Semi-analytical smoothed-particle hydrodynamics correction factors for polynomial kernels and piecewise-planar boundaries

Wawrzyniec Kostorz | Anton Esmail-Yakas

Department of Earth Science and Engineering, Imperial College London, London, UK

Correspondence
Wawrzyniec Kostorz, Department of Earth Science and Engineering, Imperial College London, Prince Consort Road, London SW7 2BP, UK.
Email: w.kostorz16@imperial.ac.uk

Abstract
This work presents a method of exactly and efficiently evaluating integrals of compactly supported (and otherwise) polynomial radial kernels near piecewise-planar boundaries. The technique is motivated by and has an immediate application in the smoothed-particle hydrodynamics method employing the semi-analytical boundary formulation. The current implementation is exact and fully general, avoiding any need for symbolic or numerical integration which previous implementations rely on. A detailed derivation and three compact implementations are provided and the latter can be easily modified to fit new and existing simulation codes. Applications and good performance are demonstrated on a number of test problems, where the semi-analytical boundary method can simulate complex boundaries using roughly half the boundary particles required in a ghost particle boundary method.

KEYWORDS
fluid–structure interaction, integral equations, particle methods

1 INTRODUCTION

The problem of accurately modeling interactions of fluid with solid walls has been one of the major challenges in the field of smoothed-particle hydrodynamics (SPH). Up to date, one of the more accurate, robust, and general ways of addressing this issue has been the so-called semi-analytical (SA) boundary formulation. This was first introduced by Kulasegaram, Bonet, and Feldman, and further developed into a more complete formulation by Ferrand et al. The SA approach relies on including boundary integral terms in the governing equations which are usually ignored for simplicity in SPH implementations. These boundary integral terms rely on integrating the peak-shaped radial particle density function over the intersection of its spherical support and the fluid domain. Away from domain boundaries the intersection is equal to the full particle support and the integration is trivial, but otherwise the support is truncated (Figure 1) and the integration becomes much more complex.

The primary interest is in evaluating the integral of the radial density function over spherical supports truncated by piecewise-linear boundaries in two dimensions and piecewise-planar polygonal boundaries in three dimensions. The latter are by far the most common method of representing realistic geometries in numerical simulations, with the unstructured triangulated mesh being the typical example. However, the general problem is highly nontrivial, especially in three
dimensions. Numerical methods capable of performing the brute force calculation are available, but their cost is generally way too high to be feasible. In recent years multiple authors addressed the issue of performing the underlying calculation in a practically viable fashion.

The two-dimensional case was handled analytically by Ferrand et al.\(^7\) for a single polynomial density function, the \(C_2\) quintic Wendland kernel.\(^8\) The three-dimensional case, for a special case of the density function, was first addressed by Mayrhofer et al.,\(^9\) later by Violeau et al.,\(^10\) and recently by Winchenbach et al.\(^11\) These solutions are not fully general, do not handle nonconvex geometries correctly and rely on involved expressions computed via mathematics software. Additionally, Reference 9 employs a relatively complex and cumbersome geometric decomposition process, Reference 10 is incomplete, and Reference 11 is formulated for (and therefore accurate near) just infinite planar boundaries. An alternative approach was proposed by Chiron et al.,\(^2\) where the problem is reduced to a two-dimensional integral performed numerically on an unstructured planar mesh. This method correctly accounts for star-convexity but is again formulated for a single kernel (Wendland \(C_2\)).\(^8\) uses a nonstandard low-order formulation, purposefully avoids handling planar regions bounded by curved edges (by partially integrating over trivial, noncontributing areas) and requires a meshing algorithm and a domain trimming process both of which are highly nontrivial. Our recent work\(^12\) introduced a general, simple and compact method for arbitrary polynomial kernels where the original problem is reduced to a series of one-dimensional integrals over straight edges approximated numerically and the rest of the solution is fully analytic. The method is robust but still requires numerically integrating a well-behaved (but fairly cumbersome) function over a finite interval and a single user-specified parameter controlling its accuracy.

This work is a natural continuation and extension of Reference 12, with the part of the solution previously approximated numerically now evaluated analytically. In order to achieve this, the antiderivatives of interest are not formed explicitly but instead their values at a point are generated via inexpensive recurrence relations. Avoiding the numerical integration means the method no longer requires user-defined parameters (see \(\delta\) in Reference 12 which controls the number of numerical integration nodes) to control the accuracy and computational cost. Instead, the method is fully exact, with computational cost controlled only by the number of boundaries, rather than the desired accuracy. No numerical quadrature need be imported or implemented, and thus as a whole, the implementation is much more user friendly.

The previously given algorithm\(^12\) is appropriately updated and improved, in particular by minimizing the number of vector–vector operations (e.g., scalar products) and providing appropriate handling of particles along walls (but away from...
element edges, where the underlying problem becomes ill posed). We therefore provide the first fully general, computationally efficient and exact solution of the problem for arbitrary piecewise-planar boundaries and polynomial kernels. MATLAB, C++, and Python implementations of the three-dimensional version of the algorithm are provided and can easily be adapted to fit existing or new SPH codes. An additional application of the methodology in conjunction with the radial basis function interpolation method is also discussed.

It should be noted that this article does not address the formulation of the SPH governing equations under the SA boundary method, but rather focuses on the efficient and exact calculation of the required terms \( \gamma \) and \( \nabla \gamma \). For a discussion of the full implementation of the SA method including how \( \gamma \) and \( \nabla \gamma \) are included in the governing equations, the reader is directed to Reference 7, which this article used as a basis for its own SPH implementation.

The rest of this article is organized as follows. The theoretical background is given in Section 2. The three-dimensional problem statement and its reduction to one-dimensional integration as in Reference 12 are briefly summarized in Section 2.1. Two special cases of line contours are motivated and discussed in Section 2.2 where the corresponding analytic solution for circular arcs is given as well. The essential part of this article, closed-form integration over straight lines previously performed numerically, is addressed in Section 2.3. A brief discussion on the closed-form two-dimensional solution follows in Section 2.4. The general two- and three-dimensional algorithms for computing the normalization factors and their gradients in SPH simulations are given in Section 3. Simplifications and an application of the methodology in conjunction with the radial basis function interpolation method are discussed in Section 4. Representative examples and conclusions follow in Sections 5 and 6 respectively.

2 | METHOD

The three-dimensional problem and its reduction to one-dimensional integration for planar boundary elements\(^{12}\) are summarized first. Two special cases are then motivated and introduced and closed-form expressions for both are given. In particular, integration along straight edges previously handled numerically is tackled analytically in Section 2.3. A short discussion on the two-dimensional version of the problem follows.

2.1 | Reduction to 1D

In this section we summarize the problem and the derivation given in Reference 12. Let \( \Omega \) be a compact subspace of \( \mathbb{R}^3 \) with boundary \( \partial \Omega \). The aim is to evaluate

\[
\gamma = \int_\Omega W \, dV, \quad \nabla \gamma = -\int_{\partial \Omega} W n \, dS, \quad (1)
\]

where \( n = n(x) \) is the unit outward normal to \( \partial \Omega \) and \( W = W(||x - x_a||) \) is a smooth scalar function dependent only on the distance between the argument \( x \) and some fixed point \( x_a \in \mathbb{R}^3 \). The gradient \( \nabla \gamma \) in the above is taken with respect to \( x_a \), since \( \gamma \) is implicitly a function of the latter; however, in further calculation we treat \( x_a \) as constant and the definition (1) of \( \nabla \gamma \) is used directly. We write

\[
r = ||x - x_a||, \quad (2)
\]

so that \( W = W(r) \) is a so-called radial function (spherically symmetric around \( x_a \)). By applying Gauss’ theorem and under mild conditions on \( W \) it can be shown\(^{12}\) that

\[
\gamma = \int_{\partial \Omega} G_w^{(3)}(r) \, (x - x_a)^T n \, dS, \quad G_w^{(3)}(r) = \int_0^1 s^2 \, W(rs) \, ds, \quad (3)
\]
so that both $\gamma$ and $\nabla \gamma$ are now expressed as surface integrals.

Next, the case of piecewise-planar $\partial \Omega$ is investigated. Let $\partial \Omega_j$ denote a planar portion of $\partial \Omega$ where

$$\partial \Omega = \bigcup_j \partial \Omega_j, \quad (4)$$

and $\partial \Omega_j$ are disjoint (they can only share edges). Clearly,

$$\gamma = \sum_j \gamma_j, \quad (5)$$

$$\nabla \gamma = \sum_j \nabla \gamma_j, \quad (5)$$

where $j$ runs over all boundary elements $\partial \Omega_j$ and $\gamma_j/\nabla \gamma_j$ are their respective contributions (surface integrals performed over $\partial \Omega_j$); it therefore suffices to consider a single planar portion of the boundary (a single $\gamma_j/\nabla \gamma_j$ pair).

The plane of element $\partial \Omega_j$ is written $P$ and $x_P$ denotes the projection of $x_a$ onto $P$. For any $x \in P$ we have

$$(x - x_a)^T n = (x_P - x_a)^T n = \lambda = \text{const.} \quad (6)$$

since $n$ is constant along a planar boundary. Here $\lambda$ is the signed distance between $x_a$ and its projection onto $P$. It also follows that for all $x \in P$

$$r^2 = \|x - x_a\|^2 = \lambda^2 + \|x - x_P\|^2 \quad (7)$$

and by defining $\rho = \|x - x_P\|$ the contributions $\gamma_j/\nabla \gamma_j$ as in (1), (3) can be written as

$$\gamma_j = \lambda \int_{\partial \Omega_j} G_W^{(3)} \left( \sqrt{\lambda^2 + \rho^2} \right) dS,$$

$$\nabla \gamma_j = -n \int_{\partial \Omega_j} W \left( \sqrt{\lambda^2 + \rho^2} \right) dS. \quad (8)$$

By then applying Stokes’ theorem the above can be integrated up once more, resulting in

$$\gamma_j = \lambda \int_{C_j} H_W^{(3)}(\rho; \lambda) \ n^T ((x - x_P) \times \hat{x}) \ dt,$$

$$\nabla \gamma_j = -n \int_{C_j} H_W^{(2)}(\rho; \lambda) \ n^T ((x - x_P) \times \hat{x}) \ dt, \quad (9)$$

where $C_j$ is the contour of $\partial \Omega_j$ oriented anticlockwise w.r.t. the (constant) element outward normal $n$, $x = x(t)$ is an arbitrary parameterization of $C_j$ and

$$H_W^{(3)}(\rho; \lambda) = \int_0^1 s G_W^{(3)} \left( \sqrt{\lambda^2 + \rho^2 s^2} \right) ds,$$

$$H_W^{(2)}(\rho; \lambda) = \int_0^1 s W \left( \sqrt{\lambda^2 + \rho^2 s^2} \right) ds. \quad (10)$$

Finally, it is observed that for any polynomial $W$, written symbolically as

$$W(r) = \sum a_n r^n \quad (11)$$

the functions $H_W^{(2,3)}$ can be given in closed form, namely,
\[
\sum a_n r^n = W(r) \rightarrow H^{(3)}_W(\rho; \lambda) = \begin{cases} 
\frac{\sum_{n=0}^{\infty} a_n}{(n+2)(n+3)} \lambda^{n(1+\kappa)} \zeta^{1+1}, & \kappa = \frac{\rho^2}{\lambda^2}, \quad \rho, \lambda \neq 0 \\
\frac{\sum_{n=0}^{\infty} a_n}{(n+2)(n+3)} \rho^n, & \lambda = 0 \\
\frac{\sum_{n=0}^{\infty} a_n}{2(n+3)} \lambda^n, & \rho = 0
\end{cases}
\]

with the expression for \(H^{(2)}_W\) being the same without the term \((n + 3)\) in each denominator in (12). Only polynomial \(W\) will be tackled in the rest of this work; for a generic \(W\) we recommend using a polynomial approximation so that the methodology presented here can be applied.

### 2.2 Special cases

Two types of contours \(C_j\) are of particular interest in this work. The first one is a straight line which enables integrating over domains bounded by polygonal (e.g., triangular) meshes, the latter being the typical way of representing geometries in numerical models. Second, we are also interested in (a part of) \(C_j\) being an arc of a circle in the plane \(P\) of the element \(\partial\Omega_j\) centered at \(x_p\). This shape occurs naturally in the SPH method,\(^\text{12}\) the latter being the main motivation and application for this work. For any such circle \(\rho = \text{const.}\) and since \(\lambda\) is also constant in the plane \(P\), along such \(C_j\) both \(H^{(2,3)}_W\) are constant. It has already been pointed out\(^\text{12}\) that if \(C_j\) is an arc with radius \(\rho_{\perp}\) spanning the angle \(\psi\) in the plane \(P\) around \(x_p\) then for \(k = 2, 3\)

\[
\int_{C_j} H^{(k)}_W(\rho; \lambda) \mathbf{n}^T((\mathbf{x} - \mathbf{x}_p) \times \mathbf{n}) \, dt = \psi \rho_{\perp}^2 H^{(k)}_W(\rho_{\perp}; \lambda)
\]

with the sign of \(\psi\) indicating the direction of integration around the normal \(\mathbf{n}\). Contour contributions of this type are therefore fully analytic for polynomial \(W\).

Previously,\(^\text{12}\) the integral (9) along straight edges was suggested to be performed numerically. The reason is a nontrivial dependence of \(H^{(2,3)}_W\) on \(\rho\) and \(\lambda\), which together with \(\rho\) itself being a nonlinear function of the natural parameterization makes it difficult to generate closed-form results. In the next section we address this problem by providing a procedure for generating closed-form antiderivatives of \(H^{(2,3)}_W\) and a method for evaluating them without constructing the analytic expressions explicitly.

### 2.3 Closed-form integration along straight edges

Consider a straight line segment \(C_j^{(0)}\) of \(C_j\) in the plane \(P\) spanned by \(z_{1,2}\) in this order, so that the unit direction along the segment is

\[
\hat{\mathbf{e}} = \frac{\mathbf{z}_2 - \mathbf{z}_1}{||\mathbf{z}_2 - \mathbf{z}_1||}
\]

The projection \(\mathbf{x}_i\) of \(\mathbf{x}_p\) onto the line \(l\) passing through \(z_{1,2}\) is then given by

\[
\mathbf{x}_i = \mathbf{z}_1 + (\hat{\mathbf{e}}^T(\mathbf{x}_p - \mathbf{z}_1)) \hat{\mathbf{e}}
\]

and in the following we employ the arc length parameterization of \(C_j^{(0)}\)

\[
\mathbf{x}(t) = \mathbf{x}_i + t \hat{\mathbf{e}}, \quad t \in [t_1, t_2].
\]

where

\[
t_i = \hat{\mathbf{e}}^T(\mathbf{z}_i - \mathbf{x}_i) = \hat{\mathbf{e}}^T(\mathbf{z}_i - \mathbf{x}_0).
\]
Only the computation of $\gamma_j$ is addressed explicitly in algebraic detail; the derivation for $\nabla \gamma_j$ is fully analogous and the corresponding results are stated when appropriate. With the above parameterization the contribution to $\gamma_j$ (9) along $C_i^{(i)}$, written $\gamma_j^{(i)}$, becomes

$$\gamma_j^{(i)} = \mu \lambda \int_{t_1}^{t_2} H_w^{(3)}(\rho(t); \lambda) \, dt$$

with

$$\mu = n^T((x_i - x_P) \times \hat{e}) = n^T((z_1 - x_a) \times \hat{e})$$

being the signed distance between $x_P$ and $x_i$ (Figure 2) and trivially from Pythagoras’ theorem

$$\rho(t)^2 = \mu^2 + t^2.$$  

(20)

We point out that both (17) and (19) can be evaluated without constructing $x_P$ and $x_i$ explicitly, but with employing just the center of the radial function and the end points of the edge; such treatment makes implementations more efficient.

The case $\rho, \lambda \neq 0$ in (12) is tackled first. By direct substitution to (18) and with the change of variables $t = |\lambda|s$ it follows that

$$\gamma_j^{(i)} = \mu \lambda \sum \frac{a_n}{(n + 2)(n + 3)} |\lambda|^{n+1} \int_{|\lambda|/|\lambda|}^{t_2/|\lambda|} \frac{1 + \frac{\mu^2}{s^2} + s^2}{\frac{\mu^2}{s^2} + s^2} ds.$$  

(21)

Next, for $\zeta > 0$ we define

$$I_\zeta(x; \phi^2) = \int x \frac{(1 + \phi^2 + s^2)^\zeta - 1}{\phi^2 + s^2} ds.$$  

(22)

(i.e., an indefinite integral/antiderivative w.r.t. $x$) so that with the above notation (and analogously for $\nabla \gamma_j^{(i)}$)

![Figure 2](image-url)
\[
\begin{align*}
\gamma_j^{(i)} &= \mu \lambda \sum \frac{a_n}{(n+2)(n+3)} |\lambda|^{n+1} \left[ I_{\gamma j+1}^{(i)} \left( \frac{t_2}{|\lambda|}; \frac{\mu^2}{|\lambda|^2} \right) - I_{\gamma j+1}^{(i)} \left( \frac{t_1}{|\lambda|}; \frac{\mu^2}{|\lambda|^2} \right) \right], \\
\nabla \gamma_j^{(i)} &= -n \mu \sum \frac{a_n}{n+2} |\lambda|^{n+1} \left[ I_{\gamma j+1}^{(i)} \left( \frac{t_2}{|\lambda|}; \frac{\mu^2}{|\lambda|^2} \right) - I_{\gamma j+1}^{(i)} \left( \frac{t_1}{|\lambda|}; \frac{\mu^2}{|\lambda|^2} \right) \right].
\end{align*}
\]

The problem is therefore solved if \( I_\zeta \) can be evaluated for half-integer \( \zeta \geq 1 \). Note that in (22) \( I_\zeta \) is defined up to the constant of integration which can be arbitrary; for convenience, we will later set it to 0 or to a value that makes the resulting expressions simpler. We now define

\[
J_\zeta(x; \phi^2) := I_{\zeta+1}(x; \phi^2) - I_{\zeta}(x; \phi^2) = \int_x^\infty (1 + \phi^2 + s^2)^{\zeta} \, ds
\]

and via integration by parts

\[
J_\zeta(x; \phi^2) = x(1 + \phi^2 + x^2)^{\zeta} - 2\zeta \int_x^\infty s^2(1 + \phi^2 + s^2)^{\zeta-1} \, ds
\]

rewritten as

\[
J_\zeta(x; \phi^2) = x(1 + \phi^2 + x^2)^{\zeta} - 2\zeta \left( J_\zeta(x; \phi^2) - (1 + \phi^2)J_{\zeta-1}(x; \phi^2) \right).
\]

Putting together index-shifted (24) and rearranged (26) yields the following coupled recurrence relation

\[
\begin{cases}
I_\zeta(x; \phi^2) = I_{\zeta-1}(x; \phi^2) + J_{\zeta-1}(x; \phi^2) \\
J_\zeta(x; \phi^2) = \frac{1}{1+2\zeta} \left[ x(1 + \phi^2 + x^2)^{\zeta} + 2\zeta(1 + \phi^2)J_{\zeta-1}(x; \phi^2) \right].
\end{cases}
\]

The \( I_\zeta \) of interest (half-integer \( \zeta \geq 1 \)) can then be constructed sequentially from

\[
\begin{cases}
I_1(x; \phi^2) = x \\
J_1(x; \phi^2) = \frac{1}{2}x^3 + (1 + \phi^2)x
\end{cases}
\]

and

\[
\begin{cases}
I_{0.5}(x; \phi^2) = \log \left( x + \sqrt{1 + \phi^2 + x^2} \right) - \frac{1}{|\phi|} \tan^{-1} \left( \frac{|\phi|}{1 + \phi^2 + \sqrt{1 + \phi^2 + x^2}} \right) \\
J_{0.5}(x; \phi^2) = \frac{1}{2} \left( x\sqrt{1 + \phi^2 + x^2} + (1 + \phi^2) \log \left( x + \sqrt{1 + \phi^2 + x^2} \right) \right)
\end{cases}
\]

where in the limit \( \phi \to 0 \)

\[
I_{0.5}(x; 0) = \log \left( x + \sqrt{1 + x^2} \right) - \frac{x}{1 + \sqrt{1 + x^2}},
\]

which follows easily from \( \tan^{-1}(x) \sim x \) for \( |x| \ll 1 \).

It is important to note that while closed-form expressions for \( I_\zeta \) of interest can now be constructed explicitly, they will generally be cumbersome and involved. A different, in particular a much more efficient approach is highly recommended in implementations. First, (27) is applied “pointwise” in the following way. For a general polynomial \( W \) (11) and given \( x, \phi^2 \) (e.g., as in (23)), (28) are evaluated if \( W \) has any even degree terms and (29) are evaluated if \( W \) has any odd degree terms. Any other required \( I_\zeta \) (with the maximum \( \zeta \) dictated by the degree of the polynomial \( W \)) are evaluated by repeatedly applying (27) to those initial values while treating \( x \) and \( \phi \) as constant. In this way the series of values is generated in a sequence of simple and cheap steps only requiring a small number of multiplications. Conversely, explicitly constructing \( I_\zeta \) starting from (29), say, would make each subsequent function increasingly involved and result in a much more computationally expensive evaluation (cf. References 9, 10).
Second, note that iterating the first equation in (27) for positive integer $k$ yields

\[ I_k = I_1 + (J_1 + \ldots + J_{k-1}), \]
\[ I_{k+0.5} = I_{0.5} + (J_{0.5} + \ldots + J_{k-0.5}). \]  

(31)

where in the above the dependence on $x, \phi^2$ (treated as constants) is implicit for simplicity of notation. The above allows for sequentially computing a series of values of $I_\zeta$ by only iterating the self-contained recurrence relation for $J_\zeta$ in (27) and storing a single value of $I_\zeta$ in memory.

Note that the case $\rho = 0$ in (12) is relevant only if integration is performed numerically (so that 0/0 does not occur in implementations). We therefore only need to consider the last remaining option $\lambda = 0$. In this case $\gamma^{(0)}_J \equiv 0$ since $\lambda$ multiplies the first integral in (8) (more generally $\gamma_J \equiv 0$ for $\lambda = 0$) and with the same parameterization and notation direct substitution yields

\[ \gamma^{(0)}_J = 0, \]
\[ \nabla \gamma^{(0)}_J = -n \sum a_n \frac{\mu}{n+2} \left[ J_{\frac{3}{2}} (t_1; \mu^2 - 1) - J_{\frac{5}{2}} (t_1; \mu^2 - 1) \right]. \]  

(32)

In the original definition (24) of $J_\zeta$ the second argument was nonnegative which is not guaranteed in the above. However, $J_\zeta$ defined as the integral in (24) is well defined for the second argument $\geq -1$, which makes the notation in (32) appropriate. In this case only the self-contained recurrence for $J_\zeta$ in (27) (second line) is used, together with $J_{0.5}$ in (29) and the limits

\[ J_0(x; \mu^2 - 1) = x, \]
\[ J_{0.5}(x; -1) = \frac{1}{2} |x| \zeta. \]  

(33)

where now the recurrence for even $n$ starts from $\zeta = 0$ (not $\zeta = 1$ as before).

### 2.4 Two dimensions

The two-dimensional version of the problem can be tackled in a similar fashion. The two integrals $\gamma / \nabla \gamma$ are defined as in (1) but in two dimensions. It has already been discussed\(^\text{12}\) that

\[ \gamma = \int_{\partial \Omega} G_{W}^{(2)}(\rho) \mathbf{P}(\mathbf{x} - \mathbf{x}_a) \, d\mathbf{t}, \]
\[ \nabla \gamma = -\int_{\partial \Omega} W(\rho) (\mathbf{x}^T \mathbf{Pn}) \, d\mathbf{t}, \]  

(34)

where $\mathbf{n} = \mathbf{n}(\mathbf{x})$ is the outward normal to $\partial \Omega$, $\mathbf{x} = \mathbf{x}(t)$ is an arbitrary parameterization of $\partial \Omega$, $\rho = \| \mathbf{x} - \mathbf{x}_a \|$, $\mathbf{P}$ is the two-dimensional $+\zeta 2$ rotation matrix and

\[ G_{W}^{(2)}(\rho) = \int_0^1 s W(\rho s) \, ds. \]  

(35)

For polynomial $W$ the above can be given explicitly

\[ \sum a_n \rho^n = W(\rho) \rightarrow G_{W}^{(2)}(\rho) = \sum \frac{a_n}{n + 2} \rho^n. \]  

(36)

For piecewise-linear boundaries in two dimensions it suffices to consider a single line element $\partial \Omega_j$ for general piecewise-planar geometries, including in the SPH method (i.e., there is no equivalent of integrating over circular arcs). As in three dimensions, the line segment of interest is spanned by $\mathbf{z}_{1,2}$ in this order, the unit direction is as in (14) and $\mathbf{x}_p \rightarrow \mathbf{x}_a$ in the definition (15) of $\mathbf{x}_i$. The parameterization (16) and the range (17) are analogous. By direct substitution to
(34) it follows that

$$\gamma_j = \mu \int_{t_1}^{t_2} G_w^{(2)} \left( \sqrt{\mu^2 + t^2} \right) dt,$$

$$\nabla \gamma_j = -n \int_{t_1}^{t_2} W \left( \sqrt{\mu^2 + t^2} \right) dt,$$  \hspace{1cm} (37)

where $n$ is constant along linear segments and

$$\mu = \hat{e}^T \mathbf{P}(z_1 - x_a)$$  \hspace{1cm} (38)

is the signed distance between $x_l$ and $x_a$. With polynomial $W$ (36) and our previous notation

$$\gamma_j = \mu \sum_a \frac{a_n}{n+2} \left[ J_n \left(t_2; \mu^2 - 1 \right) - J_n \left(t_1; \mu^2 - 1 \right) \right],$$

$$\nabla \gamma_j = -n \sum_a a_n \left[ J_n \left(t_2; \mu^2 - 1 \right) - J_n \left(t_1; \mu^2 - 1 \right) \right],$$

and the evaluation process is the same as in the special case $\lambda = 0$ in three dimensions.

3 | SMOOTHED-PARTICLE HYDRODYNAMICS

The closed-form integration method given in the previous section allows for updating the two- and three-dimensional algorithms given in Reference 12 in order to make them fully analytic. In the SPH method the radial function $W$ satisfies the additional conditions

$$\int_{\Omega_a} W \, dV = 1,$$

$$W(r \geq R) = 0,$$  \hspace{1cm} (40)

where $\Omega_a$ is the particle support, for example, a sphere (circle in 2D) with a known radius $R$ centered at $x_a$. By exploiting the above it follows that for piecewise-planar boundaries and for $x_a \in \Omega$

$$\gamma = \int_{\Omega \cap \Omega_a} W \, dV = 1 + \sum_j \left( \gamma_j - \text{sgn} (\gamma_j) \frac{A_j}{A_0} \right),$$

$$\nabla \gamma = - \int_{\partial \Omega \cap \Omega_a} W n \, dS = \sum_j \nabla \gamma_j$$  \hspace{1cm} (41)

with the index $j$ running over all boundary elements $\partial \Omega_j$ such that $\partial \Omega_j \cap \Omega_a$ is nontrivial; detection of these is an essential part of SPH codes and is not discussed in further detail. The contributions $\gamma_j / \nabla \gamma_j$ in (41) are integrated over $\partial \Omega_j \cap \Omega_a$ (or their contours in 3D), $A_0$ is the area of the unit sphere ($4\pi$ in three dimensions, $2\pi$ in two dimensions) and $A_j$ is the area of the projection of $\partial \Omega_j \cap \Omega_a$ onto the unit sphere (circle in 2D) centered at $x_a$. In a more general case, when in (40) the first equation is replaced with

$$\int_{\Omega_a} W \, dV = M$$  \hspace{1cm} (42)

the first equation in (41) is accordingly replaced with

$$\gamma = M + \sum_j \left( \gamma_j - \text{sgn} (\gamma_j) \frac{A_j M}{A_0} \right).$$  \hspace{1cm} (43)
and we note that for polynomial kernels

$$\sum a_n r^n = W(r) \rightarrow M = 4\pi R^3 \sum \frac{a_n R^n}{n+3}. \quad (44)$$

Formulation (41) splits the evaluation of $\gamma$ into the integral over the whole support $M$ (usually $M = 1$ in SPH codes), addition of piecewise-planar element contributions $\gamma_j$ and correcting for the region of $\partial \Omega_a$ obscured by $\partial \Omega_j \cap \Omega_a$. The sign factor $\text{sgn}(\gamma_j)$ guarantees the correct result in cases when the domain of integration is not star-convex w.r.t. $x_a$ and here is defined at zero via $\text{sgn}(0) = 1$. Evaluation of $\nabla \gamma$ in (41) relies only on direct contributions from boundary elements due to the natural second condition on $W$ in (40). In the rest of the section we discuss computing $\gamma_j, A_j$ and $\nabla \gamma_j$ for a single piecewise-planar (linear) element $\partial \Omega_j$ in the context of implementation in SPH simulators.

### 3.1 Two dimensions

Let a linear boundary segment spanned by $x_{1,2}$ in this order be given. Generally, the segment may be (partially) outside $\Omega_a$, where $W$ is identically zero. These regions are trivial and do not need to be integrated over and will be accounted for as previously.\(^{12}\)

First, the unit tangent is given by

$$\hat{e} = \frac{x_2 - x_1}{||x_2 - x_1||} = \mathbf{P}n,$$

$$\mathbf{P} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \quad (45)$$

and then we compute the parameterization limits $t_1$ and $t_2$ in (16) corresponding to $x_{1,2}$, namely

$$t_{1,2} = (x_{1,2} - x_a)^T \hat{e} = (x_{1,2} - x_a)^T \mathbf{P}n. \quad (46)$$

Next, the intersections of the line passing through $x_{1,2}$ and $\partial \Omega_a$ are investigated. With the parameterization (16) the intersection condition $R^2 = ||x(t) - x_a||^2$ can be rewritten as

$$R^2 = t^2 + \mu^2,$$

$$\mu = n^T(x_1 - x_a), \quad (47)$$

which describes a segment contributing in a nontrivial way only if $\mu^2 < R^2$ (the contributions $\gamma_j, A_j$ and $\nabla \gamma_j$ are all zero otherwise). Under this assumption we then truncate the parameterization $[t_1, t_2]$ via

$$t_{1,2} \leftarrow \min \left( \sqrt{R^2 - \mu^2}, \max \left(-\sqrt{R^2 - \mu^2}, t_{1,2} \right) \right) \quad (48)$$

and proceed only if after the truncation $t_2 > t_1$—otherwise the segment is outside of the particle support and all its contributions are zero. With the definition of $\mu$ in (47) and after the truncation (48) the element contributions $\gamma_j/\nabla \gamma_j$ for polynomial $W$ are as in (39), and the area of the projection of the truncated segment onto the unit circle centered at $x_a$ is given by

$$A_j = \cos^{-1} \left( \frac{\mu^2 + t_1 t_2}{\sqrt{(\mu^2 + t_1^2)(\mu^2 + t_2^2)}} \right) \quad (49)$$

with only the case $\mu = t_1 t_2 = 0$ not well defined given the current information. The latter corresponds to the fluid particle being precisely at a corner of $\partial \Omega$ and requires information about both segments connected by the corner. A simple safety mechanism dealing with this issue was described in Reference 12, where the particle is moved away from the corner by
a tiny amount in the negative mean direction of the surrounding normals. However, in practice we would not expect \( x_a \in \partial \Omega \) to occur in simulations (particles stay away from the walls).

An efficient implementation of the proposed method for polynomial \( W \) is summarized in Algorithm 1 below. The implementation is valid for all \( x_a \) inside \( \Omega \) and along \( \partial \Omega \) with the exception of corner points (detectable via \( \mu = t_1t_2 = 0 \)).

### Algorithm 1. Pseudocode for the two-dimensional algorithm for polynomial \( W \) with the support radius \( R \)

**Data:** Appropriately oriented end points of a line segment \( \partial \Omega_j \), \( x_{1,2} \) and the outward unit normal \( n \)

**Result:** Element contributions to \( \gamma/\nabla \gamma \), \( \gamma_j/\nabla \gamma_j \), and the area \( A_j \) to be used in (41)/(43)

```
1 \( \gamma_j \leftarrow 0 \);
2 \( \nabla \gamma_j \leftarrow 0 \);
3 \( A_j \leftarrow 0 \);
4 Evaluate \( \mu = n^T(x_1 - x_a) \)
5 if \( R^2 - \mu^2 > 0 \) then
6   for \( k = 1, 2 \) do
7     \( t_k = \min \left( \sqrt{R^2 - \mu^2}, \max \left( -\sqrt{R^2 - \mu^2}, (x_k - x_a)^T Pn \right) \right) \)
8   end
9   if \( t_2 > t_1 \) then
10  Evaluate \( \gamma_j/\nabla \gamma_j \) as in (39) and \( A_j \) as in (49)
11 end
12 end
```

### 3.2 Three dimensions

Let \( \partial \Omega_j \) be a convex polygon in the plane \( P \) with corners \( x_{1,n} \) oriented counter-clockwise around the outward normal \( n \). Due to the finite size of the spherical support \( \Omega_a \) the general nontrivial intersection \( \partial \Omega_j \cap \Omega_a \) will be a planar element with its edge described by a combination of straight lines and circular arcs.

First, \( \lambda \) is computed from

\[
\lambda = n^T(x_1 - x_a),
\]

where \( x_1 \) can be replaced by any other corner of the polygon. The element contributes in a nontrivial way only if \( R^2 - \lambda^2 > 0 \), which is assumed, and is skipped otherwise. The intersection \( \Omega_a \cap P \) is a circle centered at \( x_P \) with radius

\[
\rho_\perp = \sqrt{R^2 - \lambda^2}.
\]

Next, we consider a single edge \( l_i \) of the polygon, spanned by \( x_{i,i+1} \) in this order (\( x_{n+1} = x_1 \)). For this edge we compute

\[
\hat{e} = \frac{x_{i+1} - x_i}{\|x_{i+1} - x_i\|}
\]

and

\[
\mu = n^T((x_i - x_a) \times \hat{e}).
\]

Again, the segment has a nontrivial contribution only if \( \rho_\perp^2 > \mu^2 \). Also note that \( x_P \) is on the left of \( l_i \) in plane \( P \) if \( \mu \geq 0 \) and vice versa. For a contributing segment we evaluate and truncate
\[ t_{1,2} = \mathbf{e}^T (x_{i+1} - x_i), \]
\[ t_{1,2} \leftarrow \min \left( \frac{\rho_1^2 - \mu^2}{\mu^2}, \max \left( -\sqrt{\frac{\rho_1^2}{\mu^2}} - t_{1,2} \right) \right). \] (54)

while recording if any of \( t_{1,2} \) was truncated from either above or below by the \( \min/\max \) functions. As in two dimensions, the segment contributes only if \( t_2 > t_1 \) and is skipped otherwise. The contributions to \( \gamma_j/\nabla \gamma_j \) from the edge \( l_i \), written \( \gamma_j^{(i)}/\nabla \gamma_j^{(i)} \), are evaluated as in (23) with the truncated parameterization (54) and \( \lambda, \mu \) as in (50), (53). For any contributing line segment we also calculate the arc length \( \psi^{(i)} \) of its projection onto the unit circle in the plane \( P \) centered at \( x_p \), given by the same expression as in (49) but with the values of \( \mu, t_{1,2} \) as in (53) and (54).

After all sides \( l_1, \ldots, n \) are considered, we initialize
\[
\gamma_j = \sum_i \gamma_j^{(i)}, \\
\nabla \gamma_j = \sum_i \nabla \gamma_j^{(i)}, \\
\psi = -\sum_i \text{sgn} \left( \mu^{(i)} \right) \psi^{(i)},
\] (55)

where the index \( i^* \) runs only over contributing segments (i.e., those with \( \mu^2 + \lambda^2 < R^2 \)) and \( \mu^{(i)} \) is the value of \( \mu \) as defined in (53) along the (contributing) edge \( l_i \). The sign function in the above is again defined at zero as \( \text{sgn}(0) = 1 \). The above values of \( \gamma_j/\nabla \gamma_j \) are not yet exact, since they miss contributions from potential circular arcs along \( C_j \). First, if and only if along all contributing straight edges \( \mu \geq 0 \) then \( \psi \) is updated according to
\[ \psi \leftarrow \psi + 2\pi, \] (56)

which indicates that \( x_p \in \partial \Omega_j \) (if \( \partial \Omega_j \) is a convex polygon). In order to account for circular arcs we finally update
\[
\gamma_j \leftarrow \gamma_j + \lambda \psi \rho_1^2 H_W^{(3)}(\rho_1^2; \lambda), \\
\nabla \gamma_j \leftarrow \nabla \gamma_j - \mathbf{n} \psi \rho_1^2 H_W^{(2)}(\rho_1^2; \lambda),
\] (57)

which correctly evaluates the contributions \( \gamma_j/\nabla \gamma_j \) to be used in (41).

The last remaining part of the solution is the evaluation of the projection area \( A_j \) of \( \partial \Omega_j^{(a)} = \partial \Omega_j \cap \Omega_a \) onto the unit sphere centered at \( x_a \). The general shape of \( \partial \Omega_j^{(a)} \) consists of a convex polygon and a number of sections of the circle \( O_P \) with radius \( \rho_1 \) in the plane \( P \) centered at \( x_p \). The corners of the polygon can be constructed from the parameterizations (54) along the contributing edges and sections of the circle can be detected and characterized by investigating if (54) was truncated, as mentioned previously. The projection area of the smaller segment of \( O_P \) bounded by the chord with length \( 2d \) is given by
\[
A(d) = \begin{cases} 
\pi, & \lambda = 0, \; d = \rho_1 \\
\pi - 2 \left( \cos^{-1} \left( \frac{|d|}{\rho_1\sqrt{R^2-d^2}} \right) + |d| \sin^{-1} \left( \frac{d}{R\rho_1} \right) \right), & \text{otherwise}
\end{cases}
\] (58)

which is equivalent to the formula given in Reference 12 with the additional handling of \( \lambda = 0, d = \rho_1 \). The above can be expanded to provide an expression for the projection area of either the larger or the smaller section of \( O_P \), where the chord is spanned by \( y_{1,2} \) in this order
\[
A^{(i)}(y_1, y_2) = (1 + \sigma) A(\rho_1) - \sigma A \left( \frac{|y_1 - y_2|}{2} \right), \\
\sigma = \text{sgn} \left( \mathbf{n}^T (x_a - y_1) \times (y_2 - y_1) \right)
\] (59)

with \( \sigma \) positive if the chord goes around \( x_p \) in the anticlockwise fashion (corresponding to the smaller area) and vice versa. In the above we can either again use \( \text{sgn}(0) = 1 \) or the typical convention \( \text{sgn}(0) = 0 \). Finally, let

\[ A^{(i)}(y_1, y_2) = (1 + \sigma) A(\rho_1) - \sigma A \left( \frac{|y_1 - y_2|}{2} \right), \\
\sigma = \text{sgn} \left( \mathbf{n}^T (x_a - y_1) \times (y_2 - y_1) \right)
\] (59)
be an array of projections of the corners of an arbitrary convex planar polygon onto the unit sphere centered at \( \mathbf{x}_n \) (\( \| \mathbf{c}_k \| = 1 \) for all \( k = 1, \ldots, m \)). The ordering of \( \mathbf{c}_k \) should correspond to an arbitrary consecutive ordering of the corners of the original polygon. The area of the spherical polygon spanned by \( \mathbf{c}_k \) (either equal to \( A_j \) or directly contributing to it) is given by

\[
A^{[C]} = (2 - m)\pi + \sum_{k=1}^{m} \cos^{-1} \left( \frac{\mathbf{c}_k^T \mathbf{c}_{k+1} - p_k p_{k-1}}{\sqrt{(1 - p_k^2)(1 - p_{k-1}^2)}} \right).
\]

\( p_k = \mathbf{c}_k^T \mathbf{c}_{k+1} \)

with the above equivalent to the expression provided and discussed in further detail in Reference 12. Note that in implementations it is preferred to evaluate just \( p_{1,\ldots,m} \) first and then use them in the above formula in order to minimize the computational expense (the naive implementation would evaluate each \( p_k \) twice). Also observe that necessarily \( m \leq 2n \) where \( n \) is the number of sides/corners of the boundary element \( \partial \Omega_i \); the latter bound may be important in implementations if the amount of memory allocated to \( C \) has to be specified a priori.

Many of the terms involved in the calculations presented, particularly (61), produce singularities for certain inputs. While these inputs should in practice never be encountered, our experience shows that finite machine precision can often lead to situations where the function evaluations do become singular. This is because the formulae presented involve repeated floating point operations which can produce results where the values are indistinguishable to machine precision. This is often the case when an element corner is relatively close (within \( 10^{-10} \)) to \( \partial \Omega_i \). In order to avoid floating point errors, in (61) we first explicitly limit

\[
p_k \leftarrow \max(-1, \min(1, p_k))
\]

\( \text{to ensure that the argument of } \sqrt{\cdots} \text{ in the denominator of each term in (61) is nonnegative. Second, each term in the sum is modified according to}

\[
\cos^{-1} \left( \frac{\cdots}{\sqrt{\cdots}} \right) \to \cos^{-1} \left( \max \left[ -1, \min \left[ 1, \frac{\cdots}{\max(\epsilon, \sqrt{\cdots})} \right] \right] \right),
\]

where \( \epsilon \) is a small positive user-specified tolerance on the order of machine precision; in our implementations \( \epsilon = 10^{-14} \) is used. The above modifications directly prohibit singularities from occurring and guarantee that the argument of \( \cos^{-1} \) is in the appropriate range \([-1, 1]\). Similar but simpler safeties are used in (58) for the arguments of \( \sin^{-1} \) and \( \cos^{-1} \). Since both of them are positive and \( \rho_\perp, R^2 - d^2 > 0 \) are already guaranteed when \( A(d) \) is evaluated, each argument is only limited from the top via \( \min(1, \cdot) \). An explicit mechanism for avoiding singularities in (58) from division by zero can also be used (cf. \( \max(\epsilon, \cdot) \) in (63)).

The above methodology is visualized in Figure 3. The main view and the three projection angles \((a - c)\) are further supplemented by an interactive U3D model”.

An efficient and fully general implementation of our approach for polynomial \( W \) is summarized in Algorithm 2 below. MATLAB, C++ and Python implementations of the general three-dimensional algorithm together with examples corresponding to Figure 3 are provided in Appendices A, B, and C, respectively. Similarly to 2D, the method handles arbitrary \( \mathbf{x}_i \) within \( \Omega \) and along \( \partial \Omega \) with the exception of regions of nonsmoothness, namely the element edges and corners, that is, all cases when \( \mu = 0 \) and \( t_1 t_2 \leq 0 \). In those cases the projection area of the element \( A_j \) onto the unit sphere is not well defined and evaluating the appropriate contribution requires information on neighboring elements. The previously mentioned safety mechanisms\(^{12}\) applies in such scenarios.

*3D content activates on click. The U3D format is supported by Acrobat Reader (version 7 and above). For more information visit https://helpx.adobe.com/acrobat/using/enable-3d-content-pdf.html.
FIGURE 3 Visualization of the three-dimensional problem where a particle (green) with radius \( R = 1 \) (so that \( \partial \Omega_a \) is a unit sphere) interacts with a quadrilateral boundary element \( \partial \Omega_j \). The contour \( C_j \) of \( \partial \Omega_j \cap \Omega_a \) is composed of three straight lines (yellow, solid) and two arcs (red) with their end points connected by yellow dashed lines (also see views (a – c) for more detail). The projection area \( A_j \) (light blue) is the sum of the projection areas of the pentagon bounded by all yellow segments (dark blue contour, cf. \( A^{(C)} \) in (61)) and two sections of the circle bounded by red arcs and dashed yellow lines (here two spherical regions between red and blue arcs, \( A^{(s)} \) in (59)).

The correct handling of \( \mu, \lambda = 0 \) in implementations is crucial, even if the particle position \( \mathbf{x}_a \) is always assumed to be away from walls. Indeed, for \( \mathbf{x}_a \) along the line containing a straight edge of an element within the particle support (but away from the edge itself and \( \partial \Omega \) generally) one has \( \lambda = \mu = 0 \) but can be strictly on the inside of \( \Omega \) (only if \( \Omega \) is not convex).

Equations (41), (43), with contributions \( \gamma_j, \nabla \gamma_j, A_j \) as defined in this work, produce correct results for \( \mathbf{x}_a \in \overline{\Omega} \) (with the exceptions given before). For \( \mathbf{x}_a \) strictly outside \( \overline{\Omega} \), and without any change in the implementation, the resulting value of \( \nabla \gamma \) will still be computed correctly. However, in such cases the computed and true values of \( \gamma \), written \( \gamma_C, \gamma_T \) respectively, satisfy

\[
\gamma_C = 2M - \gamma_T
\]  

and \( \gamma_C > M \) if \( \mathbf{x}_a \notin \overline{\Omega} \). The latter condition is unlikely to occur and generally not desired in simulations. Still, it may potentially occur and be hard to fully eliminate in extreme cases such as very high-velocity impacts. A simple general solution ensuring the correct value of \( \gamma \) is setting

\[
\gamma \leftarrow \min(\gamma, 2M - \gamma)
\]

after the calculation (41)/(43) involving all contributing boundary elements.

4 SIMPLIFIED ALGORITHMS

The source of singularities (and the main source of complexity in three dimensions) in Algorithms 1 and 2 is evaluation of the projection area \( A_j \). The latter is necessary for the evaluation of \( \gamma \) (Equation 41) and is essentially an indirect way of integrating over the spherical region \( \partial \Omega_a \setminus \partial \Omega \).
Algorithm 2. Pseudocode for the three-dimensional algorithm for polynomial $W$ with the support radius $R$. All of $\gamma_j, A_j$ and $\nabla \gamma_j$ are computed in the same loop. An additional switch can be added after line 15, so that $t_1 t_2 \leq 0 = \mu$ raises an error.

Data: Appropriately oriented corners of a convex polygon $\partial \Omega_j, x_1, \ldots, x_n$ and the outward unit normal $n$

Result: Element contributions to $\gamma / \nabla \gamma, \gamma / \nabla \gamma_j$, and the area $A_j$ to be used in (41)/(43)

1. $\gamma_j \leftarrow 0$
2. $\nabla \gamma_j \leftarrow 0$
3. $A_j \leftarrow 0$
4. Evaluate $\lambda = n^T(x_1 - x_a)$
5. if $R^2 > \lambda^2$ then
   6. $y_1 \leftarrow [\ ]; y_2 \leftarrow [\ ]; C \leftarrow [\ ];$
   7. $\psi \leftarrow 0$, $L_0 \leftarrow$ true;
   8. Evaluate $\rho_\perp = \sqrt{R^2 - \lambda^2}$;
   9. for $i = 1, \ldots, n$ do
      10. Evaluate $\hat{e}$ and $\mu$ as in (52) and (53);
      11. if $\mu < 0$ then $L_0 \leftarrow$ false;
      12. if $\rho_\perp^2 > \mu^2$ then
         13. Evaluate and truncate $t_{1,2}$ as in (54). During the process evaluate:
            $L_1 \leftarrow$"$t_1$ was truncated from the bottom in (54)";
            $L_2 \leftarrow$"$t_2$ was truncated from the top in (54)";
      14. if $t_2 > t_1$ then
         15. Evaluate $\gamma^{(i)}_j / \nabla \gamma^{(i)}_j$ as in (23) (or (32) if $\lambda = 0$) and $\psi^{(i)}$ according to (49);
         16. if $\mu \geq 0$ then $\psi^{(i)} \leftarrow -\psi^{(i)}$;
         17. $\gamma_j \leftarrow \gamma_j + \gamma^{(i)}_j$;
         18. $\nabla \gamma_j \leftarrow \nabla \gamma_j + \nabla \gamma^{(i)}_j$;
         19. $\psi \leftarrow \psi + \psi^{(i)}$;
      20. if $L_1$ then
         21. $z = x_i + [t_1 - \hat{e}^T(x_i - x_a)]\hat{e}$;
         22. if $y_1 \neq [ \ ]$ then
            23. $A_j \leftarrow A_j + A^{(i)}(y_1, z)$ with $A^{(i)}(\cdot, \cdot)$ as in (59);
            24. else
            25. $y_2 \leftarrow z$;
         26. end
         27. Append $C \leftarrow [C, \frac{x-x_a}{\|x-x_a\|}]$;
      28. end
      29. $z = x_i + [t_2 - \hat{e}^T(x_i - x_a)]\hat{e}$;
      30. Append $C \leftarrow [C, \frac{x-x_a}{\|x-x_a\|}]$;
      31. if $L_2$ then $y_1 \leftarrow z$;
      32. end
   33. end
      34. if $L_0$ then
         35. if $\psi = 0$ then $A_j \leftarrow 2A(\rho_\perp)$ with $A(d)$ as in (58);
         36. $\psi \leftarrow \psi + 2\pi$;
      37. end
      38. Update $\gamma_j$ and $\nabla \gamma_j$ as in (57);
      39. if $y_2 \neq [ \ ]$ then $A_j \leftarrow A_j + A^{(i)}(y_1, y_2)$ with $A^{(i)}(\cdot, \cdot)$ as in (59);
      40. end
      41. if $\text{numel}(C) \geq 3$ then $A_j \leftarrow A_j + A^{[C]}$ with $A^{[C]}$ as in (61);
      42. end
   43. end
44. end
Algorithm 3. Pseudocode for the simplified three-dimensional algorithm for polynomial $W$ with the support radius $R$, where only $\nabla \gamma_j$ is computed

**Data:** Appropriately oriented corners of a convex polygon $\partial \Omega_j$, $x_1, \ldots, x_n$ and the outward unit normal $n$

**Result:** Element contribution to $V\gamma_j, \nabla \gamma_j$, to be used in (41)

1. $\nabla \gamma_j \leftarrow 0$;
2. Evaluate $\lambda = n^T(x_1 - x_a)$
3. if $R^2 > \lambda^2$ then
4. $\psi \leftarrow 0$, $L_0 \leftarrow$ true;
5. Evaluate $\rho_\perp = \sqrt{R^2 - \lambda^2}$;
6. for $i = 1, \ldots, n$ do
7. Evaluate $\hat{e}$ and $\mu$ as in (52) and (53);
8. if $\mu < 0$ then $L_0 \leftarrow$ false;
9. if $\rho_\perp^2 > \mu^2$ then
10. Evaluate and truncate $t_{1,2}$ as in (54);
11. if $t_2 > t_1$ then
12. Evaluate $\nabla \gamma_j^{(i)}$ as in (23) (or (32) if $\lambda = 0$) and $\psi^{(i)}$ according to (49);
13. if $\mu \geq 0$ then $\psi^{(i)} \leftarrow -\psi^{(i)}$;
14. $\nabla \gamma_j \leftarrow \nabla \gamma_j + \nabla \gamma_j^{(i)}$;
15. $\psi \leftarrow \psi + \psi^{(i)}$;
16. end
17. end
18. if $L_0$ then $\psi \leftarrow \psi + 2\pi$;
19. Update $\nabla \gamma_j$ as in (57);
20. end
21. end

However, Ferrand et al.\textsuperscript{7} proposed a formulation where only $\nabla \gamma$ is needed and an ODE in time for $\gamma$ is solved numerically. With their approach the calculation of $A_j$ can be omitted and our method simplifies significantly in three dimensions as summarized in Algorithm 3 below. Observe that the same approach in two dimensions (Algorithm 1) leaves the corresponding algorithm practically unchanged since the calculation of $A_j$ in two dimensions (49) is evaluated in a single inexpensive step. In both two and three dimensions the simplified algorithm is valid for all $x_a$, including along the edges and at the corners of boundary elements. At the same time, direct calculation of $\gamma$ is very desirable in implementations and naturally leads to much simpler and more stable numerical schemes; we therefore recommend the full method to be used in practice.

Finally, we note that the methodology derived in this work can be used for the purpose of more generic efficient numerical integration over two- and three-dimensional piecewise-planar domains. In order to do so, a generic function $f$ can be approximated via the Radial Basis Function (RBF) interpolation method with a polynomial basis function of infinite support (e.g., $W(r) = r^3$, the cubic spline), written as

$$f(x) \approx \sum_{k=1}^{N_w} w_k \ W(||x - \tilde{x}_k||),$$

(66)

where $w_k$ are constants and $\tilde{x}_k$ are a prespecified collection of RBF centers. Typically, $\tilde{x}_k$ are specified a priori and correspond to points at which the value of $f$ is known. The set of weights $w_k$ is then constructed by solving a $N_w \times N_w$ linear system equivalent to enforcing the agreement between the RBF surrogate and the original function. The details of the RBF interpolation method are not crucial for this work and for further information the reader is kindly referred to Reference 13.
Once the surrogate (66) is given, we approximate

$$\gamma^{(j)} = \int_{\Omega} f \, dV \approx \sum_{k=1}^{N_w} W_k \left( \int_{\Omega} W_k \, dV \right)$$

(67)

with $W_k = W(||x - \bar{x}_k||)$. Each of the integrals on the right-hand side of Equation (67) can then be evaluated via the methodology proposed. If $W$ has infinite support then the decomposition (5) applies, and $\gamma^{(j)}$ in (67) can be evaluated in a single loop over the elements of $\partial\Omega_j$, but at each element a series $N_w$ contributions from $N_w$ centers $\bar{x}_k$ are evaluated. Note that the truncation of $t_{1,2}$ and the contribution of circular arcs (controlled via $\psi$ in our notation) are no longer applicable, further simplifying the implementation. The main advantage of the approach is that only the basic boundary mesh information can be used to evaluate three-dimensional integrals without complex meshing of the interior of the domain or any degree of approximation other than constructing the RBF surrogate for $f$. Moreover, the method resolution (here the number of RBF centers $N_w$) can be adjusted just w.r.t. the complexity of $f$ rather than the domain of integration $\Omega$. Pseudocode for the appropriately simplified three-dimensional method is given in Algorithm 4. We note that a very similar approach can be used to integrate scalar functions over (open and closed) piecewise-planar surfaces in three dimensions. The method would replace $\gamma_j$ with $\nabla \gamma_j$ in Algorithm 4 but with the multiplication by $-\mathbf{n}$ omitted in both expressions for $\nabla \gamma_j$ in (23) and (32).

Algorithm 4. Pseudocode for the simplified three-dimensional algorithm for integrating over piecewise-planar domains via RBF interpolation. Now the radial function $W$ is assumed to have infinite support

Data: Appropriately oriented corners of a convex polygon $\partial\Omega_j$, $x_1,\ldots,n$, the outward unit normal $\mathbf{n}$ and $\{w_k,\bar{x}_k\}_{k=1}^{N_w}$ as in (66)

Result: Element contribution to $\gamma^{(j)}$, $\gamma_j$, to be used in (5)

1 $\gamma_j \leftarrow 0$
2 for $k = 1,\ldots,N_w$
3     Set $x_a = \bar{x}_k$
4     Evaluate $\lambda = \mathbf{n}^T(x_1 - x_a)$
5         for $i = 1,\ldots,n$
6             Evaluate $\hat{e}$ and $\mu$ as in (52) and (53)
7             Evaluate $t_{1,2}$ as in (54) without truncating;
8             Evaluate $\gamma_j^{(i)}$ as in (23) (or (32) if $\lambda = 0$);
9         end
10        Update $\gamma_j \leftarrow \gamma_j + w_k \gamma_j^{(i)}$
11 end

5 | RESULTS

In this section example applications of the methodology proposed in this work are presented, both in the context of SPH simulations and numerical integration with the aid of the RBF interpolation method.

5.1 | Smoothed-particle hydrodynamics

Generality and accuracy of the semi-analytical formulation have already been derived and established, and the discussed method of computing the correction factors $\gamma/\nabla \gamma$ is exact (maximally accurate). The crucial remaining issue is the numerical efficiency, which if low, may provide an incentive to still use more widely spread and less accurate heuristics.
In order to quantify the performance of the approach, the supplied generic C++ implementation (Appendix B) was directly ported to an existing SPH code, originally employing the (most common) fluid extension (aka ghost particle) boundary method.\textsuperscript{14} Both the original and modified (semi-analytical) simulators share the same base code and differ just in boundary handling, ensuring a fair comparison in the context of interest. The SA method implemented mostly followed the methodology of Ferrand et al.\textsuperscript{7} with some differences such as the time integration scheme and boundary pressure calculation in order to be as close as possible to the existing ghost particle implementation. Numerical efficiency was quantified on a typical SPH benchmark test case visualized in Figure 4.

Five fluid resolutions between 3000 and 100,000 particles were investigated and the \( C^2 \) quintic Wendland kernel\textsuperscript{8} was used. A direct comparison of simulation times for both formulations is given in Figure 5. At each resolution two or three independent simulations are run for both boundary methods in order to provide a basic statistical characterization (Figure 5, bottom plot).

The semi-analytical code is practically as fast as the original (fluid extension). On average, in our test it runs slightly faster at low (<10,000) and high (>90,000) resolutions and slightly slower at the intermediate ones. However, in all cases the difference is practically negligible—the relative speed of both methods is within ±3\%, making them practically equal in this respect.

It is noted that this level of performance was achieved with minimal modifications of the source supplied in Appendix B. The only difference between the provided and our code is the use of an already in place framework for handling vectors (including dot and cross products). Further increase in speed and efficiency are easily achievable; the latter two were not the only design criteria for the implementations provided, with generality and easy portability also playing a crucial role. If better performance is desired, we first recommend reducing or eliminating the need for dynamic memory allocation and precomputing and hard-wiring (i.e., fixing at compile time) the coefficients \( \frac{a_n}{(n+2)(n+3)} \), and so on instead of passing \( a_n \)'s for each element-particle pair at run time.

Finally, the generality and robustness of the semi-analytical framework is illustrated on a coastal defense example (Figure 6, on-click interactive U3D model). Here the simple obstacle from the previous test is replaced with a breakwater comprising of five irregularly spaced pieces of rock armor fixed in place. Note the complex shape of the domain in the vicinity of the defense structure. Geometric configurations such as the one presented in Figure 6 are often too complex to be handled efficiently by typical techniques such as the fluid extension method. For example, the discretization of a single breakwater such as the one in Figure 6 can be seen in Figure 7. The SA method at most discretizes just the surface of the walls, while the fluid extension method requires ghost particles to fill the boundaries up to a fixed distance, which is difficult and inefficient for such complex and high-resolution geometries.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure4.png}
\caption{Visualization of the benchmark test case (dam break problem). A body of water is initially at rest toward one end of the container (left). After release, the fluid spills and splashes against a simple obstacle (right). For the purpose of comparison, in this study the fluid-boundary interaction is modeled via the fluid extension and semi-analytical methods.}
\end{figure}
FIGURE 5  Comparison of run times of the fluid extension (FE) and the semi-analytical (SA) methods. Mean run times at five fluid resolutions are plotted at the top. The relative run times of the semi-analytical code are summarized in the bottom plot; black intervals are the typical ranges estimated from simple two- and three-point statistics.

FIGURE 6  Computational domain for the coastal defense test case in top (A), diagonal (B), and front (C) views. Two sides of the tank are removed for clarity of presentation. The geometry of interlocking rock armor pieces is highly nontrivial and contains multiple cavities and regions of nonconvexity.
Figure 7 Discretization of a single breakwater by different boundary methods. Image (A) is the stl geometry file used to initialize the method. Image (B) presents the discretization used in the SA method, which places ghost particles along the surface of the geometry. In this case a total of 4223 particles are required. Image (C) presents the discretization used in a typical fluid extension method, which fills part of the geometry with ghost particles. In this case, a total of 8016 particles are required, almost double that of the SA implementation.

In the example in Figure 6, the fixed ghost particle method requires a total of ~350,000 particles to fully model the boundaries, while the SA method at most requires only ~185,000 particles. The boundary particles in the SA method are required only if placing a single layer of boundary particles to improve the calculation of gradients and/or shear forces near the wall. For example, the SA formulation of Ferrand et al.\textsuperscript{7} requires the surfaces of the walls to be discretized by boundary particles in order to improve calculations of gradients near the boundary. However, the formulation of Kulasegaram et al.\textsuperscript{5} does not explicitly require any boundary particles. The implementation in this work is agnostic to the presence of boundary particles and can be equivalently employed by formulations with or without them. In this work, and in general, a Ferrand style formulation is used, but if there is a focus on speed and minimizing the number of particles, then a Kulasegaram style formulation can instead be used. Special note is given to Ferrand et al.’s paper which does present both formulations.\textsuperscript{7}

The full setup process is minimal and comprises of just providing a generic triangulated mesh file (cf. Figure 6) and specifying the initial body of fluid. A number of illustrative simulation snapshots are given in Figure 8. While not presented here, these results are comparable in overall dynamics to those produced by a typical ghost particle method. Other literature addresses direct comparison of the two methods.\textsuperscript{7}

5.2 Volume and surface integration

For demonstration purposes we consider the Rosenbrock (banana) function, given by

\[ f(x, y, z) = (1 - x)^2 + (1 - y)^2 + 100 \left((y^2 - x)^2 + (z^2 - y)^2\right) \] (68)

over the domain \( \Omega_0 = [-1, 1]^3 \). As suggested in Section 4, an RBF interpolant of (68) was first constructed, with the samples taken over a regular \( 8 \times 8 \times 8 \) grid (Figure 9) and the cubic spline kernel function \( W(r) = r^3 \). The relative \( L_2 \) error between \( f \) and its interpolant over \( \Omega_0 \) (estimated numerically via dense regular sampling) is under \( 10^{-4} \).

After \( f \) is approximated as a weighted sum of polynomial radial functions, integrating it over any structure (surface/volume) described by piecewise-planar polygonal elements is exact and cheap. The only source of errors is the inaccuracy of the interpolant, which can be decreased easily to arbitrarily low values via more dense sampling. In Figure 10 two examples of numerical integration of the interpolant via the method proposed are given. Note that the area and volume of any shape can be evaluated in the special case \( W(r) = 1 \) (\( a_0 = 1, a_{n \geq 1} = 0 \)) centered at an arbitrary \( x_a \).
FIGURE 8  Visualization of the sea defense example. As in the previous problem, the body of water is released from rest at the back end of the container and splashes against the now highly nontrivial obstacle. The complex interaction between the fluid and five pieces of rock armor is handled automatically by the semi-analytical approach, which only requires the basic mesh information (here a triangulated geometry in the .stl format).
FIGURE 9  Sliced visualization of the three-dimensional Rosenbrock function (A). The function was first sampled at the nodes of a regular (optimal for the RBF method) $8^3$ grid within $[-1,1]^3$ (B). The RBF interpolant (C) was then constructed from the sampled data and can be integrated easily via the methodology discussed in this work.

FIGURE 10  Illustration of the method being used for surface and volume integration. Volume integration is applicable for closed surfaces (A) while surface integration applies to both closed and open surfaces (A,B). In (A) the integral of the approximated Rosenbrock function over the volume is evaluated at 71.33 and the surface integrals in (A,B) are evaluated at 658.9 and 998.6, respectively. Note how a coarse mesh can be used in (A) without affecting stability or accuracy.

6 CONCLUSIONS

A general closed-form method for exactly evaluating integrals of polynomial radial kernels in two and three dimensions was given. The approach does not utilize any degree of numerical approximation and avoids explicitly constructing involved antiderivative expressions by employing pointwise recurrence relations. Crucially, arbitrary nonconvex domains are handled correctly by the method.

The original motivation for the technique is the semi-analytical formulation of the SPH method, where the calculation discussed is responsible for accurate and general modeling of wall boundary conditions. Alternative applications, namely volume and surface integration by employing polynomial RBF interpolants, were discussed and examples of all proposed applications were given. Fully self-contained implementations of the general three-dimensional method were supplied in three popular programming languages (Appendices A–C) and can be easily adapted to work with new or existing SPH simulators (via Equation 41). We remind the reader that the (cheap) correction (65) can be used in implementations for full generality.
Our approach is already fully analytic, hence further accuracy improvements are not possible. Conceptually and implementation-wise, the method is designed to be compact and efficient; involved calculations are performed only in cases where an edge of the boundary is within particle support. Source codes provided can be easily optimized further, for example, by employing precomputations or clever memory handling. Optimizations of this kind are implementation-specific and beyond the scope of this work, but have significant potential in particular in applications where execution speed is crucial. Still, even without such tweaks a direct port of our generic C++ source (Appendix B) was demonstrated to be practically as fast as the most common alternative (fluid extension) when coupled with the same base SPH code; the generality, stability and accuracy of the semi-analytical framework has already been achieved without a performance drop.

This work opens the possibility for the semi-analytical formulation to finally become a widely available and easily accessible standard. With the method and source codes provided, all semi-analytical formulations become vastly easier and more natural to implement, since the correction terms $\gamma/\nabla \gamma$ in the governing equations can now generally be evaluated directly, cheaply, and exactly (i.e., can be treated as “generally known”). While there is still room for improvement on the semi-analytical formulation itself, including how the governing equations are formulated, in this article we have effectively solved the issue of calculation of $\gamma/\nabla \gamma$ which had previously been the main issue holding the method back. Additionally, our method can be completely insensitive to the boundary element shape and size. While some semi-analytical SPH formulations prefer to include a layer of boundary particles and rely on a fine mesh for accurate estimates of $\gamma/\nabla \gamma$, this is not strictly necessary. For example, the formulation of Kulasegaram is independent of the existence of boundary particles, and the method presented here can be used either with or without boundary particles. In the case where none are used, the improvements in speed of the method can be even greater, with no loss in accuracy beyond the accuracy of the initial SPH formulation. This would allow any mesh suitable for the geometry of interest to be used independently of the fluid resolution. Typical methods require the smallest boundary element to have an area greater than $dx \times dx$, but this need not be the case with the semi-analytical approach.

This article solves the general case of polynomial radial kernels and piecewise-planar boundaries comprising of arbitrary convex polygons. Polynomial kernels are by far the most common type used in practice and can be used as approximations of arbitrary (sufficiently well behaved) nonpolynomial alternatives, so limiting the investigation to this class of functions is not a major issue. However, a lot of work remains regarding the evaluation of correction terms near nonplanar boundaries. One of the main strengths of the semi-analytical formulation is that it applies to arbitrary boundaries as long as the corrections can be evaluated. This principle has been discussed and employed in our recent work with promising results but so far only the simplest nonplanar shape, a sphere, has been handled as a proof of concept. Further development of methods of evaluating semi-analytical corrections near nonplanar walls remains an active area of our research.

ACKNOWLEDGMENT
The authors thank Professor Stephen Neethling (Department of Earth Science & Engineering, Imperial College London) for providing the SPH code upon which the current methods were implemented.

DATA AVAILABILITY STATEMENT
The data that support the findings of this study are available from the corresponding author upon reasonable request.

ORCID
Wawrzyniec Kostorz https://orcid.org/0000-0003-4690-2475

REFERENCES
1. Macia F, Gonzalez LM, Cercos-Pita JL, Souto-Iglesias A. A boundary integral SPH formulation: consistency and applications to ISPH and WCSPH. Prog Theor Phys. 2012;128(3):439-462.
2. Chiron L, de Leffe M, Oger G, Le Touzé D. Fast and accurate SPH modelling of 3D complex wall boundaries in viscous and non viscous flows. Comput Phys Commun. 2019;234:93-111.
3. Calderon-Sanchez J, Cercos-Pita JL, Duque D. A geometric formulation of the Shepard renormalization factor. Comput Fluids. 2019;183:16-27.
4. Vacondio R, Altomare C, De Leffe M, et al. Grand challenges for smoothed particle hydrodynamics numerical schemes. Comput Part Mech. 2020. https://doi.org/10.1007/s40571-020-00354-1
5. Kulasegaram S, Bonet J, Lewis RW, Profit M. A variational formulation based contact algorithm for rigid boundaries in two-dimensional SPH applications. Comput Mech. 2004;33(4):316-325.
6. Feldman J, Bonet J. Dynamic refinement and boundary contact forces in SPH with applications in fluid flow problems. Int J Numer Methods Eng. 2007;72(3):295-324.
7. Ferrand M, Laurence DR, Rogers BD, Violeau D, Kassiotis C. Unified semi-analytical wall boundary conditions for inviscid, laminar or turbulent flows in the meshless SPH method. *Int J Numer Methods Fluids*. 2013;71(4):446-472.

8. Wendland H. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Adv Comput Math*. 1995;4(4):389-396.

9. Mayrhofer A, Ferrand M, Kassiotis C, Violeau D, Morel FX. Unified semi-analytical wall boundary conditions in SPH: analytical extension to 3-D. *Numer Algorithms*. 2015;68(1):15-34.

10. Violeau D, Leroy A, Mayrhofer A. Exact computation of SPH wall renormalising integrals in 3-D. Paper presented at: Proceeding of the 9th SPHERIC Workshop. Conservatoire National des Arts et Metiers (CNAM), Paris, 3-5 June; 2014.

11. Winchenbach R, Akhunov R, Kolb A. Semi-analytic boundary handling below particle resolution for smoothed particle hydrodynamics. *ACM Trans Graph*. 2020;39(6).

12. Kostorz W, Esmail-Yakas A. A semi-analytical boundary integral method for radial functions with application to smoothed particle hydrodynamics. *J Comput Phys*. 2020;417:109565.

13. Buhmann MD. *Radial Basis Functions: Theory and Implementations*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge, MA: Cambridge University Press; 2003.

14. Bouscasse B, Colagrossi A, Marrone S, Antuono M. Nonlinear water wave interaction with floating bodies in SPH. *J Fluids Struct*. 2013;42:112-129.

15. Kostorz W, Esmail-Yakas A. Semi-analytical treatment of spherical solids in smoothed-particle hydrodynamics fluid simulations. *Int J Numer Methods Eng*. 2020. https://doi.org/10.1002/nme.6597

**SUPPORTING INFORMATION**

Additional supporting information may be found online in the Supporting Information section at the end of this article.

---

**How to cite this article:** Kostorz W, Esmail-Yakas A. Semi-analytical smoothed-particle hydrodynamics correction factors for polynomial kernels and piecewise-planar boundaries. *Int J Numer Methods Eng*. 2021;122(24):7271-7305. https://doi.org/10.1002/nme.6771

---

**APPENDIX A. THREE-DIMENSIONAL MATLAB SPH ALGORITHM**

```matlab
function [gamma_j, nab_gamma_j, A_j] = gamma3_rec(x, nrm, x_a, R, a_n)

% Compute contributions gamma_j, nab_gamma_j, A_j of a planar boundary element to the SPH normalisation factors. The element corners oriented % anti-clockwise around the normal nrm are stored as columns of x. x_a is % the particle position. R is the particle radius and a_n is the array of % coefficients of W(r) = {SUM}_{j=1}^{deg+1} a_n(j)∗(r^(j-1))

% Integral loop
gamma_j = 0;
nab_gamma_j = 0;
A_j = 0;
lbd = dot(nrm, x(:,1)-x_a);

if R^2>lbd^2
    N_x = length(x(1,:)); % number of corners/edges of the face
    y1 = [1]; y2 = [1]; C = zeros(3,2*N_x); nC = 0;
    psi = 0; L0 = true;
    r_p = sqrt(R^2-lbd^2);
    for i=1:N_x
        % Cyclic index shift
        ip1 = i + 1;
        if ip1>N_x; ip1 = 1; end
        e = x(:,ip1)-x(:,i); e = e/norm(e);
        mew = dot(nrm, cross(x(:,i)-x_a,e));
        if mew<0; L0 = false; end
        if r_p^2>mew^2
            %
        end
    end
end
```

---
t1 = dot(e,x(:,i)-x_a);
t2 = dot(e,x(:,ip1)-x_a);
if (mew==0.0 & t1>t2<=0); error('Element edge'); end

l_max = sqrt(r_p^2-mew^2);
L1 = (t1 < -l_max);
L2 = (t2 > l_max);

t1 = max(-l_max,min(l_max,t1));
t2 = max(-l_max,min(l_max,t2));

if t2>t1
    [gj_i,nab_gj_i] = recur_int(t1,t2,mew,lbd,a_n);
    psi_i = A_arc(mew^2,t1,t2);
    if mew>0; psi_i = -psi_i; end
    gamma_j = gamma_j + gj_i;
    nab_gamma_j = nab_gamma_j + nab_gj_i;
    psi = psi + psi_i;
    if L1
        z = x(:,i) + (t1 - dot(e,x(:,i)-x_a))*e;
        if isempty(y1)
            A_j = A_j + A_s(y1,z,lbd,r_p,R,x_a,nrm);
        else
            y2 = z;
        end
        C(:,nC+1) = (z-x_a)/norm(z-x_a); nC = nC + 1;
    end
    if L2; y1 = z; end
end
end

if L0
    if psi==0; A_j = 2*A_d(r_p,lbd,r_p,R); end
    psi = psi + 2*pi;
end

gamma_j = gamma_j + lbd*psi*r_p^2*HW_3(r_p,lbd,a_n);
nab_gamma_j = nab_gamma_j + psi*r_p^2*HW_2(r_p,lbd,a_n);

if isempty(y2); A_j = A_j + A_s(y1,y2,lbd,r_p,R,x_a,nrm); end
if nC>=3; A_j = A_j + A_C(C,nC); end

nab_gamma_j = -nrm*nab_gamma_j; % multiply by the normal

%% Integration over straight lines

% Integrate over [t1,t2]
function [gji,nab_gji] = recur_int(t1,t2,mew,lbd,a_n)
    gji = 0;
    nab_gji = 0;
    if mew ~= 0
        if lbd == 0
            J1 = J_n_half(t1,mew^2,length(a_n)-1);
            J2 = J_n_half(t2,mew^2,length(a_n)-1);
        else
            J1 = I_n_half(t1/abs(lbd),mew^2/abs(lbd)^2,length(a_n)-1);
            J2 = I_n_half(t2/abs(lbd),mew^2/abs(lbd)^2,length(a_n)-1);
        end
        gji = mew*sum(a_n.((2:length(a_n)+1)).*J2); nab_gji = mew*sum(a_n.((2:length(a_n)+1)).*abs(lbd).^((1:length(a_n))^2).*(12-I1));
    else
        if lbd == 0
            J1 = J_n_half(t1/lbd,mew^2/abs(lbd)^2,length(a_n)-1);
            J2 = J_n_half(t2/lbd,mew^2/abs(lbd)^2,length(a_n)-1);
        else
            J1 = I_n_half(t1/abs(lbd),mew^2/abs(lbd)^2,length(a_n)-1);
            J2 = I_n_half(t2/abs(lbd),mew^2/abs(lbd)^2,length(a_n)-1);
        end
        gji = mew*sum(a_n.((2:length(a_n)+1)).*abs(lbd).^((1:length(a_n))^2).*(12-I1)); nab_gji = mew*sum(a_n.((2:length(a_n)+1)).*abs(lbd).^((1:length(a_n))^2).*(12-I1));
    end
end
end
% Generate a series of values of $J_{\frac{n}{2}}$
function $J = J_{\frac{n}{2}}(x, \text{mew2}, \text{deg})$
    $J = \text{zeros}(1, \text{deg}+1)$;
    $v0 = \sqrt{\text{mew2} + x^2}$;

% Odd terms
    $J(2) = 0.5 \times (x \times v0 + \text{mew2} \times \text{log}((\text{mew2} == 0) + x + v0))$;
    for $j = 2: \text{ceil}(\text{deg}/2)$;
        $J(2 \times j) = \frac{0.5}{j} \times (x \times (\text{mew2} + x^2)^{(j-0.5)} + \ldots$
        $(2 \times j-1) \times \text{mew2} @ J(2 \times j-2)$;
    end
% Even terms
    $J(1) = x$;
    for $j = 2: \text{floor}(\text{deg}/2)+1$;
        $J(2 \times j-1) = \frac{1}{2 \times j-1} \times (x \times (\text{mew2} + x^2)^{(j-1)} + 2 \times (j-1) \times \text{mew2} @ J(2 \times j-3))$;
    end
end

% Generate a series of values of $I_{\frac{n}{2}+1}$
function $I = I_{\frac{n}{2}+1}(x, \text{phi2}, \text{deg})$
    $I = \text{zeros}(1, \text{deg}+1)$;
    $v0 = \sqrt{1 + \text{phi2} + x^2}$;

% Odd terms
    $I(1) = \text{log}(x + v0)$;
    if $\text{phi2} == 0$
        $I(1) = I(1) - \frac{x}{1 + \sqrt{1 + x^2}}$;
    else
        $I(1) = I(1) - \frac{\text{atan}(\sqrt{\text{phi2}} \times x/\ldots$
            $(1 + \text{phi2} + \sqrt{1 + \text{phi2} + x^2})) / \sqrt{\text{phi2}}$;
    end
    $J = 0.5 \times (x + v0 + (1 + \text{phi2}) \times \text{log}(x + v0))$;
    for $j = 2: \text{ceil}(\text{deg}/2)+1$;
        $I(2 \times j-2) = I(\max(2 \times j-4, 1)) + J$;
        $J = \frac{0.5}{j} \times (x \times v0^{(2 \times j-1)} + 2 \times (j-1) \times (1 + \text{phi2}) \times J)$;
    end
% Even terms
    $I(1) = x$;
    $J = \frac{x^3}{3} + (1 + \text{phi2}) \times x$;
    for $j = 2: \text{floor}(\text{deg}/2)+1$;
        $I(2 \times j-1) = I(2 \times j-3) + J$;
        $J = 1/(2 \times j+1) \times (x \times v0^{(2 \times j)} + 2 \times j \times (1 + \text{phi2}) \times J)$;
    end
end

%% Integration over circular arcs

% Evaluate $H_W^{(3)}$ at a point
function $v = HW_3(r, lbd, a_n)$
    if $lbd == 0$
        $v = \text{sum}(a_n ./ ((2 : \text{length}(a_n)+1)) \text{.*}(3 : \text{length}(a_n)+2)) \text{.*}\ldots$
            $(r.^{(0 : \text{length}(a_n)-1))})$;
    else
        $kp = (r/lbd)^2$;
        $v = \text{sum}(a_n ./ ((2 : \text{length}(a_n)+1)) \text{.*}(3 : \text{length}(a_n)+2)) \text{.*}\ldots$
            $(\text{abs}(lbd)).^{(0 : \text{length}(a_n)-1))} \text{.*}\ldots$
            $((1+kp).^{(1+(0 : \text{length}(a_n)-1)/2)-1)))/\text{kp}$;
    end
end

% Evaluate $H_W^{(2)}$ at a point
function $v = HW_2(r, lbd, a_n)$
    if $lbd == 0$
        $v = \text{sum}(a_n ./ ((2 : \text{length}(a_n)+1)) \text{.*}(r.^{(0 : \text{length}(a_n)-1))}))$;
    else
        $kp = (r/lbd)^2$;
        $v = \text{sum}(a_n ./ ((2 : \text{length}(a_n)+1)) \text{.*}\ldots$
            $(\text{abs}(lbd)).^{(0 : \text{length}(a_n)-1))} \text{.*}\ldots$
            $((1+kp).^{(1+(0 : \text{length}(a_n)-1)/2)-1})))/\text{kp}$;
    end
end

%% Projections

% 3D arc length projection
function $A = A_{\text{arc}}(\text{mew2}, t1, t2)$
    $A = \text{acos} (\text{min}(1, \text{max}(-1, ...$
\((\text{mew2}+t1^2) / \sqrt{((\text{mew2}+t1^2) \times (\text{mew2}+t2^2))})\); 

end

% A(d), area of the projection of the smaller section
function A = A_d(d, lbd, r_p, R)
    A = pi;
    if lbd \sim 0 || d \leq r_p; A = A - 2 * (\text{acos}(\text{min}(1, d \times \text{abs}(lbd) / r_p)) / \text{sqrt}(R^2 - d^2)) + \text{abs}(lbd) / R * \text{asin}(\text{min}(1, d / r_p)); end

end

% A^s(), signed section area
function A = A_s(y1, y2, lbd, r_p, R, x_a, n)
    ss = \text{sign}(\text{dot}(n, \text{cross}(x_a - y1, y2 - y1)))
    A = (1 + ss) * A_d(r_p, lbd, r_p, R) ... - ss * A_d(\text{norm}(y1 - y2) / 2, lbd, r_p, R);

end

% A^C(), spherical area of a polygon
function A = A_C(C, nC)
    p = \text{max}(-1, \text{min}(1, \text{sum}(C(:, 1:nC) \times C(:, [2:nC, 1]), 1))))
    jm = [nC, 1:(nC - 1)]
    A = (2 - nC) * pi + \text{sum}(\text{acos}(\text{min}(1, \text{max}(-1, ... \text{sum}(C(:, [2:nC, 1]) \times C(:, jm), 1) - p \times p(jm)) / ... \text{max}(1e-14, \text{sqrt}((1 - p) \times (1 - p(jm))^2)))));

end

Example
It is assumed that the above code is contained in the file \texttt{gamma3_rec.m} on the current MATLAB path. A simple test program is given below. The results correspond to the example given in Figure 3, where

\[
\gamma_j = 0.17426, \quad V\gamma_j = [0, 0, -0.10540], \quad A_j = 2.28966
\]

% Element corners
x = \text{zeros}(3, 4);
x(:, 1) = [-0.2, 0.64, 0.6]';
x(:, 2) = [-1.6, -0.72, 0.6]';
x(:, 3) = [0.8, -0.72, 0.6]';
x(:, 4) = [1.6, 0.32, 0.6]';

nrm = [0, 0, 1]';
x_a = [0, 0, 0]';

% Wendland C_2 kernel
R = 1;
a_n = 21 / (2 * pi * R^3) *[1, 0, -10, 20, -15, 4]./(R.^(0:5));

[gamma_j, nab_gamma_j, A_j] = gamma3_rec(x, nrm, x_a, R, a_n);

fprintf('gamma_j = \%f\n', gamma_j);
fprintf('nab\_gamma_j = [\%f, \%f, \%f]\n', ...
    nab\_gamma\_j(1), nab\_gamma\_j(2), nab\_gamma\_j(3));
fprintf('A_j = \%f\n', A_j);

APPENDIX B. THREE-DIMENSIONAL C++ SPH ALGORITHM

#include <math.h>
#include <iostream>
#include <cstring>
using namespace std;

// VECTOR OPERATIONS
// Dot product
double vec_dot(double *a, double *b) {
    return a[0]*b[0] + a[1]*b[1] + a[2]*b[2];
}
// Cross product

void vec_cross(double *a, double *b, double *aCb) {
    for (int k = 0; k < 3; ++k) {
        int kp = (k + 1) % 3,
            km = (k + 2) % 3;
        aCb[k] = a[kp] * b[km] - a[km] * b[kp];
    }
}

// Difference between two vectors

void vec_diff(double *a, double *b, double *aMb) {
    for (int k = 0; k < 3; ++k) aMb[k] = a[k] - b[k];
}

// Normalised difference between two vectors

void vec_diff_norm(double *a, double *b, double *aMb_unit) {
    double norm2 = 0.0;
    for (int k = 0; k < 3; ++k) {
        aMb_unit[k] = a[k] - b[k];
        norm2 += pow(aMb_unit[k], 2);
    }
    norm2 = sqrt(norm2);
    for (int k = 0; k < 3; ++k) aMb_unit[k] /= norm2;
}

// Simple 3D AXPY

void vec_axpy(double a, double *x, double *y, double *aX_p_Y) {
    for (int k = 0; k < 3; ++k) aX_p_Y[k] = a * x[k] + y[k];
}

PROJECTIONS

// Projection of a segment described by mew, t1, t2 onto the unit circle

double A_arc(double mew2, double t1, double t2) {
    return acos(min(1.0, max(-1.0, (mew2 + t1 * t2) / sqrt((mew2 + pow(t1, 2)) * (mew2 + pow(t2, 2))))));
}

// Projection of the smaller section of the circle onto the unit sphere

double A_d(double d, double lbd, double r_p, double R) {
    double ans = M_PI;
    if (lbd != 0.0 || d < r_p) ans -= 2 * (acos(min(1.0, fabs(lbd) / (r_p * sqrt(pow(R, 2) - pow(d, 2)))))) + fabs(lbd) / R * asin(min(1.0, d / r_p));
    return ans;
}

// Projection of either the smaller or the larger section of the circle onto the unit sphere

double A_s(double *y1, double *y2, double lbd, double r_p, double R, double *x_a, double *nrm) {
    double xdif[3], ydif[3], xCy[3];
    vec_diff(x_a, y1, xdif);
    vec_diff(y2, y1, ydif);
    vec_cross(xdif, ydif, xCy);

    double ans = A_d(sqrt(pow(ydif[0], 2) + pow(ydif[1], 2) + pow(ydif[2], 2)) / 2.0, lbd, r_p, R);
    if (vec_dot(xCy, nrm) > 0) ans = 2.0 * A_d(r_p, lbd, r_p, R) - ans;
    return ans;
}

// Projection area of a polygon with nC sides onto the unit sphere

double A_C(double *C, int nC) {
    double* p; p = new double[nC];
    for (int j = 0; j < nC; ++j) p[j] = max(-1.0, min(1.0, vec_dot(C + 3 * j, C + 3 * ((j + 1) % nC))));

    double A = (2.0 - nC) * M_PI;
    for (int j = 0; j < nC; ++j) {
        int jm = (j + nC) % nC;
        A += acos(max(-1.0, min(1.0, vec_dot(C + 3 * ((j + 1) % nC), C + 3 * jm)) - p[j] * p[j] / max(1.0e-14, sqrt((1.0 - pow(p[j], 2)) * (1.0 - pow(p[jm], 2))))));
    }
    delete[] p; return A;
}
// ANALYTIC CIRCULAR ARCS

// Evaluate H_W^3 at a point
double H_W_3(double r, double lbd, double *a_n, int deg) {
    double ans = 0.0;
    if (lbd != 0.0) {
        double kap = pow(r,2)/pow(lbd,2);
        for (int j=0; j<=deg; ++j) ans += a_n[j]/((j+2)*(j+3))*pow(fabs(lbd),j)*(pow(1.0+kap, double (j)/2.0+1.0)-1.0);
    } else for (int j=0; j<=deg; ++j) ans += a_n[j]/((j+2)*(j+3))*pow(r,j);
    return ans;
}

// Evaluate H_W^2 at a point
double H_W_2(double r, double lbd, double *a_n, int deg) {
    double ans = 0.0;
    if (lbd != 0.0) {
        double kap = pow(r,2)/pow(lbd,2);
        for (int j=0; j<=deg; ++j) ans += a_n[j]/(j+2)*pow(fabs(lbd),j)*(pow(1.0+kap, double (j)/2.0+1.0)-1.0);
    } else for (int j=0; j<=deg; ++j) ans += a_n[j]/(j+2)*pow(r,j);
    return ans;
}

// RECURSIVE STRAIGHT EDGES

// Recursively generate a series of values I_{n/2+1} for n = 0,...,deg
void I_n_rec( double *I, double x, double phi2, int deg) {
    double v_loc = sqrt(1.0+pow(x,2)+phi2), J;
    // Odd terms
    J = 0.5*(x*v_loc + (1+phi2)*log(x+v_loc)); I[0] = log(x+v_loc);
    if (phi2>0.0) I[0] -= atan(sqrt(phi2)*x/(1+phi2+v_loc))/sqrt(phi2);
    else I[0] -= x/(1+v_loc);
    for (int j=1; j<=int (ceil(deg/2.0)); ++j) {
        I[2*j-1] = I[max(2*j-3,0)] + J;
        J = 0.5/(j+1.0)*(x*pow(v_loc,2*j+1) + (2*j+1)*(1+phi2)*J);
    }
    // Even terms
    J = pow(x,3)/3.0 + (1.0+phi2)*x; I[0] = x;
    for (int j=1; j<=int (floor(deg/2.0)); ++j) {
        I[2*j] = I[2*j-2] + J;
        J = 0.5/(j+1.5)*(x*pow(v_loc,2*j+2) + 2*(j+1)*(1+phi2)*J);
    }
}

// Recursively generate a series of values J_{n/2} for n = 0,1,...,deg
void J_n_rec( double *J, double x, double mew2, int deg) {
    double v_loc = sqrt(pow(x,2)+mew2);
    // Odd terms
    J[1] = 0.5*(x*v_loc); if (mew2>0.0) J[1] += 0.5*mew2*log(x+v_loc);
    for(int j=1; j<=int (ceil(deg/2.0)); ++j) J[2*j+1] = 0.5/(j+1.0)* (x*pow(v_loc,2*j+1) + (2*j+1)*mew2*J[2*j-1]);
    // Even terms
    J[0] = x;
    for(int j=1; j<=int (floor(deg/2.0)); ++j) J[2*j] = 0.5/(j+0.5)* (x*pow(v_loc,2*j) + 2.0*j*mew2*J[2*j-2]);
}

// Integrate contributions to gamma_j/nabla_gamma_j along a straight
// edge characterised by t1,t2,mew,lbd and with the kernel function
// given by W(r) = SUM_{n=0}^deg (a_n[r^n]
void recur_int(double &g_ji,double &nabg_ji,double &t1, double &t2, double mew,double lbd,double &a_n,int deg) {
    g_ji = 0.0;
    nabg_ji = 0.0;
    if (mew != 0.0) {
        double *I1, *I2;
        I1 = new double[deg+1];
        I2 = new double[deg + 1];
        if (lbd == 0.0) {
double mew2 = pow(mew,2);
J_n_rec(I1,t1,mew2,deg);
J_n_rec(I2,t2,mew2,deg);
for(int j = 0; j<=deg; ++j) nabg_ji += a_n[j]/(j+2)*
(12[j]−11[j]); }
else {
double phi2 = pow(mew,2)/pow(lbd,2);
I_n_rec(I1,t1/fabs(lbd),phi2,deg);
I_n_rec(I2,t2/fabs(lbd),phi2,deg);
for(int j = 0; j<=deg; ++j) {
double dI = a_n[j]/(j+2)*
pow(fabs(lbd),j+1)*
(pow(fabs(lbd),j+1)/pow(fabs(lbd),j+1));
g_ji += dI/(j+3);
nabg_ji += dI; }
g_ji *= mew;lbd; delete[] I1; delete[] I2; }
}

// SEMI–ANALYTICAL INTEGRAL

// Compute contributions gamma_j, nab_gamma_j, A_j of a planar
// boundary element to the SPH normalisation factors. The N_x corners
// oriented anti–clockwise around the normal nrm are stored in x as
// x = [x^1(1)_x, x^1(1)_y, x^1(1)_z, x^1(2)_x, x^1(2)_y, ... , x^N(N_x)_z]
// x_a is the particle position, R is the particle radius and a_n is
// the array of coefficients of W(r) = {SUM}_{j=0}^deg a_n[j]*(r^j)

int gamma3_rec( double *x, int N_x, double *nrm,
double *x_a, double R, double *a_n, int deg,
double & gamma_j, double *nab_gamma_j, double & A_j) {

gamma_j = 0.0;
A_j = 0.0;
double v_1[3], lbd, nab_coef = 0.0;
vec_diff(x,x_a,v_1); lbd = vec_dot(nrm,v_1);
if (R>fabs(lbd)) {
double y1[3],y2[3],
C = new double[6*N_x];
in nC = 0;
bool fill_y1 = false , fill_y2 = false , L_0 = true;
double psi = 0.0, r_p = sqrt(pow(R,2)
−pow(lbd,2));
for(int i = 0; i<N_x; ++i) {
int ip1 = (i+1)%N_x;
double mew,e[3],v_2[3];
vec_diff_norm(x+3*i,x+3*i,e);
vec_diff(x+3*i,x_a,v_1); vec_cross(v_1,e,v_2);
mew = vec_dot(v_2,nrm);
if (mew<0) L_0 = false;
if (r_p>fabs(mew)) {
vec_diff(x+3*i,x_a,v_2);

double t1 = vec_dot(e,v_1),
t2 = vec_dot(e,v_2),
t_uncut = t1; // store for later
if (mew==0.0 && t1*t2<0.0) return 1; // edge detection
bool L_1 = t1 < −l_max, L_2 = t2 > l_max;
t1 = max(−l_max,min(l_max,t1));
t2 = max(−l_max,min(l_max,t2));
if (t2>t1){
double g_ji,nabg_ji;
recur_int(g_ji, nabg_ji ,t1,t2,mew,lbd,a_n,deg);
double z[3], psi_i = A_arc(pow(mew,2),t1,t2);
if (mew>=0.0) psi_i *= −1.0;
}
gamma_j += g_ji;  
nab_coef += nabg_ji;  
psi += psi_i;  
if (L_1) { 
  vec_axpy(t1−t_uncut,e,x+3∗i,z);  
  if (fill_y1) { 
    A_j += A_s(y1,z,lbd,r_p,x_a,nrm);  
  } else { 
    memcpy(y2,z,3*sizeof(double));  
    fill_y2 = true;  
  } 
  vec_diff_norm(z,x_a,C+3∗nC); nC += 1; }  
vec_axpy(t2−t_uncut,e,x+3∗i,z);  
vec_diff_norm(z,x_a,C+3∗nC); nC += 1;  
if (L_2) { 
  memcpy(y1,z,3*sizeof(double));  
  fill_y1 = true; }  
if (L_0) { 
  if (psi==0.0) A_j = 2.0∗A_d(r_p,lbd,r_p,R);  
  psi += 2.0∗M_PI; }  
gamma_j += lbd∗psi∗pow(r_p,2)∗H_W_3(r_p,lbd,a_n,deg);  
nab_coef += psi∗pow(r_p,2)∗H_W_2(r_p,lbd,a_n,deg);  
if (fill_y2) A_j += A_s(y1,y2,lbd,r_p,R,x_a,nrm);  
if (nC>=3) A_j += A_C(C,nC);  
delete[] C; }  
for(int k = 0; k<3; ++k) nab_gamma_j[k] = −nrm[k]∗nab_coef;  
return 0; }  
//  
Example  
It is assumed that the above code is contained in the file gamma3_rec.cpp on the current path. A simple test program is given below. The results correspond to the example given in Figure 3, where  
\[ \gamma_j = 0.17426, \]  
\[ \nabla \gamma_j = [0, 0, −0.10540], \]  
\[ A_j = 2.28966 \]  
#include "gamma3_rec.cpp"  
int main() {  
  int N_x = 4;  // Quadrilateral element  
  double x[12] = {−0.2,0.64,0.6,  
                  −1.6,−0.72,0.6,  
                  0.8,−0.72,0.6,  
                  1.6,0.32,0.6},  // x_1, x_2, x_3, x_4  
  nrm[3] = [0.0,0.0,1.0],  // normal  
  x_a[3] = [0.0,0.0,0.0];  // particle position  
  const int deg = 5;  // Quintic C_2 Wendland kernel  
  double a_n[deg+1] = {1.0,0.0,−10.0,20.0,−15.0,4.0}, R = 1.0;  
  for(int n = 0; n <= deg; ++n) a_n[n] *= 21.0/(2.0*M_PI*pow(R,3+n));  
  double gamma_j, nab_gamma_j[3], A_j;  
  int err_ID =  
    gamma3_rec(x,N_x,nrm,x_a,R,a_n,deg,gamma_j,nab_gamma_j,A_j);  
  cout << "gamma_j = " << gamma_j << "\n";  
  cout << "nab_gamma_j = ["  
       << nab_gamma_j[0] << ","  
       << nab_gamma_j[1] << ","  
       << nab_gamma_j[2] << "]\n";  
  cout << "A_j = ["  
       << A_j << "]\n";  
  cout << "Exit status: " << err_ID << "\n"; }
APPENDIX C. THREE-DIMENSIONAL PYTHON SPH ALGORITHM

from numpy import array, zeros, dot, cross, multiply, divide, sign, log, sum
from numpy.linalg import norm
from math import sqrt, pi, asin, acos, atan, floor, ceil

# Projections

# Length of the shadow of the segment characterised by mew, t1, t2
def A_arc(mew2, t1, t2):
    return acos(min(1.0, max(-1.0, (mew2+t1) / sqrt((mew2+t1)**2*(mew2+t2)**2)) ))

# Projection area of the smaller section of the circle in plane P
def A_d(d, lbd, r_p, R):
    A = pi if lbd != 0.0 or d < r_p:
        A = 2*(acos(min(1.0, d*abs(lbd)/ (r_p*sqrt(R**2-d**2)))) + abs(lbd)/R * asin(min(1.0, d/r_p)) )
    return A

# Oriented area of either the smaller or the larger section
def A_s(y1, y2, lbd, r_p, R, xa, nrm):
    ss = sign(dot(nrm, cross(xa-y1, y2-y1)))
    return (1.0 + ss) * A_d(r_p, lbd, r_p, R) - ss * A_d(norm(y1-y2)/2, lbd, r_p, R)

# Spherical area of a convex polygon
def A_C(C, nC):
    p = zeros(nC);
    for j in range(nC):
        p[j] = max(-1, min(1, dot(C[:,j], C[:,(j+1)%nC])))
    ans = (2-nC)*pi
    for j in range(nC):
        jm = (j-1+nC)%nC
        ans += acos(max(-1, min(1, (dot(C[:,jm], C[:,(j+1)%nC]) - p[j]*p[jm]) / (1-p[j]**2)*(1-p[jm]**2))))
        ans /= (1.0+kap)**(j/2.0+1.0) - 1.0
    return ans

# Evaluate H_W^3 at a point
def H_W_3(r, lbd, a_n, deg):
    ans = 0.0
    if lbd != 0.0:
        kap = r**2/lbd**2
        for j in range(deg+1):
            ans += a_n[j]/((j+2)**(j+3) * (abs(lbd)**j)*((r/lbd)**(j/2.0+1.0) - 1.0)
            ans /= kap
    else:
        for j in range(deg+1):
            ans += a_n[j]/((j+2)**(j+3))*r**j
    return ans

# Evaluate H_W^2 at a point
def H_W_2(r, lbd, a_n, deg):
    ans = 0.0
    if lbd != 0.0:
        kap = r**2/lbd**2
        for j in range(deg+1):
            ans += a_n[j]/((j+2)**(j+3) * ((1.0+kap)**(j/2.0+1.0) - 1.0)
            ans /= kap
    else:
        for j in range(deg+1):
            ans += a_n[j]/((j+2)**(j+3))*r**j
    return ans

# Evaluate L_n[2+1] for n = 0,1,....,deg
I = zeros(deg+1)
v_loc = sqrt(1 + x**2 + phi2)
# Odd terms
J = 0.5*(x*v_loc + (1.0+phi2)*log(x+v_loc)); I[0] = log(x + v_loc)
if phi2 > 0.0: I[0] = atan(sqrt(phi2)*x/(1+phi2+v_loc)) / sqrt(phi2)
else: I[0] = x/(1.0+v_loc)
for j in range(1,1 + int(ceil(deg/2.0))):
\[I[2j-1] = \max(2j-3,0)] + J \]
\[J = 0.5/(j+1.0)\times (xv_{loc}**2j+1) + (2j+1)\times (1.0+\phi2)\times J\]

**Even terms**
\[J = (x**3)/3.0 + (1.0+\phi2)\times x; I[0] = x\]

**Odd terms**
\[J[2j+1] = 0.5/(j+1.5)\times (xv_{loc}**2j+2) + 2.0/(j+1)\times (1.0+\phi2)\times J\]

```python
# Even terms
J = (x**3)/3.0 + (1.0+phi2)*x; I[0] = x
for j in range(1,1 + int(floor(deg/2.0))):
    I[2j] = I[2j-2] + J
    J = 0.5/(j+1.0)\times (xv_{loc}**2j+1) + (2j+1)\times (1.0+phi2)\times J
return I
```

```python
# Odd terms
J[1] = 0.5\times xv_{loc}
if mew2>0.0: J[1] += 0.5\times mew2*\log(xv_{loc})
for j in range(1,int(ceil(deg/2.0))):
    J[2j+1] = 0.5/(j+1.0)\times (xv_{loc}**2j+2) + 2.0\times (j+1)\times (1.0+phi2)\times J
return J
```

```python
# Recursively integrate the polynomial kernel over [t1,t2]
g_{ji} = 0.0
nabg_{ji} = 0.0
if mew != 0.0:
    if lbd == 0.0:
        I1 = J_n_rec(t1,mew**2,deg)
        I2 = J_n_rec(t2,mew**2,deg)
        for j in range(deg+1): nabg_{ji} += a_n[j]/(j+2)\times (I2[j] - I1[j])
    else:
        phi2 = (mew**2)/lbd**2
        I1 = J_n_rec(t1/abs(lbd),phi2,deg)
        I2 = J_n_rec(t2/abs(lbd),phi2,deg)
        for j in range(deg+1):
            dI = a_n[j]/(j+2)\times abs(lbd)\times (I2[j] - I1[j])
            g_{ji} += dI/(j+3)
            nabg_{ji} += dI
        g_{ji} *= mew*lbd
        nabg_{ji} *= mew
    return (g_{ji},nabg_{ji})
```
if $r_p > \text{abs}(\text{mew})$:
    $t_1 = \text{dot}(e,x[:,i]-xa)$
    $t_2 = \text{dot}(e,x[:,ip1]-xa)$
if (\text{mew==0.0 and } t_1 \times t_2 <= 0): \text{return } (0.0,0.0,0.0,1)$

$l_{\text{max}} = \sqrt{(r_p^2 - \text{mew}^2)}$
$L_1 = (t_1 < -l_{\text{max}})$
$L_2 = (t_2 > l_{\text{max}})$

$t_1 = \text{max}(-l_{\text{max}}, \text{min}(l_{\text{max}}, t_1))$
$t_2 = \text{max}(-l_{\text{max}}, \text{min}(l_{\text{max}}, t_2))$

if $t_2 > t_1$:
    \[GG_{ji} = \text{recur\_int}(t_1,t_2,\text{mew},lbd,a_n,deg)\]
    \[\psi_i = A_{\text{arc}}(\text{mew}^2,t_1,t_2)\]
    if \text{mew>0.0}: \psi_i *= -1
    \[\gamma_j += GG_{ji}[0]\]
    \[\text{nab\_coef} += GG_{ji}[1]\]
    \[\psi += \psi_i\]

if $L_1$:
    \[z = x[:,i] + (t_1 - \text{dot}(e,x[:,i]-xa))\times e\]
    if \text{y1 is not None}:
        \[A_j += A_{s}(y1,z,lbd,r_p,xa,nrm)\]
    else:
        \[y_2 = z\]
    \[C[:,nC] = (z-\text{xa})/\text{norm}(z-\text{xa}); nC += 1\]
    \[z = x[:,i] + (t_2 - \text{dot}(e,x[:,i]-xa))\times e\]
    \[C[:,nC] = (z-\text{xa})/\text{norm}(z-\text{xa}); nC += 1\]
    if $L_2$: \[y_1 = z\]

if $L_0$:
    if \text{psi == 0.0}: \[A_j = 2*A_d(r_p,lbd,r_p,R)\]
    \[\psi += 2*\pi\]
    \[\gamma_j += lbd*\text{psi}*(r_p^2)*H_{W3}(r_p,lbd,a_n,deg)\]
    \[\text{nab\_coef} += \text{psi}*(r_p^2)*H_{W2}(r_p,lbd,a_n,deg)\]
if \text{y2 is not None}:
    \[A_j += A_{s}(y1,y2,lbd,r_p,xa,nrm)\]
if \text{nC>3}:
    \[A_j += A_{C}(C,nC)\]

\[\gamma_j,\text{nab\_coef},\text{norm},A_j,0\]

---

**Example**

It is assumed that the above code is contained in the file `gamma3_rec_module.py` on the current Python path. A simple test program is given below. The results correspond to the example given in Figure 3, where

$$\gamma_j = 0.17426,$$
$$\nabla \gamma_j = [0,0,-0.10540],$$
$$A_j = 2.28966$$

```python
from gamma3_rec_module import gamma3_rec
from numpy import zeros, array
from math import pi

N_x = 4
x = zeros((3,N_x))
x[:,0] = array([-0.2,0.64,0.6])
x[:,1] = array([-1.6,-0.72,0.6])
x[:,2] = array([0.8,-0.72,0.6])
x[:,3] = array([1.6,0.32,0.6])
nrm = array([0,0,1.0])
xa = array([0.0,0.0,0.0])
deg = 5
a_n = array([1.0,0.0,-10.0,20.0,-15.0,4.0])
```

from gamma3_rec_module import gamma3_rec
from numpy import zeros, array
from math import pi

N_x = 4
x = zeros((3,N_x))
x[:,0] = array([-0.2,0.64,0.6])
x[:,1] = array([-1.6,-0.72,0.6])
x[:,2] = array([0.8,-0.72,0.6])
x[:,3] = array([1.6,0.32,0.6])
nrm = array([0,0,1.0])
xa = array([0.0,0.0,0.0])
deg = 5
a_n = array([1.0,0.0,-10.0,20.0,-15.0,4.0])

# Example

It is assumed that the above code is contained in the file `gamma3_rec_module.py` on the current Python path. A simple test program is given below. The results correspond to the example given in Figure 3, where

$$\gamma_j = 0.17426,$$
$$\nabla \gamma_j = [0,0,-0.10540],$$
$$A_j = 2.28966$$

```python
from gamma3_rec_module import gamma3_rec
from numpy import zeros, array
from math import pi

N_x = 4
x = zeros((3,N_x))
x[:,0] = array([-0.2,0.64,0.6])
x[:,1] = array([-1.6,-0.72,0.6])
x[:,2] = array([0.8,-0.72,0.6])
x[:,3] = array([1.6,0.32,0.6])
nrm = array([0,0,1.0])
xa = array([0.0,0.0,0.0])
deg = 5
a_n = array([1.0,0.0,-10.0,20.0,-15.0,4.0])
```
R = 1.0
for n in range(deg+1): a_n[n] = 21/(2*pi*(R**(3+n)))

ans = gamma3_rec(x,N_x,nrm,xa,R,a_n,deg)

print("gamma_j = ",ans[0])
print("nab_gamma_j = ",ans[1])
print("A_j = ",ans[2])
print("Exit status: ",ans[3])