Implementation of Some Triangular Data Fitting Schemes Using Averaging To Get Continuity

Computer Science Department
University of Waterloo
Research Report CS-2000-10
ftp://cs-archive.uwaterloo.ca/cs-archive/CS-2000-10/

Stephen Mann
smann@cgl.uwaterloo.ca, http://www.cgl.uwaterloo.ca/~smann/

Abstract

In this paper, I describe the implementation details of some functional, triangular data fitting schemes. The schemes in question use derivative information to find initial settings of control points, giving a C^0, piecewise polynomial surface with a high degree of polynomial precision. The interior control points are then modified to increase the continuity between patches without decreasing the polynomial precision. In implementing these schemes, I had to address several issues, including basis conversion for bivariate functions, finding the weights of control points used to compute derivatives, and the construction of data sets for testing. In addition, I developed and tested a new scheme that uses fewer derivatives than the other schemes discussed in this paper.

1 Introduction

In an earlier technical report [7], I presented a method for creating patches that meet with C^0 continuity, and a method for adjusting the control points of these patches to increase the continuity. The continuity adjustment process has the property that if the original patches have degree p polynomial precision, then the adjusted patches will also have degree p polynomial precision.

After writing that technical report, I implemented several of these schemes. This technical report describes some of the interesting implementation details, which required extension of some known results on basis conversion and finding weights of control points in derivative calculations. While the previous technical report describes interpolants that create one polynomial patch per data triangle and three polynomial patches per data triangle, this report only investigates the schemes that fit a single polynomial patch per data triangle. I also introduce a new data fitting scheme that is a minor variation of the earlier schemes, and I analyze the surfaces created by these schemes.

The basic problem these interpolants are solving is:

> **Given:** A triangulation $\mathcal{T}$ of a region of the plane, with z-values and derivative values at the data points.
>
> **Find:** A piecewise polynomial surface that interpolates the data points and as many derivatives as possible, with the patches meeting with as high of order of continuity as possible.

The goal in the "find" statement is a bit vague: what degree patches should we use? With what order of continuity should they meet? The vagueness results from the trade-off that are possible: by increasing degree, we can interpolate all the derivatives and have the patches meet with as high an order continuity as desired.

The approach in this paper (and in the earlier technical report) is to use the derivative information to obtain polynomial precision with as low of degree patch as possible. Once the degree is determined, the maximum continuity with which the patches meet is also determined.

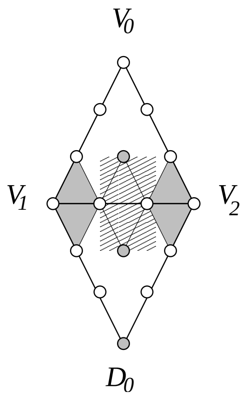

Figure 1: Two cubics meeting along a common boundary.

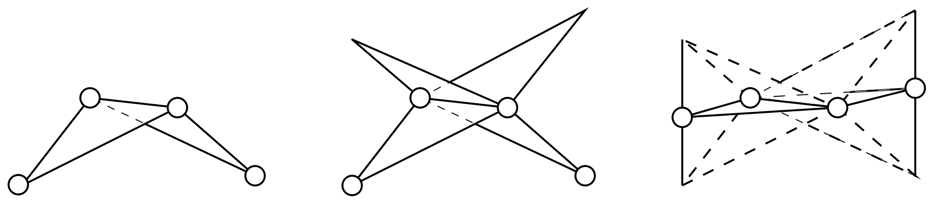

Figure 2: Adjusting the panels to meet C^1.

2 Background and a New Scheme

In this section, I give some of the background material for this paper. I will give a minimal (indeed, less than minimal) discussion of this material since it is mostly just reproduction of work in my earlier technical report; for more details, the reader is referred to that report [7]. Further, I will describe a new data fitting scheme that is a variation on the other schemes discussed in this section.

2.1 Continuity

The continuity conditions for functional triangle Bézier patches are easily described in terms of the control points. Here I will just describe the C^0 and C^1 conditions; higher order conditions can be found elsewhere [7], with a bit of discussion of C^2 conditions appearing in Section 3.2 of this paper.

For two patches to meet with C^0 continuity, they must have identical control points along their shared boundary. For the two patches to meet with C^1 continuity, adjacent panels of the control nets must be coplanar. For example, as illustrated schematically in Figure 1, the two patches must share boundary control points. To meet with C^1 continuity, each of the three panels of four control points must be coplanar.

The observation of Foley and Opitz [4] is that if two panels are not coplanar, we may make them coplanar by extending each panel over the other and averaging (Figure 2). Mann later generalized this averaging scheme to higher order continuity [7]. One advantage of using such an averaging scheme is that if both patches have degree p polynomial precision before averaging, then they will continue to have degree p polynomial precision after averaging. This is the idea of the three schemes discussed in the remainder of this section.

2.2 Scheme 1

The first scheme initially sets all the control points using derivatives at the corner vertices. For example, the left half of Figure 3 illustrates the derivatives needed at a vertex (i.e., we need the appropriate derivatives for setting the circled vertices) and on the right illustrates the control points actually set using these derivatives. The shaded vertices are set to the average value specified by the derivatives at the corner points.

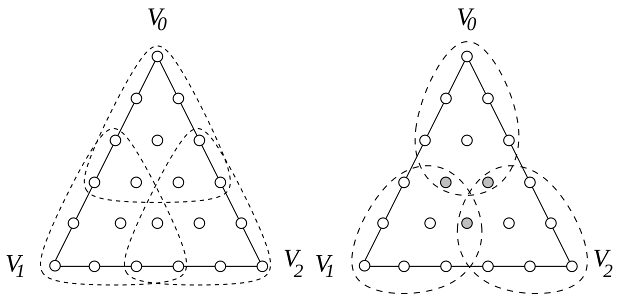

Figure 3: Derivatives needed for the quintic version of first scheme.

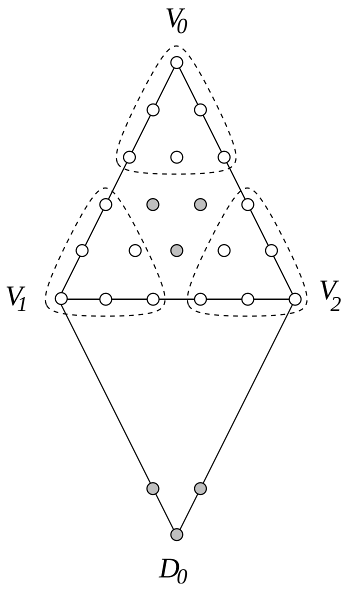

Figure 4: Construction for second scheme illustrated for $n = 5$.

After setting the control points in this manner, adjacent patches meet with C^0 continuity. We then apply the averaging scheme of Section 2.1 to increase the continuity between patches.

2.3 Scheme 2

The second scheme is similar to the first. However, instead of setting all the control points using derivatives at the corner of the data triangle, we set the center control points using the derivatives at the neighboring triangles. For example, Figure 4 illustrates the construction for quintics. The circled control points are set using derivatives at the appropriate corner. To set the shaded control points, we solve a system of linear equations using the derivatives at the neighboring corner. Each equation is an evaluation of the Bézier patch or its derivatives at the domain vertex of an adjacent triangle. E.g., one of the equations will be

$$\sum P_{ijk} B^n_{ijk}(D_0) = z(D_0).$$

For quintics, there are two more equations for interpolating the partial derivatives at D_0. For symmetry reasons, we compute three sets of these center control points (one for each neighboring triangle) and take a weighted average of the results (see the earlier technical report for the averaging scheme [7]).

2.4 A New Scheme (Scheme 3)

The problem with the first of the schemes described in the previous two sections is that it requires a large number of derivatives at the data points. The second scheme requires fewer derivatives at the data

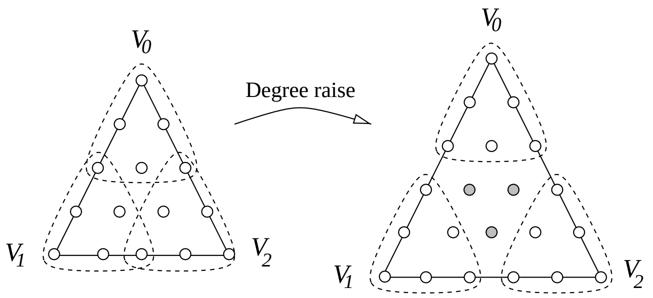

Figure 5: Construction for new scheme illustrated to achieve C^1 continuity.

points, but uses data at neighboring triangles (i.e., it is less "local"); later we will see that this use of data at neighboring triangles results in poor surface quality.

We can reduce the number of derivatives from what is required by the first of those schemes without borrowing from the neighboring triangles with the following construction for a C^k patch network:

1. Find the minimum number of derivatives needed to avoid solving the vertex consistency problem (i.e., to get a C^k join, you need degree $4k+1$ patches and $2k$ derivatives and position at the data points).
2. Fit a degree $\lfloor (3k+2)/2 \rfloor$ patch to this data using the scheme described in Section 2.2.
3. Degree raise this patch to degree $4k+1$.
4. Reset the corner groups of control points using the derivative information at the corners.
5. Adjust to continuity C^k.

Figure 5 illustrates this scheme for construction a C^1 join. On the left, we first construct a quartic patch using the scheme described in Section 2.2. Next, we degree raise this quartic patch to quintic, giving the patch on the right. We then reset the circled control points to match the derivatives at the corners, and finally we adjust the gray points to meet the neighboring patches with C^1 continuity.

An obvious variation on this scheme is to start with a degree $4k+1$ patch, set the corner control points using the derivatives, then set the remaining interior control points by assuming that patch is a degree raised patch (similar to Farin's construction for the quadratic precision point of a cubic [3]) and then adjusting for continuity. The generalization of Farin's quadratic precision point to higher degree is discussed in the next subsection.

2.5 Generalization of Farin's Quadratic Precision Point

If we have position and derivatives at three points on a triangle, we can interpolate this data using a cubic patch. However, the center control point will be left unset by the data at the corners of the triangle. One simple setting is to place the center control point at the centroid of the triangle; this setting results in a patch with linear precision. Farin found a different setting of the center control point, which achieves quadratic precision [3]:

$$P_{111} = \frac{1}{4}\big[P_{210} + P_{120} + P_{012} + P_{021} + P_{201} + P_{102}\big] - \frac{1}{6}\big[P_{300} + P_{030} + P_{003}\big].$$

The generalization of Farin's idea is that if we have position and d-derivatives at the corners of a triangle, we can interpolate this data with a degree $2d+1$ Bézier patch. This leaves a triangular block of control points on the interior of the patch unset. Our goal is to find settings of these control points such that the patch reproduces the highest degree polynomial completely specified by the derivative information (degree $\lfloor \frac{4d+4}{3} \rfloor$).

For example, we can find the "quartic precision" settings of a the shaded control points of the quintic patch appearing in Figure 5. To find these points, first note that the derivatives completely specify (over specify) a quartic patch. Now assume that our quintic is a degree raised version of a quartic. Rather than constructing that quartic and degree raising to find settings for the shaded control point (as was done in the previous section), we will find formulas for them.

The three circled groups of quintic control points are set using the derivatives at the corners. This leaves the three shaded control points unset. If we label the quintic control points P_{ijk} and label the quartic control points Q_{ijk}, then the degree raising formulas for the shaded quintic points are

$$\begin{aligned} P_{221} &= \big(2Q_{121} + 2Q_{211} + Q_{220}\big)\big/5, \\ P_{212} &= \big(2Q_{112} + Q_{202} + 2Q_{211}\big)\big/5, \\ P_{122} &= \big(Q_{022} + 2Q_{121} + 2Q_{121}\big)\big/5. \end{aligned}$$

However, we don't want to express the quintic points in terms of the quartics. Instead we use the degree raising formulas to relate all of the quartic and quintic control points, and solve to find expressions for the unknown quintic control points in terms of the known quintic control points. Here I will derive the formula for P_{212}; the formulas for the other two points follow from symmetry.

We start by finding another quintic control point that is a weighted combination (in the degree raising formula) of Q_{211}:

$$\begin{aligned} P_{311} &= \big(3Q_{211} + Q_{301} + Q_{310}\big)/5 \\ \Rightarrow Q_{211} &= \big(5P_{311} - Q_{301} - Q_{310}\big)/3 \end{aligned}$$

Next, we find expressions for Q_{301} and Q_{310}:

$$\begin{aligned} P_{401} &= \big(4Q_{301} + Q_{400}\big)/5 \\ \Rightarrow Q_{301} &= \big(5P_{401} - Q_{400}\big)/4 \\ P_{410} &= \big(4Q_{310} + Q_{400}\big)/5 \\ \Rightarrow Q_{310} &= \big(5P_{410} - Q_{400}\big)/4 \end{aligned}$$

Since we know $Q_{400} = P_{500}$, we can now write a formula for Q_{211} in terms of the quintic control points:

$$Q_{211} = \Big(5P_{311} - \big(5P_{401} - P_{500}\big)/4 - \big(5P_{410} - P_{500}\big)/4\Big)\Big/3$$

Performing a similar derivation for Q_{022} and Q_{121} (the latter formula is just an index permutation of the formula for Q_{211}) and substituting, we find

$$P_{212} = \frac{1}{15}\big(P_{005} + P_{500}\big) + \frac{1}{3}P_{320} - \frac{2}{15}P_{310} + \frac{2}{3}\big(P_{113} + P_{311}\big) - \frac{1}{6}\big(P_{104} + P_{401} + P_{014} + P_{410}\big).$$

The control points P_{221} and P_{122} have similar formulas.

Looking at the above process, we see that it works because as we solve for formulas for the Q_{ijk} in terms of the P_{ijk}, the degree raising formulas are "pushing out" towards the corners of the patch, where we know $P_{n00} = Q_{n-1\,00}$.

However, things are more complicated when we try to derive formulas for a septic patch (i.e., when we have position and 3 derivatives at the corners). The problem is that the derivatives only specify a quintic patch. We can derive formulas for most of the septic control points using the above process, but the septic control points P_{322}, P_{232}, and P_{223} are related to the sextic control point S_{222}, which is not set by the derivative information and can not be eliminated from the equations without using a new technique. At this point, we can work with the quintic patch, giving

$$S_{222} = \big(2Q_{122} + 2Q_{212} + 2Q_{221}\big)/6.$$

We now need to related through degree raising the quintic control points Q_{122}, Q_{212}, Q_{221} to the septic control points, which is a straightforward task, albeit more complicated than deriving the relations when degree raising by one degree.

In general, while Farin's quartic precision point is useful, and the quintic precision points derived above might be useful, for higher degree, it seems unlikely that the derivation of the formulas would result in sufficient benefits over constructing the lower degree patch and explicitly degree raising.

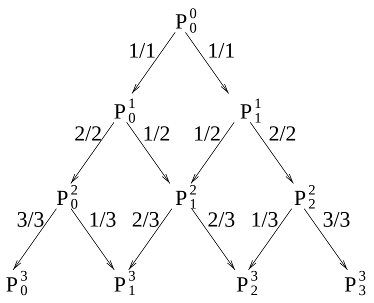

Figure 6: Degree raising triangle diagram.

2.6 Degree Raising Triangle Diagram

Triangle diagrams can be used to illustrate many properties of Bézier and B-spline curves and surfaces. The most common use is to illustrate the data flow of the de Casteljau evaluation. Here, I briefly note that there is a form of a triangle diagram for degree raising. Unlike standard triangle diagrams [5], the weights on the edges will not be related to an evaluation point in the domain. Instead, we will use the degree raising weights.

Figure 6 shows a degree raising triangle digram for curves. In this figure, the super script of the control points represents the degree. Note that this is a complete diagram, showing degree raising from a constant function to a cubic function. More typically, we would start at an intermediate level and work down. The benefits of such a diagram are less clear than they are for the de Casteljau data flow diagram, although one use is to use the diagram to determine the weights for multiple degree raisings (which could also have readily been computed analytically [8]).

A second idea would be to use the diagram to derive a simple degree reduction algorithm. For example, using just the bottom two layers in the figure, if we start with P_i^3 and want P_i^2, we could set $P_0^2 = P_0^3$ and $P_2^2 = P_3^3$. The diagram gives us two simple formulas for P_1^2:

$$\begin{aligned} P_1^3 = (P_0^2 + 2P_1^2)/3 \quad &\Rightarrow \quad P_1^2 = (3P_1^3 - P_0^2)/2 \\ P_2^3 = (P_2^2 + 2P_1^2)/3 \quad &\Rightarrow \quad P_1^2 = (3P_2^3 - P_2^2)/2. \end{aligned}$$

We could then average these two equations to get a symmetric formula for P_1^2.

This idea of a de Casteljau type diagram for degree raising extends to arbitrary dimension, although the diagrams become complicated even for degree 3 due to the necessity of labeling all the edges.

3 Implementation

This section describes relevant implementation notes for the three data fitting schemes described in the previous section. To implement these schemes, some extensions to previous work on basis conversion and triangle diagrams was needed. These extensions are discussed in this section, along with other implementation notes.

3.1 Subdivision To Convert From Monomial Basis To Bernstein Basis

The vertices of my data sets were tagged with mixed partial derivatives. I set the control points of the Bézier patch to interpolate the derivatives by performing a change of basis on the derivative data, converting from the monomial basis to the Bernstein basis. Once the data is in the Bernstein basis, we can interpolate the mixed partials at the data points by just copying the coefficients.

Barry and Goldman give an algorithm for conversion from the monomial basis to the Bernstein basis by using de Casteljau evaluations [2]. If we look at the domain and the blossom values, their algorithm (for cubic curves) takes control points $f(P,P,P), f(P,P,\delta), f(P,\delta,\delta), f(\delta,\delta,\delta)$, performs a de Casteljau evaluation at Q, and extracts control point $f(P,P,P), f(P,P,Q), f(P,Q,Q), f(Q,Q,Q)$ from the de Casteljau evaluation triangle as illustrated in Figure 7.

Note that the Barry-Goldman monomial basis is a scaled form of the standard monomial basis, and the partial derivatives need to be scaled to get them in the monomial basis; in particular, they use $\binom{n}{i}t^i$

$$\begin{tikzcd}
 & & & f(Q,Q,Q) & & & \\
 & & f(P,Q,Q) \arrow[ur] & & f(Q,Q,\delta) \arrow[ul] & & \\
 & f(P,P,Q) \arrow[ur] & & f(P,Q,\delta) \arrow[ul] \arrow[ur] & & f(Q,\delta,\delta) \arrow[ul] & \\
f(P,P,P) \arrow[ur] & & f(P,P,\delta) \arrow[ul] \arrow[ur] & & f(P,\delta,\delta) \arrow[ul] \arrow[ur] & & f(\delta,\delta,\delta) \arrow[ul]
\end{tikzcd}$$

Figure 7: Conversion from monomial to Bernstein basis.

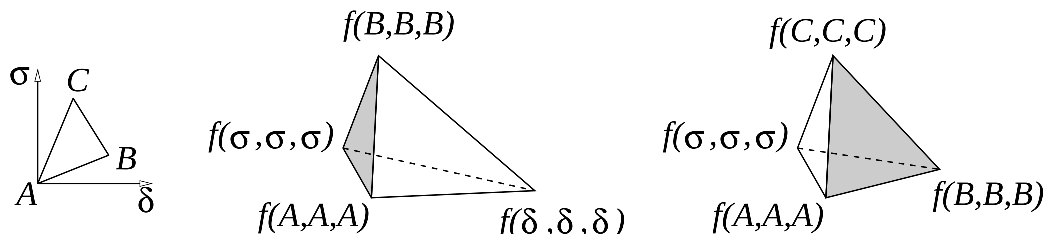

Figure 8: Monomial and Bernstein bivariate domains.

for their scaled monomial basis. For our bivariate monomial basis, we will use

$$\frac{n!}{i!j!(n-i-j)!}x^i y^j.$$

The extension of the Barry-Goldman scheme to a bivariate domain is almost trivial: just evaluate twice and you have the answer. However, for numerical reasons we may wish to select the order of the evaluations, which in turn may require us to flip the control net. For example, suppose we have a cubic with control points

$$\begin{array}{llll}
f(\sigma,\sigma,\sigma) & & & \\
f(A,\sigma,\sigma) & f(\delta,\sigma,\sigma) & & \\
f(A,A,\sigma) & f(A,\delta,\sigma) & f(\delta,\delta,\sigma) & \\
f(A,A,A) & f(A,A,\delta) & f(A,\delta,\delta) & f(\delta,\delta,\delta),
\end{array}$$

where σ and δ are the domain partial derivative directions. To convert to the Bernstein basis with domain $\triangle ABC$, we perform a de Casteljau evaluation at B, extract the face $\triangle AB\sigma$, then perform a second de Casteljau evaluation at C and extract the face over $\triangle ABC$. This process is illustrated in Figure 8. On the left of this figure is a digram of the domain, showing the partial derivative directions and the desired triangular domain. In the middle is an illustration of the de Casteljau evaluation starting from the monomial basis on the bottom, and evaluating at point B in the domain. The shaded face of this pyramid is extracted, becoming the bottom face of the pyramid at the right of this diagram. A second de Casteljau evaluation is performed at the point C, with the shaded face of this pyramid containing the desired Bézier control points.

However, for stability reasons, we may wish to evaluate at C first, and/or we may wish to extract a different intermediate triangle. In particular, consider the case when B lies on the ray (A,σ) and C lies on the ray $(A,-\delta)$ as illustrated in Figure 9. If we first evaluate at B to eliminate δ, the resulting control net will be degenerate, leaving us unable to evaluate at C. A reordering of the evaluations avoids this degeneracy, but may (as illustrated in the figure) result in a representation for triangle $\triangle ACB$ that requires a flip of the control points to get a parameterization over $\triangle ABC$.

Further, when B or C lie near one of the edges of the original domain, we need to be careful about the evaluation order to avoid numerical stabilities. As a heuristic to decide on a reasonable evaluation order, I express each of B and C relative to the frame A,δ,σ: $B = A + b_0\delta + b_1\sigma$ and $C = A + c_0\delta + c_1\sigma$. I then find the minimum in absolute value of b_0, b_1, c_0, c_1, and use this to decide at which of B and C to evaluate (the one with the minimum coordinate), and which of δ and σ to replace first (e.g., if b_0 is minimum, I evaluate at B first and eliminate δ). This results in four cases in the code, one case for

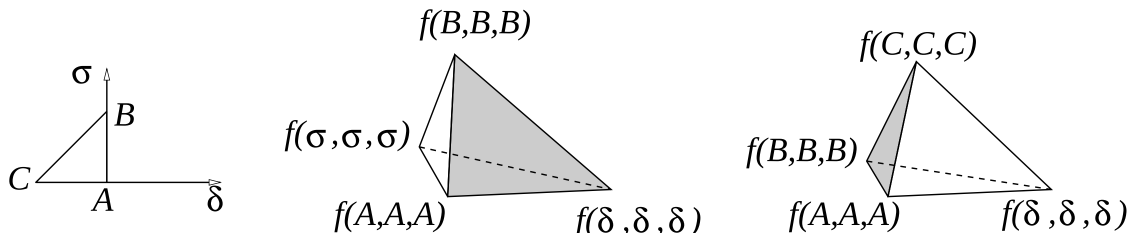

Figure 9: Triangular domain for which alternative evaluation and a flip is required.

each order of evaluation of B, C and order of replacement of δ, σ. Two cases require flipping the control points, although this "flip" could be written into the code for extracting the control points from the de Casteljau pyramid.

3.2 Continuity Adjustment

The continuity adjustment scheme is straight-forward and simple to implement. The hardest part is extracting the correct control points. This task is simplified by constructing neighbor patches so that a specified edge is the shared edge, and by rotating the center patch so that a specified edge is the shared edge.

One note, however: the continuity adjustment in the earlier technical report "pushes" values from two adjacent patches over a common boundary, averages, and "pushes" the averaged values back to get the desired point. For example, to average to get C^2 continuity (assuming we already have C^1 continuity), suppose we start with values

$$f(A,A), f(A,B), f(B,B), f(A,C), f(B,C), f(C,C)$$

and

$$g(A,A), g(A,B), g(B,B), g(A,D), g(B,D), g(D,D),$$

with $f(A,A) = g(A,A), f(A,B) = g(A,B), f(B,B) = g(B,B)$ and appropriate relations between the other values to have C^1 continuity. The scheme described in my earlier technical report computes $f(C,D)$ and $g(C,D)$, averages, and then uses this average to compute new values for $f(A,A)$ and $g(D,D)$ to get C^2 continuity.

In my implementation, I found it easier to "push" the values of one patch over the other patch and average. This alternative scheme does not require pushing values back. Working from the same example, I compute $f(D,D)$ and $g(C,C)$, and then compute

$$\begin{aligned} g^*(D,D) &= \big(f(D,D) + g(D,D)\big)/2, \\ f^*(C,C) &= \big(f(C,C) + g(C,C)\big)/2. \end{aligned}$$

The control points for $f(C,C)$ and $g(D,D)$ are then set to the values $f^*(C,C)$ and $g^*(D,D)$ respectively. Both approaches are illustrated in Figure 10.

3.3 Implementation Notes for the First Scheme

This scheme is straight-forward to implement. There are three parts to the implementation: the conversion from monomial to Bernstein basis (discussed in Section 3.1), the continuity adjustments (discussed in Section 3.2), and the averaging of control points. This last step is the only step not already discussed, and the only difficulty occurs if we don't first rotate the center patch, which results in different edges of the patch requiring different indices to extract and average the control points.

3.4 Implementation Notes for the Second Scheme

The second scheme is more difficult to implement. The parts required to implement this scheme are the conversion from monomial to Bernstein basis (discussed in Section 3.1), the interpolation of neighboring data, the continuity adjustments (discussed in Section 3.2), and averaging of control points.

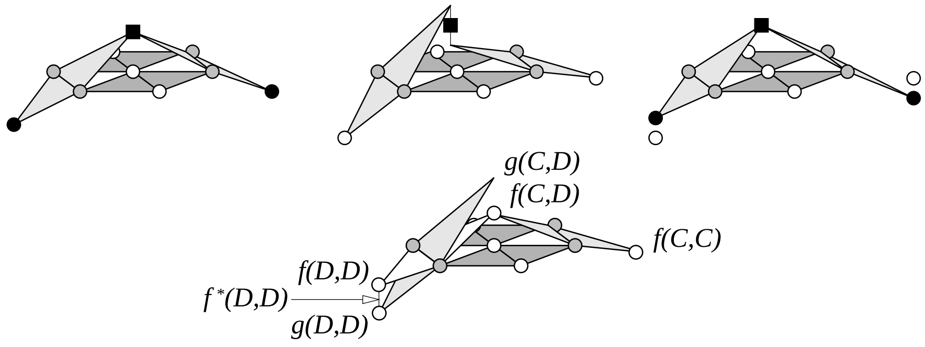

Figure 10: Top row, left: C^2 condition. Top row, middle, right: C^2 adjustment. Bottom row: alternative C^2 adjustment.

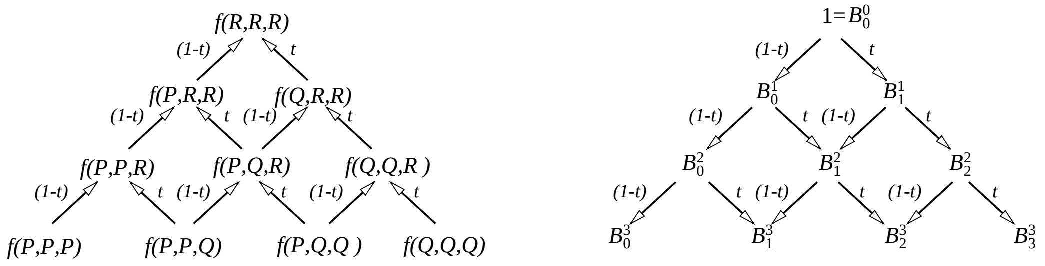

Figure 11: de Casteljau evaluation of a curve and the construction of the Bernstein polynomials.

The most difficult part of this scheme is the interpolation of the neighboring data. This step requires you to set up and solve a system of linear equations. This was my approach, and I used lapack [1] to solve the system of equations. The difficulty is in finding the coefficients for the matrix of equations.

Each equation is an evaluation of a Bézier patch (or derivatives of the Bézier patch) at the corner of a neighboring patch. Some of the Bézier coefficients are known and some are unknown. What we are looking for is the coefficients of both the known and unknown control points. For the equation representing the evaluation of the Bézier patch, we know the coefficients are the Bernstein polynomials. However, when an equation represents a derivative of the Bézier patch, the equations become more complicated.

In the curve case, we can compute the Bernstein polynomials using the reverse triangle diagram as shown in Figure 11. To compute control point weights for the derivative equation, we could try working from the derivatives formula of a Bézier curve:

$$\sum_{i=0}^{n-1} n(P_{i+1} - P_i)B_i^{n-1}(t).$$

What we are after are the coefficients of the P_i. Rewriting, we find that the coefficient C_i of P_i is

$$C_i = \begin{cases} -nB_0^{n-1}(t) & i = 0 \\ nB_{n-1}^{n-1}(t) & i = n \\ nB_{i-1}^{n-1}(t) - nB_i^{n-1}(t) & \text{otherwise} \end{cases}$$

However, these equations are more complex for higher derivatives, and for surfaces, the equations are still more complex for mixed partial derivatives (with the main problem being special cases for the corners and edges).

Fortunately there is a simpler way. Taking my cue from the triangle diagram for Bézier curves and the Bernstein polynomials (Figure 11), I used the parameter value for computing the first layers of the triangle diagram, and then used the weights of the derivative directions directions for the remaining layers. This is illustrated for both curves and surface in Figure 12. Note that in the downward de Casteljau diagram, I have reordered the weights on the edges from the upward triangle diagram; the order has no

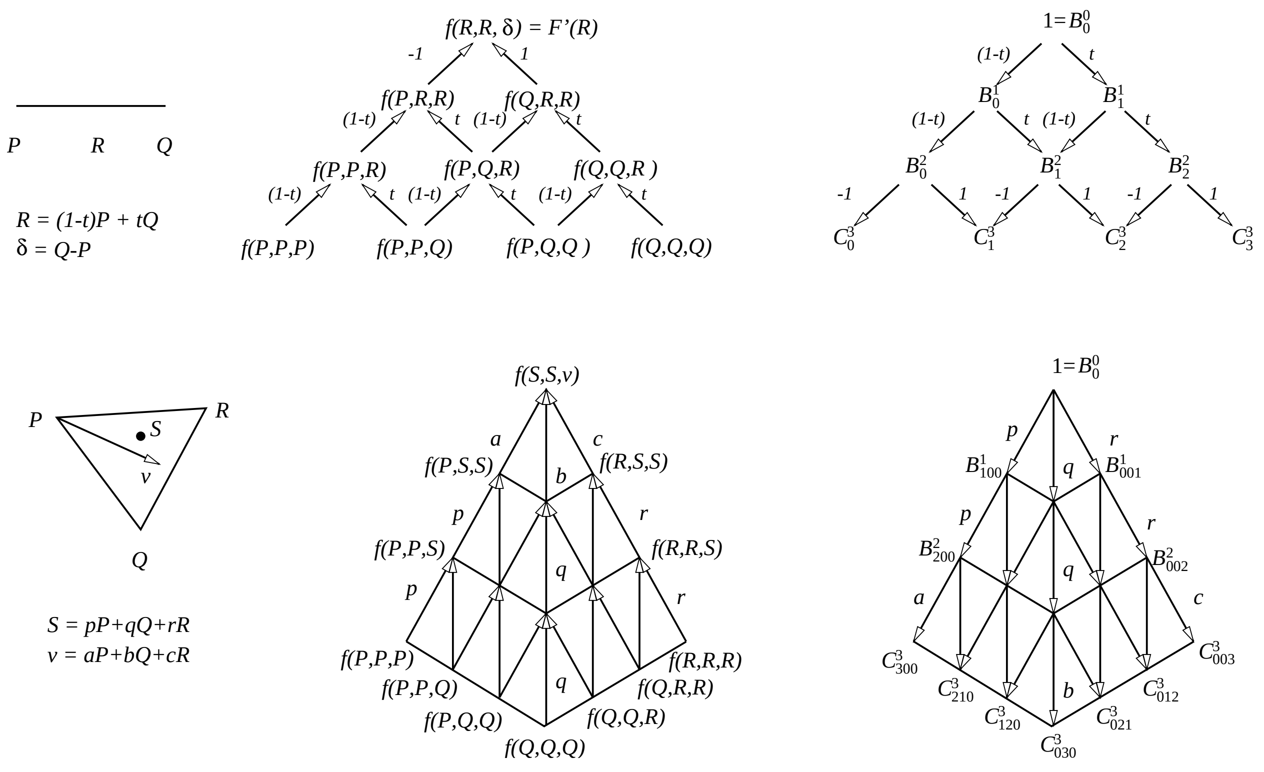

Figure 12: de Casteljau evaluation of the derivative and the construction of the weights of the control points.

effect on the result. For multiple derivatives, we apply the derivative weights at multiple levels in the pyramid, and for mixed partials, we apply the weights for each direction at as many levels as we take each derivative.

Given that we know how to compute the weights of the control points when computing derivatives, the question of which derivatives to use remains. I used the triangle directions at the vertex at which we evaluate; e.g., for the case illustrated in Figure 4m I used domain directions $V_2 - D_0$ and $V_1 - D_0$.

3.5 Implementation Notes for Third Scheme

Having implemented schemes 1 and 2, scheme 3 was easy to implement. Basically, use the initial fitter for scheme 1, degree raise, then use code similar to the initial fitter for scheme 2 to reset the corner values, and finally call the continuity adjustment. While not the most efficient implementation, this was sufficient for testing.

3.6 Rotate Patch

One of the difficulties in implementing these schemes is that the continuity conditions have to be applied to all three boundaries of the patch. When I implemented scheme 1, I implemented this as cases, requiring a different permutation of the indices for each edge. When I started to implement scheme 2, I realized that the permutations would be even worse (primarily for the averaging step), and decided to instead rotate the patch so that I could always use the same indices. This rotation greatly simplified my code.

4 Results

I ran several tests on the three schemes. The initial tests were to verify the code was working correctly; for these I used constant and linear data sets since I could check intermediate values and easily tell if they were correct. I then tested for polynomial precision and convergence, and then looked at isophote plots, shaded images, and Gaussian curvature plots.

	Method 1				Method 2			Method 3
f	n=3	n=5	n=6	n=9	n=3	n=5	n=9	n=5
$z = x^2$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
$z = x^3 + x^2y - xy^2 + x - y - 1$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
$z = x^2y^2 - x^2y - xy^2 - x^2 - y - 2$		$\surd$	$\surd$	$\surd$		$\surd$	$\surd$	$\surd$
$z = x^5/2 - y^5 + x^3y^2 - x^2y^3 + x^2 + x + 1$		$\surd$	$\surd$	$\surd$		$\surd$	$\surd$	
$z = x^6 - y^6 + x^3y^3 - x^2y^2 + x - y$			$\surd$	$\surd$			$\surd$	
$z = x^9 + x^4y^3 - y^9 + x + y + 2$				$\surd$			$\surd$	

Table 1: Test functions. A '$\surd$' means the method reproduced f.

4.1 Polynomial Precision

As a first test, I checked for polynomial reproduction. In particular, I used the functions listed in Table 1.[1] For this table, I sampled the known function for position and derivatives on a 5×5 grid, giving a 3×3 region of non-boundary patches to which the interpolants were fit. Each patch of the interpolant was sampled on a '10×10' grid (since the patches are triangular, it is really half a grid) and the z-values of the patch samples were compared to the z-values of the known function. The patches were fit over the $[-0.5, 0.5] \times [-0.5, 0.5]$ square, with the boundary patches extending outside this region.

As a further test, the patches were fit to polynomial data *without* adjusting for continuity, then sampled for position and normal. The resulting triangulation has multiple triangles meeting at each sample point. At sample points on the interior of the patches, the normals will all be equal; at sample points on the boundary of patches, the normals will be equal if the patch meets its neighbor with C^1 or higher continuity. Thus, I checked the normals for each position to verify that they were equal, since when schemes 1 and 2 fit degree n patches to degree n data (and when scheme 3 fits degree $n+1$ patches to degree n data), then all constructed patches should reproduce the polynomial and the normals at a sampled point should match.

The results of these tests matched the theory: all the schemes had the level of polynomial precision and continuity that was expected. Further, the exercise was useful as a means of further debugging and verifying my code.

4.2 Convergence

As a second test, I empirically checked the polynomial convergence. Theoretically, if the interpolant has certain derivative properties, and if it has degree d precision, then it should have an error term of $O(h^{d+1})$, where h is the distance between samples of the function. As a first test, I ran my methods on the Franke function, sampling over the $[0, 1]$ square, with samples spacings of .2, .1, .05, and .025. The result of these tests appear in Table 2. When halving the interval size, we should reduce the error by a factor of 2^{d+1}. However, we see from the table that the numbers are not quite that good. Since the ratios appear to get better for denser samplings, it is likely that there is an initial effect, since the big–O error does not hold if the known function is sampled too sparsely to catch all its features. Denser samplings might show the results we want, but for this data set, the higher degree interpolants are running out of floating point precision.

As a second test of convergence, I ran the schemes without increasing the continuity (i.e., I checked the convergence of the C^0 surfaces created). The results appear in Table 3. Interestingly, most of the schemes have identical precision with and without the continuity adjustments (up to six digits of precision). The exception was scheme 2. For degree 5, the C^0 version of scheme 2 has slightly better precision than the continuity increased versions. This is the expected behavior. However, for degree 9, the continuity adjust version of scheme 2 has better precision than the C^0 version. I have no explanation for this behavior.

Looking at both raw numbers and convergence rates, scheme 1 appears to be better than scheme 2. This is expected, since the data used in setting the center control points is more local for scheme 1 than it is for scheme 2.

[1]Additional tests were run on other data sets, including $z = x^x - y^2 + xy + x + y + 1$, $z = y^2$, $z = x^5$, $z = y^5$, $z = x^9$, $z = y^9$, but not all schemes were tested on these additional data sets.

Method	Theory	0.2	0.1	0.05	0.025
m1, d3	16	0.041201	0.00986012 (4.18)	0.000866224 (11.4)	5.89684e-05 (14.7)
d5	64	0.0288368	0.0011683 (24.7)	2.82327e-05 (41.4)	4.9238e-07 (57.3)
d6	128	0.0125883	0.000152819 (82.4)	1.19468e-06 (128)	9.66577e-09 (124)
d9	1024	0.00351799	7.17813e-06 (490)	1.33345e-08 (538)	1.51709e-11 (879)
m2, d3	16	0.0426704	0.0117674 (3.63)	0.00108287 (10.9)	7.4514e-05 (14.5)
d5	64	0.0288692	0.0014494 (19.9)	2.90774e-05 (49.8)	4.9238e-07 (59.1)
d9	1024	0.00358552	5.97093e-05 (60)	1.34982e-07 (442)	1.75238e-10 (770)
m3, d5	32	0.0288368	0.00128509 (22.4)	4.37087e-05 (29.4)	1.18989e-06 (36.7)

Table 2: Convergence rates on Franke function. Numbers in parentheses are ratio of previous to current column and should match the value in the "Theory" column.

Method	Theory	0.2	0.1	0.05	0.025
m1, d5	64	0.0288368	0.0011683 (24.7)	2.82327e-05 (41.4)	4.9238e-07 (57.3)
d6	128	0.0125883	0.000152819 (82)	1.19468e-06 (128)	9.66577e-09 (123)
d9	1024	0.00351799	7.17813e-06 (490)	1.33345e-08 (538)	1.51709e-11 (879)
m2, d5	64	0.0288368	0.0011683 (24.7)	2.82327e-05 (41.4)	4.9238e-07 (57.3)
d9	1024	0.00504409	7.25655e-05 (69.5)	1.64821e-07 (440)	2.1413e-10 (770)
m3, d5	32	0.0288368	0.00128509 (22.4)	4.37087e-05 (29.4)	1.18989e-06 (36.7)

Table 3: Convergence rates on Franke function. Numbers in parentheses are ratio of previous to current column and should match the value in the "Theory" column.

The problem with using my samplings of the Franke function as test data sets for convergence is that the multiple peaks/valleys of this data set mean that the asymptotic behavior does not occur until locally most areas appear as a single peak or valley. However, with samplings this dense, the error is already small, and we run out of floating point precision before the asymptotic convergence can be observed.

As a better test of convergence, I sampled the function

$$z(x, y) = \sin(\pi x/2 + \pi/4)\sin(\pi y/2 + \pi/4) \tag{1}$$

over the $[0, 1]$ square. This function is between 0.5 and 1 in this region, and since it is a single peak we should be able to observe the asymptotic behavior earlier than with the Franke function. Further, since the values of this function over this region are close to 1, the absolute error is approximately the relative error, and we can readily tell when we have run out of floating point precision.

Table 4 shows the results of fitting my interpolants to samplings of this function. From this table, we see that the convergence rates are almost exactly as predicted, with a few expectations. First, going from interval size 1 to 0.5 we have slightly worse behavior, suggesting that the interval size of 1 is too large to observe the asymptotic behavior. Second, the degree 9 interpolants are performing worse than expected, especially scheme 2. However, the values for degree 9 interpolants are reasonably close to the prediction (the last values are extremely poor, but this is due to loss of numerical precision).

The other surprise is scheme 3, which performed far better than expected. It is unclear from this single data set if the scheme was converging more rapidly because it hadn't reached its asymptotic behavior, or if it actually has better convergence properties than might otherwise be expected.

In short, the convergence test for this second function (Equation 1) strongly supports the theoretically predicted error terms.

4.3 Shaded Images

I next generated some isophote plots of a 5×5 sampling of the Franke function, and shaded images of the surfaces fit to an 8×8 sampling of the Franke function. I used a denser sampling of the Franke function for the shaded images, since the primary goal of the isophote plots was to analyze the changes in the surface as the continuity was increased, which was more easily observed for sparser samplings of the Franke function. The shaded images were for evaluating the shape of the constructed surfaces, which for lower degrees required denser samplings to obtain reasonable looking surfaces.

Method	Theory	1	0.5	0.25	0.125	0.0625
m1, d3	16	0.10455	0.0074179 (14.1)	0.00047831 (15.5)	3.0128e-05 (15.9)	1.8866e-06 (15.9)
d5	64	0.0087873	0.00015189 (57.9)	2.4334e-06 (62.4)	3.8259e-08 (63.6)	5.9872e-10 (63.9)
d6	128	0.00039143	3.7182e-06 (105)	3.2361e-08 (115)	2.5920e-10 (125)	2.0372e-12 (127)
d9	1024	1.0777e-05	1.1297e-08 (953)	1.1228e-11 (1006)	1.1213e-14 (1001)	2.7756e-15 (4)
m2, d3	16	0.11291	0.0081842 (13.8)	0.00053368 (15.3)	3.3711e-05 (15.8)	2.1125e-06 (16)
d5	64	0.0089772	0.00015189 (59.1)	2.4334e-06 (62.4)	3.8259e-08 (63.6)	5.9872e-10 (63.9)
d9	1024	6.1818e-05	7.4456e-08 (830)	7.6825e-11 (969)	2.6013e-13 (295)	2.8099e-13 (0.926)
m3, d5	32	0.0087873	0.00015543 (56.5)	2.5967e-06 (59.8)	4.9931e-08(52.0)	1.2119e-09 (41.2)

Table 4: Convergence rates on $\sin(xy)$ function. Numbers in parentheses are ratio of previous to current column and should match the value in the "Theory" column.

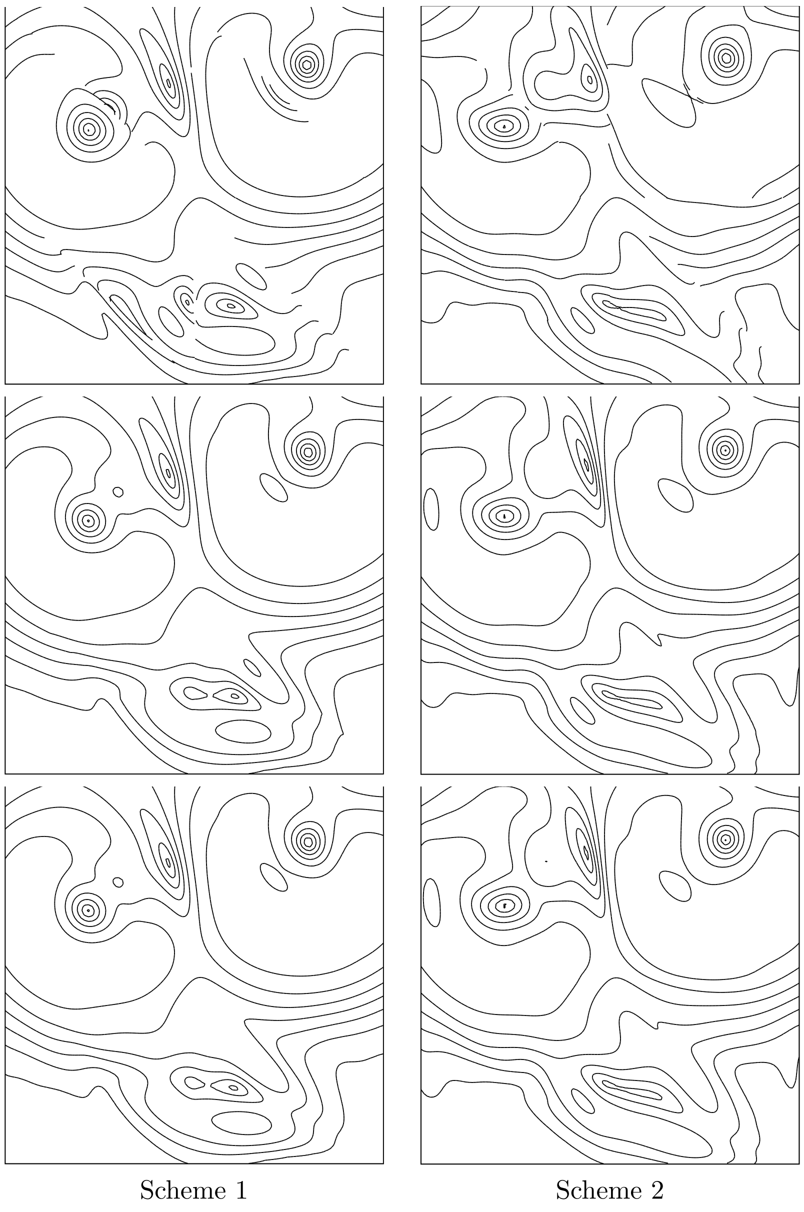

Figure 13: Degree 9 polynomials fit to Franke function. Rows: C^0, C^1, and C^2.

Figure 13 shows isophote plots of degree nine surfaces fit using schemes 1 and 2 to a sparse sampling of the Franke function. The Franke function was sampled on a 5×5 grid; patches were fit to the inner 3×3 grid of samples, creating a total of eight patches. The left column shows the first scheme; the right column shows the second scheme. The first row was computed using no continuity adjustments; the second row had just C^1 adjustments; the third row had both C^1 and C^2 adjustments.

The point of this figure is to show the effects of the continuity adjustments. In the top row, we see the expected discontinuities in the isophotes; in the second row, the isophotes are C^0, with the C^1 discontinuity being minor except in a few spots (in particular, the upper left of scheme 1). Curiously, the isophotes of the second scheme suggest it has created a better surface, but shaded images show exactly the opposite is true. Finally, note that while the last row shows an improvement in scheme 1, there are only small differences between the C^1 and C^2 surfaces for the second scheme.

To get a better feel for surface quality, I generated shaded images and curvature plots of surfaces fit to data sampled from the Franke function. Figure 14 shows a top view of the surfaces constructed by schemes 1 and 2 for degrees 3, 5, and 9 and for scheme 1, degree 6. In this figure, we can clearly see the C^0 discontinuities of the degree three interpolants. Figure 15 offers a second view of these surfaces. From this picture, we can see that for degree 5, scheme 1 produces better surfaces than scheme 2. Curvature plots of both of these degree 5 surfaces are shown in Figure 18, where we can see that the surface produced by scheme 2 has far more artifacts than the surface constructed by scheme 1.

Although not apparent in Figure 14 and 15, the degree nine surface produced by scheme 1 is of better quality than the degree nine surface produced by scheme 2. However, the shape artifacts in the degree nine surface produced by scheme 2 are subtle, and shaded images showing these artifacts do not reproduce well in print. To better see the artifacts in the degree nine surfaces, I made Gaussian curvature plots of the surfaces, shown in Figure 16 and Figure 17. In these figures, red is positive Gaussian curvature, green zero Gaussian curvature, and blue is negative Gaussian curvature. Although some of the artifacts in these figures are sampling artifacts, we can see more variation in the surfaces constructed by scheme 2 than those constructed by scheme 1.

The curvature plot of the degree 9 surface created by scheme 1 is interesting in that the curvature is very good expect in one spot. I suspect this artifact occurs because the sampling in this region is low relative to the curvature in the region. Additionally, it is interesting to note that although the degree 5 surface constructed by scheme 1 is only C^1, the Gaussian curvature plot indicates that it is close to being C^2.

Figure 18 compares scheme 3 to schemes 1 and 2 on a sampling of the Franke data set with all three schemes constructing degree 5 surfaces. Scheme 3, on the right of the figure, clearly constructed a better surface than scheme 2 (in the middle), but it is not as good of quality as the surface constructed by scheme 1 (on the left). While it is not surprising that scheme 1 did a better job than scheme 3 (since scheme 1 uses more derivatives than scheme 3 and has higher polynomial precision than scheme 3), it is a bit surprising that scheme 3 did better than scheme 2, since scheme 2 reproduces fifth degree polynomials while scheme 3 only reproduces fourth degree polynomials. In fact, the quality of scheme 2 is poor enough that I at first suspected that I had misimplemented it, but since it has the predicted polynomial precision I decided after checking the code that the poor quality is probably due to the non-locality of the data.

5 Conclusions

The purpose of this technical report was to report on the implementation of several data fitting schemes devised to test averaging as a means of achieving continuity between two patches without losing polynomial precision. The continuity construction itself worked as predicted: the desired level of continuity was achieved without losing polynomial precision. Further, the continuity scheme was easy to implement.

The implementation of the data fitting schemes devised to test the continuity scheme proved more interesting than the continuity itself. In particular, scheme 2 performed worse than expected, and appears to perform worse than the scheme 3 (the new data fitting scheme described in this paper). Further, to implement these schemes, I had to derive extensions to previous results, in particular a bivariate change of basis scheme, and a method for computing the weights of control points used to compute the derivatives of a triangular Bézier patch. I also looked at extending the idea of Farin's quadratic precision point, although such an extension is probably not worth considering for higher degree than quartic precision for quintic patches.

As a final note, while the averaging construction for obtaining continuity has only been applied here for triangular Bézier patches, the idea is easily extended to curves, and in turn, to tensor product Bézier

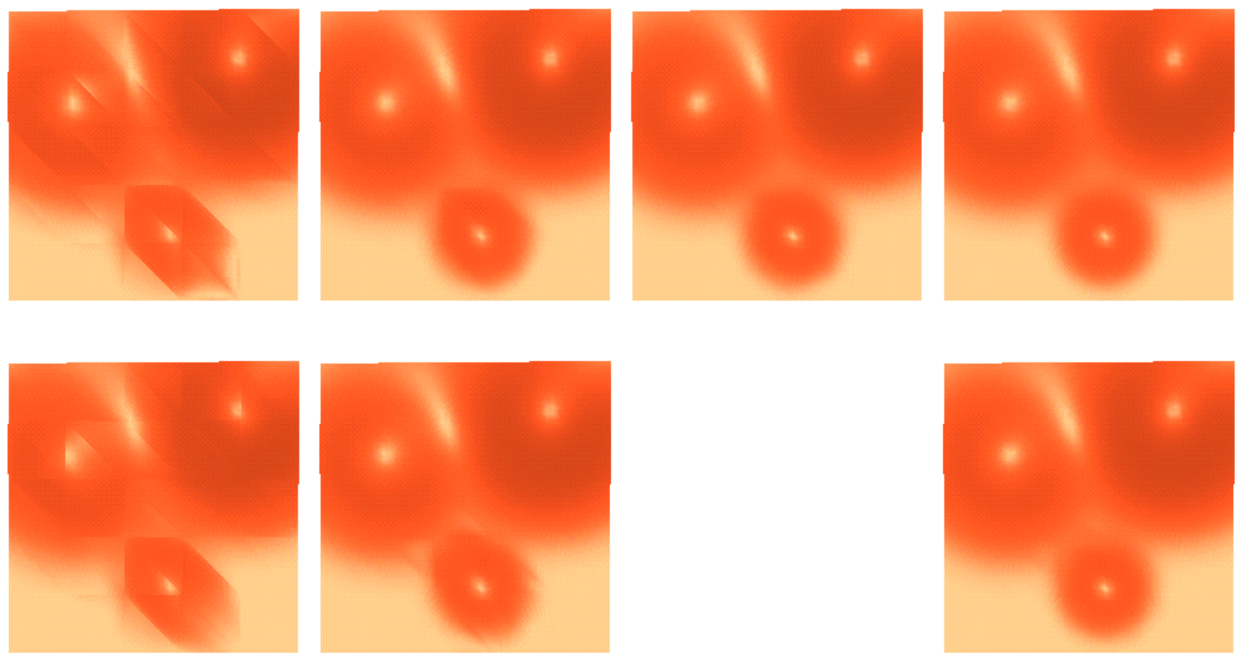
Figure 14: Surfaces fit to Franke function. Top row: scheme 1. Bottom row: scheme 2. Columns are for degrees 3,5,6, and 9.

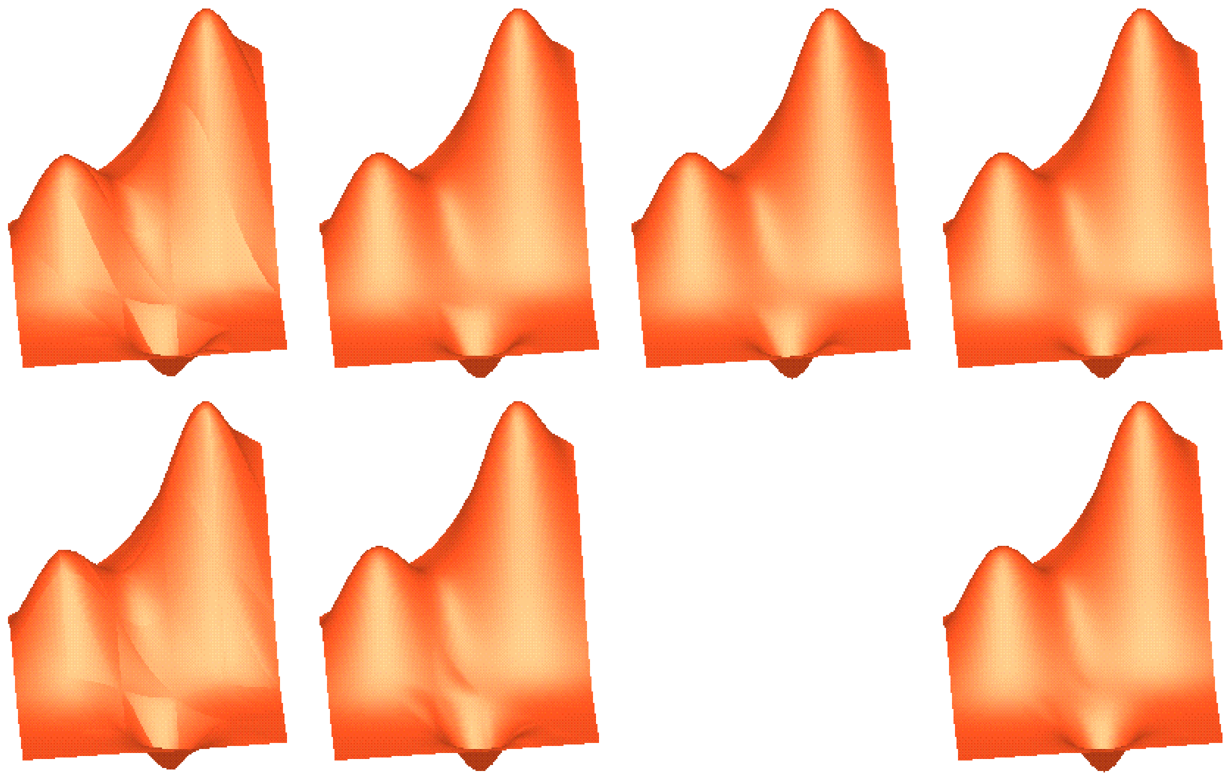
Figure 15: Surfaces fit to Franke function. Top row: scheme 1. Bottom row: scheme 2. Columns are for degrees 3,5,6, and 9.

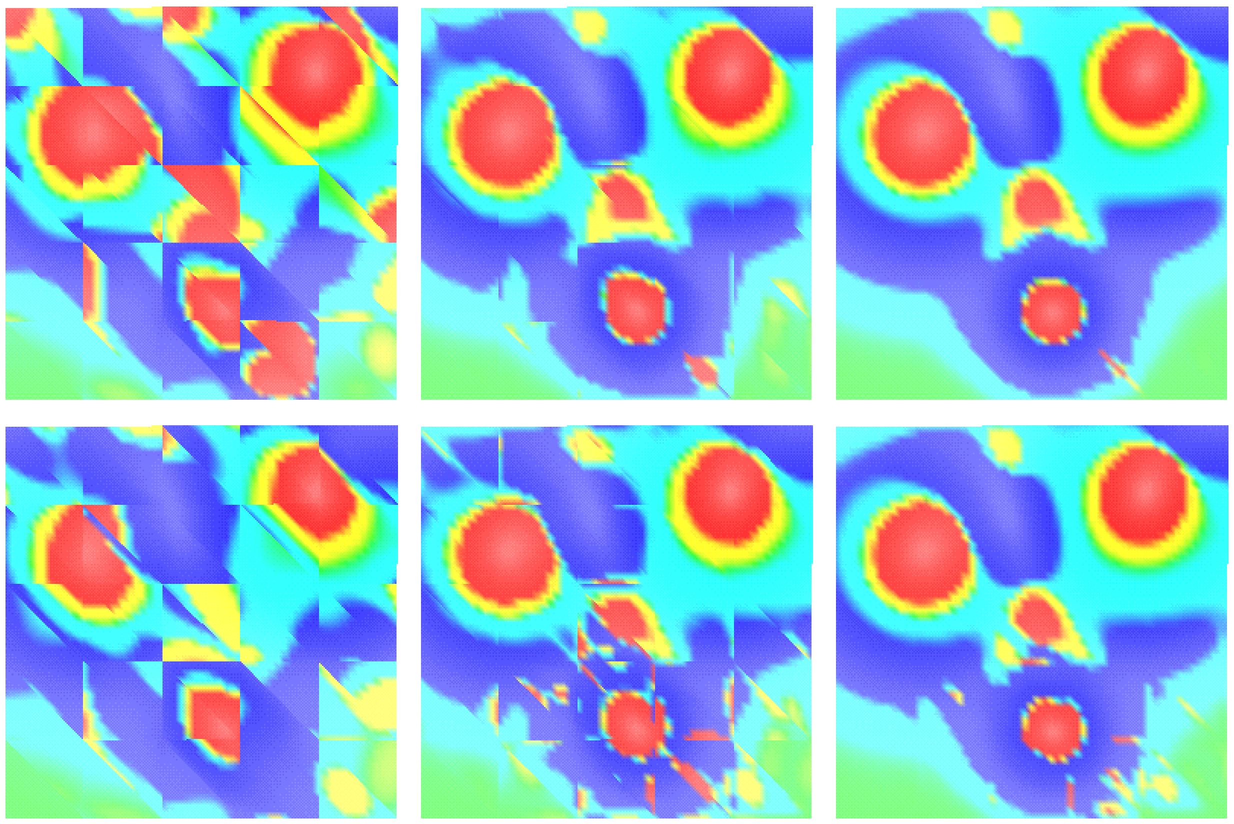

Figure 16: Curvature plots of surfaces fit to Franke function. Top row: scheme 1. Bottom row: scheme 2. Columns are for degrees 3,5, and 9.

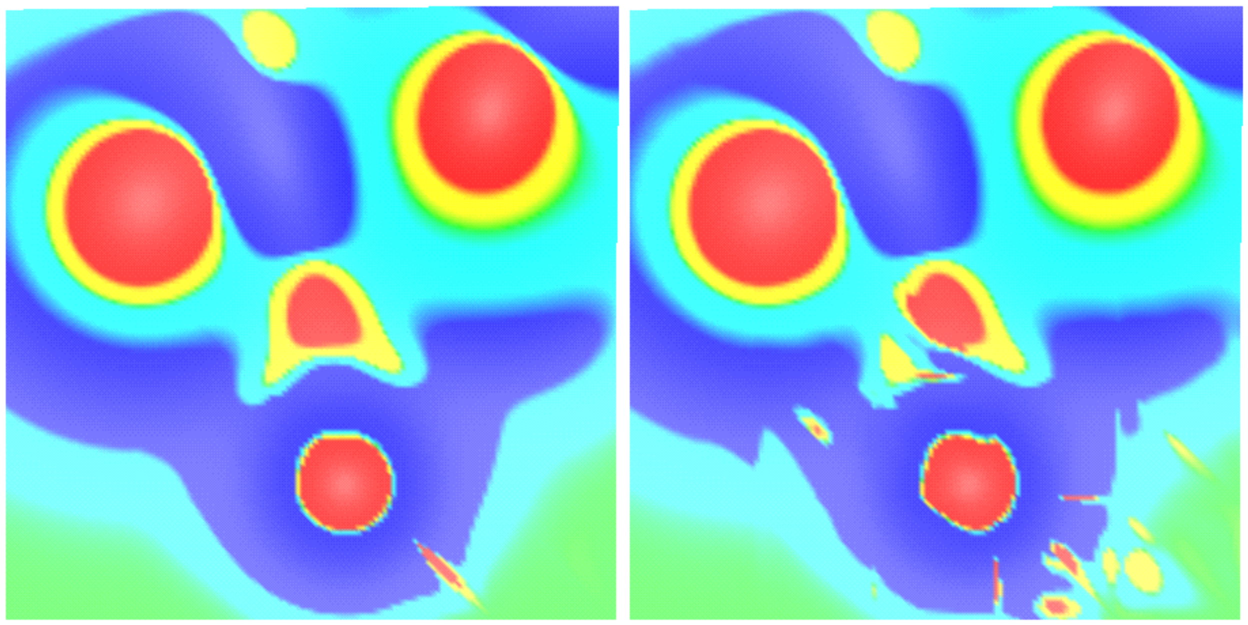

Figure 17: Curvature plots of surfaces fit to Franke function. Left: scheme 1, degree 9. Right: scheme 2, degree 9.

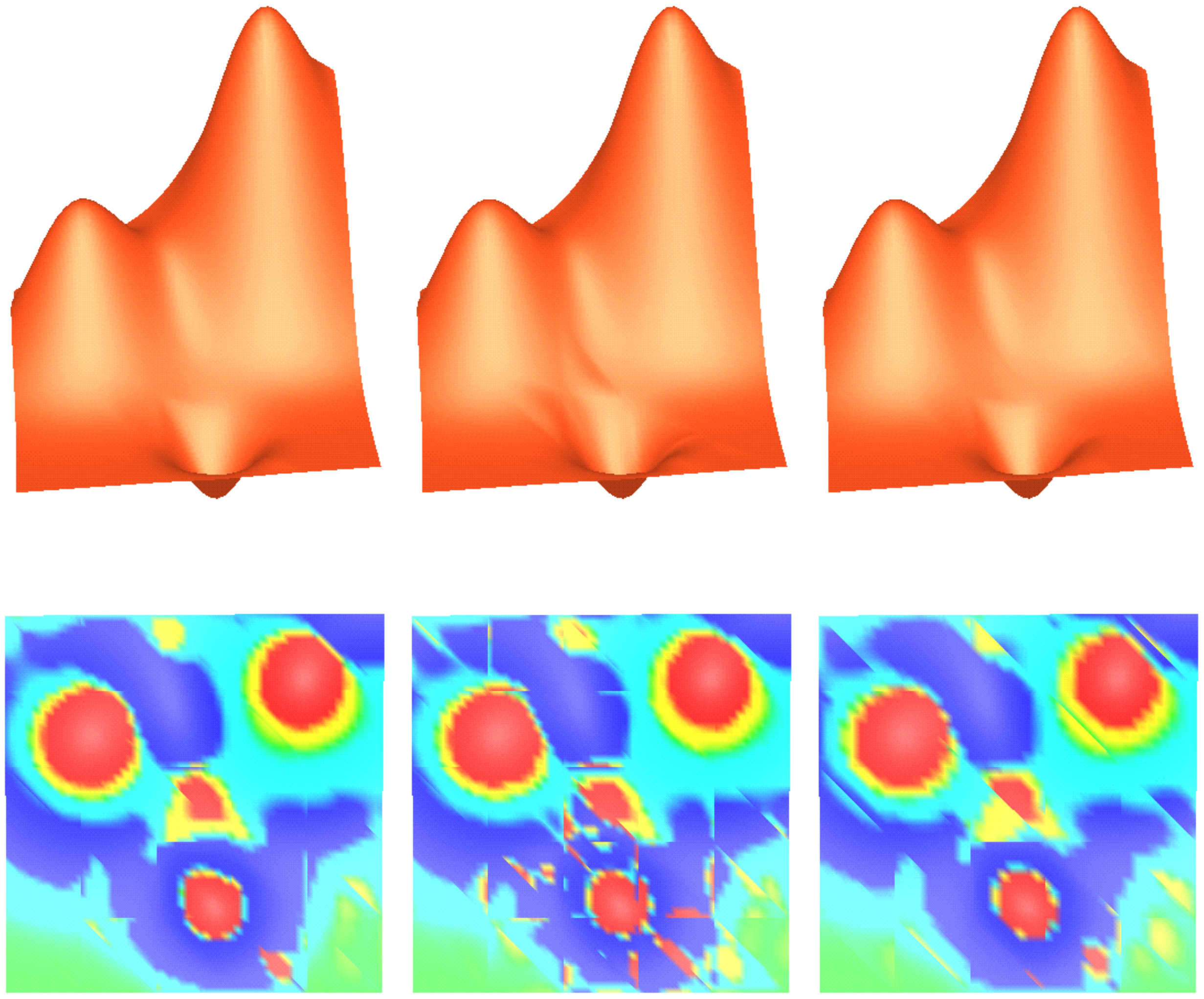

Figure 18: Surfaces fit to Franke function using the degree 5 interpolate for scheme 1, 2, and 3.

```
dfunc := proc(f,fn) local fdx, fdy, fdxdx, fdxdy, fdydy, ...;
        dprint(` f`,f,fn);
        fdx := unapply(diff(f(x,y),x),x,y);
        dprint(` fdx`,fdx,fn);
        fdy := unapply(diff(f(x,y),y),x,y);
        dprint(` fdy`,fdy,fn);
        fdxdx := unapply(diff(f(x,y),x,x),x,y);
        dprint(` fdxdx`,fdxdx,fn);
        fdxdy := unapply(diff(f(x,y),x,y),x,y);
        dprint(` fdxdy`,fdxdy,fn);
        fdydy := unapply(diff(f(x,y),y,y),x,y);
        dprint(` fdydy`,fdydy,fn);
...
        writeline(fn,`double (*partials[])(double, double) = {`);
        writeline(fn,`  f, fdx, fdy, fdxdx, fdxdy, fdydy,`);
...
        writeline(fn,`  0.};`);
end;
```

Figure 19: Maple routine to generate derivative functions.

surfaces. In the latter case, we would want (as we did for triangles) enough derivatives at the data points to consistently set enough mixed partial derivatives for the continuity adjustments we are applying to the patches.

A Using Maple to create data sets

To develop and test these schemes, I had to generate a variety of data sets. Initially, I used simple data sets (constant, linear), since it was easy to test if the code was computing the correct values for these data sets. Later, I needed data sets that had up to 6th order mixed partials. To facilitate the construction of these data sets, I wrote a C program that takes an array of C functions for the derivatives and creates a data mesh. The data mesh is sampled on a uniform grid; while a more complex sampling would perhaps better indicate how the schemes would perform on "real" data, my primary goals for these tests were merely to check (a) the polynomial reproduction and (b) the convergence rate. For these purposes, a uniform sampling was sufficient.

To generate the array of C functions for the derivatives, I wrote a Maple routine that takes the function to be sampled and generates C routines for the function and all the mixed partials up to 6th order. Part of the code for this routine appears in Figure 19; the calling sequence for generating the C routines for the Franke function appears in Figure 20.

This Maple routine generates a C file that should be linked with another C routine (`fmesh.c`) and the surface fitting library routines [6]. `fmesh` takes several parameters that allow you to select the region and density of the sampling of the function.

A few notes:

- I was using Maple V, release 4. In this version of Maple, the back-quote is the string delimiter character, and a space character is often needed at the start of the string to avoid an automatic conversion of the string to a variable. In more recent versions Maple, the double-quote is the string delimiter, and an initial space is unneeded.
- The `dprint` routine was a routine I wrote that checks handle constant functions; using the `unapply` command, constant functions are automatically converted to the constant, which won't be printed correctly. Although there are other ways to create constant functions, writing the `dprint` routine was the most obvious way for me to handle the problem.

```
> f := (x,y) -> 3./4.*exp( -(SQ(9*x-2) + SQ(9*y-2))/4. )
    + 3./4.*exp( -SQ(9*x+1)/49. - SQ(9*y+1)/10.)
    + 1./2.*exp(-( SQ(9*x-7) + SQ(9*y-3) )/4. )
    - 1./5.*exp(-SQ(9*x-4) - SQ(9*y-7));
> SQ := x->x^2;
> readlib(C);
> read(ffunc);
> dfunc(f,'f1.c');
```

Figure 20: Maple sequence to generate C derivative routines for the Franke function.

References

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[2] Phillip J. Barry and Ronald N. Goldman. Algorithms for progressive curves. In R Goldman and T Lyche, editors, *Knot Insertion and Deletion Algorithms for B-Spline Curves and Surfaces*, pages 11–63. SIAM, 1993.

[3] Gerald Farin. Smooth interpolation to scattered 3d data. In R.E. Barnhill and W. Böhm, editors, *Surfaces in CAGD*, pages 43–63. North-Holland, 1983.

[4] Thomas A. Foley and Karsten Opitz. Hybrid cubic bézier triangle patches. In T Lyche and L Schumaker, editors, *Mathematical Methods for Computer Aided Geometric Design II*, pages pp 275–286. Academic Press, 1992.

[5] Ronald Goldman and Phillip Barry. Wonderful triangle: A simple, unified, algorithmic approach to chnge of basis procedures in computer aided geometric design. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design II*, pages 297–320. Academic Press, 1992.

[6] Michael Lounsbery, Stephen Mann, Charles Loop, David Meyers, James Painter, Tony DeRose, and Kenneth Sloan. A testbed for the comparison of parametric surface mehtods. In *SPIE/SPSE Symposium on Electronic Imaging Science and Technology, Santa Clara, CA.*, 1990.

[7] Stephen Mann. Triangular interpolants to scattered data with high degree polynomial precision. Technical Report CS-2000-01, University of Waterloo, January 2000. Available as `ftp://cs-archive.uwaterloo.ca/cs-archive/CS-2000-01/`.

[8] Stephen Mann and Wayne Liu. An analysis of polynomial composition algorithms. Technical Report CS-95-24, University of Waterloo, 1995. Available as `ftp://cs-archive.uwaterloo.ca/cs-archive/CS-95-24/`.