ROCKY MOUNTAIN
JOURNAL OF MATHEMATICS
Volume 14, Number 1, Winter 1984

A TRANSFINITE C2 INTERPOLANT OVER TRIANGLES

PETER ALFELD AND ROBERT E. BARNHILL

ABSTRACT. A transfinite C^2 interpolant on a general triangle is created. The required data are essentially C^2, no compatibility conditions arise, and the precision set includes all polynomials of degree less than or equal to eight. The symbol manipulation language REDUCE is used to derive the scheme. The scheme is discretized to two different finite dimensional C^2 interpolants in an appendix.

1. Introduction and history. Scientists and engineers often take three-dimensional measurements through which they wish to pass a surface. When designing interactively the surface of a real object, designers input three-dimensional points. Because the geometric information for these two classes of problems can be located arbitrarily in three-dimensional space, the surface scheme must be able to handle arbitrarily located data. There are two broad classes of methods suitable for solving these problems (i.e., problems in which simplifying geometric assumptions cannot be made): (1) patch methods, and (2) point methods. "Patch methods" are those methods in which small curved pieces are joined together to form a smooth surface. "Point methods" are those methods in which information given only at discrete points is used to construct a surface.

This paper and its appendix introduce new patch methods which have the following properties: (1) the data may be arbitrarily located, and (2) the interpolating surface is twice continuously differentiable (C^2). We have divided the development of our new schemes into two parts. This paper is Part 1 and the accompanying appendix by Alfeld is Part 2.

(a) In Part 1, we develop schemes of interpolation to *curves* of information defined over triangles. (These are called *transfinite* interpolants because entire curves of information are interpolated.)

(b) In Part 2, we discretize these transfinite interpolants to obtain finite dimensional patches (i.e., patches which depend on only finitely many data). A reason for developing transfinite patches per se is that there is a unified theory of the interpolation properties, polynomial precision, and

This research was supported in part by The National Science Foundation with Grant MCS-8101854 and by The Department of Energy with Contract DE-AC02-82ER12046. A000 to The University of Utah.

smoothness for them. These properties can also be traced relatively easily through the discretization of the transfinite schemes, so that the interpolation, precision, and smoothness of the final result can be determined.

When would a user want a C^2 scheme? An example comes from the autombile industry, namely, feature lines. The eye can detect a discontinuity in curvature, so a C^2 surface is esthetically necessary.

2. Barnhill, Birkhoff, and Gordon triangular interpolants. We derive a scheme that interpolates to position, first, and second derivatives on the boundary of a general triangle. When this scheme is applied piecewise to each triangle of a triangulation, the resulting surface is twice continuously differentiable. In order to make the formulas easier to express, we assume that the data come from an underlying "primitive" function F. However, this is strictly a notational convenience; the data in a practical problem are "wire frame data" consisting of curves and first and second cross-boundary derivatives. (First cross-boundary derivatives are "ribbons" tangent to the given curves and second cross-boundary derivatives are "osculating ribbons".)

We assume, then, the existence of an underlying primitive function F whose gradient and Hessian exist, and are continuous. In fact, we have to use the values of higher derivatives of F at certain points. We need not assume that any mixed partial derivatives of order greater than two commute.

The approach chosen here follows that of Barnhill, Birkhoff, and Gordon [1]. To describe it, some notation is needed. Consider a general triangle, denoted by T, with vertices V_1, V_2, V_3, labeled counterclockwise.

For $i = 1, 2, 3$, the edge of T opposite the vertex V_i is denoted by e_i, i.e., $e_1 = V_3 - V_2$, $e_2 = V_1 - V_3$, $e_3 = V_2 - V_1$. It is obvious that

$$e_1 + e_2 + e_3 = 0. \tag{2.1}$$

For any function f, and any direction $e \in \mathbf{R}^2$, we consider the Gâteaux derivative, which is defined by

$$\frac{\partial f}{\partial e}(V) = \frac{d}{dt} f(V + te)\Big|_{t=0}.$$

If the gradient ∇f exists and is continuous, then

$$\frac{\partial f}{\partial e} = \nabla f \circ e. \tag{2.2}$$

In particular, we will use derivatives in the direction of the edges of T. A convenient notation is given by

$$f_i(V) = \frac{\partial f}{\partial e_i}(V), \quad i = 1, 2, 3$$

and

$$f_{ij}(V) = \frac{\partial^2 f}{\partial e_j \partial e_i}(V), \quad i, j = 1, 2, 3,$$

etc. (Note the reversal in the sequence of subscripts for higher order Gâteaux derivatives.) If the gradient of f exists and is continuous, then by virtue of (2.1) and (2.2) we obtain

$$f_1 + f_2 + f_3 = 0. \tag{2.3}$$

Similarly, if the Hessian H of f exists and is symmetric and continuous, then it follows from two applications of (2.3) that

$$f_{ii} = f_{jj} + 2f_{jk} + f_{kk} \tag{2.4}$$

whenever $\{i, j, k\} = \{1, 2, 3\}$.

Any point V in the plane can be expressed uniquely in terms of its barycentric coordinates b_1, b_2, b_3, as follows: $V = \sum_{i=1}^{3} b_i V_i$, $\sum_{i=1}^{3} b_i = 1$. The b_i, $i = 1, 2, 3$, are linear functions of the cartesian coordinates of V. It is convenient to use barycentric coordinates exclusively, although Cartesian coordinates are present implicitly. Table 1 contains the directional derivatives of the barycentric coordinates in the directions e_i, $i = 1, 2, 3$. For a more detailed introduction to barycentric coordinates and their properties see [**2**].

$j\backslash i$	1	2	3
1	0	-1	$+1$
2	$+1$	0	-1
3	-1	$+1$	0

TABLE 1. Directional derivatives $\partial b_i/\partial e_j$.

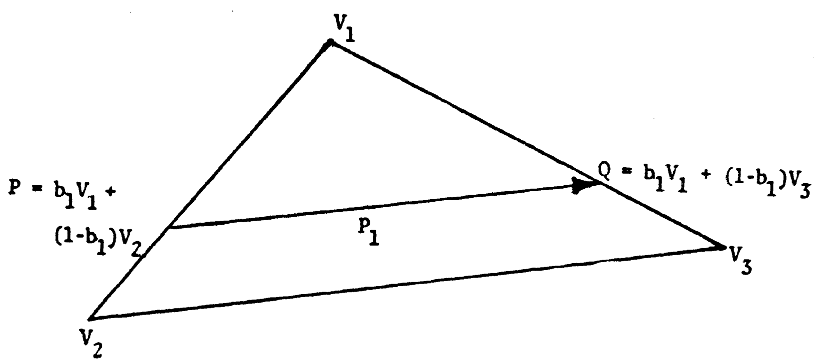

FIGURE 1. Geometry for BBG P_1F.

We use basic interpolation operators P_i which interpolate to position, and to first and second directional derivatives (in the direction of e_i), on edges e_j and e_k along lines parallel to e_i ($\{i, j, k\} = \{1, 2, 3\}$). For example, P_1 interpolates to position and derivatives at points P and Q in Figure 1.

The operator P_1 is defined formally by

$$
\begin{aligned}
(P_1F)\Big(\sum_{i=1}^{3} b_iV_i\Big) \\
= h_0(s_1)F(P) + h_1(s_1)F(Q) \\
+ \bar{h}_0(s_1)(1-b_1)F_1(P) + \bar{h}_1(s_1)(1-b_1)F_1(Q) \\
+ \bar{\bar{h}}_0(s_1)(1-b_1)^2F_{11}(P) + \bar{\bar{h}}_1(s_1)(1-b_1)^2F_{11}(Q)
\end{aligned}
\tag{2.5}
$$

where $s_1 = b_3/(1 - b_1)$, $P = b_1V_1 + (1 - b_1)V_2$ on e_3, $Q = b_1V_1 + (1 - b_1)V_3$ on e_2 (Cf. Figure 1).

The h_i, $\bar{h}_i$ and $\bar{\bar{h}}_i$ are quintic polynomials uniquely defined by the cardinal properties:

$$
\begin{array}{lll}
h_i(j) = \delta_{ij} & h_i'(j) = 0 & h_i''(j) = 0 \\
\bar{h}_i(j) = 0 & \bar{h}_i'(j) = \delta_{ij} & \bar{h}_i''(j) = 0 \\
\bar{\bar{h}}_i(j) = 0 & \bar{\bar{h}}_i'(j) = 0 & \bar{\bar{h}}_i''(j) = \delta_{ij}
\end{array}
\tag{2.6}
$$

for $i, j = 0, 1$, δ_{ij} being the Kronecker Delta.

Explicit expressions for the cardinal functions are not needed for the theoretical development. However, they are tabulated in §5.3.

The projectors P_2 and P_3 are defined similarly; for details see §4. It is easily verified (using Table 1) that

$$
\frac{\partial^i F}{\partial e_1^i}(P) = \frac{\partial^i P_1 F}{\partial e_1^i}(P) \quad \text{and} \quad \frac{\partial^i F}{\partial e_1^i}(Q) = \frac{\partial^i P_1 F}{\partial e_1^i}(Q)
$$

for $i = 0, 1, 2$, and that P_2 and P_3 have similar properties.

Barnhill, Birkhoff, and Gordon [1] used elementary projectors similar to the above that interpolate to position and first directional derivatives only. They computed the Boolean sum of all three operators and obtained a transfinite C^1 BBG interpolation scheme. In this paper, we construct the Boolean sum $Q := P_3 \oplus P_2 \oplus P_1$. (The Boolean sum of any two operators S, T is defined by $S \oplus T = S + T - ST$).

It turns out that Q does solve the interpolation problem. J. A Gregory has shown that triple Boolean sums of BBG projectors have no compatibility problems.

The algebraic manipulations are very tedious and were carried out using the symbol manipulation language REDUCE [3]. In the following §3, we describe a differentiation rule that is central to the automatic computation of Q. §4 contains the documentation of the computation of Q and a listing of Q. In §5, data requirements, the lack of compatibility conditions, and the precision of the scheme are discussed.

3. The central differentiation rule. Computation of the Boolean sum $Q = P_3 \oplus P_2 \oplus P_1$ involves the composition of some of the elementary projectors P_1, P_2, P_3, and hence the differentiation of functions restricted to edges, in the direction of other edges. In this section, we first study an example illustrating that concept, and establish a general pattern of differentiation. We then state and prove a general rule that covers all relevant cases, and that is central to the symbol manipulation approach. The proof partly follows the pattern established by the example.

EXAMPLE. Consider the problem of computing

$$\frac{\partial}{\partial e_1}[f(b_2V_2 + (1 - b_2)V_1)]$$

(i.e., we are differentiating on e_3 in the direction of e_1).

The function to be differentiated is composed thus: $V = \sum_{i=1}^{3} b_i V_i$, $z(V) := f(b_2V_2 + (1 - b_2)V_1) = f(g(\xi_2(V)))$ where $\xi_2(\sum_{i=1}^{3} b_i V_i) := b_2$, $g(b_2) := b_2V_2 + (1 - b_2)V_1$. Now consider

$$\frac{\partial}{\partial e_1} z(V) = \lim_{t \to 0} \frac{z(V + te_1) - z(V)}{t}.$$

Note that, since $e_1 = V_3 - V_2$,

$$\begin{aligned} z(V + te_1) &= f(g(\xi_2(b_1V_1 + (b_2 - t)V_2 + (b_3 + t)V_3))) \\ &= f(g(b_2 - t)) \\ &= f((b_2 - t)V_2 + (1 - b_2 + t)V_1) \\ &= f(b_2V_2 + (1 - b_2)V_1 - te_3). \end{aligned}$$

Similarly $z(V) = f(b_2V_2 + (1 - b_2)V_1)$ and hence

$$\frac{\partial}{\partial e_1} z(V) = \lim_{t \to 0} \frac{f(b_2V_2 + (1 - b_2)V_1 - te_3) - f(b_2V_2 + (1 - b_2)V_1)}{t}.$$

Thus:

$$\frac{\partial}{\partial e_1}[f(b_2V_2 + (1 - b_2)V_1)] = -\frac{\partial f}{\partial e_3}(b_2V_2 + (1 - b_2)V_1).$$

In this derivation, we did not have to assume that the gradient of f exists. We now state the Central Differentiation Rule.

CENTRAL DIFFERENTIATION RULE. *Assume that all first order directional derivatives of f exist. Then, for all $a_i \in \{0, 1, b_i, 1 - b_j (j \neq i)\}$, $i = 1, 2, 3$, such that $\sum_{i=1}^{3} a_i = 1$, and all $e \in \{e_1, e_2, e_3\}$, the following is true:*

(3.1) $$\frac{\partial}{\partial e}[f(\sum_{i=1}^{3} a_i V_i)] = -\sum_{j=1}^{3}\left(1-\left(\frac{\partial a_j}{\partial e}\right)^2\right)\frac{\partial a_{j+1}}{\partial e}\frac{\partial f}{\partial e_j}(\sum_{i=1}^{3} a_i V_i)$$

where $a_4 := a_1$.

PROOF. The proof proceeds by considering all possible cases.

Case 1. ($\sum_{i=1}^{3} a_i V_i$ is a general point in $\mathbf{R}^2$) $a_i = b_i$ for $i = 1, 2, 3$, $e = e_k$ for some $k \in \{1, 2, 3\}$. By Table 1, since $\partial b_j/\partial e_k = \pm 1$ for $j \neq k$, all but the k-th term in the right hand side of (3.1) vanish, which yields (with $b_4 := b_1$)

$$\frac{\partial}{\partial e_k}[f(\sum_{i=1}^{3} b_i V_i)] = -(1-0^2)\frac{\partial b_{k+1}}{\partial e_k}\frac{\partial f}{\partial e_k}(\sum_{i=1}^{3} b_i V_i)$$
$$= \frac{\partial f}{\partial e_k}(\sum_{i=1}^{3} b_i V_i)$$

since $\partial b_{k+1}/\partial e_k = -1$.

Case 2. ($\sum_{i=1}^{3} a_i V_i$ is a vertex of T) $a_i = a_\ell = 0$, $a_k = 1$, $\{i, \ell, k\} = \{1, 2, 3\}$. The derivative of $F(\sum_{j=1}^{3} a_j V_j)$ should be zero since $\sum_{j=1}^{3} a_j V_j = V_k$ is constant. The right hand side of (3.1) does yield zero since $\partial a_{j+1}/\partial e = 0$. for all j.

Case 3. ($\sum_{i=1}^{3} a_i V_i$ lies on edge e_k of T) $a_k = 0$, $a_\ell = 1 - b_m$, $a_m = b_m$, $\{k, \ell, m\} = \{1, 2, 3\}$ $e = e_\mu$, $\mu \in \{1, 2, 3\}$.

Case 3.1. $\mu = m$. Since b_m is constant in the direction of e_m, the left hand side of (3.1) is 0. The right hand side does yield zero since $\partial a_{j+1}/\partial e_m = 0$ for all $j = 1, 2, 3$.

Case 3.2. $\mu \neq m$. Arguing as in the above example we see that

$$\frac{\partial f(b_m V_m + (1-b_m)V_\ell)}{\partial e_\mu} = s(\mu, k)\frac{\partial f}{\partial e_k}(b_m V_m + (1-b_m)V_\ell)$$

where

$$s(\mu, k) = \begin{cases} -1 & \text{if } \mu \neq k, \\ +1 & \text{if } \mu = k. \end{cases}$$

We now turn to the right hand side of (3.1). Since $\mu \neq m$, the first term in each product vanishes whenever $j \neq k$. Since $\partial a_{k+1}/\partial e_\mu = \pm 1$ we obtain the correct expression, possibly with the wrong sign. To verify that the sign generated by (3.1) is in fact correct we use Table 2. There, the last column lists the correct sign $s(\mu, k)$ of the derivative, and the next to last column (sgn (RHS)) gives the sign generated by the right hand side of (3.1). All possible cases are covered and the signs always agree. This completes the proof of the Central Differentiation Rule.

k	ℓ	m	μ	a_{k+1}	$\operatorname{sgn}\frac{\partial a_{k+1}}{\partial e_\mu}$	sgn (RHS)	$s(\ell, m)$
1	2	3	1 2	$1 - b_3$	-1 $+1$	$+1$ -1	$+1$ -1
1	3	2	1 3	b_2	-1 $+1$	$+1$ -1	$+1$ -1
2	1	3	1 2	b_3	$+1$ -1	-1 $+1$	-1 -1
2	3	1	2 3	$1-b_1$	-1 $+1$	$+1$ -1	$+1$ -1
3	1	2	1 3	$1-b_2$	$+1$ -1	-1 $+1$	-1 $+1$
3	2	1	2 3	b_1	$+1$ -1	-1 $+1$	-1 $+1$

TABLE 2. Proof of Central Differentiation Rule.

NOTE. In the above proof we nowhere required that the gradient of F exists. All that is needed is that the directional derivatives occuring in the formula (3.1) exist.

4. Computation of the triple Boolean sum $\mathbf{Q} = \mathbf{P}_3 \oplus \mathbf{P}_2 \oplus \mathbf{P}_1$. This section is a documented listing of the REDUCE program that computes the C^2 interpolant. (The REDUCE code can be obtained by sending us a blank tape and indicating the desired tape parameters.) For a description of the REDUCE language see [3].

The purpose of listing the source code is twofold: firstly, it enables the reader to verify and reproduce the results described here, and secondly, it illustrates the simplicity and the potential of symbol manipulation in the derivation of complicated interpolation schemes.

In reading the program, some familiarity with REDUCE would be useful, but is not essential. The code is largely self-explanatory, and surprisingly simple.

At the end of this section, there is also a machine produced listing of the interpolant, in a notation close to that employed in hand work.

THE REDUCE PROGRAM (TABLE 3). Initially, we ignore output and formatting statements.

—*Declaring basic functions* (*lines 5–10*). The OPERATOR declaration (lines 5–8) instructs REDUCE that the listed identifiers denote functions. The exclamation mark in an identifier indicates that the following non-alphanumerical character is part of the identifier. The correspondence between identifiers and the notation employed in this paper is fairly obvious. The identifiers in line 5–7 denote the cardinal functions (B stands for

$^-$ and s for $^=$; !$'$ and !$''$ denote first and second derivatives respectively); $B1$, $B2$, and $B3$ are the barycentric coordinates, and F is the primitive function.

The edges e_i, $i = 1, 2, 3$, are denoted by $E1$, $E2$, $E3$. A major deviation from the true context is that within the REDUCE program the barycentric coordinates are considered functions of the scalar variables $E1, E2, E3$. Thus directional derivatives are interpreted as partial derivatives which can be handled easily in REDUCE. This interpretation is possible because all relevant rules for partial and directional derivatives are formally identical.

The DEFINE statement in line 10 instructs REDUCE to replace on input $C1$ by $B1$ $(E1, E2, E3)$, etc.

—*Defining the cardinal properties* (*lines 24–32*). Lines 24–32 contain a list of the cardinal properties (2.6). The LET statement differs from the DEFINE statement in that substitutions are carried out during computation rather than on input.

—*Defining derivatives* (*lines 34–49*). In lines 34–39 the relations between the identifiers for the cardinal functions and their derivatives are defined, lines 40–44 contain Tab e 1, and the Central Differentiation Rule is introduced in lines 46–49. DF is the differentiation operator built into REDUCE.

The first argument N of the function F indicates the derivative of the primitive function f that is being denoted. The integer N has digits 1, 2, 3 which denote the directions in which derivatives have been taken, the right most indicating the most recent derivative, etc. The last three arguments are the barycentric coordinates. Thus we have for example the correspondences

$$F(0, A, B, C) \leftrightarrow F(AV_1 + BV_2 + CV_3)$$

$$F(12, 0, 1, 0) \leftrightarrow \frac{\delta^2 F}{\delta e_2 \partial e_1}(V_2).$$

—*Defining the basic projectors* (*lines 51–76*). The notation is self-explanatory. Lines 51–58 correspond to (2.5), and lines 60–76 define the projectors P_2 and P_3, all applied to the primitive function f. If the projectors are applied to other functions, the definition has to be rewritten, with $f(\ )$ replaced by suitable expressions.

—*Computing* $P_2 \oplus P_1$ (*lines 78–85*). Again, the notation it self-explanatory. The formula for the Boolean Sum employed here is $P_2 \oplus P_1 = P_2 + P_1 - P_2P_1$. The equivalent formula $P_2 \oplus P_1 = P_1 + P_2(I - P_1)$. (where I is the identity operator) which is more convenient for hand-work, yields identical results. Note that P_2 applied to P_1 $(P2P1)$ is computed by rewriting the definition of P_2 with P_1 replacing f.

—*Computation of* $P_3 \oplus P_2 \oplus P_1$ *(lines 91–101).* Similar remarks as for the computation of $P_2 \oplus P_1$ apply.

—*Incorporating the assumptions of continuous gradient and Hessian (lines 103–111).* Up to this stage, the only assumptions that have been made are that the required directional derivatives exist. The resulting expression is now simplified in a post processing stage. To facilitate automatic cancellation of terms, all derivatives (up to second order) in the direction e_3 are replaced by combinations of derivatives in the directions e_1 and e_2, using (2.3) and (2.4) (lines 103–110). In line 111, F_{12} and F_{21} are equated. Notice that no assumptions are incorporated about the commutation of mixed directional derivatives of order higher than 2.

—*Output (lines 1, 3, 12–22, 89, 112–121).* Lines 3, 12–22, and 116–118 contain formatting statements. In the form given, the program generates the following output files:

CMPI: contains the listing of the first stage of computation.

P2BP1: contains $P_2 \oplus P_1$ in REDUCE readable form, not incorporating the assumptions on continuous gradient and Hessian.

CMP2: contains the history of the second stage of computation.

INTP.RED: contains $Q = P_3 \oplus P_2 \oplus P_1$ in REDUCE readable form. This is useful for further processing, such as investigations of compatibility and precision.

INTP.HMN: contains Q in a different notation. The first argument of F denotes the derivative as before, but the second is V in the form $V = \sum_{i=1}^{3} a_i V_i$. This notation is closer to that commonly employed in hand work, but cannot be processed further in REDUCE without introducing additional internal notation.

```
OUT CMP1;

OFF EXP;OFF NAT;

OPERATOR        H0,H1,H0B,H1B,H0S,H1S,
                H0!',H1!',H0B!',H1B!',H0S!',H1S!',
                H0!",H1!",H0B!",H1B!",H0S!",H1S!",
                B1,B2,B3,F;

DEFINE C1=B1(E1,E2,E3),C2=B2(E1,E2,E3),C3=B3(E1,E2,E3);

FACTOR  H0,H1,H0B,H1B,H0S,H1S,
        H0!',H1!',H0B!',H1B!',H0S!',H1S!',
        H0!",H1!",H0B!",H1B!",H0S!",H1S!";

ORDER   H0,H1,H0B,H1B,H0S,H1S,
        H0!',H1!',H0B!',H1B!',H0S!',H1S!',
        H0!",H1!",H0B!",H1B!",H0S!",H1S!";

FACTOR V1,V2,V3;

ORDER V1,V2,V3;

LET     H0(0)=1,       H0(1)=0,       H1(0)=0,       H1(1)=1,
        H0!'(0)=0,     H0!'(1)=0,     H1!'(0)=0,     H1!'(1)=0,
        H0!"(0)=0,     H0!"(1)=0,     H1!"(0)=0,     H1!"(1)=0,
        H0B(0)=0,      H0B(1)=0,      H1B(0)=0,      H1B(1)=0,
        H0B!'(0)=1,    H0B!'(1)=0,    H1B!'(0)=0,    H1B!'(1)=1,
        H0B!"(0)=0,    H0B!"(1)=0,    H1B!"(0)=0,    H1B!"(1)=0,
        H0S(0)=0,      H0S(1)=0,      H1S(0)=0,      H1S(1)=0,
        H0S!'(0)=0,    H0S!'(1)=0,    H1S!'(0)=0,    H1S!'(1)=0,
        H0S!"(0)=1,    H0S!"(1)=0,    H1S!"(0)=0,    H1S!"(1)=1;

FOR ALL X LET   DF(H0(X),X)=H0!'(X),      DF(H1(X),X)=H1!'(X),
                DF(H0B(X),X)=H0B!'(X),    DF(H1B(X),X)=H1B!'(X),
                DF(H0S(X),X)=H0S!'(X),    DF(H1S(X),X)=H1S!'(X),
                DF(H0!'(X),X)=H0!"(X),    DF(H1!'(X),X)=H1!"(X),
                DF(H0B!'(X),X)=H0B!"(X),DF(H1B!'(X),X)=H1B!"(X),
                DF(H0S!'(X),X)=H0S!"(X),DF(H1S!'(X),X)=H1S!"(X);

FOR ALL A,B,C LET

DF(B1(A,B,C),A)=0,DF(B1(A,B,C),B)=1,DF(B1(A,B,C),C)=-1,
DF(B2(A,B,C),A)=-1,DF(B2(A,B,C),B)=0,DF(B2(A,B,C),C)=1,
DF(B3(A,B,C),A)=1,DF(B3(A,B,C),B)=-1,DF(B3(A,B,C),C)=0;

FOR ALL A1,A2,A3,N,E LET DF(F(N,A1,A2,A3),E) =
        -(1-DF(A1,E)**2)        *DF(A2,E)       *F(10*N+1,A1,A2,A3)
        -(1-DF(A2,E)**2)        *DF(A3,E)       *F(10*N+2,A1,A2,A3)
        -(1-DF(A3,E)**2)        *DF(A1,E)       *F(10*N+3,A1,A2,A3);

S1:=C3/(1-C1);

P1:=    H0(S1)                  *F(0,C1,1-C1,0)
+       H1(S1)                  *F(0,C1,0,1-C1)
+       H0B(S1) *(1-C1)         *F(1,C1,1-C1,0)
+       H1B(S1) *(1-C1)         *F(1,C1,0,1-C1)
+       H0S(S1) *(1-C1)**2      *F(11,C1,1-C1,0)
+       H1S(S1) *(1-C1)**2      *F(11,C1,0,1-C1);

S2:=C1/(1-C2);

P2:=    H0(S2)                  *F(0,0,C2,1-C2)
+       H1(S2)                  *F(0,1-C2,C2,0)
+       H0B(S2) *(1-C2)         *F(2,0,C2,1-C2)
+       H1B(S2) *(1-C2)         *F(2,1-C2,C2,0)
+       H0S(S2) *(1-C2)**2      *F(22,0,C2,1-C2)
+       H1S(S2) *(1-C2)**2      *F(22,1-C2,C2,0);

S3:=C2/(1-C3);

P3:=    H0(S3)                  *F(0,1-C3,0,C3)
+       H1(S3)                  *F(0,0,1-C3,C3)
+       H0B(S3) *(1-C3)         *F(3,1-C3,0,C3)
+       H1B(S3) *(1-C3)         *F(3,0,1-C3,C3)
+       H0S(S3) *(1-C3)**2      *F(33,1-C3,0,C3)
+       H1S(S3) *(1-C3)**2      *F(33,0,1-C3,C3);

D2P1:=DF(P1,E2);            D22P1:=DF(D2P1,E2);

P2P1:=  H0(S2)                  *SUB(C1=0,C3=1-C2,P1)
+       H1(S2)                  *SUB(C1=1-C2,C3=0,P1)
+       H0B(S2) *(1-C2)         *SUB(C1=0,C3=1-C2,D2P1)
```

```
+       H1B(S2) *(1-C2)          *SUB(C1=1-C2,C3=0,D2P1)
+       H0S(S2) *(1-C2)**2       *SUB(C1=0,C3=1-C2,D22P1)
+       H1S(S2) *(1-C2)**2       *SUB(C1=1-C2,C3=0,D22P1);

P2BP1:=P1+P2-P2P1;

SHUT CMP1;OUT P2BP1;P2BP1:=P2BP1;SHUT P2BP1;OUT CMP2;

D3P2BP1:=DF(P2BP1,E3);  D33P2BP1:=DF(D3P2BP1,E3);

P3P2BP1:=
        H0(S3)                   *SUB(C1=1-C3,C2=0,P2BP1)
+       H1(S3)                   *SUB(C1=0,C2=1-C3,P2BP1)
+       H0B(S3) *(1-C3)          *SUB(C1=1-C3,C2=0,D3P2BP1)
+       H1B(S3) *(1-C3)          *SUB(C1=0,C2=1-C3,D3P2BP1)
+       H0S(S3) *(1-C3)**2       *SUB(C1=1-C3,C2=0,D33P2BP1)
+       H1S(S3) *(1-C3)**2       *SUB(C1=0,C2=1-C3,D33P2BP1);

P3BP2BP1:=P3+P2BP1-P3P2BP1;

FOR ALL A,B,C LET
        F(3,A,B,C)      = -F(1,A,B,C)-F(2,A,B,C),
        F(13,A,B,C)     = -F(11,A,B,C)-F(12,A,B,C),
        F(23,A,B,C)     = -F(21,A,B,C)-F(22,A,B,C),
        F(31,A,B,C)     = -F(11,A,B,C)-F(21,A,B,C),
        F(32,A,B,C)     = -F(12,A,B,C)-F(22,A,B,C),
        F(33,A,B,C)     =  F(11,A,B,C)+F(12,A,B,C)
                                  +F(21,A,B,C)+F(22,A,B,C),
        F(21,A,B,C)     =  F(12,A,B,C);

SHUT CMP2;

OUT INTP.RED;INTP:=P3BP2BP1;SHUT INTP.RED;

LET C1=B1,C2=B2,C3=B3;

FOR ALL A,B,C,N LET F(N,A,B,C)=F(N,A*V1+B*V2+C*V3);

OUT INTP.HMN;ON NAT;INTP:=P3BP2BP1;SHUT INTP.HMN;
```

TABLE 3. REDUCE SOURCE CODE.

```
INTP :=  - ((B3 - 1)*(B3*F(2,V1)*H1B'( - B3 + 1) - (F(11,V1) - F(22,

                 V1) + 2*F(13,V1) + F(33,V1))*(B3*H1S'( - B3 + 1) +

                 2*H1S( - B3 + 1)) + H0S( - B3 + 1)*(F(221,V3) - F(

                 122,V3)) - H1B( - B3 + 1)*F(23,V1) + H1B( - B3 + 1)

               *(F(2,V1) - F(13,V1) - F(33,V1)) - H0B( - B3 + 1)*(F(

                 12,V3) - F(21,V3)) + H1S( - B3 + 1)*(2*F(133,V1) +

                 F(333,V1) - F(223,V1) + F(113,V1)) + F(1,B3*V3 - V1

               *(B3 - 1)) + F(2,B3*V3 - V1*(B3 - 1)) + (F(1,V1) + F(3

                 ,V1))*(B3*H1B'( - B3 + 1) + H1B( - B3 + 1)))*H0B((

                - B2)/(B3 - 1)) + (B3 - 1)*(H0B(( - B2)/(B3 - 1))*F(3

               ,B3*V3 - V1*(B3 - 1)) + H1B(( - B2)/(B3 - 1))*(F(1,B3*

                 V3 - V2*(B3 - 1)) + F(2,B3*V3 - V2*(B3 - 1))) + H1B

                                                               2
               (( - B2)/(B3 - 1))*F(3,B3*V3 - V2*(B3 - 1))) + (B3 *((

                   F(1,V1) + F(3,V1) + F(2,V1))*H1B"( - B3 + 1) -

                 H1S"( - B3 + 1)*(F(11,V1) - F(22,V1) + 2*F(13,V1)

                   + F(33,V1))) - 2*B3*(F(11,V1)*H1S'( - B3 + 1)

                  - F(22,V1)*H1S'( - B3 + 1) + F(13,V1)*(H1B'( - B3

                   + 1) + 2*H1S'( - B3 + 1)) + F(33,V1)*(H1B'( -

                   B3 + 1) + H1S'( - B3 + 1)) + H0S'( - B3 + 1)*F(

                 122,V3) - 2*F(133,V1)*H1S'( - B3 + 1) - F(333,V1)*

                 H1S'( - B3 + 1) + F(12,V3)*H0B'( - B3 + 1) - F(21,
```

```
    V3)*HOB'( - B3 + 1) + F(23,V1)*H1B'( - B3 + 1) +

    H1S'( - B3 + 1)*F(223,V1) - H1S'( - B3 + 1)*F(113,

    V1)) - 2*F(11,V1)*H1S( - B3 + 1) + 2*F(22,V1)*H1S(

  - B3 + 1) - 4*F(13,V1)*H1S( - B3 + 1) - 2*F(33,V1)*

 H1S( - B3 + 1) + F(22,B3*V3 - V1*(B3 - 1)) + 2*F(221,

 V3)*(B3*H0S'( - B3 + 1) + 2*H0S( - B3 + 1)) - H0S( -

 B3 + 1)*(4*F(122,V3) - F(2211,V3) + F(1122,V3)) - (2*F

    (13,V1) + 2*F(33,V1) - F(133,V1) - F(333,V1))*H1B(

  - B3 + 1) - 2*H1B( - B3 + 1)*F(23,V1) + H1B( - B3 + 1

 )*F(233,V1) - (2*F(12,V3) + F(112,V3))*HOB( - B3 + 1)

  + 2*HOB( - B3 + 1)*F(21,V3) + HOB( - B3 + 1)*F(211,V3

 ) + F(11,B3*V3 - V1*(B3 - 1)) + F(2233,V1)*H1S( - B3

  + 1) - 2*H1S( - B3 + 1)*F(1333,V1) - H1S( - B3 + 1)*F

 (3333,V1) - H1S( - B3 + 1)*F(1133,V1) + 4*H1S( - B3 +

 1)*(2*F(133,V1) + F(333,V1) - F(223,V1) + F(113,V1))

  + 2*F(12,B3*V3 - V1*(B3 - 1)))*H0S(( - B2)/(B3 - 1))*

              2                                     2
( - B3 + 1)   - H0S(( - B2)/(B3 - 1))*( - B3 + 1) *F(33,B3

                                  2
*V3 - V1*(B3 - 1)) - ( - B3 + 1) *H1S(( - B2)/(B3 - 1))*(

 F(33,B3*V3 - V2*(B3 - 1)) - F(22,B3*V3 - V2*(B3 - 1))

  - 2*F(21,B3*V3 - V2*(B3 - 1)) - F(11,B3*V3 - V2*(B3

     - 1))) + (B2 - 1)*(F(1,B2*V2 - V1*(B2 - 1)) + F(3,

 B2*V2 - V1*(B2 - 1)))*H1B(( - B1)/(B2 - 1)) + (B2 - 1)

*(H1B(( - B1)/(B2 - 1))*F(2,B2*V2 - V1*(B2 - 1)) + HOB((

     - B1)/(B2 - 1))*(B2*(F(1,V2)*HOB'( - B2 + 1) + F(1,

        V3)*H1B'( - B2 + 1) + F(0,V2)*H0'( - B2 + 1) + F(

        0,V3)*H1'( - B2 + 1) + F(11,V2)*H0S'( - B2 + 1)

         + F(11,V3)*H1S'( - B2 + 1)) + HOB( - B2 + 1)*(F(

        1,V2) + F(13,V2)) - H1B( - B2 + 1)*(F(12,V3) - F(

        1,V3)) + H0( - B2 + 1)*F(3,V2) - H1( - B2 + 1)*F(

    2,V3) + H0S( - B2 + 1)*(2*F(11,V2) + F(113,V2)) -

    H1S( - B2 + 1)*(F(112,V3) - 2*F(11,V3)) + F(2,B2*V2

     - V3*(B2 - 1)))) - (F(0,B2*V2 - V3*(B2 - 1)) - HOB(

  - B2 + 1)*F(1,V2) - H1B( - B2 + 1)*F(1,V3) - H0( - B2

  + 1)*F(0,V2) - H1( - B2 + 1)*F(0,V3) - H0S( - B2 + 1)

 *F(11,V2) - H1S( - B2 + 1)*F(11,V3))*H0(( - B1)/(B2 -

        2
 1)) + (B2 *(F(1,V2)*HOB"( - B2 + 1) + F(1,V3)*H1B"( -

     B2 + 1) + F(0,V2)*H0"( - B2 + 1) + F(0,V3)*H1"( -

     B2 + 1) + F(11,V2)*H0S"( - B2 + 1) + F(11,V3)*H1S"(

      - B2 + 1)) - 2*B2*(F(12,V3)*H1B'( - B2 + 1) + F(

     112,V3)*H1S'( - B2 + 1) + F(0,V2)*H0'( - B2 + 1) +

     F(0,V3)*H1'( - B2 + 1) - F(11,V2)*H0S'( - B2 + 1)

      - F(11,V3)*H1S'( - B2 + 1) - H0'( - B2 + 1)*F(3,V2
```

```
    ) + H1'( - B2 + 1)*F(2,V3) - HOS'( - B2 + 1)*F(113,
    V2) - F(13,V2)*HOB'( - B2 + 1)) + F(122,V3)*H1B( -
  B2 + 1) - 2*F(12,V3)*H1B( - B2 + 1) + 2*HOB( - B2 + 1)
  *F(13,V2) + HOB( - B2 + 1)*F(133,V2) + HO( - B2 + 1)*F
  (33,V2) + H1( - B2 + 1)*F(22,V3) + HOS( - B2 + 1)*(2*F
    (11,V2) + 4*F(113,V2) + F(1133,V2)) + H1S( - B2 + 1
  )*(F(1122,V3) - 4*F(112,V3) + 2*F(11,V3)) - F(22,B2*V2
                                                 2
   - V3*(B2 - 1)))*HOS(( - B1)/(B2 - 1))*( - B2 + 1)  +
           2
( - B2 + 1) *(F(11,B2*V2 - V1*(B2 - 1)) - F(22,B2*V2 - V1
  *(B2 - 1)) + 2*F(13,B2*V2 - V1*(B2 - 1)) + F(33,B2*V2
   - V1*(B2 - 1)))*H1S(( - B1)/(B2 - 1)) + (B1 - 1)*(HOB
  (( - B3)/(B1 - 1))*F(1,B1*V1 - V2*(B1 - 1)) + H1B(( -
    B3)/(B1 - 1))*F(1,B1*V1 - V3*(B1 - 1))) - HO(( - B3
  )/(B1 - 1))*F(0,B1*V1 - V2*(B1 - 1)) - H1(( - B3)/(B1
   - 1))*F(0,B1*V1 - V3*(B1 - 1)) - HOS(( - B3)/(B1 - 1)
             2                                   2
)*( - B1 + 1) *F(11,B1*V1 - V2*(B1 - 1)) - ( - B1 + 1) *
H1S(( - B3)/(B1 - 1))*F(11,B1*V1 - V3*(B1 - 1)) + HO(( -
  B2)/(B3 - 1))*(F(2,V1)*H1B( - B3 + 1) - (F(11,V1) - F(
    22,V1) + 2*F(13,V1) + F(33,V1))*H1S( - B3 + 1) +
  H1B( - B3 + 1)*(F(1,V1) + F(3,V1))))
```

TABLE 4. Listing of $Q = P_3 \oplus P_2 \oplus P_1$.

5. Data requirements, compatibility, and precision.

5.1. Data requirements. Table 5 lists the data needed for the formulas in Table 4. The columns of Table 5 correspond to the edges e_1, e_2, e_3, and the vertices V_1, V_2, V_3 of T. The rows correspond to values of F and some of its directional derivatives.

The entries in Table 5 consist of the letter x if the data are needed and a blank or a hyphen otherwise. Obviously, data requirements along edge e_i imply the same requirements at vertices V_j and V_k (where $\{i, j, k\} = \{1, 2, 3\}$). However, the vertex data requirements in Table 5 correspond to terms in Table 4 that are evaluated at the indicated vertices only. In any implementation, these data and any vertex data implied by edge data need to be supplied consistently.

Note that, on the edges, only directional derivatives of order up to two are required. At the vertices, some higher derivatives are also needed.

5.2. Compatibility conditions. Many bivariate interpolation schemes, at least in their early versions, exhibit the desired interpolation properties only if the primitive function F satisfies certain compatibility conditions. These typically require that certain mixed directional derivatives commute.

	e_1	e_2	e_3	V_1	V_2	V_3
F	x	x	x	–	x	x
F_1	–	x	x	–	x	x
F_2	x	–	–	–	x	x
F_{11}	–	x	x	–	x	x
F_{12}	–	–	–	–	x	x
F_{22}	x	–	–	–	x	x
F_{112}				–	–	x
F_{113}				x	x	–
F_{122}				–	–	x
F_{133}				x	x	–
F_{211}				–	–	x
F_{221}				–	–	x
F_{223}				x	–	–
F_{233}				x	–	–
F_{333}				x	–	–
F_{1122}				–	–	x
F_{1133}				x	x	–
F_{1333}				x	–	–
F_{2211}				–	–	x
F_{2233}				x	–	–
F_{3333}				x	–	–

TABLE 5. Data Requirements for the Interpolant Q.

The scheme presented here was differentiated, using the REDUCE syntax, and then evaluated on edges. It turned out that $Q = P_3 \oplus P_2 \oplus P_1$ interpolates to F and all of its directional derivatives up to order 2 on all three edges unconditionally.

Some terms do arise, however, that may not be immediately recognized as vanishing.

For example, using the syntax established in Table 3, we find that for $s = b_3V_3 + (1 - b_3)V_1$

$$\frac{\partial(QF)}{\partial e_1}(s) = \frac{\partial F}{\partial e_1}(s) + h_1(1 - b_3)R$$

where

$$R = F_{223}(V_1) - 2F_{133}(V_1) - F_{333}(V_1) - F_{113}(V_1).$$

However, since gradient and Hessian of F are continuous we have, by (2.4) that

$$R = \frac{\partial(F_{22} - 2F_{13} - F_{33} - F_{11})}{\partial e_3}(V_1)$$
$$= \frac{\partial(0)}{\partial e_1}(V_1) = 0.$$

Thus

$$\frac{\partial(QF)}{\partial e_1}(s) = \frac{\partial F}{\partial e_1}(s).$$

No compatibility condition arises for this or any other directional derivative of order up through 2.

5.3. Polynomial precision. The *precision set* of any operator P is the set of functions F, for which P is exact, i.e., $PF = F$.

Given a REDUCE version of $Q = P_3 \oplus P_2 \oplus P_1$ it is straightforward to apply Q to any polynomial. It is useful to note that since b_1 and b_2 are linear in x and y, and since $b_3 = 1 - b_1 - b_2$, any polynomial in x and y of degree N can be expressed as a polynomial in b_1 and b_2 of degree N and vice versa.

By applying Q to basic polynomials in b_1 and b_2, we find that the precision set of Q includes all polynomials of degree through eight. Q is also precise for some polynomials of higher degree.

At this point only, in all of the work described here, explicit expressions for the cardinal function defined in (2.6) are needed. These cardinal functions are given by:

$$\begin{aligned}
h_0(x) &= -6x^5 + 15x^4 - 10x^3 + 1,\\
h_1(x) &= 6x^5 - 15x^4 + 10x^3,\\
\bar{h}_0(x) &= -3x^5 + 8x^4 - 6x^3 + x,\\
\bar{h}_1(x) &= -3x^5 + 7x^4 - 4x^3,\\
\bar{\bar{h}}_0(x) &= -(1/2)x^5 + (3/2)x^4 - (3/2)x^3 + (1/2)x^2,\\
\bar{\bar{h}}_1(x) &= (1/2)x^5 - x^4 + (1/2)x^3.
\end{aligned}$$

APPENDIX: TWO DISCRETE C^2 INTERPOLANTS

PETER ALFELD

A1. Introduction. This appendix has two purposes: it describes some general techniques for constructing approximations of transfinite information from discrete data, and, more narrowly, it describes two particular discretizations of the transfinite scheme described in the body of this paper.

The full data requirements of the transfinite scheme are given in Table 5. Thus the information needed to define the transfinite interpolant consists of position, and one particular first and one second order derivative along edges. Also required are the values of certain derivatives at the vertices of the general triangle. These, however, can be derived from the transfinite information by differentiating tangentially and taking suitable combinations of derivatives. For example

$$F_{112}(V_3) = \frac{\partial F_{11}}{\partial e_2}(V_3)$$

(where F_{11} is required along edge e_2), or

$$F_{233}(V_1) = \frac{\partial^2}{\partial e_3^2}(-F_1 - F_3)$$

(where F_1 is required along edge e_{31} and F_3 can be computed by tangential differentiation of position).

In this Appendix, it is described how the required transfinite information can be approximated from given discrete data while preserving the global C^2 smoothness. §A2 describes the derivation of a discrete scheme with quintic precision, §A3 describes a similar scheme with reduced data requirements, but only with cubic precision, and §A4 contains some simple numerical examples.

A2. A discrete scheme with quintic precision. A discrete scheme that is precise for all quintics can be obtained by using the stencil given in Figure 2. The notation means that function values, gradients, and Hessians must be supplied at the vertices of the general triangle, in addition

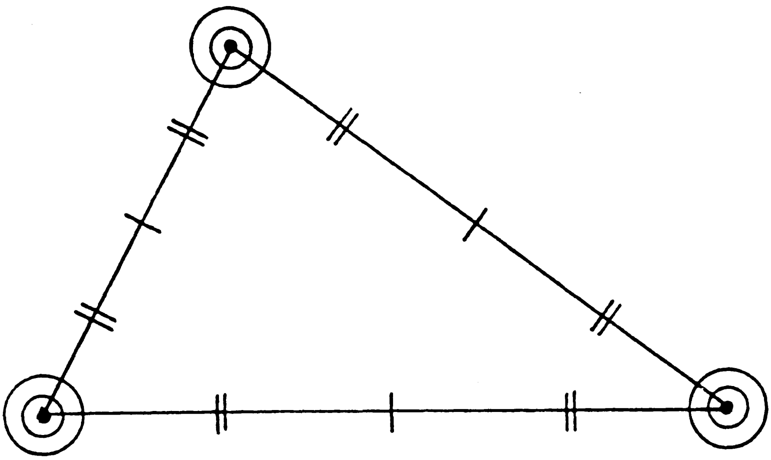

FIGURE 2. Stencil required for scheme 1.

to first order perpendicular cross-boundary derivatives at the midpoints of the edges, and second-order perpendicular cross-boundary derivatives at the (arbitrarily chosen) points $Q_{ij} := (V_i + 3 * V_j)/4$ and Q_{ji}, where i, $j = 1, 2, 3$, i and j distinct.

CONSTRUCTION OF SCHEME 1.

STEP 1. Approximate the required positional data on each side of the triangle by the univariate quintic polynomial interpolating to function values and first and second order tangential derivatives at the vertices. These univariate data can be computed from the data given at the vertices, and the univariate problem can be solved uniquely.

We exemplify the analysis by considering edge e_1. Only minor modifications will be required for the other edges. In order not to become overburdened by the notation of the function involved, it will contain no explicit reference to the edge.

The transfinite scheme requires an approximation of $f(b_2V_2 + (1 - b_2)V_3)$. Denote the approximation by $\phi(b_2) = \sum_{i=0}^{5} \alpha_i b_2^i$. The six coefficients of ϕ are defined by the linear system:

$$\begin{aligned} \phi(0) &= F(V_3), & \phi(1) &= F(V_2), \\ -\phi'(0) &= F_1(V_3) = g(V_3)^T e_1, & -\phi'(1) &= F(V_2) = g(V_2)^T e_1, \\ \phi''(0) &= F_{11}(V_3) = e_1^T H(V_3) e_1, & \phi''(1) &= F_{12}(V_5) = e_1^T H(V_2) e_1. \end{aligned}$$

Here, g and H denote the gradient and the Hessian of the primitive function F, respectively. Both are given at the vertices. Note the negative sign in the equations specifying the first derivatives. It occurs because the transfinite scheme uses the barycentric coordinate b_2 as its basic variable, whose derivative in the direction of edge e_1 is -1. A similar sign reversal occurs on edge e_3, but not on edge e_2.

STEP 2. Approximate the required first order cross-boundary derivative on each edge by the univariate quartic polynomial interpolating to the value of that derivative at the vertices and at the midpoint of the given edge, and the tangential derivatives of the cross-boundary derivative at the vertices. As in step 1, this process is well-defined by the given discrete data.

Consider again edge e_1. The transfinite scheme requires an approximation of $F_2(b_2V_2 + (1 - b_2)V_3)$. Denote the approximation by $\phi_2(b_2) = \sum_{i=0}^{4} \alpha_{2,i} b_2^i$. The following four equations for the coefficients of ϕ_2 are readily derived:

$$\begin{aligned} \phi_2(0) &= F_2(V_3) = g(V_3)^T e_2, & \phi_2(1) &= F_2(V_2) = g(V_2)^T e_2, \\ -\phi_2'(0) &= F_{21}(V_3) = e_2^T H(V_3) e_1, & -\phi'(1) &= F_{21}(V_2) = e_2 H(V_2) e_1. \end{aligned}$$

The fifth condition is $\phi_2(1/2) = F_2(M)$, where $M = (V_2 + V_3)/2$ is the midpoint of e_1, and the right hand side has to be computed from the given perpendicular cross-boundary derivative at M and the derivative of $\phi(b_2)$ in the direction of e_1. Let F_n denote the given perpendicular cross-boundary derivative at M, and let n_1 be the normal to e_1.

Then the derivatives satisfy $Eq = v$ where $E = [n_1, e_1]^T$, q is the gradient of the interpolant at M, and $v = [F_n, \ -\phi'(1/2)]^T$. Solving for q, and taking the inner product with e_2 yields $F_2(M) = e_2^T E^{-1} v$,which supplies the required right hand side of the above linear equation.

STEP 3. Approximate the required second order cross-boundary derivative on each edge by the univariate cubic polynomial interpolating to the values of that derivative at the four points implied by the stencil. This process is also well-defined.

The analysis is similar to that in step 2. At the endpoints of edge e_1, two conditions are readily obtained. At each of the points Q_{23} and Q_{32} three second order directional derivatives are available: The second order tangential derivative $\phi''(b_2)$, the tangential derivative of the cross-boundary derivative $-\phi_2'(b_2)$, and the second order perpendicular cross-boundary derivative given as data. These three derivatives determine the Hessian of the interpolant at Q_{23} and Q_{32}, which in turn determines the required second order cross-boundary derivative.

STEP 4. Approximate the required higher order derivatives at the vertices by suitably differentiating and evaluating the polynomial approximations of the transfinite information obtained in steps 1, 2, 3, proceeding as indicated by the examples in the introduction.

The following theorem states formally that the interpolation scheme so obtained is C^2 and has quintic precision.

THEOREM 1. *For any triangulation, and for any set of data implied by stencil* 1 *on each triangle*:

(a) *The interpolation scheme* 1 *defined in the above four steps yields a globally twice continuously differentiable surface.*

(b) *If the data are obtained by differentiating and evaluating a primitive function F, then the interpolant to F will equal F if F is a bivariate polynomial of degree up to* 5.

PROOF. For part (a) of the theorem, first note that the scheme is arbitrarily often differentiable in the interior of triangles. On each edge of the triangulation, all function values, and values of first and second order derivatives exist and are determined uniquely by the discrete data given on that edge. Moreover, the data entering the transfinite scheme are independent of the orientation of the triangle. Thus positions, and first and second order derivatives, match across edges, i.e., the interpolant is globally twice continuously differentiable.

For part (b) of the theorem, first note that if the underlying primitive function is a quintic bivariate polynomial, then function values along edges reduce to univariate quintic polynomials, and any first and second order directional derivatives reduce to quartic and cubic univariate polynomials, respectively. The above construction process, being based on univariate interpolation by polynomials of suitable degree, is exact for such functions.

Thus, if the primitive function is a quintic polynomial, then the discrete scheme yields an interpolant that is identical to that given by the transfinite scheme. In [1], the transfinite scheme was shown to be exact for polynomials of degree up to 8. A fortiori, it will be exact for polynomials of degree up to 5, completing the proof of the theorem.

A3. A discrete scheme with cubic precision. A discrete scheme similar to that derived in §2 can be obtained from the following stencil:

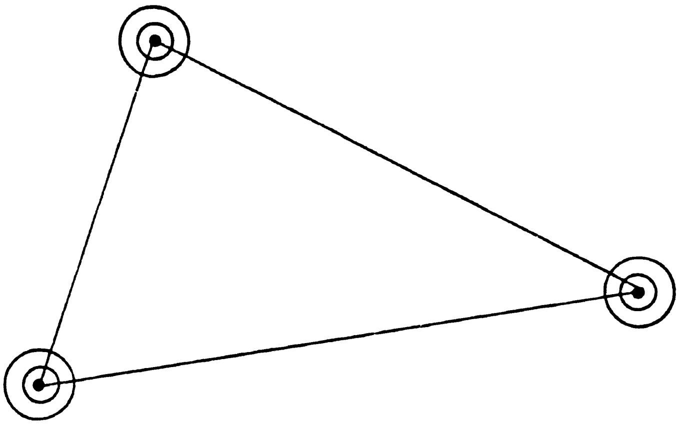

FIGURE 3. Stencil required for scheme 2.

Thus the discrete data comprise only values of position, gradient and Hessian at the vertices of the triangles. A user would not have to supply values of directional derivatives on edges of the triangles. Since there are fewer data interpolated to, the precision of the scheme is reduced and only polynomials of degree up to three will be reproduced exactly.

CONSTRUCTION OF SCHEME 2.

STEP 1. As in scheme 1, approximate the required positional data on each side of the triangle by the univariate quintic polynomial interpolating to function values and first and second order tangential derivatives at the vertices.

STEP 2. To be consistent with the quintic approximation of position, the approximation of any cross-boundary derivative along an edge must be quartic, involving five degrees of freedom. The discrete data provide only four conditions: values and tangential derivatives of the cross-boundary derivative at each of the vertices. This naturally leads to making up the missing condition by requiring that some cross-boundary derivative be cubic. To ensure global differentiability the direction of that derivative must be shared between neighboring triangles. This rules out direction defined by other edges of a triangle. Instead, we require that the first order perpendicular cross-boundary derivative be cubic. As in §2, we exemplify the analysis by considering edge e_1: We need to construct $\phi_2(b_2) = \sum_{i=0}^{4} \alpha_{2,i}\, b_2^i$, which is the approximation of $F_2(b_2v_2 + (1 - b_2)v_3)$ required by the transfinite scheme. The values of ϕ_2, and of its first order derivatives in the direction of edge e_1 at v_2 and v_3, are determined by the given discrete data. Similarly as in Step 1 of §2, the perpendicular cross-boundary derivative turns out to be given by $N(b_2) = a^T v$ where $a = E^{-1}n_1 =: [a_1, a_2]^T$, $E = [e_1, e_2]^T$, $v = [-\phi'(b_2), (b_2)]^T$, and n_1 is perpendicular to e_1 (it need not be normalized).

The function N is a quartic in b_2, whose leading coefficient is given by $\gamma_{2,4} = -5\, a_1\alpha_5 + a_2\alpha_{2,4}$ which should equal zero in order for the perpendicular cross-boundary derivative on edge e_1 to be cubic.

Thus we require that $\alpha_{2,4} = 5\, a_1\alpha_5/a_2$. A simple calculation shows that a_2 cannot be zero for a non-degenerate triangle.

STEP 3. To construct the approximation of a second order cross-boundary derivative on an edge, we proceed essentially as in Step 2 for the first order derivative. In general, any second order derivative on an edge will be a cubic polynomial, but we are given only two data implied by the discrete data at the vertices. The remaining two degrees of freedom are removed by requiring that the second order perpendicular cross-boundary derivative along the edge be linear. On edge e_1, we proceed as follows: Writing $\alpha_{22}(b_2) = \sum_{i=0}^{3} \alpha_{22,i}\, b_2^i$ for the required approximation of $F_{22}(b_2V_2 + (1 - b_2)V_3)$, we obtain, after some manipulation, $\alpha_{22,3} = 4\, a_1(-5\, a_1\alpha_5 + 2\, a_2\alpha_{2,4})/a_2^2$ and $\alpha_{22,2} = 6\, a_1(-2\, a_1\alpha_4 + a_2\alpha_{2,3})/a_2^2$. In both Steps 2 and 3, minor adjustments have to be made on edges e_2 and e_3.

STEP 4. As for scheme 1, the required higher derivatives at vertices are obtained by suitably differentiating the expressions obtained in Steps 1, 2, and 3. The properties of scheme 2 are stated formally in the following Theorem.

THEOREM 2. *For any triangulation, and for any set of data implied by stencil* 2 *on each triangle*:

(a) *The interpolation scheme* 2 *defined in the above four steps yields a globally twice continuously differentiable surface.*

(b) *If the data are obtained by differentiating and evaluating a primitive function F, then the interpolant to F will equal F if F is a bivariate polynomial of degree up to* 3.

PROOF. The proof is similar to that of Theorem 1. For part (b) of the theorem, observe that the discrete scheme yields the exact transfinite information only if the primitive function is a bivariate polynomial of degree at most 3 (which has linear second order derivatives).

A4. Numerical results. Using the symbol manipulation language REDUCE [3], the discrete schemes 1 and 2, as well as the transfinite scheme, were implemented into a FORTRAN code and run for some test examples where an underlying primitive function was known. The data required by the schemes were generated from the primitive function and were exact.

The domain in 2-space is sketched in Figure 4. There are four triangles

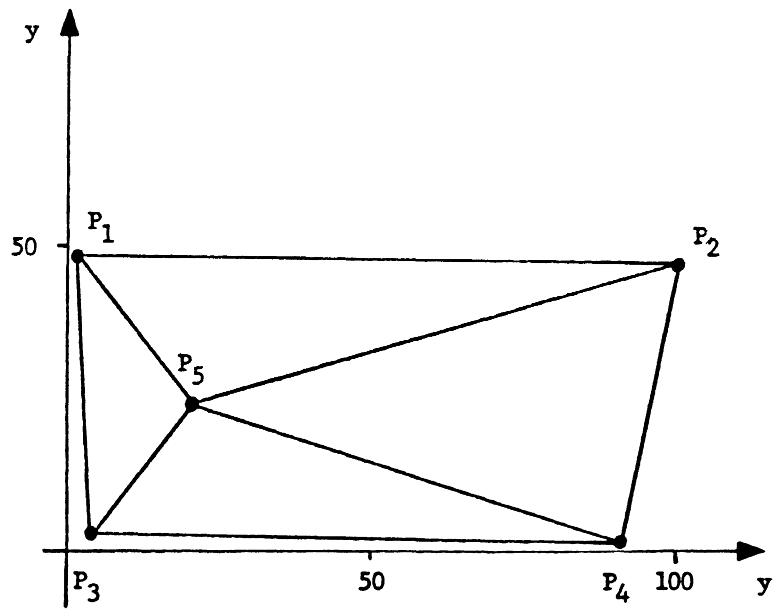

FIGURE 4. Domain picture.

($P_1 = [1, 49]^T, P_2 = [99, 47]^T, P_3 = [3, 3]^T, P_4 = [90, 1]^T, P_5 = [20, 24]^T$).

covering a quadrilateral region. The points were chosen so as to avoid any symmetry or edges parallel to coordinate axes which might introduce artifacts that are not in general present in an interpolation problem. The primitive function was chosen to be a half sphere with radius r centered at the origin, i.e., $F(x, y) = \sqrt{r^2 - x^2 - y^2}$, for several values of r. Again, the underlying principle in choosing the function F was not to introduce any artifacts due to F's having geometrical properties corresponding to properties of the domain, or to F's being a polynomial or a rational function.

The parameter r is a measure of the difficulty of the approximation problem. The point P_2 lies at distance 109.6 from the origin, and as r approaches that value, the quality of the approximation deteriorates. Table 2 below gives the maximum relative error for the three interpolation schemes, and for $r = 120, 150, 200$.

$r =$	120	150	200
transfinite scheme:	4.8E-2	4.6E-5	2.8E-7
discrete scheme 1:	5.4E-2	1.1E-3	5.5E-5
discrete scheme 2:	5.8E-2	1.2E-3	6.6E-5

TABLE 2. Numerical Results.

Note that in this example scheme 2 with quintic precision yields results that are only slightly more accurate than those given by scheme 2 with cubic precision. On the other hand, the loss in accuracy due to approximating transfinite information by discrete data is substantial. Of course, in practice transfinite information is usually unavailable so that the superior accuracy of the transfinite scheme cannot be exploited. As one would expect, the accuracy of the approximating interpolant deteriorates as the radius of the sphere defined by F decreases, and the edge of the sphere approaches the boundary of the domain of the interpolant. The accuracy of the results is remarkable in view of the fact that a substantial part of the domain of F is covered by only four triangles.

Conclusions. We have developed a C^2 interpolant to C^2 transfinite data defined over triangles.

In the appendix, two discrete bivariate interpolation schemes derived from a transfinite scheme have been described. Their relevant properties are the following:

1. The schemes require an underlying triangulation.
2. The interpolants are globally twice continuously differentiable.
3. The schemes are local, i.e., the information needed to evaluate the

interpolant at a given point is restricted to the triangle containing that point.

4. Only derivatives of order up through two are required as data.

5. The schemes are of quintic and cubic precision, respectively.

6. Limited numerical experience suggests that the gain in accuracy in going from cubic to quintic precision is marginal.

7. The cubic scheme requires data at vertices only.

It appears from the above, particularly in view of points 6 and 7, that scheme 2 is preferable over scheme 1. Note that using data on vertices only has the convenient consequence that the user does not have to be aware of the structure of the triangulation. Indeed, if the triangulation is generated by a black box routine, the user does not even need to know of its existence. A drawback of both schemes is their computational complexity. At present, only experimental codes exist, which have to be modified for each new problem.

References

1. R. E. Barnhill, G. Birkhoff, and W. J. Gordon, *Smooth Interpolation in Triangles*, Jour. of Approx. Theory **8** (1973), 114–128.

2. R. E. Barnhill, *Representation and Approximation of Surfaces*, in Mathematical Software III (J. R. Rice, ed.), Academic Press, 69–120.

3. A. C. Hearn, *REDUCE User's Manual*, 2nd ed., Report UCP-19, Department of Computer Science, University of Utah, 1973.

DEPARTMENT OF MATHEMATICS, UNIVERSITY OF UTAH, SALT LAKE CITY, UT 84112