n-Simplex Interpolation

Peter Hemingway
Client and Media Systems Laboratory
HP Laboratories Bristol
HPL-2002-320
November 20th , 2002*

E-mail: peterhemingway@acm.org

interpolation, colour, frame buffer algorithms, halftoning & dithering, level of detail algorithms, rendering, printing, texture mapping

Interpolation has wide application, and in the field of computer graphics is used for colour maps, sampling pixel images and generating points on surfaces. The performance of an interpolation algorithm is important for computer graphics because the computation is typically carried out repetitively over the typically large number of data points in an image with the modest resources of an appliance like a digital camera. The paper describes a new and practical method of interpolation that selects fewer points to interpolate between. Using fewer points for the interpolation is computationally more efficient because it reduces the number of table accesses and because there are fewer terms and operators in the interpolation equation. It is particularly good when interpolating between values that are close together (i.e. large tables) and can be just as accurate as methods that use more points. The interpolation method is based on n-simplexes so that it can be applied to problems in any dimension.

* Internal Accession Date Only
Approved for External Publication

n-Simplex Interpolation

Peter Hemingway
Hewlett Packard Laboratories
Filton Road, Stoke Gifford
Bristol BS34 8QZ

peterhemingway@acm.org

ABSTRACT

Interpolation has wide application, and in the field of computer graphics is used for colour maps, sampling pixel images and generating points on surfaces. The performance of an interpolation algorithm is important for computer graphics because the computation is typically carried out repetitively over the typically large number of data points in an image with the modest resources of an appliance like a digital camera.

The paper describes a new and practical method of interpolation that selects fewer points to interpolate between. Using fewer points for the interpolation is computationally more efficient because it reduces the number of table accesses and because there are fewer terms and operators in the interpolation equation. It is particularly good when interpolating between values that are close together (i.e. large tables) and can be just as accurate as methods that use more points. The interpolation method is based on n-simplexes so that it can be applied to problems in any dimension.

Keywords

Interpolation; Colour; Frame Buffer Algorithms; Halftoning and Dithering; Level of Detail Algorithms; Printing; Rendering; Texture Mapping.

1. INTRODUCTION

Look-up tables are a useful method of representing complex functions. If the function has a large number of parameters, or requires high precision, then the lookup table can become unacceptably large. A compromise is to reduce the number of entries in the lookup table and interpolate between a smaller number of data points. Interpolation has wide application and one of its applications in the field of computer graphics is for colour maps[1][2][7], sampling pixel images and generating points on surfaces. The performance of an interpolation algorithm is particularly important for computer graphics because the computation is typically carried out repetitively over a very large number of data points in an image.

Consider the problem of colour mapping between a camera and a printer. Each device has its own colour space, but the camera must convert to, and the printer must convert from, a common (shared) colour space. If the camera and printer are both connected to a device like a PC then the PC can mediate and perform the mapping from one colour space to another, but if a camera is to communicate directly with a printer, then each device needs to provide device specific colour mapping to a common interchange space like sRGB [8].

This paper describes a new method that interpolates between fewer points. Using fewer points for the interpolation is computationally more efficient and yet can be just as accurate as methods that use more points. The improved efficiency is achieved in two ways: because the algorithm uses fewer table accesses, and also because there are fewer terms and operators in the interpolation equation. The algorithm is particularly applicable to peer-to-peer systems where each device must take on its own share of computation when image data is transferred between peers.

2. BACKGROUND

There are many different methods of interpolation, but they all use the most significant part of the input values to address lookup table elements, and the least significant part of the input values for interpolation. Tri-linear interpolation is a well-known technique for three dimensions that uses eight entries from a sparse table (a coarse lattice) to estimate the required value, by calculating a weighted average of the eight points.

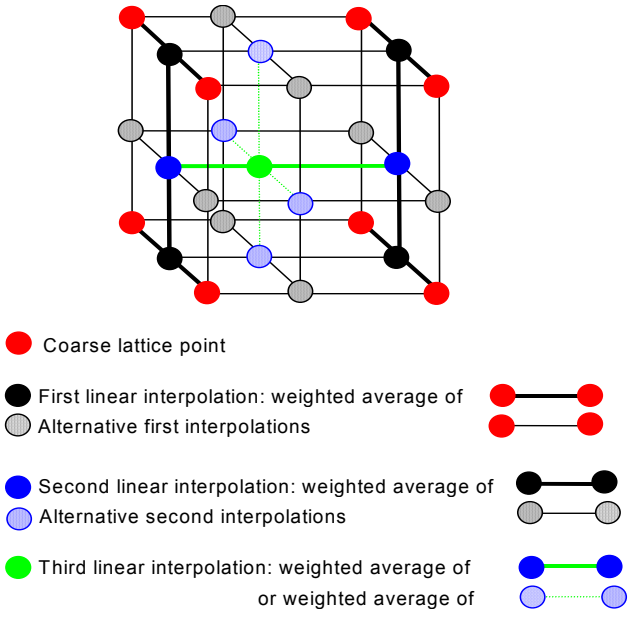


Figure 1. Tri-linear Interpolation

Tri-linear interpolation involves looking up eight table values and then calculating the weighted average that will involve eight multiplications and seven additions.

Radial interpolation [3][4] reduces the number of table accesses from eight to five. Tetrahedral interpolation [5] further reduces the number of table accesses to four. This is an improvement over tri-linear and other methods that use eight coarse lattice points for interpolation because the speed of the interpolation can be improved by using fewer table entries in the averaging step (and keeping the table size the same). This has a twofold effect: the number of table memory accesses is reduced, and the averaging can be performed with fewer multiplications and additions. Table memory accesses can have significant overhead because the table may be packed.[1]

When applied to a three dimensional problem, the n-simplex interpolation method [6] described here uses less than four table accesses, with corresponding simplification of the weighted averaging calculation. Although the interpolation is in three dimensions, interpolation within a solid (that must have a minimum of four vertices to exist in three dimensions) is not always necessary, for example when the interpolation is within a plane of the lookup table lattice. By detecting these special cases, without incurring an overhead, it is possible to use less than four values for the interpolation in three dimensions.

If the interpolation points lie within a plane rather than the volume of a solid, then only three points are required (and to minimize errors, these should be the nearest three points). Where the points lie on a face of the tetrahedron then could be treated as an interpolation between three points that make up that face of the tetrahedron. Some of these points will lie on an edge and so lie between two vertices of the tetrahedron and sometimes interpolation will not be required at all if the point coincides with one vertex of the tetrahedron. By optimising tetrahedral interpolation for these cases then the interpolation needs, on average, fewer than four table accesses for each of the interpolations. Although this example uses three dimensions, because the interpolation method is based on n-simplexes, the optimisation applies equally to other dimensions of interpolation by detecting the optimisation cases and treating them as lower dimension interpolations.

3. ALGORITHM

n-Simplex interpolation takes advantage of the fact that within a sparse lookup table, some interpolations can use fewer points. Some input values will fall precisely on the sparse lattice, in which case the result is known from the table entry and no averaging is required. If the table is large and the distance between lattice points is small, this will happen more frequently than when the table is smaller with a larger gap between coarse lattice points. The distance between coarse lattice points, K, is conveniently represented as $K = 2^N$ so that $N = \log_2(K)$. If N = 0, then the lattice distance is 2^0 and elements are adjacent (K = 1). In this case the sub-cube is the lattice point so there is no interpolation. If N = 3, then the distance between lattice points in any direction is 2^3, and the volume bounded by adjacent lattice points that make up a sub-cube is $(2^3)^3$.

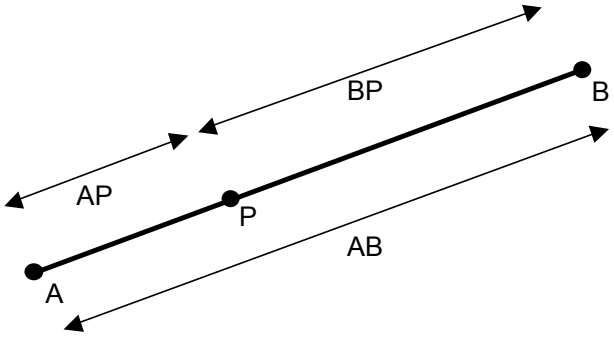
Of those input points that do not fall directly on the lattice, some will fall directly on a line between two lattice points, and here the result may be estimated as the weighted average of the two points, the weights being proportional to the distance of the input point from the lattice points.

Of those input points that do not fall on a line, some will lie within a triangle described by three coarse lattice points, and the position of the input point on the triangle determines the weight given to each of the three points when interpolating between them. The remainder of the input points must lie within a tetrahedron of four lattice points.

It will be shown that testing for these special optimisation cases can be performed without incurring any additional computational overhead.

3.1 Coefficients

Where an input point coincides with the table entry, no interpolation is required.



In two dimensions the length of a line AB with coordinates (x_a,y_a) and (x_b, y_b) is given by

$$|AB| = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$$

For a point with position P on a line length |AB| between two lattice points, position A and B, with values *a* and *b* respectively, then the interpolated value *p* is given by

$$\frac{\left(a \cdot |BP| + b \cdot |AP|\right)}{|AB|}$$

Note that this equation yields the value *a* if P coincides with A, and *b* if P coincides with B.

[1] Packed arrays use less storage but are more difficult to access.

If, and only if, P lies on AB then,

$$|AP| + |PB| \equiv |AB|$$

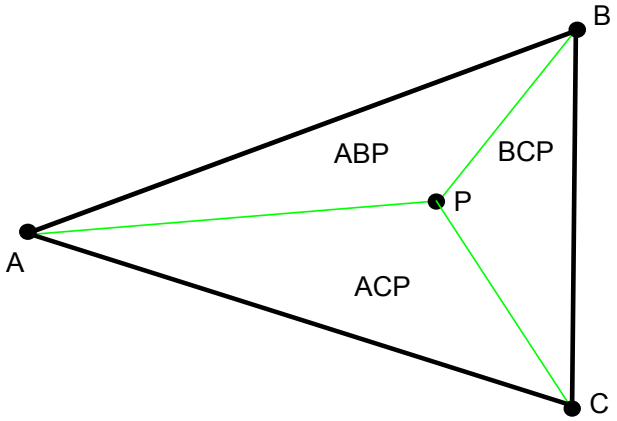
in which case

$$\frac{|AP| + |PB|}{|AB|} = 1$$

The distance between two points (x_a,y_a,z_a) and (x_b, y_b, z_b) is

$$\sqrt{\left((x_a - x_b)^2 + (y_a - y_b)^2 + (z_a - z_b)^2\right)}$$

For a point with position P within a triangle of three points



A,B,C, with values a, b and c respectively, the interpolated value p is given by

$$p = \frac{(\text{area}(A, B, P)\cdot c + \text{area}(A, C, P)\cdot b + \text{area}(B, C, P)\cdot a)}{\text{area}(A, B, C)}$$

If, and only if, P lies within the area A,B,C, then,

$$area(A,B,P) + area(A,C,P) + area(B,C,P) \equiv area(A,B,C)$$

Note that if P lies on a side, then one of the triangles will have zero area, and the equation becomes the same as that for a point on the side. If P lies on a vertex then two areas are zero, and the result is the vertex that P lies on.

The area of a triangle is given by Heron's formula:

$$\Delta = \sqrt{(s(s-a)(s-b)(s-c))}$$

where

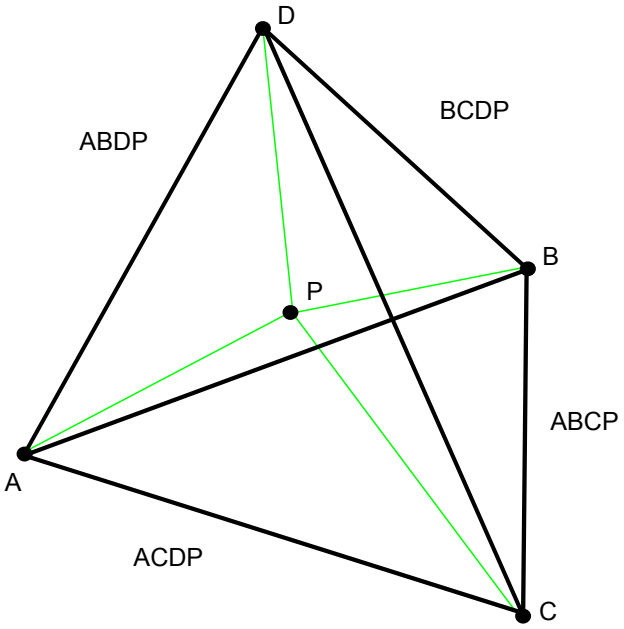
$$s = \frac{(a+b+c)}{2}$$

and *a, b* and *c* are the lengths of the three sides.

Applying the same technique in three dimensions, where *vol* represents the volume, the interpolated value *p* is given by

$$\frac{(vol(A,B,C,P)\cdot d + vol(A,B,D,P)\cdot c + vol(A,C,D,P)\cdot b + vol(B,C,D,P)\cdot a)}{vol(A,B,C,D)}$$

If, and only if, P lies in the volume A,B,C,D, then this equation will yield unity. Otherwise the equation will yield greater than unity indicating that the point P lies outside the tetrahedron A,B,C,D. If the point P lies on a face of the tetrahedron, then the volume of one of the tetrahedrons becomes zero, and the equation becomes the same as that for a point on a triangle. If P lies on an edge then two volumes will be zero (and vice versa). If P lies on a vertex, then three volumes will be zero (and vice versa).



The volume of a tetrahedron with one vertex at the origin and the other vertices at $(x_a\ y_a\ z_a)$, $(x_b\ y_b\ z_b)$, $(x_c\ y_c\ z_c)$ is given by :

$$\left|\frac{(x_a(y_b z_c - z_b y_c) - y_a(x_b z_c - z_b x_c) + z_a(x_b y_c - y_b x_c))}{6}\right|$$

This process of using n-simplexes can be extended to interpolation problems that have more than three input values by using the equation for the volume of an n-simplex.

These equations can be used to determine whether a particular point lies on a coarse lattice point, on a line, within a triangle or inside a tetrahedron, and also the relative influences of each point (the ratio of the tetrahedral volumes). Some points will fall on several lines (triangles, tetrahedrons), and so some points within the sub-cube will have several equivalent n-simplex interpolation equations. For example the point in the middle of the cube lies at the midpoint of three diagonal lines. If the space within the sub-cube is linear, then all equations should yield similar results[2].

There are eight vertices for a cube and for any point within the cube it is a simple matter of looking at all possible tetrahedral volumes by enumerating all choices of four vertices from the eight vertices that define the cubic volume and then finding out if the point of interest lies inside the tetrahedron. Once an enclosing tetrahedron is found, further tests can determine if the lies on a face (triangle), edge (line) or vertex (point) In this way the subset of the cube vertices that are involved are recorded and their coefficients derived from the ratios of volumes, areas or distance as appropriate.

The frequency with which input points fall on lines between two lattice points, within a triangle of three lattice points or within tetrahedron described by four lattice points, varies with the volume of the sub-cube which in turn is the cube of the distance between lattice points. This is shown in Table 1.

N	Inter Lattice Distance 2^N	Sub Cube Volume $(2^N)^3$	On Point	On Line	Within Triangle	In Tetra-hedron
0	1	1	1	0	0	0
1	2	8	1	7	0	0
2	4	64	1	33	30	0
3	8	512	1	85	378	48
4	16	4096	1	189	2322	1584
5	32	32768	1	397	11202	21168
6	64	262144	1	813	48918	212418
7	128	2097152	1	4579	188586	1903986

Table 1. Frequency

Assuming an even distribution of the input values, the probabilities of needing: one table access (if the input value coincides with a lattice point), or two table accesses (if the input value lies on line), or three table accesses (if the input value lies on a triangle) or at most four accesses (since the point must lie within a tetrahedron), can be calculated by dividing each frequency by the volume of the sub cube. This is shown in Table 2.

N	P (Point)	P (Line)	P (Triangle)	P (Tetra-hedron)	Av. Table Access	Best	Max
0	1.000000				1.0000	1.00	1.00
1	0.125000	0.8750			1.8750	1.00	2.00
2	0.015625	0.5156	0.4688		2.4531	1.00	3.00
3	0.001953	0.1660	0.7383	0.0938	2.9238	1.00	4.00
4	0.000244	0.0461	0.5669	0.3867	3.3401	1.00	4.00
5	0.000030	0.0121	0.3419	0.6460	3.6338	1.00	4.00
6	0.000004	0.0031	0.1866	0.8103	3.8073	1.00	4.00

Table 2. Probability

The worst case is never more than four accesses and, for small N, case is significantly the worst less than four. The best case is always one lookup, which occurs for input data that coincides with lattice points and so requires no interpolation. In some circumstances it may be useful to also perform a simple lookup with no interpolation when the input value is very close to a table lattice point. This may incur a small error but the result will be computed more quickly. If the interpolation is used for colour space mapping, snapping some values to lattice points (e.g. for background or colour business graphics) may even be

[2] In practice the results may differ, and the degree of difference is a measure of the distortion of a particular sub-cube. This can be useful in determining whether the lattice is too coarse to achieve an acceptable approximation, but is outside the scope of this paper.

advantageous if it ignores small deviations that are due to noise in the data.

For each possible point P it is possible to derive the weights and lattice points, for each interpolation point. For N=2 there are 64 points in the sub-cube[3], which can be addressed as P[x][y][z], where x, y and z are between 0 and 3 inclusive. The interpolation equations are derived using the tetrahedral areas and it is convenient to write a program to generate the equation coefficients but the efficiency of *generating* the equations is not important: it is the *execution* efficiency that is important for n-simplex interpolation. If the eight vertices of the cube (which are coarse lattice points) are referred to as V_{000}, V_{100}, V_{010}, V_{001}, V_{110}, V_{101}, V_{011}, V_{111}, then a computer program can be used to generate the following interpolation equations for N=2:

In many cases more than one interpolation equation exists. For example P[2][2][2] lies in the centre of the eight vertices and so it lies on the midpoint of four diagonals:

$$P[2][2][2] = \frac{2V_{110} + 2V_{001} + 2}{4}$$
$$= \frac{2V_{100} + 2V_{011} + 2}{4}$$
$$= \frac{2V_{111} + 2V_{000} + 2}{4}$$
$$= \frac{2V_{010} + 2V_{101} + 2}{4}$$

The equations are equivalent since they all access only two elements and the overall weighting is the same for each equation, but each equation may yield different results if the volume bounded by V_{000}, V_{100}, V_{010}, V_{001}, V_{110}, V_{101}, V_{011}, V_{111} is not linear (and the greater the non-linearity, the greater the disparity between the above equations). In this example all the weights can be simplified by dividing by two. In some cases it may be preferable to use one equation over another. When interpolating colour tables it is often more appropriate to average along the neutral (grey) axis.

```
P[ 0 ][ 0 ][ 0 ] = (  4 V000  + 2)/4
P[ 0 ][ 0 ][ 1 ] = (  3 V000  +  1 V001  + 2)/4
P[ 0 ][ 0 ][ 2 ] = (  2 V000  +  2 V001  + 2)/4
P[ 0 ][ 0 ][ 3 ] = (  1 V000  +  3 V001  + 2)/4
P[ 0 ][ 1 ][ 0 ] = (  3 V000  +  1 V010  + 2)/4
P[ 0 ][ 1 ][ 1 ] = (  3 V000  +  1 V011  + 2)/4
P[ 0 ][ 1 ][ 2 ] = (  2 V000  +  1 V001  +  V011  + 2)/4
P[ 0 ][ 1 ][ 3 ] = (  1 V010  +  3 V001  + 2)/4
P[ 0 ][ 2 ][ 0 ] = (  2 V000  +  2 V010  + 2)/4
P[ 0 ][ 2 ][ 1 ] = (  2 V000  +  1 V010  +  V011  + 2)/4
P[ 0 ][ 2 ][ 2 ] = (  2 V010  +  2 V001  + 2)/4
P[ 0 ][ 2 ][ 3 ] = (  1 V010  +  2 V001  +  V011  + 2)/4
P[ 0 ][ 3 ][ 0 ] = (  1 V000  +  3 V010  + 2)/4
P[ 0 ][ 3 ][ 1 ] = (  3 V010  +  1 V001  + 2)/4
P[ 0 ][ 3 ][ 2 ] = (  2 V010  +  1 V001  +  V011  + 2)/4
P[ 0 ][ 3 ][ 3 ] = (  1 V000  +  3 V011  + 2)/4
P[ 1 ][ 0 ][ 0 ] = (  3 V000  +  1 V100  + 2)/4
P[ 1 ][ 0 ][ 1 ] = (  3 V000  +  1 V101  + 2)/4
P[ 1 ][ 0 ][ 2 ] = (  2 V000  +  1 V001  +  V101  + 2)/4
P[ 1 ][ 0 ][ 3 ] = (  1 V100  +  3 V001  + 2)/4
P[ 1 ][ 1 ][ 0 ] = (  3 V000  +  1 V110  + 2)/4
P[ 1 ][ 1 ][ 1 ] = (  3 V000  +  1 V111  + 2)/4
P[ 1 ][ 1 ][ 2 ] = (  1 V100  +  1 V010  +  V001  + 2)/4
P[ 1 ][ 1 ][ 3 ] = (  1 V110  +  3 V001  + 2)/4
P[ 1 ][ 2 ][ 0 ] = (  2 V000  +  1 V010  +  V110  + 2)/4
P[ 1 ][ 2 ][ 1 ] = (  1 V100  +  2 V010  +  V001  + 2)/4
P[ 1 ][ 2 ][ 2 ] = (  2 V010  +  1 V001  +  V101  + 2)/4
P[ 1 ][ 2 ][ 3 ] = (  1 V110  +  2 V001  +  V011  + 2)/4
P[ 1 ][ 3 ][ 0 ] = (  1 V100  +  3 V010  + 2)/4
P[ 1 ][ 3 ][ 1 ] = (  3 V010  +  1 V101  + 2)/4
P[ 1 ][ 3 ][ 2 ] = (  2 V010  +  1 V101  +  V011  + 2)/4
P[ 1 ][ 3 ][ 3 ] = (  1 V100  +  3 V011  + 2)/4
P[ 2 ][ 0 ][ 0 ] = (  2 V000  +  2 V100  + 2)/4
P[ 2 ][ 0 ][ 1 ] = (  2 V000  +  1 V100  +  V101  + 2)/4
P[ 2 ][ 0 ][ 2 ] = (  2 V100  +  2 V001  + 2)/4
P[ 2 ][ 0 ][ 3 ] = (  1 V100  +  2 V001  +  V101  + 2)/4
P[ 2 ][ 1 ][ 0 ] = (  2 V000  +  1 V100  +  V110  + 2)/4
P[ 2 ][ 1 ][ 1 ] = (  2 V100  +  1 V010  +  V001  + 2)/4
P[ 2 ][ 1 ][ 2 ] = (  2 V100  +  1 V001  +  V011  + 2)/4
P[ 2 ][ 1 ][ 3 ] = (  1 V110  +  2 V001  +  V101  + 2)/4
P[ 2 ][ 2 ][ 0 ] = (  2 V100  +  2 V010  + 2)/4
P[ 2 ][ 2 ][ 1 ] = (  2 V100  +  1 V010  +  V011  + 2)/4
P[ 2 ][ 2 ][ 2 ] = (  2 V110  +  2 V001  + 2)/4
P[ 2 ][ 2 ][ 3 ] = (  1 V110  +  2 V001  +  V111  + 2)/4
P[ 2 ][ 3 ][ 0 ] = (  1 V100  +  2 V010  +  V110  + 2)/4
P[ 2 ][ 3 ][ 1 ] = (  2 V010  +  1 V110  +  V101  + 2)/4
P[ 2 ][ 3 ][ 2 ] = (  2 V110  +  1 V001  +  V011  + 2)/4
```

Figure 2. Interpolation Equations for N=2

3.2 Example Code

n-Simplex interpolation would be of little value if the saving from the reduction in table lookups were offset by the overhead of testing for simpler cases. Fortunately the algorithm can be written so a single test determines which table entries need to be

[3] The sub-cube does not include the points that lie on the faces furthest from the reference point V_{000}, since these are in the adjacent sub-cubes.

accessed, for example by using a computed goto or case statement[4].

The high order input bits are used to provide an offset into the table, as is normally the case, but the low order bits are used choose which case to execute. Each case corresponds to one of the positions that lies between coarse lattice points, and so each case need only access the necessary coarse lattice points. The interpolation coefficients for the lattice points are known integers between unity and (2^N-1), with the sum of the coefficients being 2^N. A program that uses the tetrahedral volume ratios described earlier can generate the case statement automatically. The final step (which can be performed outside the case statement) is to round (by adding $(2^N)/2$) and normalise (by dividing by 2^N). For example, mapping three input bytes to one output byte the table size will be $(2^{8-N}+1)^3$, and there will be $(2^N)^3$ cases.

```
#define N 4
#define SIZE 16
#define STRIDE ((1<<(8-N))+1)
#define ROUND 8

unsigned char table[(STRIDE)*(STRIDE)*(STRIDE)];

/* define offsets from the reference element V000:
   an x offset adds the square of the stride, a
   y offset adds the stride and a z offset adds 1
*/

#define V000 offset[0]
#define V100 offset[STRIDE*STRIDE]
#define V110 offset[STRIDE*STRIDE + STRIDE]
#define V111 offset[STRIDE*STRIDE + STRIDE+1]
#define V101 offset[STRIDE*STRIDE + 1]
#define V010 offset[STRIDE]
#define V011 offset[STRIDE + 1]
#define V001 offset[1]

char interpolate(xin,yin,zin) int xin,yin,zin; {

  char p;

  int swvar = ((xin & (SIZE-1)) << (2*N))|
     ((yin & (SIZE-1)) << N) | (zin & (SIZE-1));

  char *offset = &table[ (xin>>N)*STRIDE*STRIDE) +
                      (yin>>N)*STRIDE + (zin>>N)];

  switch(swvar){
    case 0: p = 16*V000; break;
    case 1: p = 15*V000 + V001; break;
    case 2: p = 14*V000 + 2*V001; break;
    case 3: p = 13*V000 + 3*V001; break;

    ...

    case 4077: p = 2*V100 + V010 + 13*V111; break;
    case 4078: p = 2*V100 + V011 + 13*V111; break;
    case 4079: p = V010 + 2*V101 + 13*V111; break;
    case 4080: p = V000 + 15*V110; break;
    case 4081: p = 15*V110 + V001; break;
    case 4082: p = 14*V110 + V101 + V011; break;
    case 4083: p = 13*V110 + V001 + 2*V111; break;
    case 4084: p = 12*V110 + V001 + 3*V111; break;
    case 4085: p = 11*V110 + V001 + 4*V111; break;
    case 4086: p = 10*V110 + V001 + 5*V111; break;
    case 4087: p = 9*V110 + V001 + 6*V111; break;
    case 4088: p = 8*V110 + V001 + 7*V111; break;
    case 4089: p = 7*V110 + V001 + 8*V111; break;
    case 4090: p = 6*V110 + V001 + 9*V111; break;
    case 4091: p = 5*V110 + V001 + 10*V111; break;
    case 4092: p = 4*V110 + V001 + 11*V111; break;
    case 4093: p = 3*V110 + V001 + 12*V111; break;
    case 4094: p = 2*V110 + V001 + 13*V111; break;
    case 4095: p = V000 + 15*V111; break;
  };
  p = (p + ROUND)/16;
  return p;
};
```

Figure 3. Example C Code

The code can be made to execute slightly faster, at the expense of increased code size, by performing the weighted average and rounding in a single step. In this way the separate rounding step is removed and some of the cases can be simplified. For example, case 0 would become p:=V000, and case 2 would become p:=(7*V000 + V001 + 4)/8. Although the overall code size becomes larger, the overall execution is reduced, for example case 0 removes a multiplication and addition and case 2 removes a multiplication.

4. ANALYSIS

As N increases, the case statements become larger (and so does the code size for the case statement). The distances between the coarse lattice points also increases with N as does interpolation error (although the table size will decrease). For colour mapping, the task of mapping three input bytes to three output bytes (therefore having three assignment statements per line instead of only one assignment per line in the above example) can be achieved for various values of N. The code size in the following chart is for a program compiled using cc with no flags on a

[4] The cases start from zero and are consecutive so the overhead of a case statement is about 6 instructions.

Hewlett Packard model 735 Unix workstation, as reported by the Unix size program, and although these values are representative, there would be differences with different compilers or target instruction sets. However it is reasonable to say, from Table 2 that for N=3 n-simplex interpolation will reduce the number of table accesses from 4 for tetrahedral to 2.9238 (a reduction of 17%) and the interpolation expressions will also be implified by a similar factor.

To map three input bytes to three output bytes the table size will be $3.(2^{8-N}+1)^3$ bytes and code will have $(2^N)^3$ cases[5].

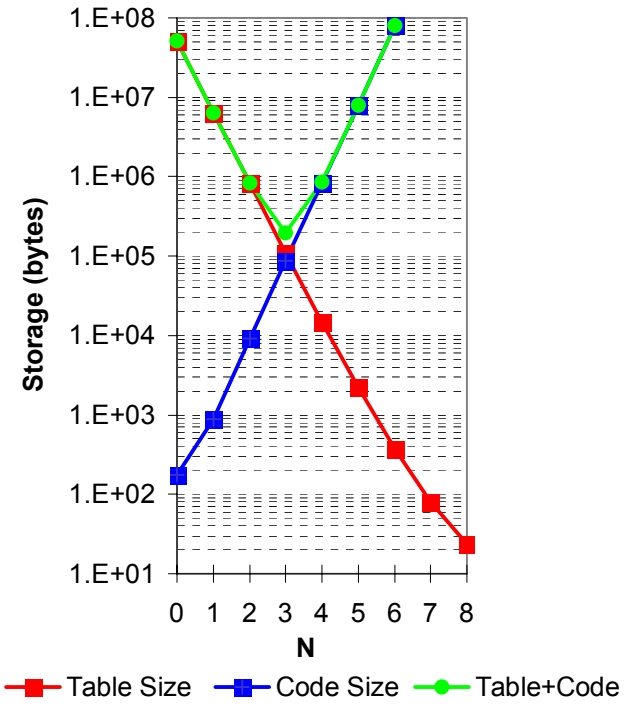


Figure 4. Storage versus Lattice Spacing

For this example, the smallest overall storage (200k bytes) occurs for N= 3 (and the distance between table elements is eight). Increasing N above 3 would offer no advantage. Indeed, it would present several disadvantages: the table would have fewer entries (and probably less precise), the average number of lookups will increase, so the interpolation will be slower and some compilers may not be able to compile large switch statements.

If more storage is available, reducing N will speed up the interpolation and also tend to increase the precision of the results.

[5] The code size for N=5 and N=6 have been extrapolated from the code size for N<=4 because the compiler needed more than 1Gb memory when trying to compile such large switch statements.

With k input values, each input value having B bits, then the number of table entries is

$$(2^{B-N}+1)^k$$

with the number of case statements being

$$\left(2^N\right)^k$$

In practice, because the code required for a case statement exceeds the storage for a table entry then the optimal solution occurs when

$$N \le \frac{B}{2}$$

Increasing k (and increasing the complexity of the case statement code) the optimal solution may have N much smaller than B/2.

5. CONCLUSION

A practical interpolation method has been presented that uses fewer table lookups and uses fewer execution steps than existing methods. The method is most advantageous when interpolating over smaller ranges. Smaller interpolation ranges will become more common as memory cost falls, and the cost of larger tables will be more acceptable. These larger tables will naturally offer greater precision.

The algorithm has been described for three dimensional interpolation problems, and more specifically colour mapping. It has been shown to be computationally more efficient than other three dimensional interpolation algorithms, including tetrahedral interpolation. The algorithm is not limited to colour mapping and can be applied to other three dimensional problems. It is also not limited to just three dimensions, and can be used to advantage to interpolate in more than three dimensions.

6. ACKNOWLEDGMENTS

The author would like to thank Gary Vondran for sharing his extensive knowledge of interpolation algorithms and for his assessment of the algorithm presented here. The author is grateful to Dave Grosvenor and Huw Oliver for commenting on earlier drafts of this document.

7. REFERENCES

[1] Kasson, J. M., Nin, S. I., Plouffe, W., and Hafner, J. L. “Performing color space conversions with three-dimensional linear interpolation”, Journal of Electronic Imaging 4(3) (July 1995), 226-250.

[2] Kanamori, K., and Kotera, H., “Color correction technique for hard copies by 4-neighbors interpolation method”, J. Imaging Sci. Technol. 36(1), 73-80 (Jan./Feb. 1992).

[3] Vondran, Gary L., ”Non-symmetric Radial and Non-symmetric Pruned Radial Interpolation”, U.S. Patent No. 5,966,474, 12 October 1999

[4] Vondran, Gary L. et al., “Radial and Pruned Radial Interpolation”, U.S. Patent No. 6,040,925, 21 March 2000.

[5] Katsuhiro Kanamori, Hideiko Kawakami, and Hiroaki Kotera. “A novel color transformation algorithm and its applications”. Image Processing Algorithms and Techniques, 1244:272-281, 1990.

[6] Hemingway, P. “Method of Interpolation”, Patent Application No. EP99302139, 27 September 2000.

[7] Kang, Henry R., “Color Technology for Electronic Imaging Devices”, Chapter 4, pages 64-101, Spie Press, 1996.

[8] International Electrotechnical Commission, “Colour Measurement and Management in Multimedia Systems and Equipment – Part 2-1: Default RGB Colour Space – sRGB”, IEC 61966-2-1, October 1999.