Lecture 20: Lagrange Interpolation and Neville's Algorithm

for I will pass through thee, saith the LORD. Amos 5:17

1. Introduction

Perhaps the easiest way to describe a shape is to select some points on the shape. Given enough data points, the eye has a natural tendency to interpolate smoothly between the data. Here we are going to study this problem mathematically. Given a finite collection of points in affine space, we shall investigate methods for generating polynomial curves and surfaces that pass through the points. We begin with schemes for curves and later extend these techniques to surfaces.

2. Linear Interpolation

Two points determine a line. Suppose we want the equation of the line $P(t)$ passing through the two points P and Q in affine space. Then we can write

$$P(t) = P + t(Q - P) \tag{2.1}$$

The curve $P(t)$ passes though P at $t = 0$ and Q at $t = 1$. Moreover as t varies the points on $P(t)$ extend in the direction along the vector from P to Q; thus, these points lie along the line in affine space generated by P and Q. Rearranging terms, we can rewrite (2.1) as

$$P(t) = (1 - t)P + tQ \tag{2.2}$$

Equation (2.2) is called *linear interpolation;* this equation is the foundation of all we plan to accomplish in this lecture.

One subtle issue. We saw that in (2.2) $P(t)$ passes through P at $t = 0$ and through Q at $t = 1$. We did not specify this requirement in the original problem. All we wanted was a line passing through the two points P and Q; the parameters t at which the line was to pass through these points was not mentioned. Suppose, however, that we do wish to specify these parameters as well. That is, now we require a line $P_{01}(t)$ to pass through P_0 at $t = t_0$ and through P_1 at $t = t_1$. Mimicking (2.2), we expect to write an equation of the form

$$P_{01}(t) = (1 - f(t))P_0 + f(t)P_1. \tag{2.3}$$

Moreover, still emulating (2.2), we want $f(t)$ to be linear and to satisfy

$$f(t_0) = 0 \qquad \text{and} \qquad f(t_1) = 1.$$

These equations for $u = f(t)$ represent another linear interpolation problem; this time in the tu-plane. That is, now we need to find the line in the coordinate plane interpolating the data $(t_0, 0)$ and $(t_1, 1)$. Of course you learned long ago, when you first studied analytic geometry, how to solve such problems. This line is given by the equation

$$f(t) = \frac{(t - t_0)}{(t_1 - t_0)} \tag{2.4}$$

as you can readily verify by evaluating $f(t)$ at $t = t_0$ and $t = t_1$. Substituting (2.4) into (2.3), we obtain

$$P_{01}(t) = \frac{t_1 - t}{t_1 - t_0} P_0 + \frac{t - t_0}{t_1 - t_0} P_1 \tag{2.5}$$

where we have used the identity $1 - f(t) = (t_1 - t)/(t_1 - t_0)$.

Equation (2.5) is so fundamental that we are going to represent it graphically with a simple diagram.

$$\begin{tikzcd}
 & P_{01}(t) & \\
P_0 \arrow[ur, "\frac{t_1 - t}{t_1 - t_0}"] & & P_1 \arrow[ul, "\frac{t - t_0}{t_1 - t_0}"']
\end{tikzcd}$$

(a) Normalized

$$\begin{tikzcd}
 & P_{01}(t) & \\
P_0 \arrow[ur, "t_1 - t"] & & P_1 \arrow[ul, "t - t_0"']
\end{tikzcd}$$

(b) Unnormalized

Figure 1: Diagrams representing Equation (2.5).

In Figure 1(a) the value at the apex of the triangle is computed by multiplying the points at the base by the values along the arrows and then adding the results. The end product is just Equation (2.5). Figure 1(b) represents exactly the same computation as Figure 1(a). Here, however, we have removed the normalization in the denominator to simplify the diagram. The denominator can be retrieved by summing the numerators, since in affine space the functions multiplying the points must sum to one. The advantage of Figure 1(b) is that it is much less cluttered than Figure 1(a), so in the future we shall usually draw these graphs in this unnormalized form. You should get used to this simple diagram now because you are going to see many more like it throughout this lecture.

3. Neville's Algorithm

Let us try a slightly harder problem. Suppose we now have three points P_0, P_1, P_2 in affine space that we wish to interpolate at the parameters t_0, t_1, t_2. How shall we proceed?

We already have a way to interpolate P_0, P_1 at t_0, t_1; we can join these points with the straight line

$$P_{01}(t) = \frac{t_1 - t}{t_1 - t_0} P_0 + \frac{t - t_0}{t_1 - t_0} P_1 \ .$$

Similarly, by reindexing, we can interpolate P_1, P_2 at t_1, t_2 with the straight line

$$P_{12}(t) = \frac{t_2 - t}{t_2 - t_1} P_1 + \frac{t - t_1}{t_2 - t_1} P_2 \ .$$

The piecewise linear curve given by

$$\begin{aligned} P(t) &= P_{01}(t) \qquad && t \le t_1 \\ &= P_{12}(t) && t \ge t_1 \end{aligned}$$

certainly interpolates the points P_0, P_1, P_2 at the parameters t_0, t_1, t_2. However, this curve is not smooth; it has a sharp point at P_1. Sharp points are potentially dangerous and hence undesirable in objects designed for human consumption. We seek a smooth curve that does the job.

To generate a smooth curve, apply linear interpolation to the two curves $P_{01}(t)$ and $P_{12}(t)$:

$$P_{012}(t) = \frac{t_2 - t}{t_2 - t_0} P_{01}(t) + \frac{t - t_0}{t_2 - t_0} P_{12}(t) \ . \tag{3.1}$$

By substitution it is easy to verify that $P_{012}(t)$ interpolates P_0 and P_2 at t_0 and t_2, since by (3.1)

$$P_{012}(t_0) = P_{01}(t_0) = P_0$$
$$P_{012}(t_2) = P_{12}(t_2) = P_2$$

To verify that $P_{012}(t)$ also interpolates P_1 at t_1, observe that $P_{01}(t)$ and $P_{12}(t)$ both interpolate P_1 at t_1. Therefore

$$\begin{aligned} P_{012}(t_1) &= \frac{t_2 - t_1}{t_2 - t_0} P_{01}(t_1) + \frac{t_1 - t_0}{t_2 - t_0} P_{12}(t_1) \\ &= \frac{t_2 - t_1}{t_2 - t_0} P_1 + \frac{t_1 - t_0}{t_2 - t_0} P_1 \\ &= P_1 \ . \end{aligned}$$

If we were to expand the right hand side of (3.1), we would find that $P_{012}(t)$ is a quadratic polynomial in *t,* since $P_{01}(t)$ and $P_{12}(t)$ are both linear in *t*. Thus we have constructed a smooth curve that interpolates the given points at the specified parameter values (see Figure 2). Figure 3 is a graphical representation of Equation (3.1).

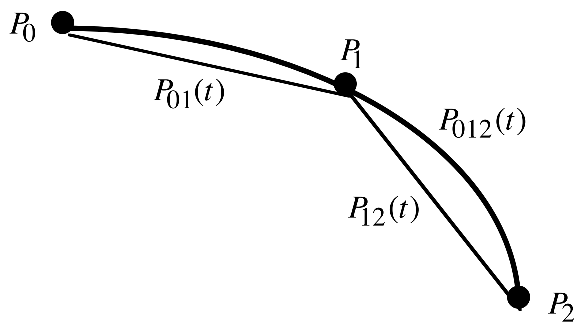


Figure 2: The two lines $P_{01}(t)$ and $P_{12}(t)$, and the quadratic interpolant $P_{012}(t)$.

$$\begin{tikzcd}
& & P_{012}(t) & & \\
& P_{01}(t) \arrow[ur, "t_2 - t"] & & P_{12}(t) \arrow[ul, "t - t_0"'] & \\
P_0 \arrow[ur, "t_1 - t"] & & P_1 \arrow[ul, "t - t_0"'] \arrow[ur, "t_2 - t"] & & P_2 \arrow[ul, "t - t_1"']
\end{tikzcd}$$

Figure 3: A graphical representation of Equation (3.1). The first level of the diagram is just two juxtaposed copies of Figure 1, one for $P_{01}(t)$ and one for $P_{12}(t)$. The second level represents the linear interpolation step joining $P_{01}(t)$ and $P_{12}(t)$. Here we have adopted our convention of leaving off the denominators to avoid cluttering the diagram.

What if we want to interpolate four points P_0, P_1, P_2, P_3 at parameter values t_0, t_1, t_2, t_3? We already know how to build quadratic curves to interpolate portions of this data. We can construct $P_{012}(t)$ to interpolate P_0, P_1, P_2 at t_0, t_1, t_2 and $P_{123}(t)$ to interpolate P_1, P_2, P_3 at t_1, t_2, t_3. Diagraming $P_{123}(t)$ yields Figure 4.

$$\begin{tikzcd}
& & P_{123}(t) & & \\
& P_{12}(t) \arrow[ur, "t_3 - t"] & & P_{23}(t) \arrow[ul, "t - t_1"'] & \\
P_1 \arrow[ur, "t_2 - t"] & & P_2 \arrow[ul, "t - t_1"'] \arrow[ur, "t_3 - t"] & & P_3 \arrow[ul, "t - t_2"']
\end{tikzcd}$$

Figure 4: A graphical representation for the curve $P_{123}(t)$.

Figure 3 and Figure 4 share the little subtriangle with vertex $P_{12}(t)$. Overlapping these two figures and joining $P_{012}(t)$ and $P_{123}(t)$ by yet another linear interpolation step

$$P_{0123}(t) = \frac{t_3 - t}{t_3 - t_0} P_{012}(t) + \frac{t - t_0}{t_3 - t_0} P_{123}(t),$$

we arrive at Figure 5.

$$\begin{tikzcd}
 & & & P_{0123}(t) & & & \\
 & & P_{012}(t) \arrow[ur, "t_3 - t"] & & P_{123}(t) \arrow[ul, "t - t_0"'] & & \\
 & P_{01}(t) \arrow[ur, "t_2 - t"] & & P_{12}(t) \arrow[ul, "t - t_0"'] \arrow[ur, "t_3 - t"] & & P_{23}(t) \arrow[ul, "t - t_1"'] & \\
P_0 \arrow[ur, "t_1 - t"] & & P_1 \arrow[ul, "t - t_0"'] \arrow[ur, "t_2 - t"] & & P_2 \arrow[ul, "t - t_1"'] \arrow[ur, "t_3 - t"] & & P_3 \arrow[ul, "t - t_2"']
\end{tikzcd}$$

Figure 5: Neville's algorithm for cubic interpolation.

Now it is easy to verify directly from the diagram that $P_{0123}(t)$ interpolates the given data at the specified parameter values. By substitution, we see that

$$P_{0123}(t_0) = P_{012}(t_0) = P_0$$
$$P_{0123}(t_3) = P_{123}(t_3) = P_3 \ .$$

Moreover we already know that

$$P_{012}(t_k) = P_{123}(t_k) = P_k \qquad k = 1,2$$

and since the labels on the arrows exiting $P_{012}(t)$ and $P_{123}(t)$ sum to one (remember the normalization), it follows that

$$P_{0123}(t_k) = P_k \qquad k = 1,2 \ .$$

The algorithm for computing $P_{0123}(t)$ represented by Figure 5 is called *Neville's algorithm.* We shall have a lot more to say about this algorithm shortly. The curves generated by Neville's algorithm are called *Lagrange interpolating polynomials.* We illustrate an example of a Lagrange interpolating polynomial in Figure 6.

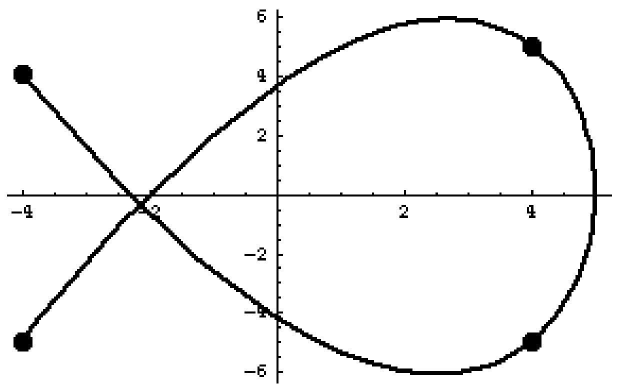


Figure 6: The cubic Lagrange polynomial for the control points: $P_0 = (-4, 4)$, $P_1 = (4, -5)$, $P_2 = (4, 5)$, $P_3 = (-4, -5)$ (dots), interpolated at the nodes $t_k = k$, $k = 0, ..., 3$.

We could go on introducing more and more data points and constructing higher and higher order curves, but by now it should be clear how to proceed. Instead, let us summarize what we expect to be true in the following theorem.

Theorem 1: *Given affine points* $P_0, ..., P_n$ *and distinct parameters* $t_0, ..., t_n$, *there is a polynomial curve* $P_{0\cdots n}(t)$ *of degree n that interpolates the given points at the specified parameters. That is,* $P_{0\cdots n}(t_k) = P_k$, $k = 0, ..., n$.

Proof: The proof is by induction on *n*. We have already established this result by construction for $n = 0, 1, 2, 3$. Suppose this result is true for $n-1$. Then by the inductive hypothesis, there are polynomial curves $P_{0\cdots n-1}(t)$ and $P_{1\cdots n}(t)$ of degree $n-1$ that interpolate the points $P_0, ..., P_{n-1}$ at the parameters $t_0, ..., t_{n-1}$ and the points $P_1, ..., P_n$ at the parameters $t_1, ..., t_n$. Define

$$P_{0\cdots n}(t) = \frac{t_n - t}{t_n - t_0} P_{0\cdots n-1}(t) + \frac{t - t_0}{t_n - t_0} P_{1\cdots n}(t) \ . \qquad (3.2)$$

Then applying the same arguments we used in the quadratic and cubic cases, you can easily verify that

$$P_{0\cdots n}(t_k) = P_k \qquad k = 0, ..., n \ .$$

Moreover since $P_{0\cdots n-1}(t)$ and $P_{1\cdots n}(t)$ are polynomials of degree $n-1$, it follows from (3.2) that $P_{0\cdots n}(t)$ is a polynomial of degree *n*.

The parameter values $t_0,...,t_n$ at which the interpolation occurs are called *nodes* and the points $P_0,..., P_n$ that are interpolated are called *control points* (see Figure 6) . In general, if we change the nodes, then the interpolating curve $P_{0\cdots n}(t)$ changes even if we leave the control points fixed (see Exercise 2).

4. The Structure of Neville's Algorithm

Equation (3.2) is a recursive formula for $P_{0\cdots n}(t)$. It asserts that we can compute $P_{0\cdots n}(t)$ by calculating $P_{0\cdots n-1}(t)$ and $P_{1\cdots n}(t)$ and then taking a specific affine combination of the results. Continuing in this manner, we can also compute $P_{0\cdots n-1}(t)$ and $P_{1\cdots n}(t)$ recursively. This recursion bottoms out at the constant functions $P_k(t) = P_k$.

$$\begin{tikzcd}
 & & & P_{0123}(t) & & & & \\
 & P_{012}(t) \arrow[urr] & & & & P_{123}(t) \arrow[ull] & & \\
P_{01}(t) \arrow[ur] & & P_{12}(t) \arrow[ul] & & P_{12}(t) \arrow[ur] & & P_{23}(t) \arrow[ul] & \\
P_0 \arrow[u] & P_1 \arrow[ul] & P_1 \arrow[u] & P_2 \arrow[ul] & P_1 \arrow[u] & P_2 \arrow[ul] & P_2 \arrow[u] & P_3 \arrow[ul]
\end{tikzcd}$$

Figure 7: Performing interpolation by recursive calls: the cubic case.

If we proceed in this manner, we arrive at an algorithm with the structure of a binary tree as illustrated in Figure 7. This algorithm is very inefficient because it requires an exponential number of procedure calls. Moreover, all the interior nodes -- that is, all the nodes not lying along the periphery of the diagram -- are computed twice; for example, $P_{12}(t)$ is computed once during the computation of $P_{012}(t)$ and once again during the computation of $P_{123}(t)$. Thus, implementing (3.2) recursively is not a good idea.

There is a better way: apply dynamic programming. In dynamic programming, we first compute all the constant interpolants, then all the linear interpolants, then all the quadratic interpolants, continuing to build higher and higher order interpolants as we go. In this fashion, each interpolant is computed only once. This approach leads to an $O(n^2)$ algorithm -- there are n linear interpolants, $n-1$ quadratic interpolants, $n-2$ cubic interpolants, etc. so altogether there are $n+(n-1)+\cdots+1 = n(n+1)/2 = O(n^2)$ interpolants -- rather than an exponential algorithm. It is precisely this technique that is illustrated for cubic curves in Figure 5.

Moreover, while the time complexity of this dynamic programming algorithm is $O(n^2)$, the space complexity is only $O(n)$. Indeed once we have computed the interpolants of order $k+1$ we can discard the interpolants of order k, since they are no longer needed to compute the higher order interpolants. This space efficiency is another advantage of dynamic programming.

This dynamic programming approach to interpolation is called *Neville's algorithm.* This algorithm and algorithms like it are at the heart of what we plan to study throughout these lectures. Get accustomed to it now because it will be fundamental to all our work later on. In particular, be sure you understand the difference between the dynamic programming algorithm illustrated in Figure 5 and the recursive procedure illustrated in Figure 7.

Neville's algorithm has an interesting structure. Call the base of the diagram the zeroth level and the apex the n^{th} level. Then the k^{th} level of the algorithm represents k^{th} order interpolants because, by construction, $P_{j+1\cdots j+k}(t)$ interpolates the control points $P_{j+1},\ldots,P_{j+k}$ at the nodes $t_{j+1},\ldots,t_{j+k}$. Notice that in the diagram the points $P_{j+1},\ldots,P_{j+k}$ lie in the span of the curve $P_{j+1\cdots j+k}(t)$; that is, the points $P_{j+1},\ldots,P_{j+k}$ form the base of the triangle with apex $P_{j+1\cdots j+k}(t)$. Thus each subtriangle reproduces in the small the structure of the entire triangle in the large.

The diagram for Neville's algorithm is easy to remember. Start with $P_{0\cdots n}(t)$ at the apex. Strip off the index n and place $P_{0\cdots n-1}(t)$ below it to the left; strip off the index 0 and place $P_{1\cdots n}(t)$ below it to the right. Since the index n was removed to the left, label the left arrow with $t_n - t$; since the index 0 was removed to the right, label the right arrow with $t - t_0$. Now proceed recursively stripping off labels from $P_{0\cdots n-1}(t)$ and $P_{1\cdots n}(t)$ and labeling the arrows accordingly. Remember to join $P_{1\cdots n-1}(t)$ to both $P_{0\cdots n-1}(t)$ and $P_{1\cdots n}(t)$ to generate a dynamic programming algorithm instead of a recursive procedure. Refer to Figure 5 for an illustration of the cubic case.

There is another important structural property of Neville's algorithm that is an artifact of this construction. Look at Figure 5. Pick any direction and consider the labels along the arrows as you ascend the triangle in that direction. Notice that these labels appear to be identical; this observation holds for any degree. In fact, these labels are not really identical because we have suppressed the denominators. Only the numerators match; the denominators differ from level to level.

Nevertheless this *parallel property* of matching numerators along parallel arrows is fairly important and we shall return to it again in subsequent lectures.

Neville's algorithm has one additional significant property. Suppose we have already interpolated the control points $P_0,\dots,P_n$ at the nodes $t_0,\dots,t_n$ and later we discover that we need to interpolate one additional point P_{n+1} at the parameter t_{n+1}. We need not restart our computations from the beginning. If we have saved the original triangular computation for $P_{0\cdots n}(t)$, then we need only add the edge computing $P_{n,n+1}(t),\dots,P_{0\cdots n+1}(t)$. That is, in the dynamic programming algorithm for $P_{0\cdots n+1}(t)$, we need only add the computation of one curve of each degree. Thus, at the cost of increasing our storage from $O(n)$ to $O(n^2)$, to complete our calculation, we need only add a computation of $O(n)$ instead of redoing work of $O(n^2)$. This saving is yet another advantage of the dynamic programming approach to interpolation.

5. Uniqueness of Polynomial Interpolants and Taylor's Theorem

Theorem 1 asserts that given any arbitrary sequence of points $P_0,\dots,P_n$ and any collection of distinct parameters $t_0,\dots,t_n$, there *exists* a polynomial curve $P_{0\cdots n}(t)$ of degree n that interpolates the given points at the specified parameters. Here we are going to show that this polynomial curve is *unique*, extending the result that two points determine a unique line. Notice, however, that uniqueness requires us to specify the nodes as well as the control points. We begin by recalling some simple facts about polynomials.

Theorem 2: *(Taylor's Theorem)*
Let $P(t)$ be a polynomial of degree n, and let r be a real number. Then

$$P(t) = P(r) + P'(r)(t-r) + P''(r)\frac{(t-r)^2}{2!} + \cdots + P^{(n)}(r)\frac{(t-r)^n}{n!} .$$

Proof: Since $P(t)$ is a polynomial of degree *n*, there must be constants $c_0,\dots,c_n$ such that

$$P(t) = c_0 + c_1 t + \cdots + c_n t^n .$$

Let $Q(t) = P(t+r)$. Then

$$Q(t) = c_0 + c_1(t+r) + \cdots + c_n(t+r)^n .$$

Expanding the powers of $(t+r)$ and collecting the coefficients of the powers of *t,* we see that $Q(t)$ is also a polynomial of degree *n* in *t,* so there must be constants $d_0,\dots,d_n$ such that

$$Q(t) = d_0 + d_1 t + \cdots + d_n t^n \ .$$

But $P(t) = Q(t-r)$, so by substitution

$$P(t) = d_0 + d_1(t-r) + \cdots + d_n(t-r)^n \ .$$

Differentiating both sides k times and evaluating at $t = r$ yields $d_k = P^{(k)}(r)/k!$.

Corollary 3: *Let $P(t)$ be a polynomial of degree n. Then r is a root of $P(t)$ if and only if $t - r$ is a factor of $P(t)$.*

Proof: Let $P(t)$ be a polynomial of degree n. Then by Taylor's theorem

$$P(t) = P(r) + P'(r)(t-r) + P''(r)\frac{(t-r)^2}{2!} + \cdots + P^{(n)}(r)\frac{(t-r)^n}{n!} \ .$$

Therefore, by inspection, $P(r) = 0$ if and only if $t - r$ is a factor of $P(t)$.

Corollary 4: *Every nonzero polynomial of degree n has at most n roots.*

Proof: This result is an immediate consequence of Corollary 3, since a polynomial of degree n can have at most n linear factors.

Corollary 5: *Let $P(t)$ and $Q(t)$ be two polynomials of degree n that agree at $n+1$ parameter values. Then $P(t) = Q(t)$.*

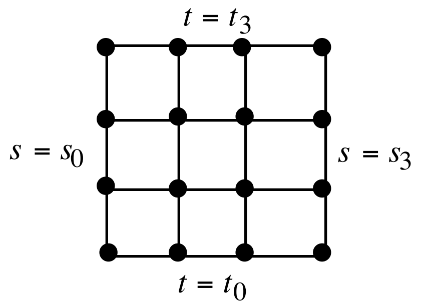
Proof: Let $R(t) = Q(t) - P(t)$. Then $R(t)$ is a polynomial of degree n. Moreover since $P(t)$ and $Q(t)$ agree at $n+1$ parameter values, $R(t)$ has $n+1$ roots. Therefore by Corollary 4, $R(t)$ must be the zero polynomial, so $P(t) = Q(t)$.

Theorem 6: *Given points $P_0,\ldots,P_n$ and distinct parameters $t_0,\ldots,t_n$, there exists a unique polynomial curve of degree n that interpolates the given points at the specified parameters.*

Proof: Existence has already been established in Theorem 1; it remains to demonstrate uniqueness. Suppose that $P(t)$ and $Q(t)$ are two polynomial curves of degree n that interpolate the given control points at the specified nodes. Then $P(t)$ and $Q(t)$ are polynomials of degree n that agree at the $n+1$ parameter values $t_0, \ldots, t_n$. Hence by Corollary 5 $Q(t) = P(t)$, so the interpolating polynomial is unique.

6. Tensor Product Lagrange Surfaces

Tensor product surfaces are some of the simplest surfaces to construct, but they are also some of the most important surfaces in computer graphics. In the tensor product construction, we start with two rectangular arrays of size $(m+1) \times (n+1)$: one for the control points $\{P_{ij}\}$ and one for the nodes $\{(s_{ij}, t_{ij})\}$, where $0 \le i \le m$ and $0 \le j \le n$ (see Figure 8).



$$\begin{array}{cccc}
P_{03} & P_{13} & P_{23} & P_{33} \\
P_{02} & P_{12} & P_{22} & P_{32} \\
P_{01} & P_{11} & P_{21} & P_{31} \\
P_{00} & P_{10} & P_{20} & P_{30}
\end{array}$$

(a) Domain -- Rectangular Grid (b) Range -- Rectangular Array of Points

Figure 8: Data for a tensor product bicubic interpolant: (a) represents the nodes in the domain and (b) represents the control points in the range. The nodes lie on a rectangular grid, but the control points may be in arbitrary positions. The surface $P(s,t)$ must interpolate the control points P_{ij} at the nodes (s_i, t_j) -- that is, $P(s_i, t_j) = P_{ij}$.

The nodes are in special position because they lie on a rectangular grid in the parameter plane -- that is, they lie along the parameter lines $s = s_i$ and $t = t_j$. We shall assume further that $s_0 < \cdots < s_m$ and $t_0 < \cdots < t_n$. We seek a bivariate polynomial $P(s,t)$ of degree m in s and degree n in t that interpolates the given control points at the specified parameter values -- that is, we seek a bivariate polynomial $P(s,t)$ of bidegree (m,n) such that

$$P(s_i, t_j) = P_{ij}.$$

This bivariate interpolation problem is easy to solve using univariate methods. Let $P_i(t)$, $i = 0,\ldots,m$, be the Lagrange interpolating curve to the control points $P_{i,0},\ldots,P_{i,n}$ at the nodes $t_0,\ldots,t_n$ -- that is,

$$P_i(t_j) = P_{i,j}. \tag{6.1}$$

For each fixed value of *t*, let $P(s,t)$ be the Lagrange interpolating curve to the control points $P_0(t),\ldots,P_m(t)$ at the nodes $s_0,\ldots,s_n$ -- that is,

$$P(s_i,t) = P_i(t) \tag{6.2}$$

Then as *t* varies from t_0 to t_n and *s* varies from s_0 to s_m, the curves $P(s,t)$ sweep out a surface (see Figure 9). Moreover,

$$P(s_i,t_j) = P_i(t_j) = P_{ij},$$

so we have indeed solved this interpolation problem. This surface is called a *tensor product Lagrange surface*.

Notice that if we restrict to the domain $s_0 \le s \le s_m$ and $t_0 \le t \le t_n$, then we get a four sided surface patch. Moreover, it is easy to see that the boundary curves of this rectangular patch are the Lagrange polynomial curves that interpolate the boundary control points.

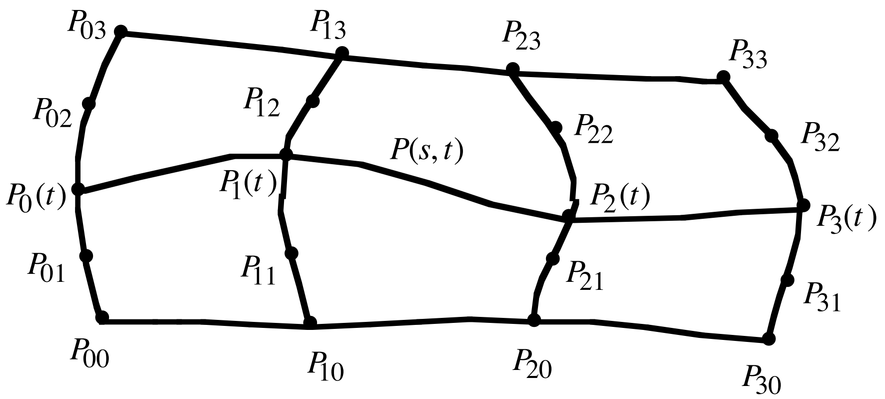
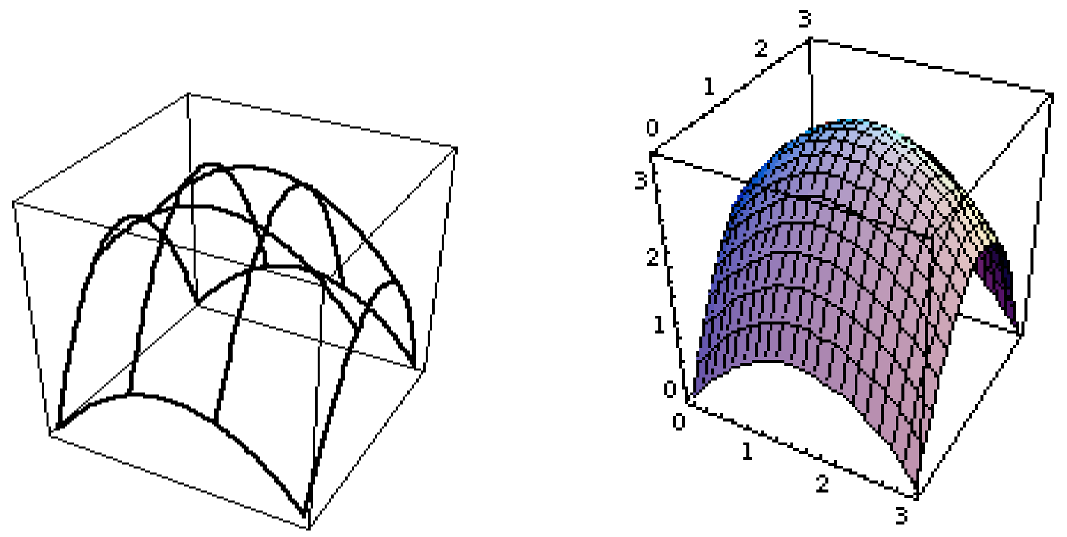


Figure 9: Schematic view of bicubic interpolation. The curve $P_k(t)$ interpolates the control points $P_{k0},\ldots,P_{k3}$, and the surface $P(s,t)$ interpolates the control curves $P_0(t),\ldots,P_3(t)$. The boundary curves are the interpolating curves of the boundary control points. See also Figure 10.



(a) Mesh of cubic space curves (b) Interpolating bicubic surface

Figure 10: Bicubic interpolation.

Equations (6.1) and (6.2) lead to a bivariate version of Neville's algorithm. First apply Neville's algorithm $m+1$ times to calculate points on each of the curves $P_0(t), ..., P_m(t)$; then apply Neville's algorithm one more time with $P_0(t), ..., P_m(t)$ as control points to compute $P(s,t)$ (see Figure 11).

$$\begin{tikzcd}
 & & & & P(s,t) & & & & \\
 & & P_{01}(s) \arrow[urr, "s_2 - s"] & & & & P_{12}(s) \arrow[ull, "s - s_0"'] & & \\
 & P_0(t) \arrow[ur, "s_1 - s"] & & & P_1(t) \arrow[ull, "s - s_0"'] \arrow[urr, "s_2 - s"] & & & P_2(t) \arrow[ul, "s - s_1"'] & \\
P_{01}(t) \arrow[ur, "t_2 - t"] & & P_{12}(t) \arrow[ul, "t - t_0"'] & P_{01}(t) \arrow[ur, "t_2 - t"] & & P_{12}(t) \arrow[ul, "t - t_0"'] & P_{01}(t) \arrow[ur, "t_2 - t"] & & P_{12}(t) \arrow[ul, "t - t_0"'] \\
P_{00} \arrow[u, "t_1 - t"] & P_{01} \arrow[ul, "t - t_0"'] \arrow[ur, "t_2 - t"] & P_{02} \arrow[u, "t - t_1"'] & P_{10} \arrow[u, "t_1 - t"] & P_{11} \arrow[ul, "t - t_0"'] \arrow[ur, "t_2 - t"] & P_{12} \arrow[u, "t - t_1"'] & P_{20} \arrow[u, "t_1 - t"] & P_{21} \arrow[ul, "t - t_0"'] \arrow[ur, "t_2 - t"] & P_{22} \arrow[u, "t - t_1"']
\end{tikzcd}$$

Figure 11: Neville's algorithm for a biquadratic patch. The three lower triangles in the two bottom tiers represent univariate interpolation in the t direction. The triangle at the top interpolates these results in the s direction.

7. Ruled Surfaces and Lofted Surfaces

So far we have performed only discrete interpolation; that is, we have developed curve and surface techniques to interpolate finite collections of control points. But if we replace the control points by curves, then essentially the same techniques can be applied to accomplish *transfinite interpolation* -- that is, interpolation of an infinite collection of points on a finite collection of curves.

Suppose, for example, that we are given two curves $U_0(s)$ and $U_1(s)$ and we require a surface to pass through these curves. We can perform linear interpolation on the curves to generate the surface

$$R(s,t) = (1-t)U_0(s) + tU_1(s) \; .$$

This expression is the same linear interpolation formula we first used to interpolate two points, only now the two points have been replaced by two curves. If we fix the value of $s = s^*$, then $R(s^*,t)$ is the line connecting the two points $U_0(s^*)$ and $U_1(s^*)$. Thus a line passes through each point on this surface. For this reason $R(s,t)$ is called a *ruled surface* (see Figure 12).

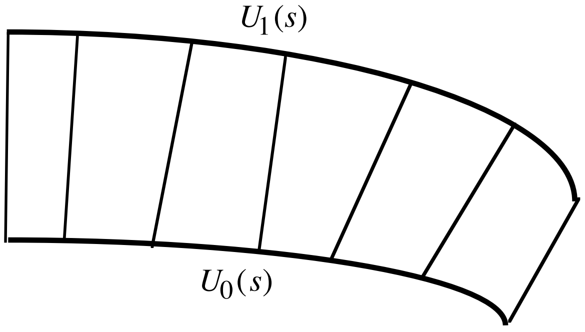


Figure 12: A ruled surface interpolating the curves $U_0(s)$ and $U_1(s)$.

When $U_1(s)$ is a translate of $U_0(s)$ -- that is, when $U_1(s) = U_0(s) + v$ -- then

$$R(s,t) = U_0(s) + tv$$

is called a *cylinder* over $U_0(s)$. When $U_1(s)$ collapses to a single point V, then

$$R(s,t) = (1-t)U_0(s) + tV$$

is called the *cone* over $U_0(s)$ with vertex V (see Figure 13).

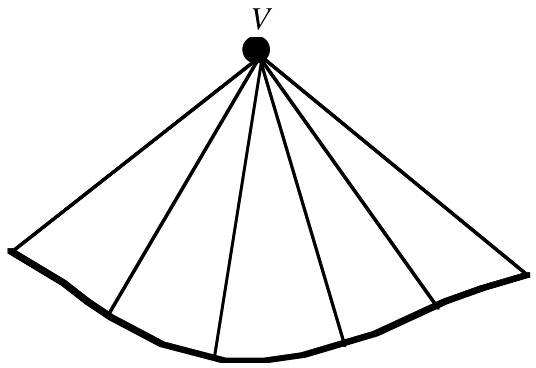


Figure 13: The cone over $U_0(s)$ with vertex *V*.

Lofting generalizes ruled surfaces by applying Lagrange interpolation to an arbitrary finite number of curves $U_0(s),...,U_n(s)$. For each fixed value of *s*, let $L_U(s,t)$ be the Lagrange interpolating curve to the control points $U_0(s),...,U_n(s)$ at the nodes $t_0,\ldots,t_n$ -- that is,

$$L_U(s,t_j) = U_j(s)$$

Then the curves $L_U(s,t)$ sweep out a surface that interpolates the curves $U_0(s),...,U_n(s)$ (see Figure 14). The surface $L_U(s,t)$ is called the *lofted surface* generated by the *rail curves* $U_0(s),..., U_n(s)$. Provided we have some procedure for computing points along the rails $U_0(s),..., U_n(s)$, we can compute points on this lofted surface using Neville's algorithm for interpolating curves: simply replace the control points $P_0,..., P_n$ by the rails $U_0(s),..., U_n(s)$ (see Figure 15).

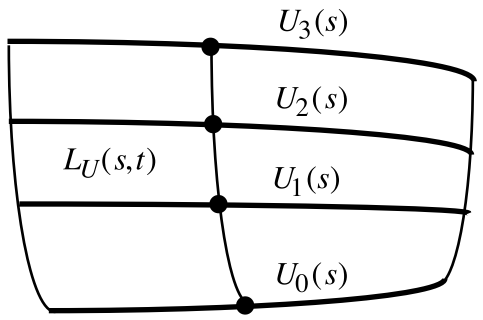


Figure 14: A lofted surface $L_U(s,t)$ interpolating the curves $U_0(s),..., U_3(s)$.

$$\begin{tikzcd}
 & & & L_U(s,t) = U_{0123}(s,t) & & & \\
 & & U_{012}(s,t) \arrow[ur, "t_3 - t"] & & U_{123}(s,t) \arrow[ul, "t - t_0"'] & & \\
 & U_{01}(s,t) \arrow[ur, "t_2 - t"] & & U_{12}(s,t) \arrow[ul, "t - t_0"'] \arrow[ur, "t_3 - t"] & & U_{23}(s,t) \arrow[ul, "t - t_1"'] & \\
U_0(s) \arrow[ur, "t_1 - t"] & & U_1(s) \arrow[ul, "t - t_0"'] \arrow[ur, "t_2 - t"] & & U_2(s) \arrow[ul, "t - t_1"'] \arrow[ur, "t_3 - t"] & & U_3(s) \arrow[ul, "t - t_2"']
\end{tikzcd}$$

Figure 15: Neville's algorithm for a lofted surface $L_U(s,t)$ interpolating the curves $U_0(s),..., U_3(s)$.

8. Summary

The key ideas in this lecture are linear interpolation coupled with a dynamic programming procedure (Neville's algorithm) for generating higher order interpolating curves and surfaces. Linear interpolation coupled with another dynamic programming procedure for generating approximating curves and surfaces will be central again in our next lecture, where we will study Bezier curves and the de Casteljau algorithm.

Exercises:

1. Complete the proof of Theorem 1 by showing that $P_{0\cdots n}(t_k) = P_k$.

2. Give an example to show that changing the nodes alters the interpolating curve $P_{0\cdots n}(t)$ even if we leave the control points fixed.

3. Let $P(t)$ be the Lagrange interpolating polynomial for the control points $P_0,..., P_n$ and nodes $t_0,...,t_n$. Form a new Lagrange interpolating curve $Q(t)$ by replacing each node t_k by the node $\tau_k = at_k + b$ for some fixed constants $a > 0$ and b. Show that changing all the nodes in this way has no affect on the shape of the interpolating curve. In particular, using Neville's algorithm, show that $Q(at+b) = P(t)$. What happens if we choose $a < 0$?

4. Implement Neville's algorithm. Experiment with curves of different degrees.
   a. How does changing the order of the control points without altering the order of the nodes affect the shape of the curve?
   b. How does changing the values of the nodes affect the shape of the curve?
   c. Place the nodes at the integers $0, 1, ..., n$, and graph the curves with control points at $P_k = (k, 0),\ k \neq j$, and $P_j = (j, 1)$.

5. Prove that the polynomials $1, (t-r), ..., (t-r)^n$ are linearly independent. Conclude that the polynomials $1, (t-r), ..., (t-r)^n$ form a basis for the polynomials of degree n and use this fact to provide an alternative proof of Taylor's theorem.

6. A polynomial $P(t)$ is said to have a root of multiplicity m at the parameter r if

   $P^{(k)}(r) = 0, \quad k = 0, ..., m-1.$

   a. Show that a polynomial $P(t)$ has a root of multiplicity m at r if and only if $(t-r)^m$ is a factor of $P(t)$.
   b. Show that every nonzero polynomial of degree n can have no more than n roots counting multiplicities.

7. Let $P(t)$ be a polynomial of degree n, and let $P_{0\cdots n}(t)$ be the polynomial that interpolates the control points $P(t_0), ..., P(t_n)$ at the nodes $t_0, ..., t_n$. Prove that $P_{0\cdots n}(t) = P(t)$.

8. Let $f(t)$ be a polynomial of degree n and let r be an arbitrary constant.
   a. Using long division of polynomials, show that there is a polynomial $g(t)$ of degree $n-1$ such that $f(t) = (t-r)g(t) + f(r)$.
   b. Using part a, conclude that $f(r) = 0 \Leftrightarrow t-r$ is a factor of $f(t)$.

9. Prove that the boundary curves of an interpolating tensor product patch are the Lagrange polynomials that interpolate the boundary control points.

10. Show that a lofted surface is equivalent to a tensor product surface if all the rails are polynomial curves.