Tetrahedral Interpolation on Regular Grids

Róbert Bán
rob.ban@inf.elte.hu

Gábor Valasek
valasek@inf.elte.hu

Eötvös Loránd University, Budapest, Hungary

Abstract

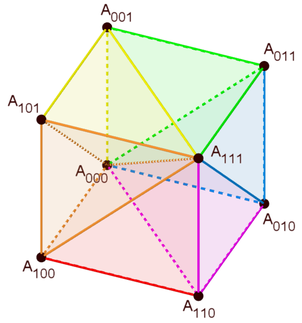
This work proposes the use of barycentric interpolation on enclosing simplices of sample points to infer a reconstructed function from discrete data. In particular, we compare the results of trilinear and tetrahedral interpolation over regular 3D grids of second order spherical harmonics (SH) light probes. In general, tetrahedral interpolation only requires four data samples per query in contrast to the 8 samples necessary for trilinear interpolation, at the expense of a more expensive weight computation. Our tetrahedral implementation subdivides the cubical cells into six tetrahedra and uses the barycentric coordinates of the query position as weights to blend the probe data. We show that barycentric coordinates can be calculated efficiently in shaders for our particular tetrahedral decomposition of the cube, resulting only in simple arithmetic and conditional move operations.

1. Introduction

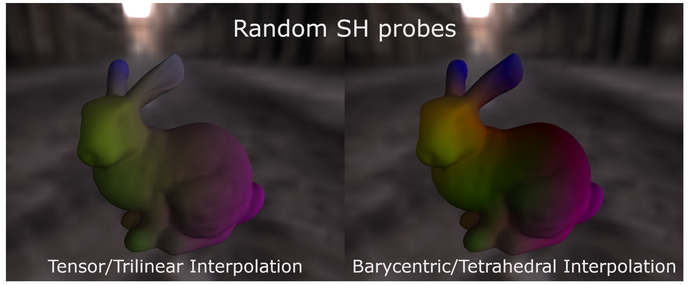
In computer graphics, multivariate function representation are often represented as regular 3D grids of samples. There are several techniques to reconstruct a continuous function from the stored data. The standard method is trilinear interpolation that uses 8 samples and 7 linear interpolations to compute a filtered value.
An alternative approach is to subdivide the regular grid cells into tetrahedra and use the barycentric coordinates of the query position to weight the data of the vertices of the smallest enclosing tetrahedron. Kasson et al. [1] proposed a similar solution for color conversion. The main advantage compared to trilinear interpolation is the reduced memory bandwidth. The contribution of our work is two-fold: (i) we propose to use the method in light probe interpolation and (ii) we present an efficient algorithm for the barycentric weight calculation.

2. Barycentric interpolation

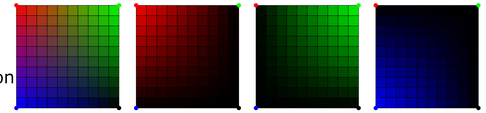
Barycentric interpolation on an n-dimensional simplex is a linear combination of the vertices where the weights sum to 1. We can compare this to tensor interpolation which is defined on an n-cube as a tensor product of linear interpolations – also called bilinear and trilinear interpolation in 2 and 3 dimensions, respectively.
For barycentric interpolation, the cube is subdivided into a disjoint union of simplices. This subdivision is not unique. We decided to use the one shown in the figure in 3D so that we can use a simplified barycentric weight computation algorithm.
Note that while barycentric and tensor interpolations yield different results, one is not necessarily better than the other. Nevertheless, while both produce C^∞ reconstructed functions within their domains (i.e. tetrahedra and boxes), in general, the continuity of the inferred signal is only C^0 along the connections of these domains and a tetrahedral decomposition results in more of these boundaries.



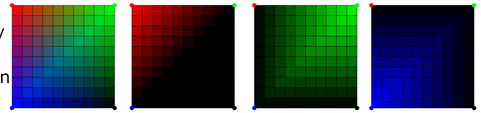

```
void barycentricWeight(vec3 r, out vec4 bary,
            out ivec3 vert2, out ivec3 vert3) {
  vert2 = ivec3(0,0,0); vert3 = ivec3(1,1,1);
  bvec3 c = greaterThanEqual(r.xyz, r.yzx);
  bool c_xy = c.x, c_yz = c.y, c_zx = c.z;
  bool c_yx =!c.x, c_zy =!c.y, c_xz =!c.z;
  bool cond;  vec3 s;
#define ORDER(X,Y,Z)                    \
  cond = c_ ## X ## Y && c_ ## Y ## Z; \
  s = cond ? r.X ## Y ## Z : s;         \
  vert2.X = cond ? 1 : vert2.X;         \
  vert3.Z = cond ? 0 : vert3.Z;
  ORDER(x,y,z)   ORDER(x,z,y)   ORDER(z,x,y)
  ORDER(z,y,x)   ORDER(y,z,x)   ORDER(y,x,z)
  bary = vec4(1 - s.x, s.z, s.x - s.y, s.y - s.z);
}
```



Tensor / Bilinear Interpolation



Barycetric / Triangular Interpolation



3. The algorithm

To compute the barycentric coordinates of a query point, first we have to identify which tetrahedron contains it, then compute the barycentric coordinates with respect to the vertices of that tetrahedron. The calculations are done in normalized cell coordinates ($\boldsymbol{x} = (x, y, z) \in [0, 1]^3$). The enclosing tetrahedron can be found by ordering the coordinates, since the six tetrahedra are

$$z \le y \le x, \qquad y \le z < x, \qquad z < x \le y, \qquad x \le z < y, \qquad y < x \le z, \qquad x < y < z. \tag{1}$$

Let us derive the weights for the first tetrahedron, highlighted with red in the figure. The vertices are A_{000}, A_{111}, A_{100}, and A_{110}, where the coordinates of A_{ijk} is $[i, j, k]^T \in \{0, 1\}^3$. The two equations defining the weights are

$$\begin{bmatrix} x & y & z \end{bmatrix}^T = w_0 \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T + w_1 \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^T + w_2 \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T + w_3 \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}^T, \text{ and} \tag{2}$$

$$1 = w_0 + w_1 + w_2 + w_3. \tag{3}$$

The solution is $w_0 = 1 - x$, $w_1 = z$, $w_2 = x - y$, $w_3 = y - z$. The weights in the remaining five tetrahedra can be calculated similarly but we can write this more concisely. Let $a \le b \le c$, where $a = \min(x, y, z)$, $c = \max(x, y, z)$ and b the remaining coordinate. Then

$$w_0 = 1 - c, \qquad w_1 = a, \qquad w_2 = c - b, \qquad w_3 = b - a. \tag{4}$$

The vertices are in a similar order: A_{000} and A_{111} are always the first two, w_2 corresponds to the remaining vertex closer to A_{000} and w_3 to the vertex closer to A_{111}.
Our GLSL implementation is shown on the left. The function returns the barycentric weights and the 3D indices of the unknown vertices of the enclosing tetrahedron. The compiled code only uses conditional moves instead of branches, making it optimal for GPUs.

4. Results

We compared trilinear and barycentric interpolation of light probes. Probes represent irradiance as second order SH functions containing $3 \times 9 = 27$ scalars. The bunny figure shows 8 random colored probes with trilinear (left) and tetrahedral (right) interpolation. The table on the right compares nearest point sampling, manual trilinear interpolation, and tetrahedral interpolation of 2 million root searches on a 3D function represented by polynomials consisting of 1, 4, 10, and 20 scalars per sample in IEEE binary32 format, in 3D textures. The numbers are in milliseconds.
On both architectures, tetrahedral interpolation increases performance considerably for larger sample footprints. It even retains this gain compared to hardware accelerated trilinear interpolation from about 20 scalars per sample.

	AMD RX 5700				NVIDIA 2080			
scal./samp.	1	4	10	20	1	4	10	20
nearest	0.21	0.27	0.29	0.55	0.37	0.44	0.52	0.64
trilinear	0.73	0.99	2.30	4.42	**0.50**	**0.79**	1.40	2.37
tetrahedral	**0.58**	**0.72**	**1.13**	**2.25**	0.69	0.81	**1.05**	**1.58**

5. Conclusion

In summary, this work justifies the use of tetrahedral interpolation in probe light blending and provides an optimized algorithm for the weight calculations. We showed qualitative comparisons in 2D and 3D tests.

[1] James M. Kasson, Wil Plouffe, and Sigfredo I. Nin. Tetrahedral interpolation technique for color space conversion. In Ricardo J. Motta and Hapet A. Berberian, editors, *Device-Independent Color Imaging and Imaging Systems Integration*, volume 1909, pages 127 – 138. International Society for Optics and Photonics, SPIE, 1993.

Acknowledgement *EFOP-3.6.3-VEKOP-16-2017-00001: Talent Management in Autonomous Vehicle Control Technologies – The Project is supported by the Hungarian Government and co-financed by the European Social Fund.*

Supported by the ÚNKP-20-3 New National Excellence Program of the Ministry for Innovation and Technology from the source of the National Research, Development and Innovation Fund.

We would like to thank Visual Concepts for providing the AMD GPU used in the tests.