Title:
Hermite Interpolation of Scalar Fields in Computer Graphics

Authors:
Róbert Bán, rob.ban@inf.elte.hu, Eötvös Loránd University, Hungary
Gábor Valasek, valasek@inf.elte.hu, Eötvös Loránd University, Hungary

Keywords:
Geometric Modelling, Computer Graphics

DOI: 10.14733/cadconfP.2024.249-253

Introduction:

Scalar fields, i.e. $f : \mathbb{R}^n \to \mathbb{R}$ functions, are used in a wide range of applications as means to represent quantities such as height maps, implicit surfaces, and radiance maps. A potential representation of these fields is to store the function in symbolic form or as procedural code, and evaluate it when a sample is needed. However, this is not always possible – either we do not know the exact function or it is too complicated and cannot be stored easily. The practical alternative is to sample the original field and discretize it. We then store the samples and if we need to evaluate the field at an arbitrary location, we read the nearby samples and use an interpolation method. We call this step filtering. The discrete field therefore consist of three parts: the sample positions, the stored data, and the reconstruction method.

The simplest sample arrangement is a regular axis-aligned grid, and the simplest sample is the function value at the sample location. On a regular cubic grid, multi-linear (tensor product linear) interpolation can be used to fill the space between the samples. This is bilinear interpolation for two, and trilinear interpolation for three dimensions. The sample positions are vertices of cells, and interpolation methods usually define polynomials inside cells.

Our main contributions are 1) formalism for Hermite interpolation independent of dimension and order, 2) a generalization of the Adini twist calculation method to 3D domain, and 3) we applied the interpolation method and mixed partial derivative calculation for implicit surface representation and rendering.

Hermite interpolated fields:

The smoothness of the reconstructed functions is often important for the application. However, multi-linear interpolation only guarantees C^0-continuity along cell boundaries. Various interpolation techniques exist that ensure a higher order of continuity [3] at the expense of requiring more samples. Instead, we propose to store partial derivative data in the samples and guarantee higher order continuity using Hermite interpolation based on only the closest samples of the grid.

In one dimension, the order n Hermite interpolation problem ($n \in \mathbb{N}$) can be stated as follows. Given $f_0^0, f_1^0, f_0^1, f_1^1, \ldots, f_0^n, f_1^n \in \mathbb{R}$, find a polynomial p of degree $2n+1$ such that

$$\forall x \in \{0,1\}\colon p(x) = f_x^0,\;\; p'(x) = f_x^1,\;\; p''(x) = f_x^2,\;\; \ldots,\;\; p^{(n)}(x) = f_x^n. \tag{2.1}$$

The problem can be solved directly by using the order n Hermite polynomials which we describe next. Let $n \in \mathbb{N}$ be the maximal order. We define the order n Hermite polynomials through a set of equations.

Definition 1. *The* order n Hermite polynomial *of base position x interpolating the k-th derivative ($k \in \{0,1,\ldots,n\}$ and $x \in \{0,1\}$) is $\alpha_x^{k,n} : \mathbb{R} \to \mathbb{R}$. The defining equations for $\alpha_x^{k,n}$ are*

$$\forall j \in \{0,1,\ldots,n\}: \ \forall y \in \{0,1\}: (\alpha_x^{k,n})^{(j)}(y) = \delta_{k,j}\delta_{x,y}, \tag{2.2}$$

where $f^{(j)}$ is the jth derivative of f and

$$\delta_{a,b} = \begin{cases} 1, & a = b, \\ 0, & a \neq b. \end{cases} \tag{2.3}$$

This means that the values and the derivatives of $\alpha_x^{k,n}$ are zero at 0 and 1 except for the k-th derivative at x, where it is 1. The number of order n Hermite polynomials is therefore $2n+2$ – one for each interpolated position and derivative up to n. As there are $2n+2$ equations for each polynomial, there is a unique polynomial of degree $2n+1$ satisfying all requirements for each (k,n,x) triplet.

The following polynomial solves the interpolation problem in Equation (2.1)

$$p(x) = \sum_{i=0}^{1} \sum_{k=0}^{n} f_i^k \cdot \alpha_i^{k,n}(x) \qquad x \in [0,1]. \tag{2.4}$$

These polynomials can be defined in higher dimensional space as well. In N dimensions, we can use the tensor-product of the basis polynomials to define the new basis functions:

$$\alpha_{\boldsymbol{y}}^{\boldsymbol{k},n}(\boldsymbol{x}) = \prod_{m=1}^{N} \alpha_{y_m}^{k_m,n}(x_m), \quad \text{where } \boldsymbol{y} \in \{0,1\}^N, \boldsymbol{k} \in \{0,1,\ldots,n\}^N, \boldsymbol{x} \in [0,1]^N. \tag{2.5}$$

These multivariate polynomials have similar interpolating properties as their one-dimensional counterparts:

$$\forall \boldsymbol{x}, \boldsymbol{y} \in \{0,1\}^N, \forall \boldsymbol{j}, \boldsymbol{k} \in \{0,1,\ldots,n\}^N: \quad \partial^{\boldsymbol{j}} \alpha_{\boldsymbol{y}}^{\boldsymbol{k},n}(\boldsymbol{x}) = \delta_{\boldsymbol{x},\boldsymbol{y}} \cdot \delta_{\boldsymbol{j},\boldsymbol{k}}. \tag{2.6}$$

Using these polynomials, we can solve a similar problem to Equation (2.1). Given function values and partial derivatives in the vertices of the unit hypercube (square in 2D, cube in 3D), we can interpolate all values with the following multivariate polynomial

$$p(\boldsymbol{x}) = \sum_{\boldsymbol{y} \in \{0,1\}^N} \sum_{\boldsymbol{k} \in \{0,1,\ldots,n\}^N} f_{\boldsymbol{y}}^{\boldsymbol{k}} \cdot \alpha_{\boldsymbol{y}}^{\boldsymbol{k},n}(\boldsymbol{x}) \qquad \boldsymbol{x} \in [0,1]^N, \tag{2.7}$$

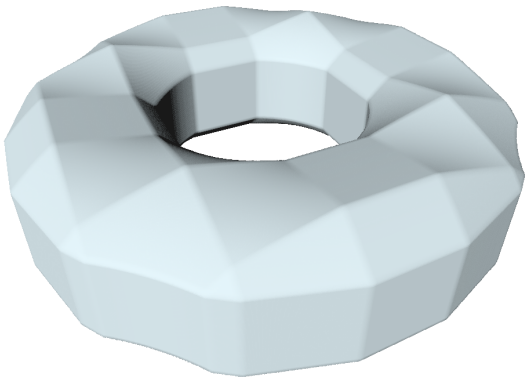
where $\forall \boldsymbol{y} \in \{0,1\}^N, \forall \boldsymbol{k} \in \{0,1,\ldots,n\}^N : f_{\boldsymbol{y}}^{\boldsymbol{k}} \in \mathbb{R}$.

Using Hermite interpolation we can guarantee higher order continuity. We keep the regular grid of the sample positions, but store the discussed partial derivatives besides the function value. Inside the cells Hermite interpolation is used, and the shared samples of neighbouring cells guarantee the desired higher order continuity.

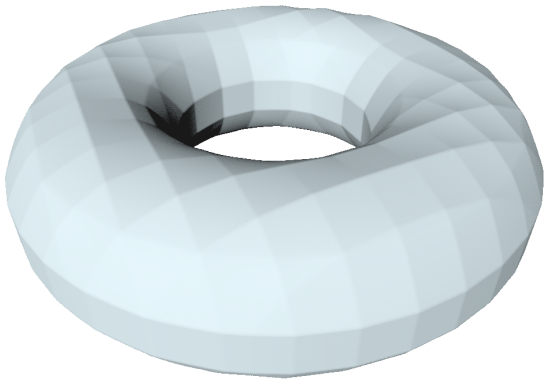
First order Hermite fields in 3D:

In practice, for three-dimensional use cases, it is often enough to store the first order derivatives. More specifically, to reduce the amount of data, we can discard mixed partial derivatives, as their contribution to the final interpolated value is usually smaller. This means that instead of 8 values, we only store the function value and the three partial derivatives corresponding to each axis for each sample position.

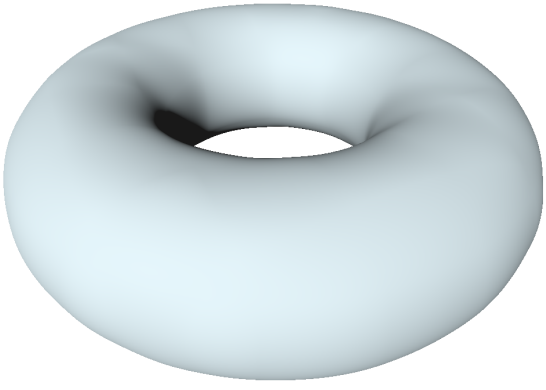
The mixed partial derivatives can be either assumed to be zero or approximated based on the stored data. The first option is similar to how Ferguson patches [2] are constructed, and as such we call it



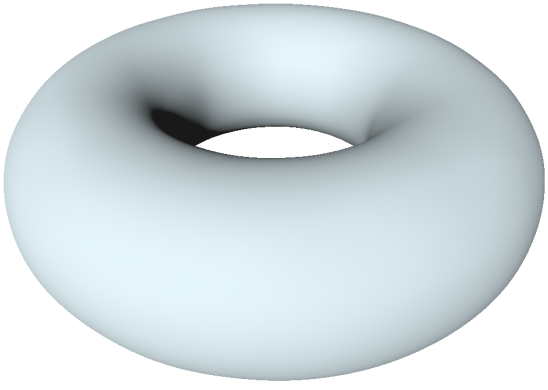
(a) Res: 8^3, Trilinear



(b) Res: 16^3, Trilinear



(c) Res: 8^3, Ferguson-Hermite

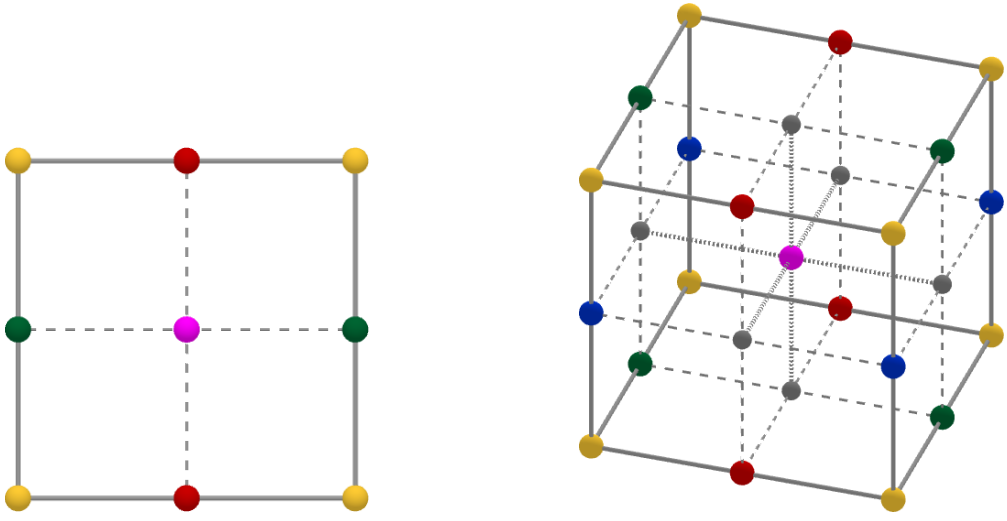


(d) Res: 8^3, Hermite with Adini-twist

Fig. 1: 3D example of an implicit surface stored as a discrete field. The trilinear case only interpolates the function value, while Hermite-interpolation interpolates the first derivatives as well. 1c) assumes zero twist, and 1d) calculates an approximated mixed partial derivatives similarly to the Adini method.

Ferguson-Hermite interpolation. Although zeroing the mixed partial derivatives can be a sensible choice, it often results in visible flat spots in the interpolated result. See Figure 1c and 1d for a comparison. Note the "wrinkles" and the slightly more angular silhouette in the first image where mixed partials are zeroed out.

For better approximation, and therefore better visual quality, the missing partial derivatives can be either calculated from the original function – or approximated from the rest of the stored data. We propose a particular instance of the latter in three dimensions. Our method is similar to how the Adini twist vectors [2] are defined for parametric surfaces. For parametric surface patches, the Adini twist is defined by fitting a Coons patch [1] to the boundary of the four neighbouring patches, and evaluating the mixed partial derivative at the appropriate parameter value. For applying this to our scalar filed in three dimensions, we first define the 3D equivalent of the Coons patch, then we fit a Coons volume to the surrounding 8 cells of a sample, and finally evaluate the partial derivatives of the interpolating function in the middle point. These partial derivatives will then be used as the missing values for the Hermite interpolation.



(a) 2D domain: parametric surface patch

(b) 3D domain: implicit surface

Fig. 2: Values needed for the Adini twist calculation. To calculate the mixed partial derivatives at the center sample (magenta), we need to use the function value at the corner samples (yellow), and the function value and partial derivatives at the edge midpoint samples (red, green and blue).

For ease of notation, let us denote linear interpolation between two values a_0 and a_1 as

$$\sum_{u,i} a_i := \sum_{i=0}^{1} (1-u)^{1-i} u^i a_i = (1-u)a_0 + ua_1. \tag{2.8}$$

Using this, we define the Coons-volume $s : [0,1]^3 \to \mathbb{R}$ as

$$s(u,v,w) = \sum_{v,j}\sum_{w,k} s(u,j,k) + \sum_{u,i}\sum_{w,k} s(i,v,k) + \sum_{u,i}\sum_{v,j} s(i,j,w) - 2\sum_{u,i}\sum_{v,j}\sum_{w,k} s(i,j,k), \tag{2.9}$$

where $s(u,j,k), s(i,v,k), s(i,j,w)$ are the given function values to interpolate along the edges of the cube, and $s(i,j,k)$ are the values at the vertices of the cube. Similarly to the Coons patch, we first interpolate the parallel edges of the domain for the whole volume in each dimension, and subtract the interpolation of the vertices, so the final volume patch interpolates all given entities.

We then use the Coons volume to derive the missing mixed partial derivatives for our Hermite interpolation, akin to the Adini twist method. For the calculation at each sample position, we fit a Coons volume to the 8 cubic cells surrounding the point – the point is a vertex of each cell as shown in Figure 2b. The values at the vertices of the cube are given by the sample values, while the values along the edges are the Hermite interpolated function values. Each edge therefore consists of two cubic polynomial segments. Then we calculate the partial derivatives of the interpolating volume at the center point. For this, we need to calculate the partial derivatives of the function along the edges but only at the center points which coincide with the given gradient values of the grid. To summarize, for the final calculated mixed derivatives $(\partial_{uv}s, \partial_{uw}s, \partial_{vw}s, \partial_{uvw}s)$, we need the function value at the vertices, and the function value and partial derivative in the edge direction at the edge middle points. The required function values and derivatives are shown in Figure 2b.

Optimization for direct rendering:

For direct rendering of implicit surfaces, one needs to find the first intersection point of a ray and the surface. Formally, the parametric form of the ray is $\boldsymbol{p}(t) = \boldsymbol{p}_0 + t\boldsymbol{v}$, where $\boldsymbol{p}_0 \in \mathbb{R}^3$ is the starting point, $\boldsymbol{v} \in \mathbb{R}^3$ ray direction, and $0 < t \in \mathbb{R}$ is the ray parameter. If $f : \mathbb{R}^3 \to \mathbb{R}$ is the implicit representation of the surface then the parameter of an intersection point satisfies $f(\boldsymbol{p}(t)) = 0$. For rendering, we need the first of these, i.e. the smallest positive t value. There are many algorithms to find the intersection point and they usually depend on sampling the value of the f function along the ray. If we use the order 1 constructions discussed above, we can directly calculate a degree 9 polynomial in t for each cell the ray travels through, and use the exact polynomial for a more robust root finding algorithm.

A further possible optimization, that may be precalculated, is to decide if a whole cell is empty, or more precisely, if it does not contain any surface points. First, we can check whether there are two vertices of the cell, where the function values have different sign – in which case the surface crosses the cell. Otherwise, we transform the Hermite representation into the Bernstein basis. If all Bernstein coefficients have the same sign, the cell is empty, due to the convex hull property of the basis. In the other cases, we may subdivide the cell and evaluate the previous two steps for each subcell if needed.

Conclusions:

We implemented the first order Hermite interpolation methods for 3D implicit surface rendering on the GPU. The Ferguson-Hermite interpolation method proved to be a high performance method. The generalized Adini twist calculation method increases visual quality, however, with diminishing margins as resolutions increase and at a reduced frame rate. In practice, the Adini method is only viable in real-time if the mixed partials are precalculated. It must be taken into account that this increases the per-sample data size twofold, and we measured an average increase in frame times up to 1.5x compared to the Ferguson-Hermite method. The higher order constructions can be used as a form of compression, in which case the stored data is processed before sampling. For example, it may be resampled as a higher resolution order 0 field – only function values – and interpolated with hardware accelerated multi-linear interpolation.

Acknowledgement:

Supported by the ÚNKP-23-4 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.

Róbert Bán, https://orcid.org/0000-0002-8266-7444
Gábor Valasek, https://orcid.org/0000-0002-0007-8647

References:

[1] Coons, S. A.: Surfaces for computer-aided design of space forms. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1967.

[2] Farin, G.: Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide, Morgan Kaufmann Publishers Inc., 2001.

[3] Csébfalvi, B.: Beyond trilinear interpolation: higher quality for free, ACM Transactions on Graphics, 38(4), article 56, 2019, 1-8. https://doi.org/10.1145/3306346.3323032