Investigating the Accuracy of a Scheme for Tricubic Interpolation

Erick Velez

Department of Mathematics, San Francisco State University, San Francisco, CA 94132

Investigating the Accuracy of a Scheme for Tricubic Interpolation

An Independent Study
Presented to the Department of Mathematics
San Francisco State University
in partial fulfillment of the requirements for
MATH 699 Independent Study
by

Erick Velez
San Francisco, California

May 22, 2020

Contents

Introduction

In 2005, Francois Lekien and Jerrold Marsden presented a scheme for tricubic interpolation presented in *International Journal for Numerical Methods in Engineering.*

The motivation for developing this scheme of comes from the authors' work in ocean dynamics. Their work in ocean dynamics involved smooth Lagrangian structures. Through linear interpolation of the data, however, they found that the results were not smooth. Thus, their focus in developing this scheme was smoothness of the interpolation. Indeed, the major findings of this paper reveal that a tricubic interpolant is required to guarantee C^1 smoothness.

Along with their findings, a software library was created implementing their scheme. We will observe the accuracy of their algorithm by building a program that interacts with the library.

CHAPTER 1

Tricubic Interpolation

The basis of the tricubic interpolation is a regular mesh, but we take it as a unit cube. The function that is interpolated throughout the cube takes the form

$$f(x,y,z) = \sum_{i,j,k=0}^{N} a_{ijk} x^i y^j z^k \tag{1.1}$$

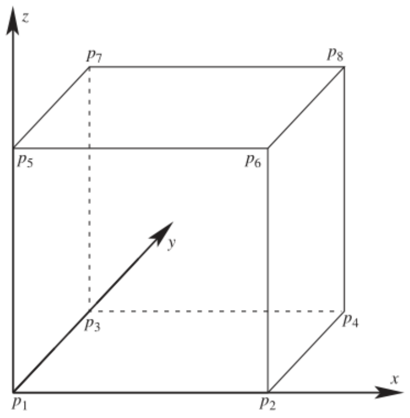
There are $N+1$ coefficients associated with this piecewise polynomial corresponding to the x, y, and z coordinates. Thus, since $N = 3$ we must have, and find, 64 coefficients of the form a_{ijk}.

1. Constraints on the Coefficients

32 of the necessary coefficients are found naturally through the corners of the unit cube. f is assumed to be known at the corners of the cube, yielding 8 coefficients. In order to guarantee C^1 smoothness, then the derivatives of the function must also exist at these corners. Then 24 more coefficients are yielded through the derivatives

$$\left\{ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right\}$$

It is shown later that not only does this guarantee C^1 at the corners of the cube but also for the entire interpolant.



The next 32 coefficients are chosen with smoothness in mind. Thus, Lekien and Marsden use additional derivatives at the 8 corners. These are

$$\left\{ \frac{\partial^2 f}{\partial x \partial y}, \frac{\partial^2 f}{\partial x \partial z}, \frac{\partial^2 f}{\partial y \partial z}, \frac{\partial^3 f}{\partial x \partial y \partial z} \right\}$$

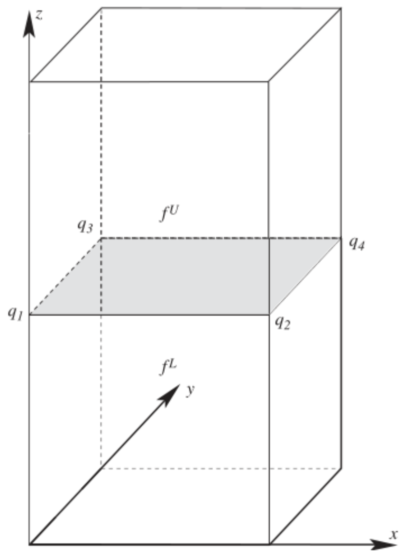
which are justified by analysis.

2. Continuity Across the Faces of the Unit Cube

A series of lemmas are presented to show that the function f and its first derivatives are continuous across the faces of the cube and that the previous derivatives are the only ones that guarantee our desired smoothness. Lekien and Marsden present proofs for these lemmas, but we will discuss them briefly.

Lemma 1.1. *The interpolated function f is continuous on the horizontal faces of the elements.*

This is the lengthiest proof presented. To prove the lemma, two unit cubes are stacked vertically.



Both have their own interpolating functions distinguished by being the upper or lower cube.

$$f^L(x, y, 1) = \sum_{i,j,k=0}^{N} a^L_{ijk} x^i y^j z^k \tag{1.2}$$

$$f^U(x,y,0) = \sum_{i,j,k=0}^{N} a^U_{ijk} x^i y^j z^k \tag{1.3}$$

Notice that value for z is fixed for both interpolation equations. Lekien and Marsden convert these into functions of x and y (a common theme in these lemmas) such that, for (1.2),

$$f^L(x,y,1) = \sum_{i,j=0}^{N} b^L_{i,j} x^i y^j \tag{1.4}$$

where

$$b^L_{ij} = \sum_{k=0}^{N} a^L_{ijk} \tag{1.5}$$

The interpolant becomes a bicubic polynomial. Skipping some technical details, they derive the linear equation

$$\vec{p} = M\vec{\beta} \tag{1.6}$$

where $\vec{p}$ is a column vector. It contains the values of f and its derivatives at the four corners of the face shared by the stacked unit cubes. The derivatives it uses are

$$\left\{ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial x \partial y} \right\}$$

β is a vector containing the coefficients b^L_{ij} and M "describes the relationship between the coefficients of the interpolant to a horizontal plane".

We can construct an equation like (1.6) for the other cube, giving us.

$$M\vec{\beta}^L = \vec{p} = M\vec{\beta}^U \tag{1.7}$$

Solving this gives us

$$M^{-1}M\vec{\beta}^L = M^{-1}M\vec{\beta}^U \tag{1.8}$$

$$\vec{\beta}^L = \vec{\beta}^U \tag{1.9}$$

Thus, the coefficients of the interpolants are equal, and so the function must be continuous along the *horizontal* faces of the cube.

LEMMA 1.2. *The derivatives*

$$\left\{\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial^2 f}{\partial x \partial y}, \frac{\partial^2 f}{\partial x^2}, \frac{\partial^2 f}{\partial y^2}, \frac{\partial^3 f}{\partial x^2 \partial y}, \frac{\partial^3 f}{\partial x \partial y^2}. \frac{\partial^3 f}{\partial x^3}, \frac{\partial^3 f}{\partial y^3}\right\}$$

are continuous on the horizontal faces of the elements.

This proof is simple as the work was done for the first lemma. Since the tricubic interpolant can be expressed as a bicubic polynomial, then the derivatives in x and y are identical on the faces. Derivatives in respect to z must be proven separately.

LEMMA 1.3. *The partial derivative $\frac{\partial f}{\partial z}$ is continuous on the horizontal faces of the elements.*

When taking the derivative of f with respect to z, f becomes

$$\frac{\partial f}{\partial z}(x, y, z) = \sum_{i,j=0}^{N} \sum_{k=1}^{N} a_{ijk} k x^i y^j z^{k-1} \tag{1.10}$$

We take similar reasoning as lemma 1.1 and make f a function of only x and y. We also define g such that

$$g = \frac{\partial f}{\partial z} \tag{1.11}$$

Again, by the reasoning of lemma 1.1, the following derivatives

$$\left\{g, \frac{\partial g}{\partial x}, \frac{\partial g}{\partial y}, \frac{\partial^2 g}{\partial x \partial y}\right\}$$

are identical for both the upper and lower unit cubes.

LEMMA 1.4. *The derivatives*

$$\left\{\frac{\partial^2 f}{\partial x \partial z}, \frac{\partial^2 f}{\partial y \partial z}, \frac{\partial^3 f}{\partial x \partial y \partial z}, \frac{\partial^3 f}{\partial x^2 \partial z}, \frac{\partial^3 f}{\partial y^2 \partial z}\right\}$$

are continuous on the horizontal faces of the elements.

Similarly to lemma 1.2, the work was done in the previous lemma. $\frac{\partial f}{\partial z}$ is a polynomial expressed in terms of x and y. Thus, its derivatives in x and y are identical across the faces of the cube.

LEMMA 1.5. *The functions*

$$\left\{f, \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}, \frac{\partial^2 f}{\partial x \partial y}, \frac{\partial^2 f}{\partial y \partial z}, \frac{\partial^3 f}{\partial x \partial y \partial z}\right\}$$

are continuous.

Lekien and Marsden present a table of derivatives that shows whether the variables x, y, z are continuous on $[0, 1]$. If all of the variables are continuous within the interval, then that derivative for f is continuous globally. They state, "the smoothness of any of these derivatives depends only on its smoothness through the faces of the elements."

THEOREM 1.6. *The tricubic interpolated function f is C^1 in three dimensions.*

PROOF. Lemma 1.5 implies that f is continuous and its three first derivatives are also continuous and therefore f is C^1. □

The set of functions which will give us 64 coefficients is

$$\left\{f, \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}, \frac{\partial^2 f}{\partial x \partial y}, \frac{\partial^2 f}{\partial x \partial z}, \frac{\partial^2 f}{\partial y \partial z}, \frac{\partial^3 f}{\partial x \partial y \partial z}\right\} \tag{1.12}$$

3. Interpolation Equation

With the necessary derivatives found to provide 64 coefficients, the interpolation equation can be defined. It takes the form of a simple linear equation.

$$B\vec{\alpha} = \vec{b} \tag{1.13}$$

where $\vec{\alpha}$ is a vector which will hold the 64 coefficients for the function defined in 1.1. $\vec{b}$ is the vector that holds all the values for each corner from the set of derivatives 1.2. B is known a priori and found similarly to how M was found in lemma 1.1. After its inverse is found, it is no longer used.

Thus, $\vec{\alpha}$ is found by inverting B

$$\vec{\alpha} = B^{-1}\vec{b} \tag{1.14}$$

Our progress then is dependent on finding $\vec{b}$, the vector holding the values of f at the 8 corners evaluated in each function in (1.12). Once $\vec{b}$ is found, α is easily found.

CHAPTER 2

The Tricubic Library and Analysis Programs

As previously mentioned, Lekien and Marsden created a software library that implements their tricubic interpolation algorithm in C++.

1. The Tricubic Library

The library is very simple in structure, only containing 5 source files.

libtricubic.cpp contains two implementations of the interpolation algorithm along with helper functions. It is has a header file associated with it.

ltricubic_utils.cpp only contains two functions which help identify the x, y, z coordinates at a corner of the unit cube and identify the place of a coefficient in $\vec{\alpha}$. It also has a header file associated with it.

$coeff.h$ contains the values of the inverted B matrix initialized as a 64×64 array.

Packaged along are three code examples of using the library.

2. Developing an Analysis Program

To begin analysis of the library, a program was built which interacts with it.

Random numbers were generated to test those functions using the Mersenne Twister pseudorandom number generator built in to the *random* header in the C++ standard library. The numbers were initially generated in the interval $[0, 1]$ for the sake of the unit cube, but they can now be created between any interval.

Six different functions were used to interact with and test the library.

(1) $f(x, y, z) = x^2 + y^2 + z^2$
(2) $f(x, y, z) = \sin(x + y + z)$
(3) $f(x, y, z) = \frac{1}{\sqrt{x^2+y^2+z^2}}$
(4) $f(x, y, z) = x^2 * y^2 + z^2$
(5) $f(x, y, z) = \frac{e^{-kr}}{r}$ where $r = \sqrt{x^2 + y^2 + z^2}$ and $k = 0.5$
(6) $f(x, y, z) = e^{\sqrt{x^2+y^2+z^2}}$

These functions, and in some cases their derivatives, were implemented in C++.

Now, the general flow of the program, and indeed of interpolation, can be described:

(1) Generate test points within the interval of your container (usually the unit cube)
(2) Iterate through the functions available to test and for each, create $\vec{\alpha}$, the vector which will contain the 64 coefficients and find the 64 constraints, $\vec{b}$.
(3) Pass both to the *tricubic_get_coeff* library which will calculate $\vec{\alpha}$. Now, the tricubic interpolant has been effectively found.
(4) For all test points, pass the calculated $\vec{\alpha}$ and the test point to *tricubic_eval*, which will return an approximation for that test point.

This is the main engine of the program.

3. Measuring Accuracy

After getting the approximate value for each test point, the exact value is found through the exact function used to find the values of the set of derivative constraints.

In order to measure accuracy, the following error formula was implemented

$$E = \sqrt{\frac{||V_E - V_A||}{||V_E||}} \tag{2.1}$$

where V_E and V_A are the lists of exact and approximate values. A *list* data structure is used to ensure order of the values such that the values coming from the same test point are compared.

4. Moving the Unit Cube

The next step in analysis was moving the unit cube to a different interval. Recall that x, y, and z for the unit cube in Lekien and Marsden are defined between $[0, 1]$. This component of the program moves the unit cube horizontally. For example x, y, z will be defined between $[2, 3]$.

In order to do this, the components of every coordinate is transformed through the formula

$$C = a + (b - a) * c \tag{2.2}$$

where C is the coordinate's new shifted component, a is the new minimum value of the interval, b is the new maximum of the interval, and c is the original coordinate's component, with $c \in [0, 1]$.

This is first done to the corners of the original unit cube as they are needed immediately to form $\vec{b}$ to calculate $\vec{\alpha}$. However, the function that creates the interpolant does not use the shifted test points. Instead, it uses the originals.

This was a logical misstep that occurred during development. It would not make sense for the interpolant to use coordinates shifted to a different interval since it is already being interpolated within a structure confined to the interval.

For example, an interpolant is created within a unit cube on the interval $[2, 3]$. A test coordinate was created in the interval $[0, 1]$, our original interval, and let it be $(0.5, 0.5, 0.5)$. The shifted coordinate would be $(2.5, 2.5, 2.5)$. Originally, the program used the shifted coordinate with the interpolant. The logical misstep comes when not realizing that the algorithm interpolates for the *entire* unit cube. $(0.5, 0.5, 0.5)$ is equivalent to $(2.5, 2.5, 2.5)$.

Thus, the exact function, the one we use to compare the approximated values against, does use the shifted test points.

5. Working with Non-unit Enclosures

Our last section of interest is working with enclosures that are not perfect unit cubes.

For the purpose of the program and this paper, enclosures will refer to a structure which contains the interpolant or its unit cube. Therefore, a unit cube is an enclosure, but a different enclosure might contain a unit cube.

So far, we have worked with points that already lie within an interval that is convenient to enclose within a unit cube i.e. an interval of length 1. We may come across a set of data that is not contained within an interval of length 1.

Lekien and Marsden note that the unit cube is merely a generalization of a regular mesh that can be interpolated. The authors state, in fact, the any parallelepiped can be used with the interpolation algorithm. While we do not go as far to generalize to parallelepipeds, we do account for non-convenient sets of data.

We approach this problem in two ways.

5.1. Creating an Enclosure around Data. The first approach is creating an enclosure around the data. This approach is as follows:

(1) Find the minimum and maximum x, y, z values among the data set.
(2) Given a distance value, create 8 coordinates using a combination of these values that distance away.

Then, the original points used to determine the minimums and maximums is contained within the enclosure. This will inherently produce less accurate results which we will see later.

5.2. Using the data to create an enclosure. The second approach is to use the minimum and maximum x, y, z values as the corners of the enclosure itself. We transform those points to the position of a unit cube. It takes similar steps to the ones described above, but there is no distance added.

These results will be more accurate if the distance of the first form of enclosures is not large. Both are considered as either approach might be useful.

CHAPTER 3

Results

There are two main areas of focus within the results: the accuracy of the interpolation within a unit enclosure and interpolating a group of incoherent points. By incoherent points, we refer to points in space that are not defined with a concrete interval. These points could be very far or very close, but we work to interpolate them. That is where we create enclosures.

1. Interpolation Accuracy

The tests are run with 1 million test points randomly created with $x, y, z \in [0, 1]$. A set is used to ensure uniqueness of the points.

Functions	Error
$f(x,y,z) = x^2 + y^2 + z^2$	7.9349e-17
$f(x,y,z) = sin(x+y+z)$	0.00457217
$f(x,y,z) = \frac{1}{\sqrt{x^2+y^2+z^2}}$	Undefined
$f(x,y,z) = x^2 * y^2 * z^2$	0
$f(x,y,z) = \frac{e^{-kr}}{r}$	0.752251
$f(x,y,z) = e^{\sqrt{x^2+y^2+z^2}}$	0.0227097

There are two functions that immediately stand out, functions 3 and 5. Function 3 is undefined when $x, y, z \in [0, 1]$, so the interpolation cannot exist. The interpolation for function 5 has a very large error. Below, we will see that different positions of the unit cube will actually make the error better (smaller). The sixth function is small, but it will also become smaller.

1.1. Shifting the unit cube.

Now, we will change the interval of where x, y, z are defined. This is done as described in Section 4 of Chapter 2. A set of 1 million test points is generated where $x, y, z \in [0, 1]$ for the first interval. They are then shifted to the appropriate interval for each iteration.

Functions	$[0,1]$	$[1,2]$	$[2,3]$
$f(x,y,z) = x^2+y^2+z^2$	7.9349e-17	1.9978e-16	1.35237e-16
$f(x,y,z) = \sin(x+y+z)$	0.00457217	0.00456053	0.00453597
$f(x,y,z) = \frac{1}{\sqrt{x^2+y^2+z^2}}$	Undefined	0.00141072	0.000213935
$f(x,y,z) = x^2 * y^2 + z^2$	0	2.52585e-16	2.82614e-16
$f(x,y,z) = \frac{e^{-kr}}{r}$	0.752251	0.00333079	0.000721183
$f(x,y,z) = e^{\sqrt{x^2+y^2+z^2}}$	0.0227097	0.00355041	0.00221184 9

Functions	$[3,4]$	$[4,5]$	$[5,6]$
$f(x,y,z) = x^2+y^2+z^2$	1.42228e-16	1.14356e-16	1.4049e-16
$f(x,y,z) = \sin(x+y+z)$	0.00449571	0.00443497	0.0043463
$f(x,y,z) = \frac{1}{\sqrt{x^2+y^2+z^2}}$	6.62155e-5	2.8518e-5	1.48252e-5
$f(x,y,z) = x^2 * y^2 + z^2$	2.68586e-16	3.10452e-16	2.8978e-16
$f(x,y,z) = \frac{e^{-kr}}{r}$	0.00029951	0.000161326	9.88411e-5
$f(x,y,z) = e^{\sqrt{x^2+y^2+z^2}}$	0.0016496	0.00133244	0.00113022

Functions	$[6,7]$	$[7,8]$	$[8.9]$	$[9,10]$
$f(x,y,z) = x^2+y^2+z^2$	1.28342e-16	1.40958e-16	1.07114e-16	1.62401e-16
$f(x,y,z) = \sin(x+y+z)$	0.00421901	0.00404018	0.0038024	0.00352879
$f(x,y,z) = \frac{1}{\sqrt{x^2+y^2+z^2}}$	8.69061e-6	5.53461e-6	3.74449e-6	2.65279e-6
$f(x,y,z) = x^2 * y^2 + z^2$	2.95463e-16	2.78401e-16	3.19885e-16	3.14888e-16
$f(x,y,z) = \frac{e^{-kr}}{r}$	6.48723e-5	4.40839e-5	3.02741e-5	2.0535e-5
$f(x,y,z) = e^{\sqrt{x^2+y^2+z^2}}$	0.000991672	0.000892006	0.000817732	0.000760868

It's easily observed that for a quadratic function like $f(x,y,z) = x^2+y^2+z^2$, the error value stays near 0. The cubic interpolant approximates it exactly in all intervals. As we noted before, $f(x,y,z) = \frac{1}{\sqrt{x^2+y^2+z^2}}$ is undefined when $x,y,z \in [0,1]$. When it is moved out of that interval, we can observe the accuracy of the interpolation, which also approaches 0.

Here, we only present 9 steps of shifting (the original interval $[0,1]$ is not shifted). The program can take any number of steps desired, but we take 9 for clarity.

2. Irregular Enclosures

We test irregular enclosures similarly to the normal accuracy testing. 8 points are randomly generated (the same number as there are corners in a cube). They are then transformed to the positions of a unit cube on the interval $[0,1]$. This done using both "distanced" enclosures (the first kind described) and "in-place" enclosures (the second). Here is a sample of those results when the distanced enclosure is 1×10^{-14} away.

Functions	Distanced	in-place
$f(x,y,z) = x^2 + y^2 + z^2$	0.229371	0.22933
$f(x,y,z) = \sin(x+y+z)$	0.143818	0.144135
$f(x,y,z) = \frac{1}{\sqrt{x^2+y^2+z^2}}$	0.422477	0.42253
$f(x,y,z) = x^2 * y^2 + z^2$	2.71192	2.71116
$f(x,y,z) = \frac{e^{-kr}}{r}$	0.577764	0.577823
$f(x,y,z) = e^{\sqrt{x^2+y^2+z^2}}$	0.0999504	0.0999624

It should be noted that the error values for the fourth function are quite large. This is potentially due to a small discrepancy in the z values of the transformed unit cube. Sometimes, the z value is not positioned perfectly to where it should be to form the unit cube (usually 0.5 away). This will need to be fixed, but these results are still interesting when the distance is changed.

After manually checking, the discrepancy between a distanced and in-place enclosure becomes more apparent when the distance is 1×10^{-5}.

Functions	Distanced	In-place
1	0.176981	0.176961
2	0.161059	0.161011
3	0.371554	0.371447
4	0.163298	0.163413
5	0.363027	0.362885
6	0.0998448	0.0998228

Here, we see an expected trend. The in-place enclosure, which should have better accuracy in all places, does. In most cases, the advantage is tiny, such as the first function where in-place is smaller by only 0.00002.

Of course, when the distance is 1 (quite large), in-place is far more accurate. The distanced enclosure is too far away, thus interpolating so much more space, to give accurate results.

3. Future Direction

There are a number of directions the analysis program can be taken. There is also much work to do on the existing program.

First, the enclosure creation and cube shifting components do not interact. Ideally, we would expect the results to be exactly the same as if we tested points in the normal unit cube and shifted it (as in Section 1.1 of this chapter).

General improvements such as being able to choose how many points to test, what interval to start in, etc. would be great to have in a finished software package. This also brings up the point that the results we see here are taken from one execution of the program. This could be flawed, especially since the testing points are randomly generated on each start of the program.

For quadratic function, this will not be a problem. If we ran the program several times and took the mean of the error on a single interval for a quadratic, we would expect it to be near 0. For cubic functions and higher, it might prove insightful (and more reliable) to take the mean over several iterations of the program.

Appendix A. Technical Details

The analysis program was originally developed on an Intel CPU, but some development also occurred on an AMD CPU. It would be interesting to see the differences (if there were any) to how the smaller numbers were represented between the two. Determining which CPU was more accurate (if there were a difference) would also be interesting, but would require identical points tested across both machines.

Modern hardware should be competent enough to produce near-identical results, but with such a hardware-dependent language as C++, it could be worthwhile.

The code can be found at this link. There is still much improvement to be done on the code base, including creating detailed documentation.

From an organization perspective, I do not believe it would make sense to create a new version of Lekien and Marsden's library including this analysis program. It would create clutter that does not align with the original intent of their software: a simple interpolation library. If this program was a traditional testing program (to ensure the functionality of software), then it would make sense. The analysis program is a separate entity that contains the tricubic library within it.

Bibliography

[1] Francois Lekien, Gerrold Marsden, *Tricubic interpolation in three dimensions*, International Journal for Numerical Methods in Engineering, 2005.