# Intersection of tetrahedra

CONOR MCCOID, McMaster University, Canada
MARTIN J. GANDER, Université de Genève, Switzerland

When intersecting non-matching three dimensional lattices, one needs to calculate the intersections of tetrahedra. The authors' previously published two dimensional triangle-triangle intersection algorithm suggests a novel approach in three dimensions based on parsimony. The algorithm presented here expands on this two dimensional algorithm and introduces new strategies necessitated by the increase in dimension. An extensive proof is given for the consistency of the algorithm. Thus, the algorithm is shown to be robust to numerical error arising from floating-point arithmetic. Example problems demonstrate its use and effectiveness.

CCS Concepts: • **Mathematics of computing** → Computation of transforms; • **Software and its engineering** → **Consistency**; **Software reliability**; • **Computing methodologies** → *Collision detection.*

**ACM Reference Format:**
Conor McCoid and Martin J. Gander. 2025. Intersection of tetrahedra. *ACM Trans. Math. Softw.* 1, 1, Article 1 (January 2025), 26 pages. https://doi.org/10.1145/3729532

## 1 Introduction

When considering complex problems in three dimensional space it is sometimes necessary to consider a secondary lattice overlapping a primary. While one hopes these lattices align in some way, this is not guaranteed. It is then a problem to project information from one lattice onto the other. In such instances it is necessary to intersect the lattices, observing how much two given volumes share a space. The intersection between two tetrahedra must be calculated, ideally quickly and robustly in floating-point arithmetic.

This paper continues the work done in [15]. The authors presented there an algorithm for triangle-triangle intersections in 2D. It goes on to prove that the algorithm is robust in floating-point arithmetic. This algorithm is summarized here as Algorithm 1 to show the necessary modifications when moving to three dimensions. It calculates the intersection between two triangles $U$ and $V$. Note that the $\text{sign}(p)$ function used here is defined as

$$\text{sign}(p) = \begin{cases} 0 & p < 0, \\ 1 & p \geq 0. \end{cases}$$

Naturally, the tetrahedral version of this algorithm has some added complications. Most crucially, there are now two types of intersections to find, those between the edges of $U$ and the faces of $V$ and those between the edges of $V$ and the faces of $U$. The 3D version of line 11 in Algorithm 1 prevents nesting these two types of calculations, as there are now multiple ways to arrive at the same intersection.

Authors' Contact Information: Conor McCoid, McMaster University, Hamilton, ON, Canada, mccoidc@mcmaster.ca; Martin J. Gander, Université de Genève, Genève, Switzerland, martin.gander@unige.ch.


Manuscript submitted to ACM

**Algorithm 1** PANG2 - intersection subroutine

1: Find $X$, the barycentric coordinates of $U$ with respect to $V$, and set $Y$ as the reference triangle with edges of length 1 aligned with the coordinate axes
2: **for** all vertices $\mathbf{x}_i$ of $X$ **do**
3:  **if** all coordinates are non-negative **then**
4:  The $i$–th vertex of $U$ lies in $V$
5:  **end if**
6: **end for**
7: **for** all lines $\{p = 0\}$, $p$ a barycentric coordinate **do**
8:  **for** all pairs of vertices such that $\text{sign}(p_i) \neq \text{sign}(p_j)$ **do**
9:  Calculate the two unknown coordinates $q_p^{ij}$ of the intersections
10:  **end for**
11:  Connect $\text{sign}(q_p^{ij})$ with $\text{sign}(p_q^{ij})$ with known relation
12:  **if** $\text{sign}(q_p^{ij}) \neq \text{sign}(q_p^{ik})$ **then**
13:  The vertex of $Y$ at $P_p \cap P_q$ lies in $X$
14:  **end if**
15: **end for**

Several algorithms for the intersection of tetrahedra already exist. For a succinct survey of current intersection algorithms, see [18]. For a comparison of major algorithms, see [13]. We describe a few of these algorithms that have particular relevance to the current paper.

Firstly, there is the Cyrus-Beck algorithm [4], which has become ubiquitous in applications. The key step of this algorithm is to take two vertices of one tetrahedron $U$ and test if they lie on opposite sides of a plane that contains a face of the other tetrahedron $V$. If so, then an intersection is calculated between this edge of $U$ and this face of $V$. We have adapted this step in line 8 of Algorithm 1, and will adapt it further for the 3D version.

Secondly, parametric equations are frequently used in such algorithms [14]. We have largely eschewed these by using barycentric coordinates. This presents greater upfront cost but minimizes calculations during runtime.

Finally, we mention the Sutherland-Hodgman algorithm [19]. This algorithm defines one of the tetrahedra, $V$ as the space bounded by planes. The second tetrahedron $U$ is sequentially intersected with these planes, so that at each step the intersection is that portion of $U$ in the 'positive' half-space defined by the plane with a face equal to the slice of $U$ that lies within the plane. This is highly asymmetric, as the planes of $V$ now have an order by which they intersect $U$. The algorithm we will present includes Sutherland-Hodgman as a subset of calculations, up to numerical error. However, we regain symmetry between the planes of $V$.

The principal strategy employed in the algorithm presented here, namely determining the signs of coordinates by avoiding direct calculation, is not mutually exclusive with other strategies. For example, one can employ snap-rounding [5, 12], and then use the algorithm presented here on the integer-valued coordinates. Several aspects would have to be adapted to this new context, but in principle a combined approach is possible.

Simplicial intersections are performed as subroutines in multiscale methods applied to finite element methods [7]. In particular, 3D Arlequin methods require tetrahedral intersections [20]. See [6] for details on the Arlequin method and [1] for an example of its use of triangular intersections. Mortar methods [17, 21] also use simplicial intersections, though primarily 1D and 2D. Tetrahedral intersections would be a part of 4D mortar methods, but the authors are unaware of any use of such methods. The predecessors of the algorithm presented here, PANG and PANG-3D [9, 10], have been employed in [2, 3, 11, 16].

Robust simplicial intersection algorithms are but one necessary component of robust lattice intersection algorithms. The algorithm presented here must be combined with some strategy for progressing through lattices, such as an advancing front algorithm [8–10]. If the strategy employed is not robust in its own right, i.e. small errors in the calculation of any one simplicial intersection cause large errors in the resulting lattice intersection, then using a robust simplicial intersection algorithm will not salvage it. The original advancing front algorithm used in PANG and PANG-3D was susceptible to small errors that would prematurely terminate the advance of the front. An update released with [15] improved the algorithm, but it is not known if it is robust.

## 2 Change of coordinates

To simplify calculations, transform the tetrahedron $V$ into the reference tetrahedron $Y$ with vertices at the positions $(0,0,0)$, $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$. The tetrahedron $U$ is likewise transformed under the same affine transformation into the tetrahedron $X$. To do so, one must determine the nature of the affine transformation.

Represent the positions of the vertices of $V$ by the matrix

$$\mathbf{v}_0\mathbf{1}^\top + \begin{bmatrix}\mathbf{0} & \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3\end{bmatrix},$$

where $\mathbf{v}_0$ is the position of the vertex to be mapped to the origin, $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{v}_3$ are the vectors spanning between $\mathbf{v}_0$ and the remaining vertices, $\mathbf{0}$ is a column vector with three zeros and $\mathbf{1}^\top = \begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix}$. Ideally, $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{v}_3$ are orthogonal. The best choice of $\mathbf{v}_0$ is one in which this is true, or nearly so.

The process of transforming from the vertices of $V$ to the vertices of $Y$ can be written as an affine transformation:

$$A\left(\mathbf{v}_0\mathbf{1}^\top + \begin{bmatrix}\mathbf{0} & \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3\end{bmatrix}\right) + \mathbf{b}\mathbf{1}^\top = \begin{bmatrix}0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}.$$

The vector $\mathbf{b}$ is then $-A\mathbf{v}_0$ and the matrix $A$ is the inverse of the matrix $\begin{bmatrix}\mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3\end{bmatrix}$.

This affine transformation must be applied to the 'subject' tetrahedron $U$ to acquire its transformation $X$. As with $V$, the position of the vertices of $U$ may be represented by the matrix $\mathbf{u}_0\mathbf{1}^\top + \begin{bmatrix}\mathbf{0} & \mathbf{u}_1 & \mathbf{u}_2 & \mathbf{u}_3\end{bmatrix}$. Let the $i$–th vertex of $X$ have position $(x_i, y_i, z_i)$. These values may then be found by solving the system

$$\begin{bmatrix}\mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3\end{bmatrix}\begin{bmatrix}x_0 & x_1 & x_2 & x_3\\ y_0 & y_1 & y_2 & y_3\\ z_0 & z_1 & z_2 & z_3\end{bmatrix} = \mathbf{u}_0\mathbf{1}^\top + \begin{bmatrix}\mathbf{0} & \mathbf{u}_1 & \mathbf{u}_2 & \mathbf{u}_3\end{bmatrix} - \mathbf{v}_0\mathbf{1}^\top. \tag{1}$$

It will be useful to add a fourth coordinate to each vertex of $X$. Let $w_i = 1 - x_i - y_i - z_i$ for all $i$. By including this coordinate the sum of the coordinates is equal to 1. Thus, these are the barycentric coordinates of $U$ with respect to $V$.

Snippet 1 gives a simple four line procedure for computing these coordinates in MATLAB. The coordinates of the tetrahedra $U$ and $V$ are taken as input and the barycentric coordinates of $X$ are given as output.

## 3 Corners of the intersection

The intersection between the tetrahedra $X$ and $Y$ is a polyhedron $Z$. There are four types of corners to this polyhedron: vertices of $X$ that lie inside $Y$; intersections between the edges of $X$ and the faces of $Y$; intersections between the edges of $Y$ and the faces of $X$ and; vertices of $Y$ that lie inside $X$. These corners form a hierarchy, with each type informing the calculations of later types. The levels of this hierarchy will be considered one at a time.

```
v0=V(:,1);
A=V(:,2:4)-v0;
X=A\(U-v0);
X=[1-sum(X,1);X];   % barycentric coord.s of U
```

Snippet 1. MATLAB code snippet of the change of coordinates. V and U are $3 \times 4$ matrices with each row corresponding to an original coordinate and each column a tetrahedral vertex. The resulting $4 \times 4$ matrix X is the transformation of U into the barycentric coordinate system of V.

```
sX=X>=0;              % signs of barycentric coordinates
if prod(sum(sX,2))==0
  W=[];
  return
end
W=U(:,prod(sX)==1);   % vertices of U in V
```

Snippet 2. MATLAB code snippet of the check of vertices of $X$ that lie inside $Y$. Signs are stored using the binary-valued sign function. First, a check is made if $X$ lies entirely on one side of a plane $P_p$, which implies no intersection between $X$ and $Y$. Then, if the product of all signs in a column is 1, then the corresponding vertex of $U$ lies in $V$.

Geometrically, this hierarchy is symmetric. The calculations of the intersections between edges of $X$ and faces of $Y$ are identical to those between the edges of $Y$ and faces of $X$, if one exchanges the roles of $X$ and $Y$. However, in floating-point arithmetic, these calculations may become inconsistent. In particular, if one of these sets fails to find intersections, the numerical result may be two dimensional and therefore essentially eliminated. It is therefore crucial that each level of the hierarchy be consistent with the previous levels.

### 3.1 Vertices of $X$ that lie inside $Y$

The reference tetrahedron $Y$ is bounded by four infinite planes: $P_x = \{x = 0\}$, $P_y = \{y = 0\}$, $P_z = \{z = 0\}$ and $P_w = \{x + y + z = 1\} = \{w = 0\}$. Each plane is defined by the subspace of $\mathbb{R}^3$ where one of the coordinates is zero. The tetrahedron $Y$ lies at the intersection of the non-negative half-spaces bounded by these planes. Therefore, the $i$–th vertex of $X$, $\mathbf{x}_i = (w_i, x_i, y_i, z_i)$, lies in $Y$ if and only if all coordinates are non-negative. Snippet 2 gives this check in MATLAB code.

The edge between two vertices intersects $P_p$ only if $\text{sign}(p_i) \neq \text{sign}(p_j)$, where $p$ is one of the four coordinates. The signs of $p_i$ then indicate the number of intersections between the edges of $X$ and the plane $P_p$ to calculate. Snippet 3 shows MATLAB code that determines which edges intersect which planes.

The case where $p_i = 0$ is considered in [15] and will be briefly summarized here. Moving $\mathbf{x}_i$ an imperceptible distance into $Y$ does not change the shape of the polyhedron of intersection. Thus, the degenerate case where $p_i = 0$ can be treated as the non-degenerate case where $0 < p_i < \epsilon_m$, i.e. $p_i$ is positive but below machine precision. It is therefore practical to use the binary-valued sign function previously defined.

Proposition 1. *Only zero, three or four intersections may occur between the edges of $X$ and the plane $P_\gamma$.*

Proof. For an intersection to exist, $\text{sign}(p_i)$ and $\text{sign}(p_j)$ must disagree. There are four $p_i$ ($i = 0, ..., 3$), and $\text{sign}(p_i)$ may take one of two values. There are only three ways to partition four objects ($p_i$) into two groups (either 0 or 1), which may be proven by the partition function. These partitionings are listed in Table 1, where $m(a)$ and $m(b)$ are the multiplicities of elements labelled $a$ and $b$, respectively. A pair is formed by taking one element of each group. The number of pairs is then the product of the two multiplicities. □

```
S1=false(4,6);                      % indicators of intersections
jmap=[1,1,2,1,2,3;                  % ordering of edges
      2,3,3,4,4,4];

for i=1:4                           % for each ref. plane
  for j=1:6                         % for each pair of vertices of X
    j1=jmap(1,j); j2=jmap(2,j);     % vertices of this pair
    if sX(i,j1)~=sX(i,j2)           % determine if they intersect
      S1(i,j)=true;
    end
  end
end
```

Snippet 3. MATLAB code snippet that determines which edges of $X$ intersect which planes of $Y$. Also included is a declaration of those pairs of vertices of $X$ that make up a given edge. A consistent ordering of these edges must be used throughout any implementation.

| $m(a)$ | $m(b)$ | pairs |
|---|---|---|
| 4 | 0 | 0 |
| 3 | 1 | 3 |
| 2 | 2 | 4 |

Table 1. Ways to partition four elements into two parts.

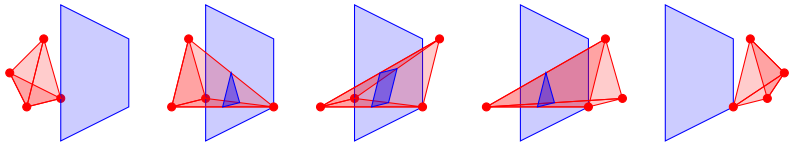

Fig. 1. The five possible configurations of $X$ with respect to a plane of $Y$. There are either zero, three or four intersections in the plane.

This proposition tells us that the part of $X$ that intersects the plane of $Y$ is a triangle, a quadrilateral, or does not exist, see Figure 1. This allows us to consider the intersection of these shapes with the face of $Y$ that lies in the plane, thereby reducing to a two dimensional calculation.

### 3.2 Intersections between edges of $X$ and faces of $Y$

Suppose $\text{sign}(p_i) \neq \text{sign}(p_j)$ for some $p$. Then there is an intersection between the edge of $X$ lying between the $i$–th and $j$–th vertices and the plane $P_p$. This intersection lies in the plane $P_p$ and so its value of $p$ is zero. The remaining three coordinates must be found through calculation.

Let $q_p^{ij}$ be one of the three coordinates of this intersection. Note it is specified by two vertices of $X$, $i$ and $j$, and the particular plane $P_p$ this edge intersects. Using Cramer's rule, it is equal to

$$q_p^{ij} = \frac{q_j p_i - q_i p_j}{p_i - p_j} = \frac{\begin{vmatrix} q_i & p_i \\ q_j & p_j \end{vmatrix}}{\begin{vmatrix} 1 & p_i \\ 1 & p_j \end{vmatrix}}. \tag{2}$$

Examples of the numerators of this expression are presented in Table 2. The intersections $w_p^{ij}$ and $q_w^{ij}$ are presented

| Plane $P_p$ | Numerator of $x_p^{ij}$ | Numerator of $y_p^{ij}$ | Numerator of $z_p^{ij}$ | Numerator of $w_p^{ij}$ |
|---|---|---|---|---|
| $x = 0$, $p = x$ | $\begin{vmatrix} x_i & y_i \\ x_j & y_j \end{vmatrix}$ | | $\begin{vmatrix} x_i & z_i \\ x_j & z_j \end{vmatrix}$ | $\begin{vmatrix} x_i & 1-y_i-z_i \\ x_j & 1-y_j-z_j \end{vmatrix}$ |
| $y = 0$, $p = y$ | $\begin{vmatrix} y_i & x_i \\ y_j & x_j \end{vmatrix}$ | | $\begin{vmatrix} y_i & z_i \\ y_j & z_j \end{vmatrix}$ | $\begin{vmatrix} y_i & 1-x_i-z_i \\ y_j & 1-x_j-z_j \end{vmatrix}$ |
| $z = 0$, $p = z$ | $\begin{vmatrix} z_i & x_i \\ z_j & x_j \end{vmatrix}$ | $\begin{vmatrix} z_i & y_i \\ z_j & y_j \end{vmatrix}$ | | $\begin{vmatrix} z_i & 1-x_i-y_i \\ z_j & 1-x_j-y_j \end{vmatrix}$ |
| $x + y + z = 1$, $p = w$ | $\begin{vmatrix} 1-y_i-z_i & x_i \\ 1-y_j-z_j & x_j \end{vmatrix}$ | $\begin{vmatrix} 1-x_i-z_i & y_i \\ 1-x_j-z_j & y_j \end{vmatrix}$ | $\begin{vmatrix} 1-x_i-y_i & z_i \\ 1-x_j-y_j & z_j \end{vmatrix}$ | |

Table 2. Numerators of intersection coordinates for each plane of $Y$.

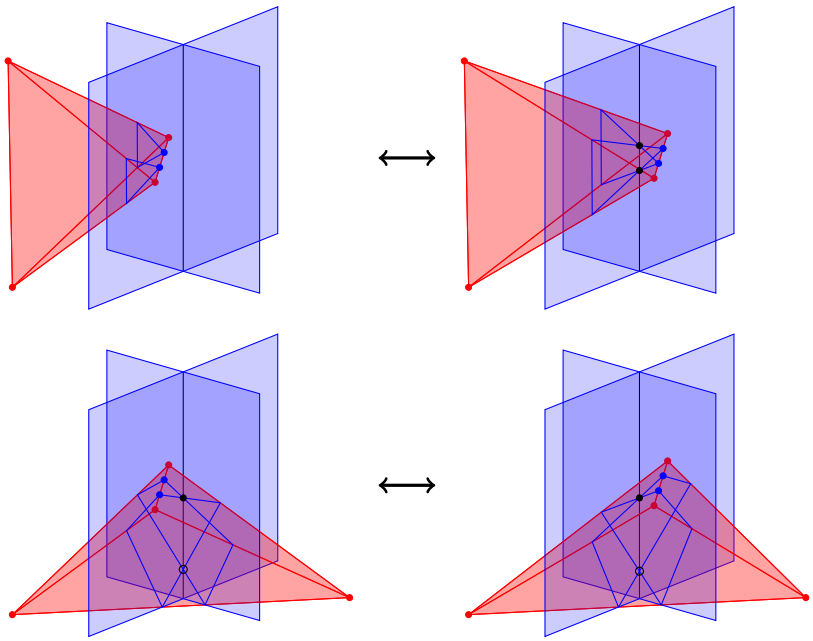

Fig. 2. Intersections between edges of $X$ and planes of $Y$ are connected along the same edge. A change in the sign of one of them causes a commensurate change in the sign of the other.

without reference to the barycentric coordinate $w$, should one wish to avoid storing this coordinate. Each value in Table 2 appears twice. Thus, of twelve entries only six need to be calculated. Moreover, Table 2 connects these intersections together, see Figure 2.

Before codifying this relationship between signs, we must first introduce a new operator for the function $\text{sign}(p)$. When using trinary-valued sign functions, regular multiplication allows the product of two negative signs to be positive, so that the product of the signs of two numbers equals the sign of the product of the two numbers. For binary-valued sign functions, this behaviour is preserved with the logical biconditional operator.

*Definition 1 (Logical biconditional operator).* The operator $\otimes$ that acts as $0 \otimes 0 = 1$, $1 \otimes 1 = 1$ and $0 \otimes 1 = 0$ is called the logical biconditional operator.

```
sQ=false(4,6,4);                % signs of intersections (# faces of Y x # edges of X x # coord.s)
Q=zeros(4,6,4);                 % intersection coord.s (# coord.s x # int.s x # faces)
indi=1:4;
for j=1:6                       % for each pair of vertices (edge of X)
  j1=jmap(1,j); j2=jmap(2,j);   % pair of vertices
  flagQ=false(4,4);             % flags for known signs (# planes x # coords.)
  for i=indi(S1(:,j))           % for each ref. plane with an intersection
    indk=indi(indi~=i);         % don't need to look at coord. assoc. w/ plane i
    sQ(i,j,i)=true;             % coord. assoc. w/ plane i is zero and therefore positive
    for k=indk                  % for each remaining coord.
      if ~S1(k,j)               % no int. for this edge of X in k direction
        sQ(i,j,k)=sX(k,j1);                                 % inherited sign
        flagQ(i,k)=true;                                    % sign now known
      elseif flagQ(k,i)                                     % paired sign is already known
        sQ(i,j,k)=~xor(sQ(k,j,i),~xor(sX(i,j1),sX(k,j2)));
        flagQ(i,k)=true;                                    % sign now known
      elseif ~flagQ(i,k)                                    % check if sign has been found
        q=EdgeIntersect(X([i,k],j1),X([i,k],j2));           % calc. coord.
        Q(k,j,i)=q;                                         % store coord.
        sQ(i,j,k)=q>=0;                                     % determine sign directly
        flagQ(i,k)=true;                                    % sign now known
      end
    end
  end
end
```

Snippet 4. MATLAB code for determining signs of intersection coordinates, using previous knowledge to avoid calculations. The `~xor()` function operates as $\otimes$, see Definition 1. The `EdgeIntersect()` function can implement equation (2) or any other method for finding the intersection of two lines with endpoints `X([i,k],j1)` and `X([i,k],j2)`.

Lemma 2. *Suppose* $\operatorname{sign}(q_i) \neq \operatorname{sign}(q_j)$ *and* $\operatorname{sign}(p_i) \neq \operatorname{sign}(p_j)$, *then*

$$\operatorname{sign}(q_p^{ij}) \otimes \operatorname{sign}(p_q^{ij}) = \operatorname{sign}(q_i) \otimes \operatorname{sign}(p_j).$$

Proof. The result is found by considering the ratio between $q_p^{ij}$ and $p_q^{ij}$:

$$\begin{aligned}\operatorname{sign}(q_p^{ij}) \otimes \operatorname{sign}(p_q^{ij}) &= \operatorname{sign}\left(\frac{q_p^{ij}}{p_q^{ij}}\right) = \operatorname{sign}\left(\frac{\begin{vmatrix} q_i & p_i \\ q_j & p_j \end{vmatrix}}{\begin{vmatrix} 1 & p_i \\ 1 & p_j \end{vmatrix}} \frac{\begin{vmatrix} 1 & q_i \\ 1 & q_j \end{vmatrix}}{\begin{vmatrix} p_i & q_i \\ p_j & q_j \end{vmatrix}}\right) = \operatorname{sign}\left(-\frac{q_j - q_i}{p_j - p_i}\right) \\ &= \operatorname{sign}(q_i) \otimes \operatorname{sign}(p_j),\end{aligned}$$

thus giving the statement of the lemma. □

An intersection lies on $Y$ if and only if all of its coordinates are non-negative. The sign of each coordinate is therefore important. This sign can be retrieved without calculation if the two vertices $\mathbf{x}_i$ and $\mathbf{x}_j$ have the same sign in that coordinate. Then the sign of the intersection is inherited. Note in that case the intersection's pair in Table 2 does not need to be found, as it fails the test $\operatorname{sign}(p_i) \neq \operatorname{sign}(p_j)$. This allows for efficient implementation, see Snippet 4 for an example in MATLAB.

```
if prod(sQ(i,j,:))==1                                % all coord.s positive
  for k=indk(indk~=1)                                % for all coord.s (except zeroth coord.)
    if Q(k,j,i)==0                                   % this coord. not yet calculated
      Q(k,j,i)=EdgeIntersect(X([i,k],j1),X([i,k],j2));
    end
  end
  W=[W, v0+A*Q(2:end,j,i)];                          % add this int. to Z
end
```

Snippet 5. MATLAB code for determining if an intersection lies on $Y$ and transforming it into original coordinates. This snippet is to be inserted between lines 23 and 24 in Snippet 4 for optimal efficiency.

Lemma 3. *Suppose* $\text{sign}(q_i) = \text{sign}(q_j)$ *and* $\text{sign}(p_i) \neq \text{sign}(p_j)$, *then*

$$\text{sign}(q_p^{ij}) = \text{sign}(q_i) = \text{sign}(q_j).$$

Proof. This may be proven directly:

$$\text{sign}(q_p^{ij}) = \text{sign}\left(\frac{q_j p_i - q_i p_j}{p_i - p_j}\right) = \text{sign}(q_j) \otimes \text{sign}(p_i)^{\otimes 2} \otimes \text{sign}\left(\frac{|q_j|\,|p_i| + |q_i|\,|p_j|}{|p_i| + |p_j|}\right) = \text{sign}(q_j),$$

where $s^{\otimes 2} = s \otimes s$ and is always equal to 1 by Definition 1. □

If an intersection is found to lie on $Y$ then we require its coordinates in the original system. Because we use barycentric coordinates retrieving original coordinates is a straightforward matter. If a point has barycentric coordinates $(w, x, y, z)$ then its position in the original coordinates is

$$\mathbf{v}_0 + x\mathbf{v}_1 + y\mathbf{v}_2 + z\mathbf{v}_3. \tag{3}$$

See Snippet 5 for this transformation. The fourth coordinate, $w$, is not used to reverse the transformation. Its purpose is solely to determine the relative position with respect to $P_w$.

*3.2.1 Straight line principle.* A line between two points is naturally convex. This fact has been used to inherit signs of intersections as described above, as well as implicitly in Lemma 2. This ensures the projection of the intersections onto the plane $P_p$ have signs consistent with that of a straight line for any choice of $p$. However, it does not prevent those intersections being consistent in $\mathbb{R}^3$.

To explain further, consider one edge of $X$ between vertices 0 and 1 that passes through the planes $P_x$, $P_y$ and $P_z$. Without loss of generality, $\text{sign}(x_0) = \text{sign}(y_0) = \text{sign}(z_0) = +$, while $\text{sign}(x_1) = \text{sign}(y_1) = \text{sign}(z_1) = -$. Three intersections must be calculated, one for each plane the line passes through. No signs are inherited and Lemma 2 may be applied. Doing so means only three signs need to be calculated and the others may be found through the use of Lemma 2. If those calculations are subjected to round-off error, it is possible to find that each intersection has one positive and one negative coordinate. However, this is impossible for this straight line. We must therefore change some of the signs to enforce its straightness.

Suppose the line represents the path of a particle through space that begins at $\mathbf{x}_0$ and travels to $\mathbf{x}_1$. As it does so, it passes through the planes $P_x$, $P_y$ and $P_z$. Once it passes through each, the particle's coordinate in that direction has changed sign. That is, before the particle has passed through $P_x$, it has a positive sign. Once it passes through $P_x$, it has a negative sign.

| | $x$ | $y$ | $z$ |
|---|---|---|---|
| $\mathbf{x}_0$ | + | + | + |
| $\cap P_x$ | | + | + |
| $\cap P_y$ | [-] | | |
| $\cap P_z$ | [-] | | |
| $\mathbf{x}_1$ | - | - | - |

| | $x$ | $y$ | $z$ |
|---|---|---|---|
| $\mathbf{x}_0$ | + | + | + |
| $\cap P_x$ | | [+] | (+) |
| $\cap P_y$ | - | | + |
| $\cap P_z$ | (-) | [-] | |
| $\mathbf{x}_1$ | - | - | - |

Table 3. Signs of the edge of $X$ between vertices 0 and 1. Square brackets indicate the signs have been found using Lemma 2, while parentheses indicate the use of the straight line principle.

```
if sum(S1(:,j))>2                       % if there are enough int.s for this edge of X
  diff1=xor(permute(sQ(i,j,indk),[3,2,1]),sX(indk,j1)); % find which coord.s have different signs from
  diff2=xor(permute(sQ(i,j,indk),[3,2,1]),sX(indk,j2)); %     the vertices on this edge of X
  if sum(diff1)==1                      % if there is 1 int. between this int. and the 1st vertex
    ind_diff=indk(~diff1);              % for those coord.s which differ
    k=indk(diff1);                      % for the plane that causes the different sign
    sQ(k,j,ind_diff)=sQ(i,j,ind_diff);  % fill in the signs for the intermediary int.
    flagQ(k,ind_diff)=true;             % flag that we now know them
  elseif sum(diff2)==1                  % else if there is 1 int. between this int. and the 2nd vertex
    ind_diff=indk(~diff2);
    k=indk(diff2);
    sQ(k,j,ind_diff)=sQ(i,j,ind_diff);  % fill in the signs for the intermediary int.
    flagQ(k,ind_diff)=true;             % flag that we now know them
  end
end
```

Snippet 6. Straight line principle coded in MATLAB. This is to be inserted between lines 23 and 24 in Snippet 4, after Snippet 5.

The particle must pass through each plane one at a time. Without loss of generality, suppose the particle passes first through $P_x$, then through $P_y$ and finally through $P_z$. Since its sign in the $x$-direction must change once it has passed through $P_x$, $\text{sign}(x_q^{01}) = \text{sign}(x_1)$ for $q = y, z$. Likewise, its sign in the $z$-direction can only change once it has passed through $P_z$, and so $\text{sign}(z_q^{01}) = \text{sign}(z_0)$ for $q = x, y$.

Let us refer to this idea as the straight line principle. It only needs to be applied when one has calculated the signs of the middle intersection along a given line. To see this, consider Table 3, which uses the example described above. On the right, we assume the intersection with $P_x$ has been calculated first and Lemma 2 has provided two other signs. The remaining two signs, those of $z_y^{01}$ and $y_z^{01}$, are linked by Lemma 2, meaning only one is needed to complete the table. The sign of $z_y^{01}$ may be either + or -, and the straight line principle will not be violated.

Meanwhile, on the left we assume the intersection with $P_y$ has been calculated first. Again, Lemma 2 reduces the unknown signs to two, this time $z_x^{01}$ and $x_z^{01}$. The only way to maintain the straight line principle is to have $\text{sign}(z_x^{01}) = +$.

Note this implies that for the six signs associated with this line, there are only two degrees of freedom, rather than the three one expects as a result of Lemma 2. However, it is not known which two signs will immediately provide the other four, as the order of the intersections is determined by the signs.

### 3.3 Intersections between edges of $Y$ and faces of $X$

The edges of $Y$ lie at the intersection of two of the planes, $P_p \cap P_q$. Since from the previous step we have polygons $X \cap P_p$ that lie within $P_p$, we can intersect these polygons with $P_q$ to find the intersection between these edges and faces of $X$. The vertices of these polygons are the intersections between edges of $X$ and faces of $Y$. This level of the

hierarchy is then significantly complicated compared to the previous, as we replace vertices with intersections and triangular faces with polygons.

The procedure is identical in this restricted space. Without loss of generality suppose we are intersecting $X \cap P_x$ with $P_y$. There are three non-zero coordinates in the plane $P_x$: $y$, $z$ and $w$. By Proposition 1 there are either zero, three or four intersections in $X \cap P_x$. An intersection occurs between $X \cap P_x$ and $P_y$ if and only if $\text{sign}\left(y_x^{ij}\right) \neq \text{sign}\left(y_x^{ik}\right)$. There may then be zero, two, three or four edges of $X \cap P_x$ that intersect $P_y$, where each edge lies between a pair of these intersections. If one excludes edges of $X \cap P_x$ on its interior, which we can do since $X \cap P_x$ is convex, then there can be only zero or two such edges that intersect $P_y$.

Code for determining whether this type of intersection occurs is nearly identical to Snippet 3. However, instead of looping over edges of $X$, the snippet loops over faces of $X$ and must therefore associate the previously calculated intersections with particular faces. This is done through careful ordering of objects throughout the implementation.

Let us consider one of those edges, and let us suppose the intersections it connects are indexed by $ij$ and $ik$. Then $\text{sign}\left(y_x^{ij}\right) \neq \text{sign}\left(y_x^{ik}\right)$. The intersection that results has only two non-zero coordinates now, those of $z$ and $w$. These may be written as

$$z_{xy}^{ijk} = \frac{z_x^{ij} y_x^{ik} - z_x^{ik} y_x^{ij}}{y_x^{ik} - y_x^{ij}}, \; w_{xy}^{ijk} = \frac{w_x^{ij} y_x^{ik} - w_x^{ik} y_x^{ij}}{y_x^{ik} - y_x^{ij}}. \tag{4}$$

This makes the intersection calculation recursive.

There are multiple ways to arrive at the same edge of $Y$. In the above example we considered $X \cap P_x$ intersected with $P_y$, but we could also have done $X \cap P_y$ intersected with $P_x$. However, this would have changed the formulas we used to arrive at the intersection, even though this intersection should not depend on the order in which we choose planes. This could easily lead to problems of consistency in the algorithm if these formulas produce different results. To avoid this, we may use a more general formula for these intersections.

Lemma 4. *The $r$–coordinate of the intersection between the face of $X$ with vertices $\mathbf{x}_i$, $\mathbf{x}_j$ and $\mathbf{x}_k$ and $P_p \cap P_q$, where $r$, $p$ and $q$ are chosen from $x$, $y$, $z$ and $w$, is equal to*

$$r_{pq}^{ijk} = \frac{\begin{vmatrix} r_i & p_i & q_i \\ r_j & p_j & q_j \\ r_k & p_k & q_k \end{vmatrix}}{\begin{vmatrix} 1 & p_i & q_i \\ 1 & p_j & q_j \\ 1 & p_k & q_k \end{vmatrix}}. \tag{5}$$

Proof. Let $(w, x, y, z)$ be the coordinates of the intersection. These coordinates are barycentric and the intersection lies on $P_p$ and $P_q$, meaning

$$\begin{bmatrix} \mathbf{1}^\top \\ \mathbf{e}_p^\top \\ \mathbf{e}_q^\top \end{bmatrix} \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix},$$

where $\mathbf{1}$ is a column vector of length 4 with entries equal to 1 and $\mathbf{e}_p$ is the $p$–th column of the $4 \times 4$ identity matrix. Since it lies on the face between the vertices $i$, $j$ and $k$, these coordinates can be represented as

$$\begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} w_i & w_j & w_k \\ x_i & x_j & x_k \\ y_i & y_j & y_k \\ z_i & z_j & z_k \end{bmatrix} \mathbf{w},$$

for some vector $\mathbf{w}$. These facts together give a matrix system for $\mathbf{w}$:

$$\begin{bmatrix} \mathbf{1}^\top \\ \mathbf{e}_p^\top \\ \mathbf{e}_q^\top \end{bmatrix} \begin{bmatrix} w_i & w_j & w_k \\ x_i & x_j & x_k \\ y_i & y_j & y_k \\ z_i & z_j & z_k \end{bmatrix} \mathbf{w} = \begin{bmatrix} 1 & 1 & 1 \\ p_i & p_j & p_k \\ q_i & q_j & q_k \end{bmatrix} \mathbf{w} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

Using Cramer's rule and minor determinant manipulation, the solution $\mathbf{w}$ is

$$\mathbf{w} = \frac{1}{\begin{vmatrix} 1 & 1 & 1 \\ p_i & p_j & p_k \\ q_i & q_j & q_k \end{vmatrix}} \left( \begin{vmatrix} 1 & 0 & 0 \\ p_i & p_j & p_k \\ q_i & q_j & q_k \end{vmatrix}, \begin{vmatrix} 0 & 1 & 0 \\ p_i & p_j & p_k \\ q_i & q_j & q_k \end{vmatrix}, \begin{vmatrix} 0 & 0 & 1 \\ p_i & p_j & p_k \\ q_i & q_j & q_k \end{vmatrix} \right).$$

The $r$–coordinate, which is one of $w$, $x$, $y$ or $z$, is the inner product between $\mathbf{w}$ and $(r_i, r_j, r_k)$. This is equivalent to the form given in the statement of the lemma, after matrix transposition. □

As before, this intersection lies on $Y$ if and only if its coordinates are non-negative in the remaining two directions. As well, if its parent intersections from the previous generation have the same sign in one of those coordinates then its sign is inherited. This becomes more complicated in this higher dimensional case as the intersection has two parent pairs, both of which must be checked.

Take the example we considered earlier, intersecting $X \cap P_x$ with $P_y$. In $P_x$ the two parents were indexed by $ij$ and $ik$. Suppose they have different signs in the $z$–coordinate. As pointed out, we could have instead intersected $X \cap P_y$ with $P_x$. Suppose in $X \cap P_y$ there are two intersections indexed by $ij$ and $jk$, which also gives an intersection in $X \cap P_x \cap P_y$ indexed by $ijk$. However, in this plane suppose these intersections have the same sign in the $z$–coordinate. Then the intersection $ijk$ in $P_x \cap P_y$ has the same sign in the $z$–coordinate as its parents in $P_y$, even though its parents in $P_x$ have different signs.

Like the pairing of numerators from Table 2, this form of the intersection has a tripling of its numerators. Keeping $i$, $j$ and $k$ fixed one can alternate the roles of $r$, $p$ and $q$ to give the same numerator up to a change in sign. This fixes the position of the face $ijk$ of $X$ with respect to the vertex of $Y$ at $P_p \cap P_q \cap P_r$, see Figure 3.

The denominators in this case are connected to the numerators of Table 2. One can use this fact to keep the algorithm consistent with previous calculations. The following lemma gives this connection. However, the corollary that follows combines this fact and the tripling of the numerators for a powerful relation between the signs of intersections' coordinates. This relation is sufficient for the algorithm to be consistent, as we will show in Theorem 6.

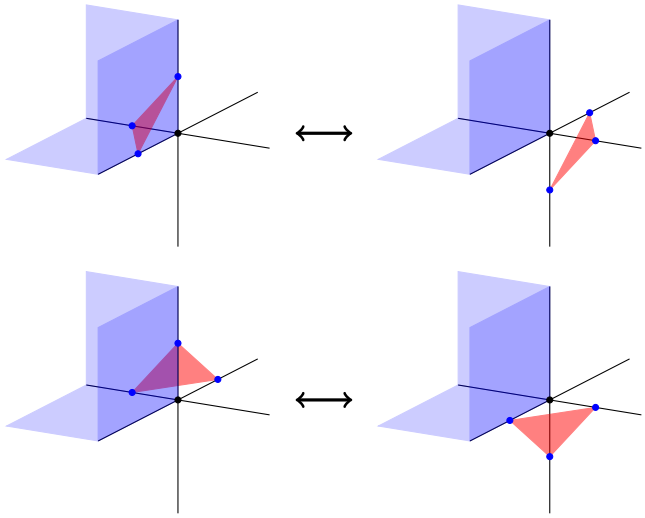

Fig. 3. Intersections between faces of $X$ and lines $P_p \cap P_q$ are connected in triplets. Each red triangle represents the portion of a face of $X$ that intersects three lines $P_p \cap P_q$. The three blue planes represent faces of $Y$. A change in the sign of one coordinate causes two other coordinates of two other intersections to change as well.

Lemma 5. *Suppose the face $ijk$ of $X$ intersects $P_p \cap P_q$. Suppose* $\text{sign}(p_i) \neq \text{sign}(p_j) = \text{sign}(p_k)$ *and* $\text{sign}(q_p^{ij}) \neq \text{sign}(q_p^{ik})$. *Then*

$$\text{sign}\left(\begin{vmatrix} 1 & q_i & p_i \\ 1 & q_j & p_j \\ 1 & q_k & p_k \end{vmatrix}\right) = \text{sign}\left(\begin{vmatrix} q_i & p_i \\ q_j & p_j \end{vmatrix}\right).$$

Proof. By assumption $q_p^{ij} - q_p^{ik}$ has the same sign as $q_p^{ij}$. We then investigate $q_p^{ij} - q_p^{ik}$ by expanding it:

$$q_p^{ij} - q_p^{ik} = \frac{\begin{vmatrix} q_i & p_i \\ q_j & p_j \end{vmatrix}}{p_j - p_i} - \frac{\begin{vmatrix} q_i & p_i \\ q_k & p_k \end{vmatrix}}{p_k - p_i} = \frac{\begin{vmatrix} 0 & q_i & p_i \\ p_j - p_i & q_j & p_j \\ p_k - p_i & q_k & p_k \end{vmatrix}}{(p_j - p_i)(p_k - p_i)} = \frac{-p_i \begin{vmatrix} 1 & q_i & p_i \\ 1 & q_j & p_j \\ 1 & q_k & p_k \end{vmatrix}}{(p_j - p_i)(p_k - p_i)}.$$

We now compare the signs of the two sides:

$$\begin{aligned} \text{sign}\left(q_p^{ij} - q_p^{ik}\right) &= \text{sign}(p_j - p_i) \otimes \text{sign}\left(\begin{vmatrix} q_i & p_i \\ q_j & p_j \end{vmatrix}\right) \\ &= \text{sign}(p_j - p_i) \otimes \text{sign}(-p_i) \otimes \text{sign}(p_k - p_i) \otimes \text{sign}\left(\begin{vmatrix} 1 & q_i & p_i \\ 1 & q_j & p_j \\ 1 & q_k & p_k \end{vmatrix}\right). \end{aligned}$$

Cancelling known signs ($\text{sign}(-p_i) = \text{sign}(p_k - p_i)$ by assumption) we are left with the statement of the lemma. □

Corollary 1. *Suppose* $\text{sign}(p_i) \neq \text{sign}(p_j) = \text{sign}(p_k)$, $\text{sign}(q_p^{ij}) \neq \text{sign}(q_p^{ik})$ *and* $\text{sign}(r_p^{ij}) \neq \text{sign}(r_p^{ik})$, *then*

$$\text{sign}(r_{pq}^{ijk}) \otimes \text{sign}(q_{pr}^{ijk}) = \text{sign}(r_p^{ij}) \otimes \text{sign}(q_p^{ik}).$$

```
indV=false(4,1);                    % indicators for vertices of V
Vmap=[6,2;6,1;1,4;1,3];             % edge/coord. combos of Y for each vertex of V
for vertex=1:4                      % for each vertex of V
  edge=Vmap(vertex,1);              % pick an edge of Y that passes through this vertex
  coord=Vmap(vertex,2);             % pick the coord. that corresponds to this vertex on this edge
  indj=1:4; indj=indj(S2(edge,:)); % pick out faces of X that intersect this edge of Y
  if ~isempty(indj) && sR(coord,edge,indj(1))~=sR(coord,edge,indj(2)) % if int.s lie on opposite sides
    indV(vertex)=true;                                                % of vertex
  end
end
W=[W,V(:,indV)];                                                      % add vertices to P
```

Snippet 7. MATLAB code that determines if a vertex of $V$ lies in $U$. The object S2 contains binary indicators of intersections between faces of $X$ and edges of $Y$. The object sR contains signs of these intersections.

Proof. Take the quotient of $r_{pq}^{ijk}$ and $q_{pr}^{ijk}$, which has the same sign as their product. Expand and use Lemma 5:

$$\operatorname{sign}\left(\frac{r_{pq}^{ijk}}{q_{pr}^{ijk}}\right) = \operatorname{sign}\left(\frac{\begin{vmatrix} r_i & p_i & q_i \\ r_j & p_j & q_j \\ r_k & p_k & q_k \end{vmatrix} \begin{vmatrix} 1 & p_i & r_i \\ 1 & p_j & r_j \\ 1 & p_k & r_k \end{vmatrix}}{\begin{vmatrix} 1 & p_i & q_i \\ 1 & p_j & q_j \\ 1 & p_k & q_k \end{vmatrix} \begin{vmatrix} q_i & p_i & r_i \\ q_j & p_j & r_j \\ q_k & p_k & r_k \end{vmatrix}}\right) = -\operatorname{sign}\left(\frac{\begin{vmatrix} p_i & r_i \\ p_j & r_j \end{vmatrix} \begin{vmatrix} 1 & p_i \\ 1 & p_j \end{vmatrix}}{\begin{vmatrix} p_i & q_i \\ p_i & q_j \end{vmatrix} \begin{vmatrix} 1 & p_i \\ 1 & p_j \end{vmatrix}}\right)$$

$$= -\operatorname{sign}(r_p^{ij}) \otimes \operatorname{sign}(q_p^{ij}) = \operatorname{sign}(r_p^{ij}) \otimes \operatorname{sign}(q_p^{ik}),$$

which is equivalent to the statement of the corollary. □

Let us note briefly that there is no equivalent of the straight line principle, see Section 3.2.1, for intersections between faces of $X$ and edges of $Y$. All intersections that lie on the plane between three vertices of $X$ are already consistent due to Corollary 1.

Code for sign determinations of these intersections appears very similar to Snippet 4. The only major difference is that the inheritance and paired sign checks must be repeated for both of the other intersections in the triple. The reverse transformation can be implemented in identical fashion to Snippet 5.

### 3.4 Vertices of $Y$ that lie inside $X$

Every pair of planes $P_p \cap P_q$ intersects both remaining planes. At each intersection is a vertex of $Y$. There are either zero or two intersections in $X \cap P_p \cap P_q$. Each has an $r$–coordinate, $r_{pq}^{ijk}$ and $r_{pq}^{ijl}$. If $\operatorname{sign}(r_{pq}^{ijk}) \neq \operatorname{sign}(r_{pq}^{ijl})$ then the intersection $P_p \cap P_q \cap P_r$ lies inside $X$. As this is a vertex of $Y$ the corresponding vertex of $V$ lies inside $U$.

The vertex corresponds to the plane of $Y$ that is not in the intersection. For example, if the vertex is at $P_w \cap P_y \cap P_z$ then the only non-zero coordinate is $x$, which must equal one. Using equation (3) gives the position as $\mathbf{v}_0 + \mathbf{v}_1$, as expected.

Each vertex of $Y$ has three edges extending from it. This test can therefore occur up to three times. If the algorithm is consistent it only needs to occur once. The remaining edges will then agree on the results.

Snippet 7 provides MATLAB code for determining if a vertex of $Y$ lies inside $X$. It requires a list of those edges of $Y$ that intersect faces of $X$ and the signs of those intersections.

## 4 Algorithm

We present pseudocode for the algorithm in Algorithm 2. We explain each step of the algorithm.

**Algorithm 2** Tetrahedral intersection algorithm

1: Transform $V \longrightarrow Y$ and $U \longrightarrow X$, see equation (1)
2: **for** all vertices $\mathbf{x}_i$ of $X$ **do**
3: **if** sign$(p_i) = 1$ for all coordinates $p$ **then**
4: $\mathbf{u}_0 + \mathbf{u}_i \in V$
5: **end if**
6: **end for**
7: **for** all planes $P_p$ **do**
8: **for** all pairs of vertices such that sign$(p_i) \neq$ sign$(p_j)$ **do**
9: Flag the edge $ij$ as having an intersection with plane $P_p$
10: **end for**
11: **end for**
12: **for** each intersection **do**
13: Determine the signs of its coordinates, see Lemma 2 and Section 3.2.1
14: **if** all coordinates are non-negative **then**
15: Transformed intersection lies in $Y$, see equation (3)
16: **end if**
17: **end for**
18: Repeat lines 7-17 for all lines $P_p \cap P_q$, see Lemma 4 and Corollary 1
19: **for** all vertices $P_p \cap P_q \cap P_r$ **do**
20: **if** sign$(r_{pq}^{ijk}) \neq$ sign$(r_{pq}^{ijl})$ **then**
21: Vertex lies in $X$
22: **end if**
23: **end for**

**Line 1** Perform a change of coordinates. The coordinates of $Y$ are already known, while the coordinates of $X$ are found by solving equation (1), see Section 2 and Snippet 1.

**Lines 2-6** Find which vertices of $X$ lie within $Y$. Using the barycentric coordinates, these vertices have entirely non-negative coordinates, see Section 3.1 and Snippet 2.

**Lines 7-11** Find which edges of $X$ intersect the planes $P_p$, see Snippet 3.

**Lines 12-17** Determine the signs of the intersections found in the previous step, either through inheritance, Lemma 2, the straight line principle, or direct calculation, see Section 3.2 and Snippets 4 and 6. Section 4.1 that follows discusses these sign determinations further. If all coordinates of an intersection are non-negative, calculate them and reverse the change of coordinates using equation (3), see Snippet 5.

**Line 18** Find which faces of $X$ intersect the lines $P_p \cap P_q$. Determine the signs of these coordinates through inheritance, Corollary 1, or direct calculation. If all coordinates of an intersection are non-negative, calculate them and use equation (3) to reverse the transformation. See Sections 3.3 and 4.1 for discussion.

**Lines 19-23** For each vertex choose one line $P_p \cap P_q$ that extends from it. Compare sign$(r_{pq}^{ijk})$ and sign$(r_{pq}^{ijl})$, the $r-$coordinate of the two intersections along this line. If they differ then the vertex lies in $X$, see Section 3.4 and Snippet 7.

Direct calculation of intersection coordinates can be achieved through equations (2) and (5). However, these are by no means the only options. Any method that finds the intersection between two lines can be used to calculate these coordinates. This aspect of the code should be considered modular.

Much work in Section 3 has been to ensure the algorithm produces a consistent result. With this extensive theory in mind we prove Algorithm 2 does in fact satisfy this high bar of robustness.

THEOREM 6. *Algorithm 2 is consistent with respect to shape.*

PROOF. For the algorithm to be consistent all calculations on the tetrahedron $X$ must be compatible with one another geometrically. We will consider each calculation step by step and show they agree with one another on the fundamental geometry.

The first step of the algorithm on $X$ is determining the signs of the four coordinates of $\mathbf{x}_i$ for all $i$ for the purposes of finding which vertices of $X$ lie within $Y$. As this is the first step there are no previous calculations with which to be compatible. Moreover, each coordinate and vertex are independent and so the calculations cannot be inconsistent with one another.

Next are the calculations of the intersections between $X$ and the planes $P_p$. An intersection with $P_p$ is found if and only if $\text{sign}(p_i) \neq \text{sign}(p_j)$ for some pair $i$ and $j$. This ensures these calculations are consistent with the previous determinations of signs.

For these calculations to be consistent with one another they must agree at the points of their intersections. That is, if there is an intersection $X \cap P_p \cap P_q$ then intersecting $P_q$ with $X \cap P_p$ must produce the same results as intersecting $P_p$ with $X \cap P_q$. Put succinctly, for every pair of intersections in $P_p$ indexed by $J_1$ and $J_2$ such that $\text{sign}(q_p^{J_1}) \neq \text{sign}(q_p^{J_2})$ there exists a pair of intersections in $P_q$ indexed by $J_3$ and $J_4$ such that $\text{sign}(p_q^{J_3}) \neq \text{sign}(p_q^{J_4})$. The sets $J_k$ contain two elements that denote the two vertices on the edge of $X$ that intersects the given plane. For example, if the edge between $\mathbf{x}_0$ and $\mathbf{x}_1$ intersects $P_p$, then $J_1 = 01$.

We may assume the two edges of $X \cap P_p$, indexed by $J_1$ and $J_2$, share a vertex as this restricts the intersection in $P_p \cap P_q$ to the boundary of $X$. Any interior intersections may be ignored. Without loss of generality suppose $J_1$ and $J_2$ have two vertices each from amongst the three $\mathbf{x}_0$, $\mathbf{x}_1$ and $\mathbf{x}_2$. Firstly, since $\text{sign}(q_p^{J_1}) \neq \text{sign}(q_p^{J_2})$ at least one of $q_0$, $q_1$ or $q_2$ has a different sign than the others. Otherwise, the signs of the intersections would both be inherited from their parents and they would both be equal to one another, see left of Figure 4. Without loss of generality, suppose $\text{sign}(q_0) \neq \text{sign}(q_1) = \text{sign}(q_2)$. Then there exist intersections with $p$–coordinates $p_q^{01}$ and $p_q^{02}$. It remains to show these have different signs.

The sets $J_k$ are then four combinations of two elements from a set of three. By the pigeonhole principle two of them are identical. Without loss of generality, suppose $J_1 = 01$. Lemma 2 gives the following relation between the signs of the coordinates:

$$\text{sign}(q_p^{01}) \otimes \text{sign}(p_q^{01}) = \text{sign}(q_0) \otimes \text{sign}(p_1).$$

There are then two cases to consider: either $J_2 = J_3 = 02$ or $J_3 \neq J_2 = 12$. In the former, $q_p^{02}$ and $p_q^{02}$ have the same relationship as $q_p^{01}$ and $p_q^{01}$, see centre of Figure 4. Thus,

$$\begin{aligned}\text{sign}(p_q^{02}) &= \text{sign}(q_0) \otimes \text{sign}(p_2) \otimes \text{sign}(q_p^{02}) \\ &\neq \text{sign}(q_0) \otimes \text{sign}(p_1) \otimes \text{sign}(q_p^{01}) = \text{sign}(p_q^{01}).\end{aligned}$$

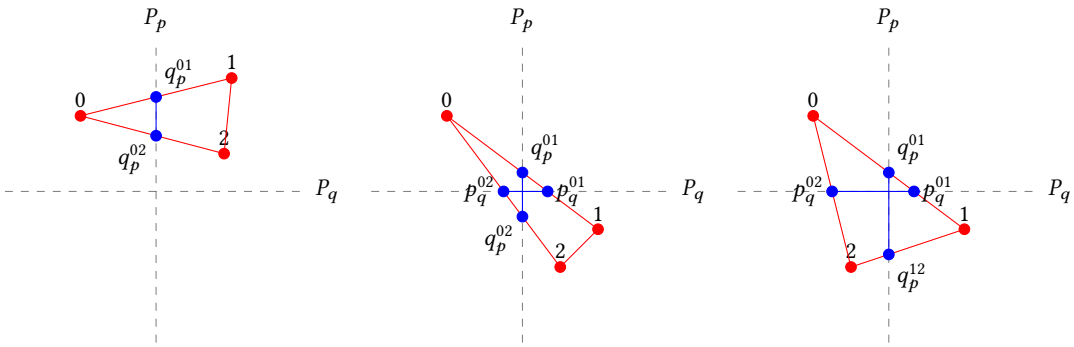


Fig. 4. Configurations of two intersections $q_p^{J_1}$ and $q_p^{J_2}$ with respect to two planes $P_p$ and $P_q$.

In the latter, $\text{sign}(q_p^{12})$ is inherited from $q_2$ and $q_1$ while $\text{sign}(p_q^{02})$ is inherited from $p_0$ and $p_2$, see right of Figure 4. Then

$$\begin{aligned}\text{sign}(p_q^{02}) = \text{sign}(p_0) \neq \text{sign}(p_1) &= \text{sign}(q_0) \otimes \text{sign}(q_p^{01}) \otimes \text{sign}(p_q^{01}) \\ &= \text{sign}(q_1) \otimes \text{sign}(q_p^{12}) \otimes \text{sign}(p_q^{01}) = \text{sign}(p_q^{01}).\end{aligned}$$

In both cases, the two coordinates have different signs. The four intersections, indexed by $J_1$, $J_2$, $J_3$ and $J_4$, form a cross around the intersection at $X \cap P_p \cap P_q$, which is itself indexed by the three vertices of $X$ that form the relevant face of $X$, $J_1 \cup J_2 = J_3 \cup J_4 = 012$.

We next find the intersections between $X$ and $P_p \cap P_q$. Since the calculations of $X \cap P_p$ and $X \cap P_q$ are consistent, we may consider $X \cap P_p \cap P_q$ as the intersection of $X \cap P_p$ with $P_q$. By Proposition 1 there are either zero, three or four intersections in $X \cap P_p$. If there are zero then certainly $X \cap P_p \cap P_q$ is empty as well. If there are three then by the same reasoning behind Proposition 1 there are either zero or two intersections in $X \cap P_p \cap P_q$, see [15] for further discussion.

If there are four intersections and $X \cap P_p$ is convex then there are also either zero or two intersections in $X \cap P_p \cap P_q$, since a line can intersect a convex polygon along either none or two of its edges. However, if the algorithm fails to maintain convexity in floating-point arithmetic, then there may be four intersections due to twisting of the polygon's edges. We suppose for the moment that convexity is maintained and will consider the non-convex case later in the proof.

For the edges $X \cap P_p \cap P_q$ to be consistent with one another they must agree at their intersection, just as in the case of $X \cap P_p$. The statement to prove in this case is then: For every pair of intersections $(J_1, J_2)$ such that $\text{sign}(r_{pq}^{J_1}) \neq \text{sign}(r_{pq}^{J_2})$ there exist two pairs $(J_3, J_4)$ and $(J_5, J_6)$ such that $\text{sign}(q_{pr}^{J_3}) \neq \text{sign}(q_{pr}^{J_4})$ and $\text{sign}(p_{qr}^{J_5}) \neq \text{sign}(p_{qr}^{J_6})$.

The pair $(J_1, J_2)$ can be arrived at from two ways: either they are intersections of $X \cap P_p$ with $P_q$, or those of $X \cap P_q$ with $P_p$. We consider the former. Then there are two pairs $(K_1, K_2)$ and $(K_3, K_4)$ such that

$$\begin{aligned}\text{sign}(q_p^{K_1}) &\neq \text{sign}(q_p^{K_2}), & \text{sign}(q_p^{K_3}) &\neq \text{sign}(q_p^{K_4}), \\ \text{sign}(q_p^{K_1}) &= \text{sign}(q_p^{K_4}), & \text{sign}(q_p^{K_2}) &= \text{sign}(q_p^{K_3}), \\ K_1 \cup K_2 &= J_1, & K_3 \cup K_4 &= J_2.\end{aligned}$$

Note it is possible that $K_2 = K_3$ or $K_1 = K_4$ but not both simultaneously.

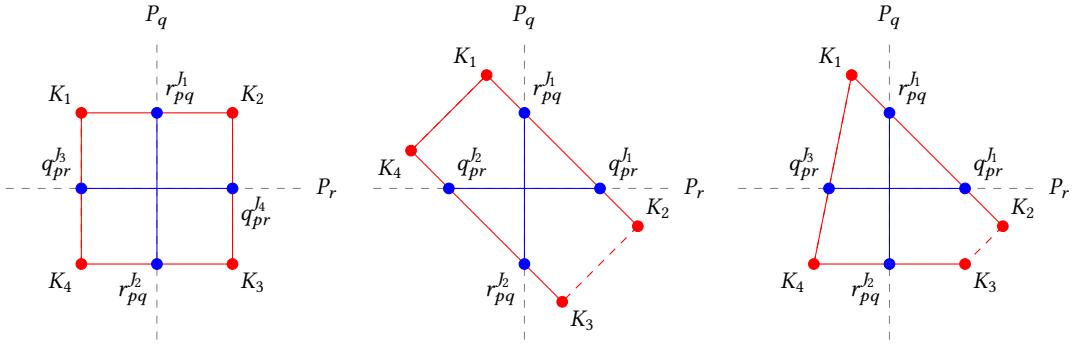


Fig. 5. Convex configurations of $X \cap P_p$. The dashed lines represent zero or one edges, depending on how many intersections have been found for $X \cap P_p$.

Since $\text{sign}(r_{pq}^{J_1}) \neq \text{sign}(r_{pq}^{J_2})$ at least one of the intersections in $X \cap P_p$ has a different sign than the others in the $r-$coordinate. There are a limited number of convex configurations satisfying these conditions. These are presented in Figure 5. In all cases there are two intersections between $X \cap P_p$ and $P_r$, indexed by $J_3 = K_2 \cup K_3$ and $J_4 = K_1 \cup K_4$.

If the sets $J_3$ and $J_4$ are not identical to $J_1$ and $J_2$ then the edge between $K_1$ and $K_2$ does not intersect $P_r$, neither does the one between $K_3$ and $K_4$. Then $\text{sign}(r_{pq}^{J_1}) = \text{sign}(r_p^{K_1}) = \text{sign}(r_p^{K_2})$ and likewise for $J_2$. Moreover, $\text{sign}(q_{pr}^{J_3}) = \text{sign}(q_p^{K_1}) = \text{sign}(q_p^{K_4})$ and likewise for $J_4$. This configuration is represented in the left of Figure 5.

Suppose instead $J_4 = J_1$, then $\text{sign}(r_p^{K_1}) \neq \text{sign}(r_p^{K_2})$. Then we may use Corollary 1 to establish

$$\text{sign}(r_{pq}^{J_1}) \otimes \text{sign}(q_{pr}^{J_1}) = \text{sign}(r_p^{K_2}) \otimes \text{sign}(q_p^{K_1}).$$

If $J_3 = J_2$ also, then $\text{sign}(r_p^{K_4}) \neq \text{sign}(r_p^{K_3})$ and, using Corollary 1 again,

$$\begin{aligned}\text{sign}(q_{pr}^{J_2}) &= \text{sign}(r_{pq}^{J_2}) \otimes \text{sign}(r_p^{K_3}) \otimes \text{sign}(q_p^{K_4}) \\ &\neq \text{sign}(r_{pq}^{J_1}) \otimes \text{sign}(r_p^{K_2}) \otimes \text{sign}(q_p^{K_1}) = \text{sign}(q_{pr}^{J_1}).\end{aligned}$$

This configuration is represented in the centre of Figure 5.

Finally, suppose $J_4 = J_1$ and $J_2 \neq J_3$. Then

$$\begin{aligned}\text{sign}(q_{pr}^{J_3}) = \text{sign}(q_p^{K_1}) &= \text{sign}(r_p^{K_2}) \otimes \text{sign}(r_{pq}^{J_1}) \otimes \text{sign}(q_{pr}^{J_1}) \\ &\neq \text{sign}(r_p^{K_3}) \otimes \text{sign}(r_{pq}^{J_2}) \otimes \text{sign}(q_{pr}^{J_1}) = \text{sign}(q_{pr}^{J_1}).\end{aligned}$$

This configuration is represented in the right of Figure 5.

We have so far assumed the intersection in $X \cap P_p$ is convex. However, under numerical error this may fail for particularly pathological examples. These cases may be recognized by the fact they indicate four intersections are to be calculated along the lines of either $P_p \cap P_q$ or $P_p \cap P_r$. These non-convex configurations are presented in Figure 6. Other convexity issues are resolved by the straight line principle, see Section 3.2.1.

The following subroutine will reduce these non-convex configurations to their convex counterparts. Any other non-convex configurations do not affect consistency.

- For those lines $P_p \cap P_q$ that have four intersections, take the smallest $r_{pq}^{J_1}$ and largest $r_{pq}^{J_2}$ and discard the remaining two.

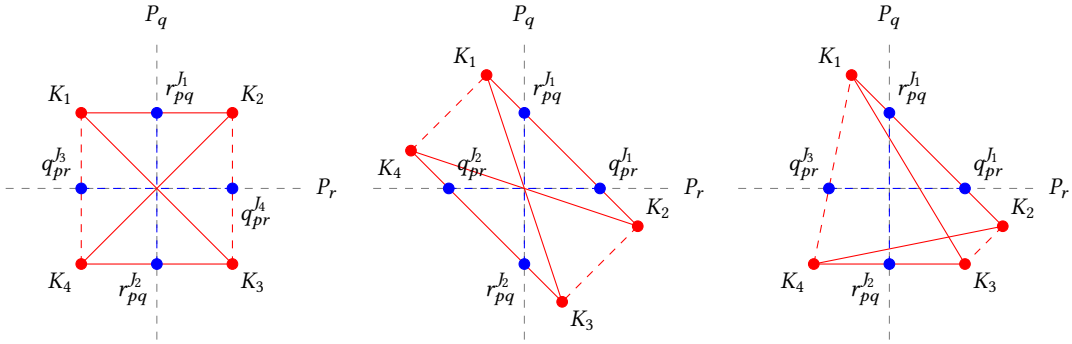


Fig. 6. Non-convex configurations of $X \cap P_p$. These can only occur with four intersections in $X \cap P_p$ and result in four intersections in either or both of $X \cap P_p \cap P_r$ and $X \cap P_p \cap P_q$.

- For any other line $P_p \cap P_r$ calculate additional intersections between this line and the remaining two edges of $X \cap P_p$ not considered, which must be done using equation (4). Make sure these intersections inherit their signs if applicable. Again take the smallest $q_{pr}^{J_3}$ and largest $q_{pr}^{J_4}$ and discard the rest.

The proof of consistency for these configurations is identical to the one presented above, reversing the roles of $K_3$ and $K_4$.

This proves the existence of the pair $(J_3, J_4)$ such that $\text{sign}(q_{pr}^{J_3}) \neq \text{sign}(q_{pr}^{J_4})$. For the existence of the pair $(J_5, J_6)$, reverse the roles of $p$ and $q$. This gives consistency of this step of the algorithm.

This means that if a vertex of $Y$, the intersection of $P_p$, $P_q$ and $P_r$, is surrounded by two intersections in $X \cap P_p \cap P_q$ then it is surrounded by four more in $X \cap P_p \cap P_r$ and $X \cap P_q \cap P_r$. The final step, determining if a vertex of $Y$ is inside $X$, is then consistent with all previous calculations. □

The following corollary shows this idea of consistency is equivalent to robustness, as long as each individual calculation, which includes coordinate transformations and line-line intersections, is robust. As some of these subroutines are modular in Algorithm 2, we will assume their robustness.

Corollary 2. *Suppose the calculated position of vertices after the coordinate transformation and intersections between edges and faces is accurate up to a distance of $\delta$. Then the volume of the intersection calculated by Algorithm 2 differs from the volume of the exact intersection by at most $\left\|A^{-1}\right\| 8\delta$, where the matrix $A$ defines the affine transformation of the algorithm.*

Proof. Let $Z$ be the exact intersection between the reference tetrahedron $Y$ and the arbitrary tetrahedron $X$, with corners $\mathbf{z}_i$. Since Algorithm 2 is consistent with respect to shape, the intersection calculated by the algorithm is a polyhedron, $Z + \Delta Z$, with corners represented by $\mathbf{z}_i + \Delta\mathbf{z}_i$.

The difference between these two polyhedra, $\Delta Z$, is the union of a finite number of smaller polyhedra. Specifically, $\Delta Z$ is the union of at most 8 polyhedra, as $Z$ has at most 8 sides. Each of these smaller polyhedra has at least one side of length $|\Delta\mathbf{z}_i| \leq \delta$ by assumption. All other side lengths are less than or equal to 1, as $Z + \Delta Z$ lies inside $Y$, so the total volume of $\Delta Z$ is less than or equal to $8\delta$.

The polyhedra $\Delta Z$ must be transformed back into the original coordinates using the affine transformation $A^{-1}(\mathbf{z}_i + \Delta\mathbf{z}_i) - \mathbf{b}$. This scales the volume of $\Delta Z$ by $\left\|A^{-1}\right\|$. □

### 4.1 Practical concerns of determining signs

There are four ways to determine the sign of a coordinate for a given intersection:

- the sign is inherited from the previous generation;
- the sign is linked either via Lemma 2 or Corollary 1 to the sign of another coordinate on the same edge or face of $X$;
- the sign is enforced via the straight line principle, or;
- the sign is found through direct calculation.

Inheritance can only occur if the edge or face of $X$ does not have the maximum number of intersections with the planes of $Y$. If an edge of $X$ intersects the plane $P_p$ but not the plane $P_q$ then $\text{sign}(q_p^{ij})$ is inherited from $q_i$ and $q_j$, which must have equal sign in the $q$–coordinate. If a face of $X$ intersects the line $P_p \cap P_q$ but not $P_p \cap P_r$ then $\text{sign}(r_{pq}^{ijk}) = \text{sign}(r_p^{ij}) = \text{sign}(r_p^{ik})$.

This means we only use Lemma 2 and Corollary 1 when the sign tests show different signs for multiple coordinates. That is, we use Lemma 2 only when $\text{sign}(p_i) \neq \text{sign}(p_j)$ and $\text{sign}(q_i) \neq \text{sign}(q_j)$. We use Corollary 1 only when $\text{sign}(q_p^{ij}) \neq \text{sign}(q_p^{ik})$, $\text{sign}(r_p^{ij}) \neq \text{sign}(r_p^{ik})$, $\text{sign}(p_q^{ik}) \neq \text{sign}(p_q^{jk})$ and $\text{sign}(r_q^{ik}) \neq \text{sign}(r_q^{jk})$. Since the algorithm has been shown to be consistent, it would also be that $\text{sign}(p_r^{ij}) \neq \text{sign}(p_r^{jk})$ and $\text{sign}(q_r^{ij}) \neq \text{sign}(q_r^{jk})$.

If we know a set of intersection coordinates can be connected then it remains to find one of their signs through direct calculation. This leaves direct calculation as a last resort, though necessary if we are to use Lemma 2 and Corollary 1. The straight line principle is only employed if an edge of $X$ intersects at least three planes $P_p$, and direct calculations have returned the middle of the three intersections.

**Algorithm 3** Subroutine for lines 12-17

```
1:  for all edges ij of X do                                   ▷ all faces ijk ...
2:      for all P_p such that ij is flagged, see line 9 of Algorithm 2, do   ▷ all P_p ∩ P_r ...
3:          for all remaining coordinates q do
4:              if ij has not been flagged for the plane P_q then   ▷ ...for the line P_p ∩ P_q
5:                  sign(q_p^ij) is inherited                  ▷ sign(q_pr^ijk) ...
6:              else if p_q^ij has been calculated then        ▷ r_pq^ijk ...
7:                  sign(q_p^ij) is found through Lemma 2      ▷ ...Corollary 1
8:              else
9:                  q_p^ij must be calculated, see equation (2)   ▷ ...Lemma 4
10:             end if
11:         end for
12:         if all coordinates q_p^ij are non-negative then
13:             Calculate any remaining coordinates, see equation (2)   ▷ ...Lemma 4
14:             Apply the reverse transformation, see equation (3)
15:         end if
16:         if this edge has more than two intersections then   ▷ Omitted
17:             if sign(q_p^ij) ≠ sign(q_i), while sign(r_p^ij) = sign(r_i) for all r ≠ q, p then
18:                 Set sign(r_q^ij) = sign(r_p^ij), see Section 3.2.1
19:             end if
20:         end if
21:     end for
22: end for
```

| Calculated | | | Reassessed | | |
|---|---|---|---|---|---|
| $x$ | $y$ | $z$ | $x$ | $y$ | $z$ |
| $> 0, < 1$ | $> 0, < 1$ | $> 0, < 1$ | No change | | |
| $< 0$ | $> 0, < 1$ | $> 0, < 1$ | 0 | 1/2 | 1/2 |
| $< 0$ | $> 1$ | $< 1$ | 0 | 1 | 0 |
| $< 0$ | $> 1$ | $> 1$ | 0 | 1/2 | 1/2 |

Table 4. Possible implementation of guardrails when the calculated position of the intersection between a face of $Y$ and an edge of $X$ disagrees with its determined signs.

We present a subroutine for lines 12-17 of Algorithm 2 as Algorithm 3. It identifies when one can use inheritance or Lemma 2, or when direct calculation is necessary. The equivalent subroutine for line 18 is nearly identical with minor alterations. These alterations are noted as comments in Algorithm 3. Note, however, that both inheritance and applicability of Corollary 1 need to be checked for both the lines $P_p \cap P_q$ and $P_r \cap P_q$.

*4.1.1 Guardrails.* Since direct calculation is used as a last resort to determine the signs of the intersections, it is possible that a particular intersection is found to lie within $Y$ based on its signs but when calculating its position it is placed outside $Y$. This can most easily occur when an edge of one tetrahedron lies in the plane of a face of the second.

This is a numerical error in the calculation of the intersection and can be guarded against in a number of ways. The most straightforward course of action is to implement guardrails. If an intersection is determined to lie within $Y$, then all of its coordinates must be less than or equal to 1 and greater than or equal to 0. If the calculation of one of its coordinates is greater than 1, then the guardrails should limit the value to 1. Likewise, if the calculation returns a value less than 0, then the guardrails should enforce a value of 0.

When implementing guardrails, it is important to remember that the coordinates are barycentric and must therefore sum to 1. Remember also that any intersection possesses coordinates equal to 0, as they lie on a particular face or edge of $Y$. For intersections on edges of $Y$, there are only two non-zero coordinates, and so if guardrails change one value to 1, then the other must be changed to 0, and vice versa.

For intersections on faces of $Y$, there is additional freedom as there are three non-zero coordinates. One can choose any point along the projection of the edge of $X$ onto the face of $Y$, though this will require additional calculations. A less sophisticated but faster implementation would be to choose an arbitrary point along an edge of $Y$. Table 4 gives one possible set of choices of such arbitrary points.

## 5 Examples

### 5.1 Combinatorial example

Much of the mathematics that ensures the robustness of the algorithm is combinatorial in nature. Let us consider a particular example and see how the combinatorics of the algorithm play out.

Suppose the vertices of $X$ have coordinates as dictated by Table 5. This can occur if $X$ resembles Figure 7, for example. We run through the sign tests performed for $P_x$:

$$\begin{aligned}
&\text{sign}(x_0) = \text{sign}(x_1) \implies \text{sign}(x_p^{01}) = +, && \text{sign}(x_0) \neq \text{sign}(x_3) \implies \exists q_x^{03},\\
&\text{sign}(x_0) = \text{sign}(x_2) \implies \text{sign}(x_p^{02}) = +, && \text{sign}(x_1) \neq \text{sign}(x_3) \implies \exists q_x^{13},\\
&\text{sign}(x_1) = \text{sign}(x_2) \implies \text{sign}(x_p^{12}) = +, && \text{sign}(x_2) \neq \text{sign}(x_3) \implies \exists q_x^{23}.
\end{aligned}$$

| | $w$ | $x$ | $y$ | $z$ |
|---|---|---|---|---|
| 0 | + | + | + | + |
| 1 | + | + | + | + |
| 2 | + | + | + | - |
| 3 | + | - | - | - |

Table 5. Coordinates of the combinatorial example. Since we are considering only the combinatorics the only important aspect of each coordinate is whether it is non-negative.

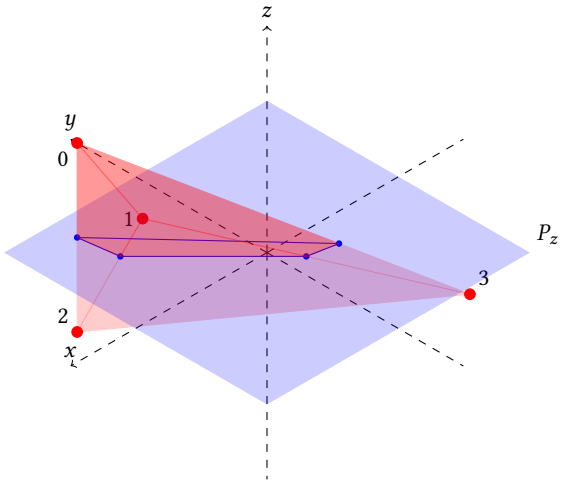


Fig. 7. Geometric example that could give rise to the combinatorial example found in Table 5. The plane $P_z$ and the intersection $X \cap P_z$ are shown to give perspective.

The same sign tests are also performed for $P_y$ and $P_z$. No sign tests are required for $P_w$ as all coordinates are positive in this direction, indicating no intersections with $P_w$. The sign tests indicate ten intersections, three in $P_x$, the same number in $P_y$ and four in $P_z$. These tests also show some of these intersections have signs inherited from the vertices. The flags from line 9 of Algorithm 2 can be summarized as

$$S = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}.$$

The rows correspond to the planes $P_w$, $P_x$, $P_y$ and $P_z$, respectively, while the columns correspond to the edges 01, 02, 03, 12, 13 and 23, respectively.

Table 6 begins with the information obtained from the sign tests in parentheses. Note there are no pairs for these coordinates to use Lemma 2. Now we must compute some of the remaining coordinates. We start with $y_x^J$. We may now

| | | $x$ | $y$ | $z$ |
|---|---|---|---|---|
| | 03 | | - | + |
| $P_x$ | 13 | | + | + |
| | 23 | | - | (-) |
| | 03 | [+] | | + |
| $P_y$ | 13 | [-] | | + |
| | 23 | [+] | | (-) |
| | 02 | (+) | (+) | |
| $P_z$ | 03 | [-] | [-] | |
| | 12 | (+) | (+) | |
| | 13 | [-] | [-] | |

Table 6. Intersections between edges of $X$ and planes $P_p$. Parentheses denote the sign has been inherited from Table 5, while square brackets indicate the sign has been made to satisfy Lemma 2 from a previous calculation.

use these along with Lemma 2 to find the signs of the $x$–coordinates in $P_y$:

$$\begin{aligned}
\text{sign}(x_y^{03}) &= \text{sign}(x_0) \otimes \text{sign}(y_3) \otimes \text{sign}(y_x^{03}) = (+)(-)(-) = +, \\
\text{sign}(x_y^{13}) &= \text{sign}(x_1) \otimes \text{sign}(y_3) \otimes \text{sign}(y_x^{13}) = (+)(-)(+) = -, \\
\text{sign}(x_y^{23}) &= \text{sign}(x_2) \otimes \text{sign}(y_3) \otimes \text{sign}(y_x^{23}) = (+)(-)(-) = +.
\end{aligned}$$

These signs are added to Table 6 and indicated with square brackets. We then compute $z_x^J$ and $z_y^J$ and again use Lemma 2 to complete the table. See Figure 8 for a visual representation of these intersections.

Four of the intersections have only non-negative coordinates, meaning they lie within $Y$ and form a corner of $X \cap Y$. This identifies eight coordinates that need to be calculated, three of which have already been found. That leaves 12 unnecessary coordinates, of which we calculated four, ultimately avoiding the calculation of eight coordinates.

A small aside: The intersection between the edge 03 and $P_x$ is the middle of three intersections along this edge. This means the sign of $z_y^{03}$ should have been obtained through the straight line principle, see Section 3.2.1. This would not have changed the number of coordinates we needed to calculate, as $z_y^{03}$ is positive.

We then perform the sign tests for $P_x \cap P_y$:

$$\begin{aligned}
\text{sign}(y_x^{03}) \neq \text{sign}(y_x^{13}) &\implies \exists z_{xy}^{013}, \\
\text{sign}(y_x^{03}) = \text{sign}(y_x^{23}) &\implies \text{sign}(y_{xz}^{023}) = -, \\
\text{sign}(y_x^{13}) \neq \text{sign}(y_x^{23}) &\implies \exists z_{xy}^{123}.
\end{aligned}$$

We repeat these tests five more times, once for each combination of $P_p \cap P_q$ and $P_q \cap P_p$. We find two intersections for each line, three of which have inherited their signs. The information is collected in Table 7.

Now we need only calculate one of the remaining coordinates, for example $z_{xy}^{123}$. Using Lemma 5 we can find the signs of the other coordinates:

$$\begin{aligned}
\text{sign}(y_{xz}^{123}) &= \text{sign}(z_x^{13}) \otimes \text{sign}(y_x^{23}) \otimes \text{sign}(z_{xy}^{123}) = (+)(-)(-) = +, \\
\text{sign}(x_{yz}^{123}) &= \text{sign}(z_y^{13}) \otimes \text{sign}(x_y^{23}) \otimes \text{sign}(z_{xy}^{123}) = (+)(+)(-) = -.
\end{aligned}$$

The two coordinates along each line have different signs, meaning the vertex of $Y$ at their intersection lies inside $X$. This identifies the last four corners of the intersection $X \cap Y$: the intersection 013 in $P_x \cap P_y$, 123 in $P_x \cap P_z$, 023 in

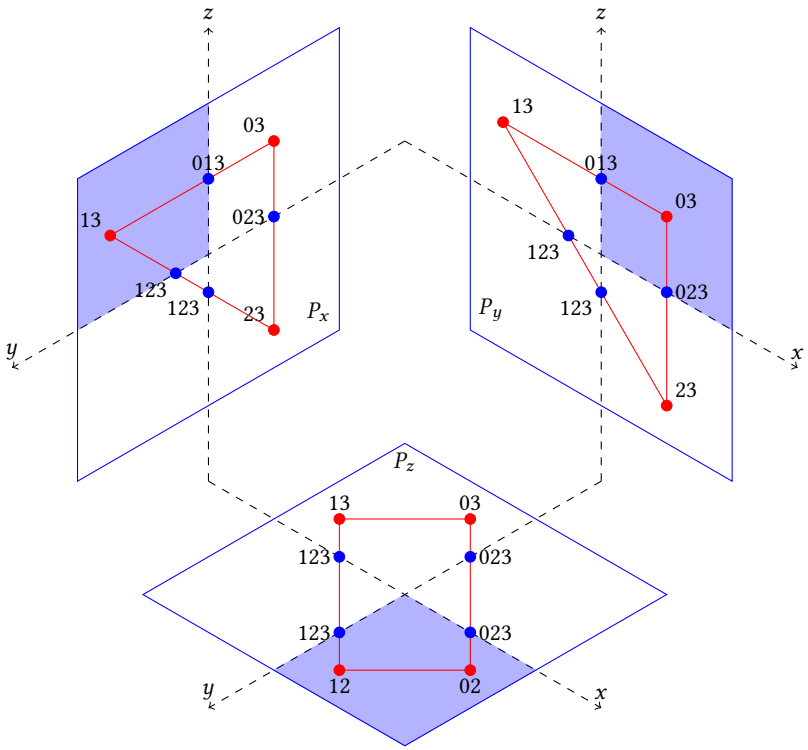


Fig. 8. The planes $P_x$, $P_y$ and $P_z$ and their intersections. Those parts of the planes that are faces of $Y$ are shaded in blue.

| | | $x$ | $y$ | $z$ |
|---|---|---|---|---|
| $P_x \cap P_y$ | 013 | | | (+) |
| | 123 | | | - |
| $P_x \cap P_z$ | 023 | | (-) | |
| | 123 | | [+] | |
| $P_y \cap P_z$ | 023 | (+) | | |
| | 123 | [-] | | |

Table 7. Intersections between faces of $X$ and lines $P_p \cap P_q$. Parentheses denote the sign has been inherited from Table 6, while square brackets indicate the sign has been made to satisfy Corollary 1 from a previous calculation.

$P_y \cap P_z$ and the vertex of $Y$. The final intersection is presented in Figure 9, minus the two vertices of $X$ found to lie within $Y$, $\mathbf{x}_0$ and $\mathbf{x}_1$.

### 5.2 Numerical example

Let us now consider a numerical example to compare the accuracy and efficiency of Algorithm 2 against its predecessor PANG and the Sutherland-Hodgman algorithm [19]. We take two tetrahedra $U$ and $V$ whose coordinates depend on an angle $\alpha$ which dictates the aspect ratio of $V$. The coordinates of the tetrahedra are presented in Table 8. The tetrahedron

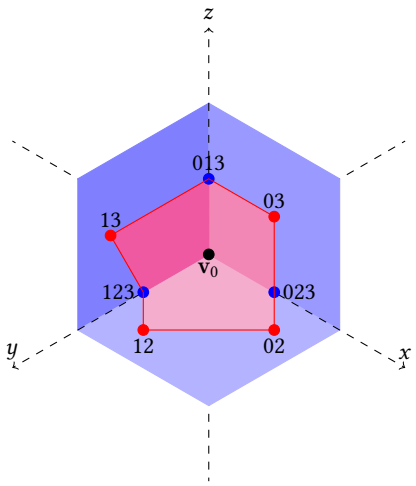


Fig. 9. The final intersection of $X \cap Y$ on the boundary of $Y$. To complete the intersection take the two vertices $\mathbf{x}_0$ and $\mathbf{x}_1$ that are found to lie within $Y$.

| | $V$ | | | | $2U$ | | | |
|---|---|---|---|---|---|---|---|---|
| $x$ | 0 | 0 | 0 | $\cos(\alpha)$ | $\cos(\alpha/2)$ | $\cos(\alpha/2+2\pi/3)$ | $\cos(\alpha/2+4\pi/3)$ | 0 |
| $y$ | 0 | 0 | 1 | $\sin(\alpha)$ | $\sin(\alpha/2)$ | $\sin(\alpha/2+2\pi/3)$ | $\sin(\alpha/2+4\pi/3)$ | 0 |
| $z$ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |

Table 8. Coordinates of the clipping and subject tetrahedra. The coordinates of $U$ must be halved.

$V$ widens with angle $\alpha$, while $U$ rotates such that the intersection is always symmetric along the plane down the centre of $V$.

The tetrahedra are plotted in Figure 10 for $\alpha = \pi/3$ and $3\pi/4$. The intersection is shaped like a pyramid, with a quadrilateral base and an apex where a vertex of $U$ falls along a line of $V$. However, with large enough $\alpha$ part of this pyramid is clipped, taking a triangular tip off. There are then two regimes of behaviour, one for small $\alpha$ and one for large $\alpha$. The volume of the intersection can be computed as

$$\frac{1}{12}\frac{\sin(\pi/6)}{\sin(\pi/2-\pi/6-\alpha/2)} - \begin{cases} 0 & 1/2 < \cos(\alpha/2), \\ \frac{\cos(\alpha/2)-1/2}{3(\cos(\alpha/2)-1)}\left(\frac{1}{2}-\cos(\alpha/2)\right)^2\tan(\pi/6) & 1/2 \geq \cos(\alpha/2). \end{cases}$$

The regimes are then split by $\alpha = 2\cos^{-1}(1/2)$.

The left of Figure 11 shows the relative error of the computed intersection as a function of the angle $\alpha$. For the vast majority of values of $\alpha$ all algorithms tested with this example perform to machine precision. However, certain values of $\alpha$ in the second regime produce large errors when using PANG. Algorithm 2 and Sutherland-Hodgman have essentially identical relative error, up to minor fluctuations in machine precision.

Algorithm 2 is shown to take the least computation time, see right of Figure 11, especially in the second regime where its combinatorial tests prevent unnecessary calculations. The computation time remains at the same order of magnitude for all algorithms. Most notably, Sutherland-Hodgman has nearly identical computation time to Algorithm 2, except in the second regime where it must perform additional calculations.

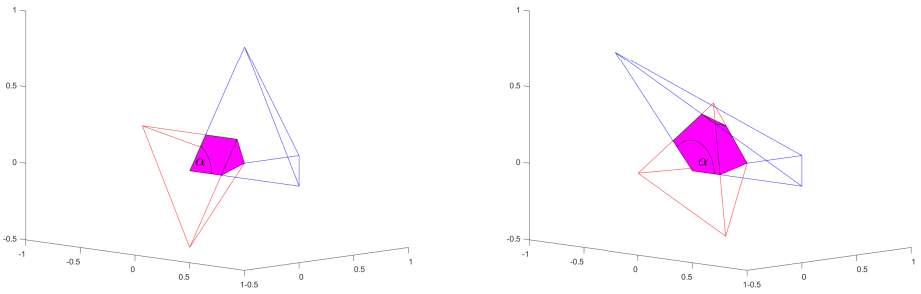

Fig. 10. Intersecting tetrahedra from Table 8 for $\alpha = \pi/3$ (left) and $3\pi/4$ (right).

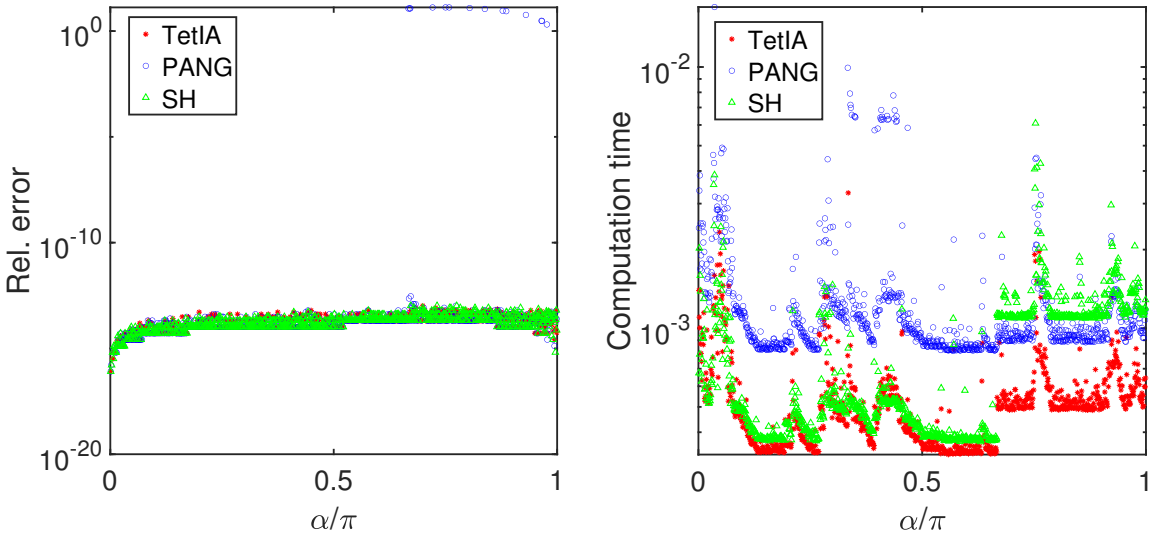


Fig. 11. Relative error (left) and computation time (right) for running the example from Table 8. Algorithm 2 is labelled as TetIA, for tetrahedral intersection algorithm.

Algorithm 2 shows no loss of accuracy or efficiency compared to Sutherland-Hodgman, and provides significant improvements in accuracy over PANG. This suggests Algorithm 2 is the strictly better algorithm, as it has proven robustness. Note that Sutherland-Hodgman is almost certainly robust itself, given it is formed from a subset of the calculations of Algorithm 2, but no proof is currently published.

## 6 Conclusions

Algorithm 2 is proven to be robust to errors that affect the shape of the intersection between tetrahedra. Since the algorithm is consistent, a significant intersection cannot vanish under small numerical error, such as those caused by floating-point arithmetic. The algorithm can be complicated to implement, as discussed in Section 4.1, but it successfully avoids errors that can seriously affect these alternatives, see Section 5.2, without loss of computation time, again see Section 5.2.

## 7 Acknowledgements

The authors would like to thank Prof. Jerónimo Rodriguez Garcia and his student Miguel Picos Maiztegui for field testing and debugging the algorithm.

This work is funded by the Swiss National Science Foundation, the Centre de Recherches Mathématiques, and Université Laval.

## References

[1] J. Albella, H. Ben Dhia, S. Imperiale, and J. Rodríguez, *Mathematical and numerical study of transient wave scattering by obstacles with a new class of Arlequin coupling*, SIAM Journal on Numerical Analysis, 57 (2019), pp. 2436–2468.

[2] P. Ciarlet, E. Jamelot, and F. D. Kpadonou, *Domain decomposition methods for the diffusion equation with low-regularity solution*, Computers & Mathematics with Applications, 74 (2017), pp. 2369–2384.

[3] S. Coniglio, C. Gogu, and J. Morlier, *Weighted average continuity approach and moment correction: New strategies for non-consistent mesh projection in structural mechanics*, Archives of Computational Methods in Engineering, 26 (2019), pp. 1415–1443.

[4] M. Cyrus and J. Beck, *Generalized two-and three-dimensional clipping*, Computers & Graphics, 3 (1978), pp. 23–28.

[5] O. Devillers, S. Lazard, and W. J. Lenhart, *Rounding meshes in 3d*, Discrete & Computational Geometry, 64 (2020), pp. 37–62.

[6] H. B. Dhia and G. Rateau, *The arlequin method as a flexible engineering design tool*, International journal for numerical methods in engineering, 62 (2005), pp. 1442–1462.

[7] Y. Efendiev and T. Y. Hou, *Multiscale finite element methods: theory and applications*, vol. 4, Springer Science & Business Media, 2009.

[8] P. E. Farrell, *Galerkin projection of discrete fields via supermesh construction*, PhD thesis, Imperial College London, 2010.

[9] M. J. Gander and C. Japhet, *An algorithm for non-matching grid projections with linear complexity*, in Domain decomposition methods in science and engineering XVIII, Springer, Berlin, Heidelberg, 2009, pp. 185–192.

[10] ———, *Algorithm 932: PANG: software for nonmatching grid projections in 2D and 3D with linear complexity*, ACM Transactions on Mathematical Software, 40 (2013), p. 6.

[11] S. Guinard, R. Bouclier, M. Toniolli, and J.-C. Passieux, *Multiscale analysis of complex aeronautical structures using robust non-intrusive coupling*, Advanced Modeling and Simulation in Engineering Sciences, 5 (2018), p. 1.

[12] D. Halperin and E. Packer, *Iterated snap rounding*, Computational Geometry, 23 (2002), pp. 209–225.

[13] I. Kolingerová, *3d-line clipping algorithms—a comparative study*, The Visual Computer, 11 (1994), pp. 96–104.

[14] Y.-D. Liang and B. A. Barsky, *A new concept and method for line clipping*, ACM Transactions on Graphics, 3 (1984), pp. 1–22.

[15] C. McCoid and M. J. Gander, *A provably robust algorithm for triangle-triangle intersections in floating-point arithmetic*, ACM Transactions on Mathematical Software, 48 (2022), pp. 1–30.

[16] F. Moro and M. Guarnieri, *Efficient 3-D domain decomposition with dual basis functions*, IEEE Transactions on Magnetics, 51 (2015), pp. 1–4.

[17] M. A. Puso, *A 3D mortar method for solid mechanics*, International Journal for Numerical Methods in Engineering, 59 (2004), pp. 315–336.

[18] V. Skala, *A brief survey of clipping and intersection algorithms with a list of references (including triangle-triangle intersections)*, Informatica, 34 (2023), pp. 169–198.

[19] I. E. Sutherland and G. W. Hodgman, *Reentrant polygon clipping*, Communications of the ACM, 17 (1974), pp. 32–42.

[20] H. Talebi, M. Silani, S. P. Bordas, P. Kerfriden, and T. Rabczuk, *A computational library for multiscale modeling of material failure*, Computational Mechanics, 53 (2014), pp. 1047–1071.

[21] B. I. Wohlmuth and R. H. Krause, *Monotone multigrid methods on nonmatching grids for nonlinear multibody contact problems*, SIAM Journal on Scientific Computing, 25 (2003), pp. 324–347.