7-1985

Appendices - Parametric Keyframe Interpolation Incorporating Kinetic Adjustment and Phrasing Control

Scott N. Steketee

Norman I. Badler
University of Pennsylvania, badler@seas.upenn.edu

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-85-18.

PARAMETRIC KEYFRAME INTERPOLATION INCORPORATING KINETIC ADDJUSTMENT AND PHRASING CONTROL

Scott N. Steketee
Norman I. Badler

MS-CIS-85-18
GRAPHICS LAB 04

Department Of Computer and Information Science
Moore School
University of Pennsylvania
Philadelphia, PA 19104

July 1985

R E F E R E N C E S :

[Badl84] Badler, N. I., "Design of a Human Movement Representation Incorporating Dynamics", CIS Technical Report, Department of Computer and Information Science, School of Engineering and Applied Science, University of Pennsylvania, 1984.

[Bars83a] Barsky, B. and J. Beatty, "Local Control of Bias and Tension in Beta-splines", Computer Graphics 17:3 (Proc. SIGGRAPH '83), pp. 193-218.

[Bars83b] Barsky, B. and T. DeRose, "The Beta2-spline: A special Case of the Beta-spline Curve and Surface Representation", Report No. UCB/CSD 83/152, Computer Science Division, University of California, Berkeley, CA, November 1983.

[Burt76] Burtnyk, N. and M. Wein, "Interactive Skeleton Techniques for Enhancing Motion Dynamics in Keyframe Animation", Communications of the Association for Computing Machinery, 19:10 (October 1976), pp. 564-569.

[Catm78] Catmull, E. "The Problems of Computer-Assisted Animation", Computer Graphics 17:3 (Proc. SIGGRAPH '83), p. 348.

[Cox72] Cox, M. G. "The Numerical Evaluation of B-Splines", J. Inst. Maths. Applics., 10, 134-149.

[Curr66] Curry, H. and I. Schoenberg, "The fundamental spline functions and their limits", J. d'Analyse Math., 17 (1966), 71-107; p. 115.

[deBo72] deBoor, C., "On Calculating with B-Splines", J. Approx. Th. 6, pp. 50-62.

[deBo78] deBoor, C, A Practical Guide to Splines, Springer-Verlag, New York, 1978.

[Faux79] Faux, I. D. and M. J. Pratt, Computational Geometry for Design and Manufacture, Ellis Horwood, Chichester, England, 1979.

[Gera78] Gerald, Curtis F., Applied Numerical Analysis, Addison-Wesley, Reading, MA, 1978.

[Koch84] Kochanek, D. and R. Bartels, "Interpolating Splines with Local Tension, Continuity, and Bias Control", Computer Graphics 18:3 (Proc. SIGGRAPH '84).

[Loom83] Loomis, J., H. Poizner, U. Bellugi, A. Blakemore, and J. Hollerbach, "Computer Graphic Modeling of American Sign Language, Computer Graphics 17:3 (Proc. SIGGRAPH '83), pp. 105-114.

[Reev81] Reeves, W. "Inbetweening for computer animation utilizing moving point contstraints", Computer Graphics 15:3 (Proc. SIGGRAPH '81), Aug. 1981, pp. 263-269.

[Shel82] Shelley, K. and D. Greenberg, "Path Specification and Path Coherence", Computer Graphics 16:3 (Proc. SIGGRAPH '82), pp. 157-166.

[Webs71] Webster's Third New International Dictionary, G. & C. Merriam Co., Springfield, MA 1971

APPENDIX A

A SUMMARY OF INTERPOLATION METHODS, INCLUDING B-SPLINES

The interpolation of motion between keyframes and the smoothing or "phrasing" of motions at transitions from one motion to another require the fitting of smooth curves to the motion variables. To facilitate control of the motion, these curves must be easily manipulated by the user. Thus the choice of interpolation method is of considerable importance. The following discussion considers several important interpolation techniques and briefly describes some of their advantages and disadvantages in the context of interpolation for animation.

Polynomial Interpolation

The most straight-forward method of fitting a curve to a set of data points is polynomial interpolation. To fit n data points, a polynomial of order n is constructed using coefficients chosen so that each data point is on the curve. (The "order" of a polynomial is defined as the degree plus one. Thus a cubic polynomial is of order 4.) The brute-force method involves constructing the polynomial in the form

$$P(X) = a_{n-1}X^{n-1} + a_{n-2}X^{n-2} + \dots + a_1X + a_0$$

and substituting the n data points to generate n equations in n unknowns. For large values of n, this is rather inefficient, and there are two other forms which are more commonly used: the Lagrange form and the Newton form. The Lagrange form is shown in Figure A1; the expression for the polynomial can be written down directly from the data points. Although this form is very easy to generate, it is not so easy to evaluate; the number of multiplications and divisions required can become rather large, so that the difficulty in evaluation soon comes to outweigh the efficiency in generating the expression for the polynomial.

Lagrange Form

$$l_i(X) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{X - X_j}{X_i - X_j}$$

$$p(X) = \sum_{i=1}^{n} y_i \, l_i(X)$$

Example

x	y
0	0
1	1
8	2
27	3

$$p(x) = 0 + 1\,\frac{(x-0)(x-8)(x-27)}{(1-0)(1-8)(1-27)} + 2\,\frac{(x-0)(x-1)(x-27)}{(8-0)(8-1)(8-27)} + 3\,\frac{(x-0)(x-1)(x-8)}{(27-0)(27-1)(27-8)}$$

Figure A1: The Lagrange Form of Polynomial Interpolation.

Newton Form

$$p(x) = \sum_{i=0}^{n-1} \alpha_i (x-x_1)(x-x_2)\ldots(x-x_i)$$

e.g., for n = 4,

$$p(x) = \alpha_0 + \alpha_1(x-x\ ) + \alpha_2(x-x\ )(x-x\ ) + \alpha_3(x-x\ )(x-x\ )(x-x\ )$$

The α_i are calculated using "divided differences."

Example

$$\begin{array}{c|c|cccc}
x & y & & & & \\
\hline
0 & 0 \; (\alpha_0) & & & & \\
 & & \frac{1-0}{1-0} = 1 \; (\alpha_1) & & & \\
1 & 1 & & \frac{\frac{1}{7} - 1}{8-0} = -.107 \; (\alpha_2) & & \\
 & & \frac{2-1}{8-1} = \frac{1}{7} & & \frac{-.0035 - -.107}{27-0} = .0038 \; (\alpha_3) & \\
8 & 2 & & \frac{\frac{1}{19} - \frac{1}{7}}{27-1} = -.0035 & & \\
 & & \frac{3-2}{27-8} = \frac{1}{19} & & & \\
27 & 3 & & & &
\end{array}$$

So $p(x) = 0 + 1(x-0) + {}^{-}.107(x-0)(x-1) + .0038(x-0)(x-1)(x-8)$

Figure A2: The Newton Form of Polynomial Interpolation.

The Newton form is shown in Figure A2; although it is somewhat more complex to generate the polynomial in this form than in the Lagrange form, the result is much easier to evaluate, using a method which is similar to Horner's Rule. For most purposes, the Newton form of polynomial interpolation strikes a very good balance between ease of generation and ease of evaluation.

Problems with Polynomial Interpolation

Polynomial interpolation has several problems, however. Large slopes and multiple-valued functions (desirable, for instance, to model two-dimensional or three-dimensional paths in space) cannot be expressed in terms of polynomials. (See Figure A3 for examples.) This problem can be solved by the use of parametric interpolation, in which the dependent variable is expressed as a function of some parameter other than the original independent variable. Arc lengths and angles are two common choices for these parameters.

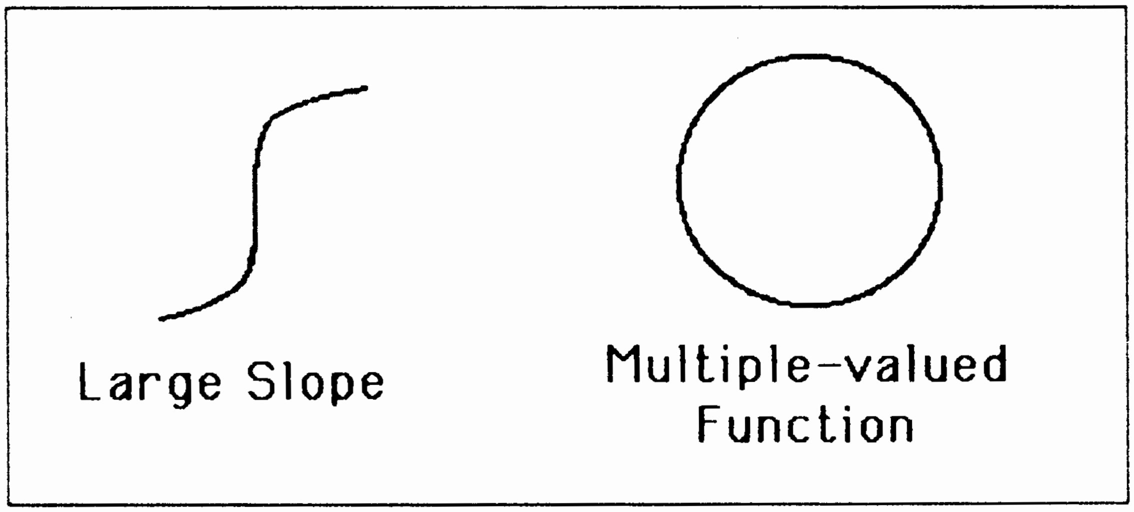

Figure A3: Curves Not Easily Expressed Using Polynomial Interpolation

Another difficulty with polynomial interpolation, and one that is less easily solved, is that higher-order polynomials have a tendency to oscillate, and the tendency increases with increasing order of the polynomial. Figure A4 shows an example of the severe oscillations associated with high-degree polynomials. This difficulty makes polynomial interpolation quite unacceptable for a large class of problems, including the animation problem dealt with in this paper.

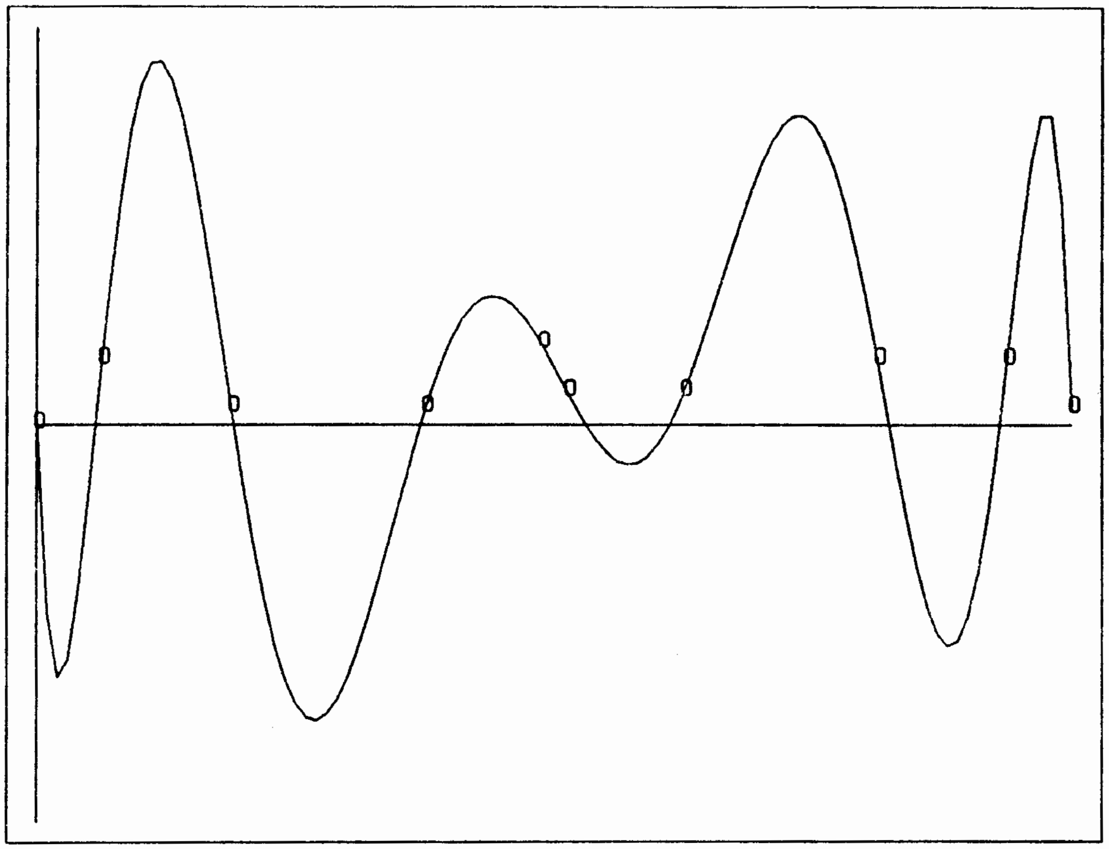
Figure A4: Oscillations Can Occur with Polynomial Interpolation. The interpolant here is a polynomial of degree 9; it oscillates wildly between data values.

One way to avoid the problems of oscillation when using polynomial interpolation is to keep the order of the polynomial sufficiently low. (The "order" of a polynomial is one more than its degree; thus a cubic function is considered to be of order 4.) However, a single polynomial of order n can interpolate only n data points, assuming we want the interpolant to actually pass through the data points. One solution is to find a best-fit polynomial of low order. This method is often unsatisfactory, as it results in an interpolant which does not actually pass through the data points.

Piecewise Polynomials

Another solution is to use not one single interpolant, ʰut a number of them, each of low order. This method is called "piecewise polynomial interpolation". If we use a different polynomial for each interval from one data point to the next, and if we can make the polynomials join each other in a reasonable way, we derive the computational benefits of polynomial interpolation (ease of generating and evaluating the interpolant) without the disadvantages of oscillations. Figure A5 shows an example using nine different polynomials to interpolate the same data points as shown in Figure A4.

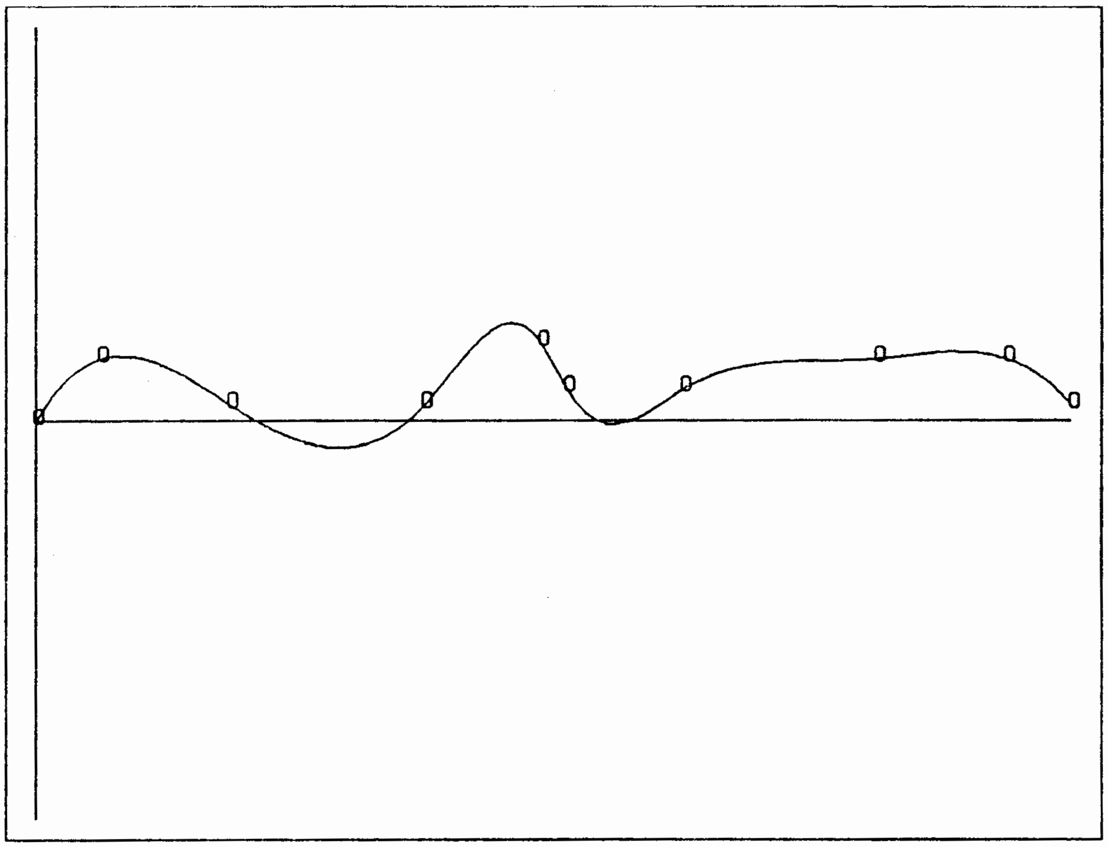
Figure A5: Piecewise Polynomial Interpolation Reduces Oscillations. The same data points as in Figure A4 are shown here, but the interpolant is actually nine different polynomials, one for each interval between data points.

In piecewise polynomial interpolation, each individual polynomial is defined over a limited domain, or "span". The x-values at the end points of these spans are called "knots" or "break points" ("knots" because it is at the knots that the individual pieces are tied together; "break points" because it is at the break points that we break from using one polynomial and begin using the next). In Figure A5, the knots are identical with the x-values of the data points; that is, each individual polynomial interpolates the span from one data point to the next. This is not a necessary feature of piecewise polynomials; it is certainly possible (and sometimes desirable) to define the polynomials in such a way that the knots fall in between the data points.

There are several different ways to choose the individual polynomials which will interpolate our data points. We need to choose both the order of the polynomials to be used, and how they are to be joined to each other. For many purposes cubic polynomials have been found to be quite satisfactory: they are of low enough order that they show little tendency to oscillate, and of high enough order that they can be controlled to fit both the data and each other in a reasonable way. There are times, of course, when other choices will be appropriate, but in the following discussion of how the polynomials are to be specified, we will assume the use of cubic polynomials.

One way of determining the polynomial pieces to be used, called Hermite interpolation, requires us to specify both the value and the first derivative at each data point. Each polynomial piece is then required to fit four conditions (two at each end point), and this completely determines a cubic polynomial. This method provides the user with control over both the value and the slope of the curve at each data point. If control over slope is desired, Hermite interpolation is a good choice. However, this control is often undesirable; if no reliable information is available about the value of the derivative at the data points, Hermite interpolation should be avoided. Another disadvantage of Hermite interpolation is that the interpolant is discontinuous in the second derivative at each data point; for many applications this discontinuity is perceptible and therefore unacceptable.

Spline Interpolation

A second method of piecewise polynomial interpolation is everywhere continuous in the function itself and in its first two derivatives. This method provides the maximum smoothness possible for the order of the polynomials used; it is called "spline" interpolation after the draftsman's spline (a thin flexible strip which the draftsman can bend to create a smooth curve passing through a number of points). In cubic spline interpolation (that is, piecewise polynomial interpolation using cubic polynomial pieces which are continuous in the second derivative where they join), each piece must pass through the two data points which mark the beginning and end of that piece, and both the first and second derivatives of the piece must match the corresponding derivatives of the adjacent pieces at the data points.

Let us analyze the number of conditions needed to completely determine the cubic pieces which interpolate n data points. Since n-1 pieces are required, and each piece is defined by four coefficients, a total of 4n-4 conditions are needed. Each piece must pass through the two data points which mark its beginning and end; this requirement supplies 2n-2 of the conditions. We require continuity of two derivatives at each of n-2 internal data points; this requirement supplies 2n-4 conditions. Thus we are only two conditions short. We can specify the additional two conditions in any of several different ways. A discussion of some commonly used conditions follows.

End conditions

The remaining two conditions which we must supply to determine our piecewise cubic polynomial are commonly called end conditions, because they are most commonly specified at one end or another of the interpolant. (In fact, they could instead be specified at some internal point, if desired.) One common method is to specify the value of either the first or second derivative at one end or the other. Since any combination of two conditions will do, we could specify the first derivative at the beginning and the second derivative at the end, or the second derivative at the beginning of the interpolant and the first derivative at some point internal to the interpolant, or any other combination we desire.

If there is any information available about derivatives at the end points of the spline, that information should be used to establish appropriate end conditions. It is commonly the case that such information is not available, and we are left with the problem of generating two end conditions without any satisfactory basis for making our choice. Several alternatives present themselves. We could specify a zero second derivative at both ends of the spline; the spline thus generated is often called a "natural" spline. In the absence of any justification for this choice, it is actually likely to distort the curve and give less satisfactory results than can be otherwise achieved. [DeBo78, p. 55] A better choice in the absence of any actual information about end point derivatives is the not-a-knot condition, in which we force the first and last polynomial pieces each to interpolate two spans. (Figure A5 above is actually a spline interpolant generated using not-a-knot end conditions.)

Basis Functions

Any given polynomial can be expressed in a number of different ways. The normal "power" form, the Lagrange form, and the Newton form are three examples. In each case, a polynomial of order k is expressed as the sum of k linearly independent terms, with each term consisting of a coefficient multiplied by some polynomial function of the independent variable. These linearly independent polynomial functions (of order less than or equal to k) are called "basis" functions, since they form a satisfactory basis for expressing any polynomial of the appropriate order. (Note that if the functions are linearly independent, at least one of them must be of order k.) Several different sets of basis functions which can be used to express cubic polynomials are shown in Figure A6; formulas for the individual functions making up each basis are shown in Figure A7.

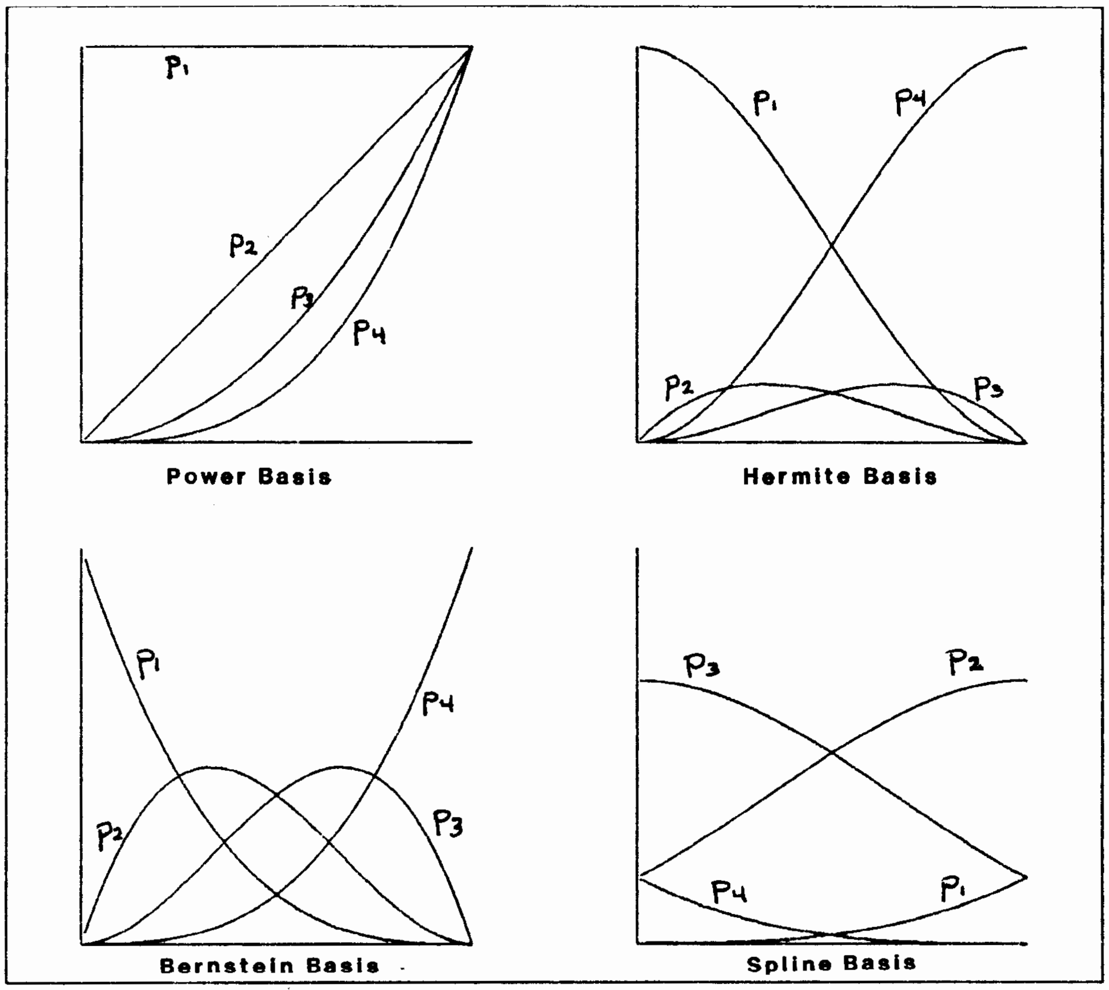

Figure A6: Four Sets of Basis Functions. The functions are all defined on the domain (0, 1); they may readily be adjusted for other domains.