

# Robust Hex-Dominant Mesh Generation using Field-Guided Polyhedral Agglomeration

Xifeng Gao, Wenzel Jakob, Marco Tarini, Daniele Panozzo

## Supplement 1

This supplement contains a detailed description of two key operations used by the operation field optimization of our method. The first is an efficient technique to search for the optimal matching for a pair of orientation field values. The second searches for the smallest rotation to align an orientation field value to a given direction, which is needed to realize boundary constraints.

## 1. 3D Crosses as Quaternions

We use unit quaternions to internally represent “3D crosses” sampling the field. This representation is compact, and allows to efficiently perform all needed operations.

Notation: here, a quaternion  $\mathbf{p} = p_x i + p_y j + p_z k + p_w$  is denoted by  $(p_x, p_y, p_z, p_w)$ , with the real part at the end.

### 1.1. The representation

The unit quaternion  $\mathbf{q}$  defines the rotation going from the *canonical* frame to the local frame, whose axis are the directions of the cross (at a point  $p$ ).

$$\text{canonical frame} \xrightarrow{\mathbf{q}} \text{local cross}$$

The six directions of the 3D cross are given by rotating the three axes of the canonical frame, which are the imaginary parts of, respectively:

$$\begin{aligned} \mathbf{q} \cdot (1, 0, 0, 0) \cdot \bar{\mathbf{q}} \\ \mathbf{q} \cdot (0, 1, 0, 0) \cdot \bar{\mathbf{q}} \\ \mathbf{q} \cdot (0, 0, 1, 0) \cdot \bar{\mathbf{q}} \end{aligned}$$

and by their three opposites, i.e. by the 6 vectors:

$$\begin{aligned} \pm (1, 0, 0) + 2(q_x, q_y, q_z) \times (q_w, +q_z, -q_y) \\ \pm (0, 1, 0) + 2(q_x, q_y, q_z) \times (-q_z, q_w, +q_x) \\ \pm (0, 0, 1) + 2(q_x, q_y, q_z) \times (+q_y, -q_x, q_w) \end{aligned}$$

The represented 3D crosses are, by construction, orthonormal.

There is a group of 24 rotations (including the identity), which map each axis of the canonical frame over either an axis or its opposite; each of these rotations can be equivalently encoded by two different unit quaternions (one the opposite of the other); this gives a set of 48 quaternions which we denote as  $\mathbf{r}_1 \dots \mathbf{r}_{48}$ .

If we post-multiply a unit quaternion  $\mathbf{q}$  by any  $\mathbf{r}_i$ , then its six directions are just (consistently) permuted, and so the represented 3D cross does not change:

$$\text{canonical fr.} \xrightarrow{\mathbf{r}_i} \text{permuted axes} \xrightarrow{\mathbf{q}} \text{same loc. cross}$$

(recall  $\mathbf{q} \cdot \mathbf{r}_i$  encodes the rotation encoded  $\mathbf{r}_i$  followed by the rotation encoded by  $\mathbf{q}$ ). Therefore, for any given 3D cross, we have exactly 48 distinct but equivalent representations  $\mathbf{q} \cdot \mathbf{r}_1 \dots \mathbf{q} \cdot \mathbf{r}_{48}$ .

### 1.2. The operations

All needed operations over 3D crosses can be performed efficiently with this representation.

#### 1.2.1. Averaging two 3D crosses

In order to blend together two 3D crosses represented by quaternions  $\mathbf{p}$  and  $\mathbf{q}$ , we first find the equivalent representation  $\mathbf{q}'$  of the latter which is closest to  $\mathbf{p}$ : i.e.,  $\mathbf{q}' = \mathbf{q} \cdot \mathbf{r}_j$  for some appropriately chosen  $j$  (this sub-problem is dealt with in the following sections).

Next, we interpolate the two quaternions  $\mathbf{q}'$  and  $\mathbf{p}$ .

Because we are only interested in the (non-weighted) average, i.e. the *half-way* interpolation, the result can be directly computed by simply averaging of four coefficients  $\mathbf{q}'$  and  $\mathbf{p}$ , then re-normalizing (conversely, for generic linear interpolation of quaternions, a linear interpolation of the coefficients would only constitute an approximation of the intended result). Due to the final normalization, we simply sum  $\mathbf{q}'$  and  $\mathbf{p}$  rather than averaging them.

#### 1.2.2. Enumerating quaternions $\mathbf{r}_i$

Let us first enumerate the 48 quaternions  $\mathbf{r}_1 \dots \mathbf{r}_{48}$ . It will be useful to divide them into 3 classes (of 8, 24, and 16 elements), according to the number of zeros:

$$\begin{aligned} & \left. \begin{aligned} & \begin{pmatrix} \pm 1, & 0, & 0, & 0 \end{pmatrix} \\ & \begin{pmatrix} 0, & \pm 1, & 0, & 0 \end{pmatrix} \\ & \begin{pmatrix} 0, & 0, & \pm 1, & 0 \end{pmatrix} \\ & \begin{pmatrix} 0, & 0, & 0, & \pm 1 \end{pmatrix} \leftarrow \text{ide} \end{aligned} \right\} \text{class } A \ (4 \times 2) \\ & \left. \begin{aligned} & \begin{pmatrix} 0, & \pm 1, & \pm 1, & 0 \end{pmatrix} / \sqrt{2} \\ & \begin{pmatrix} \pm 1, & 0, & \pm 1, & 0 \end{pmatrix} / \sqrt{2} \\ & \begin{pmatrix} \pm 1, & \pm 1, & 0, & 0 \end{pmatrix} / \sqrt{2} \\ & \begin{pmatrix} \pm 1, & 0, & 0, & \pm 1 \end{pmatrix} / \sqrt{2} \\ & \begin{pmatrix} 0, & \pm 1, & 0, & \pm 1 \end{pmatrix} / \sqrt{2} \\ & \begin{pmatrix} 0, & 0, & \pm 1, & \pm 1 \end{pmatrix} / \sqrt{2} \end{aligned} \right\} \text{class } B \ (6 \times 4) \\ & \left. \begin{aligned} & \begin{pmatrix} \pm 1, & \pm 1, & \pm 1, & \pm 1 \end{pmatrix} / 2 \end{aligned} \right\} \text{class } C \ (1 \times 16) \end{aligned}$$

Note that to multiply any  $\mathbf{q} = (q_x, q_y, q_z, q_w)$  with the elements of class  $A$  simply means to permute and/or flip its coordinates (a single “swizzle” op on GPU). Specifically, we will denote with  $\mathbf{q}_1 \dots \mathbf{q}_4$  the permutation/flipping given by multiplying with the four positive elements of class  $A$ :

$$\begin{aligned} \mathbf{q}_1 &= \mathbf{q} \cdot (1, 0, 0, 0) = (q_w, +q_z, -q_y, -q_x) \\ \mathbf{q}_2 &= \mathbf{q} \cdot (0, 1, 0, 0) = (-q_z, q_w, +q_x, -q_y) \\ \mathbf{q}_3 &= \mathbf{q} \cdot (0, 0, 1, 0) = (+q_y, -q_x, q_w, -q_z) \\ \mathbf{q}_4 &= \mathbf{q} \cdot (0, 0, 0, 1) = (+q_x, +q_y, +q_z, q_w) \end{aligned}$$

To multiply with any other  $\mathbf{r}_i$ , means to sum the  $\mathbf{q}_i$  corresponding to its non zero entries, flipped when the entry is -1; then, for class  $B$  or  $C$ , divide by  $\sqrt{2}$  or by 2 respectively. E.g.:

$$\mathbf{q} \cdot (+1, 0, -1, 0) / \sqrt{2} = (\mathbf{q}_1 - \mathbf{q}_3) / \sqrt{2},$$

and so on.

### 1.2.3. Picking the closest representation

Given two 3D crosses represented by quaternions  $\mathbf{p}$  and  $\mathbf{q}$ , we want to find  $\mathbf{r}_i$ ,  $i \in 1, \dots, 48$  which minimizes  $\|\mathbf{p} - \mathbf{q} \cdot \mathbf{r}_i\|^2$ , the distance between  $\mathbf{p}$  and  $\mathbf{q} \cdot \mathbf{r}_i$ .

One solution would be to try each of the 48 possibilities, and pick the one minimizing the distance function above; however, we use a much quicker way.

Recall that minimizing the distance between any two unit quaternions  $\mathbf{a}$  and  $\mathbf{b}$  is equivalent to maximize their dot product  $\langle \mathbf{a}, \mathbf{b} \rangle$  (the dot product of the corresponding 4 dimensional vectors; note: it is not the quaternion multiplication), because:

$$\begin{aligned} \|\mathbf{a} - \mathbf{b}\|^2 &= \\ (\mathbf{a} - \mathbf{b}) \cdot (\bar{\mathbf{a}} - \bar{\mathbf{b}}) &= \\ \|\mathbf{a}\|^2 + \|\mathbf{b}\|^2 - 2\langle \mathbf{a}, \mathbf{b} \rangle &= \\ 2(1 - \langle \mathbf{a}, \mathbf{b} \rangle) \end{aligned}$$

Let the scalars  $a_1..a_4$  be the absolute values of dot products of  $\mathbf{q}_i$  with  $\mathbf{p}$  and  $s_1..s_4$  their *sign* ( $s_i = +1$  or  $-1$ ):

$$\begin{aligned} a_i &= |\langle \mathbf{q}_i, \mathbf{p} \rangle| \\ s_i &= \text{sign}(\langle \mathbf{q}_i, \mathbf{p} \rangle) \end{aligned}$$

Let  $M$  and  $N$  be the indices of the largest and second largest  $a_i$ , that is,  $a_M \geq a_N \geq a_{k \neq M, N}$

We choose rotation  $\mathbf{r}_i$  according to which one of three scalar values  $v_A, v_B, v_C$  is larger:

$$\begin{aligned} v_A &= a_M \\ v_B &= (a_M + a_N)/\sqrt{2} \\ v_C &= (a_1 + a_2 + a_3 + a_4)/2 \end{aligned}$$

- if  $v_A$  is larger: pick the rotation (in class  $A$ ) with  $s_M$  as the single non-zero component at position  $M$ ;
- if  $v_B$  is larger: pick the rotation (in class  $B$ ) with  $s_M$  and  $s_N$  (over  $\sqrt{2}$ ) as the two non-zero components at position  $M$  and  $N$ ;
- if  $v_C$  is larger: pick the rotation (in class  $C$ ) given by  $(s_1, s_2, s_3, s_4)/2$ .

**Soundness:** the choice of rotation  $\mathbf{r}_i$  maximizes  $d = \langle \mathbf{p}, \mathbf{q} \cdot \mathbf{r}_i \rangle$ , as required. Sketch of proof: the tested value for each case ( $v_A, v_B, v_C$ ) is the value of  $d$  for the corresponding choice of  $\mathbf{r}_i$ . Moreover, as it is easy to check, in each of the three cases, the choice of  $\mathbf{r}_i$  for that case dominates over any alternative in the same class.

**Efficiency:** the method requires only four dot products, to find  $a_i$ , and a single GPU operation to find all  $s_i$ . Identification of  $M, N$  takes 4 comparisons, and 2 more are needed to identify the largest of three tested values: therefore we are choosing among 48 possibilities with a total of 6 comparisons.

### 1.2.4. Aligning a 3D cross to a normal

A 3D cross represented by  $\mathbf{q}$  which is sampled on the object boundary must be rotated by the least possible amount so that one of its six directions matches the local object surface normal  $\vec{n} = (n_x, n_y, n_z)$ . To efficiently do so, we first map back  $\vec{n}$  into canonical space, using the inverse rotation  $\bar{\mathbf{q}}$ :

$$\text{local cross} \xrightarrow{\bar{\mathbf{q}}} \text{canonical frame}$$

Let  $\vec{n}' = (n'_x, n'_y, n'_z)$  be the normal expressed in canonical frame, i.e. the imaginary part of

$$\bar{\mathbf{q}} \cdot (n_x, n_y, n_z, 0) \cdot \mathbf{q}.$$

Next, we need to identify the smallest rotation  $\mathbf{s}$  which aligns an axis, or a negated axis, of the canonical frame to  $\vec{n}'$ . Clearly, we must choose the axis corresponding to the coordinate of  $\vec{n}'$  which is largest in module, negated if that coordinate is negative. Finally, the aligned 3D cross is simply found as  $\mathbf{q} \cdot \mathbf{s}$ . This way, the chosen canonical axis is mapped first, by  $\mathbf{s}$ , into  $\vec{n}'$  and then, by  $\mathbf{q}$ , into  $\vec{n}$ , as we wanted:

$$\text{a canonical axis} \xrightarrow{\mathbf{s}} \vec{n}' \xrightarrow{\mathbf{q}} \vec{n}$$

We note that a rotation  $\mathbf{s}'$  of exactly twice the required angle for  $\mathbf{s}$  is found by simply swizzling the coordinates of  $\vec{n}'$ . Specifically, depending on the chosen axis (but independently from its sign), we have:

$$\mathbf{s}' = \begin{cases} \begin{pmatrix} 0, & +n'_z, & -n'_y, & n'_x \end{pmatrix} & \text{if axis} = \pm X \\ \begin{pmatrix} -n'_z, & 0, & +n'_x, & n'_y \end{pmatrix} & \text{if axis} = \pm Y \\ \begin{pmatrix} +n'_y, & -n'_x, & 0, & n'_z \end{pmatrix} & \text{if axis} = \pm Z \end{cases}$$

Proof (sketch): call  $\vec{c}$  the chosen canonical axis; let  $\alpha$  be the angle from  $\vec{c}$  to  $\vec{n}'$ , and  $\vec{r}$  be the unit vector orthogonal to both. Then  $\mathbf{s}$  is the rotation of angle  $\alpha$  around a line defined by  $\vec{r}$ . When  $\vec{c}$  is a positive canonical axis, it is easy to check that the imaginary part of  $\mathbf{s}'$  is  $\vec{n}' \times \vec{c}$ , which is  $\sin(\alpha) \cdot \vec{r}$ , and its real part is  $\langle \vec{n}', \vec{c} \rangle = \cos(\alpha)$ . Therefore,  $\mathbf{s}'$  rotates by  $2\alpha$  around  $\vec{r}$ . When  $\vec{c}$  is a negative canonical axis, the sign of both real and imaginary part of  $\mathbf{s}'$  are flipped, and the rotation does not change.

To halve the angle of  $\mathbf{s}'$  and thus find  $\mathbf{s}$ , we simply average  $\mathbf{s}'$  with either quaternion representing the identity rotation,  $(0, 0, 0, \pm 1)$ , chosen in accordance to the sign of the real part of  $\mathbf{s}'$  (then renormalize). This is sound because the interpolation is half-way, and the rotation amplitude is necessarily  $< \pi/2$ .

The division by 2 of the averaging can be omitted due to the final normalization. In total, up to a renormalization, we have:

$$\mathbf{s} = (s'_x, s'_y, s'_z, \text{sign}(s'_w) + s'_w)$$

where  $\text{sign}(x)$  is the sign of  $x$  (+1 or -1).