–157.
2. Gil, J. and D. Keren, New approach to the arc length parameterization problem, 13th Spring Conference on Computer Graphics, (1997), 27–34.
3. Guenter, B. and R. Parent, Computing the arc length of parametric curves, IEEE Comp. Graph. Appl. **5** (1990), 72–78.
4. Walter, M. and A. Fournier, Approximate arc length parameterization, Proceedings of the 9th Brazilian Symposium on Computer Graphics and Image Processing, Oct, (1996), 143–150.
5. Wang, F.C. and D. C. H. Yang, Nearly arc-length parameterized quintic spline interpolation for precision machining, Computer-Aided Design **25(5)** (1993), 281–288.
6. Willemsen, P., J. Kearney, and H. Wang, Ribbon networks for modeling navigable paths of autonomous agents in virtual urban environments, IEEE Virtual Reality Conference, (2003), 79–86.

Hongling Wang and Joseph Kearney and Kendall Atkinson
Department of Computer Science
The University of Iowa
Iowa City, IA 52242
`howang|kearney|atkinson@cs.uiowa.edu`