# Solid Angle of the Off-Axis Circle Sector

Richard J. Mathar*
*Hoeschstr. 7, 52372 Kreuzau, Germany*
(Dated: May 6, 2024)

The solid angle of a circular sector specified by circle radius, angle of the sector, and distance of the circle plane to the observer is calculated in terms of inverse trigonometric functions and elliptic integrals of the third kind. This generalizes previous results for the full circle that have appeared in the literature. [viXra:1403.0977]

PACS numbers: 02.30.Cj, 02.30.Gp, 07.77.-n

## I. GEOMETRY

The solid angle covered by a circle at general position relative to the observer has been discussed in the literature [1–8]. The principal generic application is the calibration of particle detectors of circular shape that receive isotropic radiation from sources which are far reaching—photonic or closer than the Bragg peak to a particle detector. The following calculation generalizes the case of the full circle to the case of a circular sector, characterized by the interior angle $\alpha$, radius $R$ and planar area $\alpha R^2/2$ of the section.

The solid angle of a circular segment might be calculated by subtraction of the solid angles of the associated circular sector and of the common triangle of both regions. This solid angle of the triangle is published [9–11]; so this work also assists to the calculation of the circular sections.

The main variables of the geometry are characterized in Figure 1:

- The shortest (perpendicular) distance $h > 0$ of the observer to the plane of the circle.
- The radius $R > 0$ of the circle.
- The interior angle $\alpha$ of the sector, in the range $0 \le \alpha \le \pi$.
- The distance $D \ge 0$ between the location of the observer perpendicularly projected onto the plane of the circle and the circle center. We fasten the origin of the plane of the circle to that foot point of the observer's position.

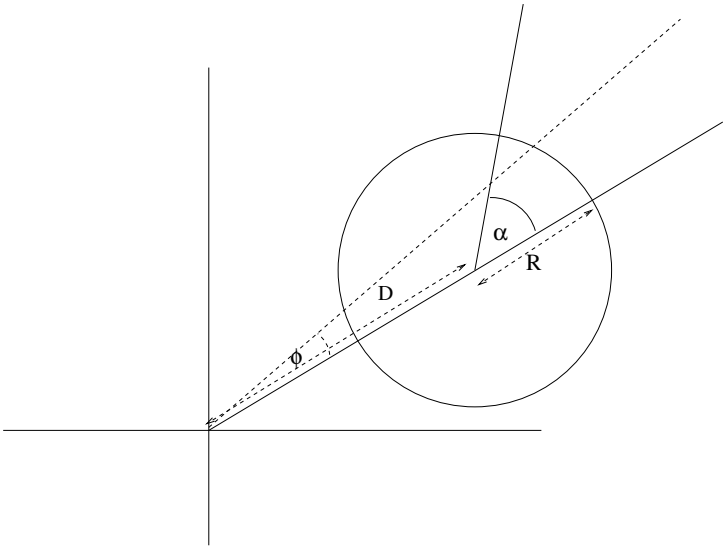


FIG. 1. The circle of radius $R$ at distance $D$, its sector of angle $\alpha$, and the azimuthal angle $\phi$ of the spherical coordinates seen by the observer looking into a direction perpendicular to the circle's plane where the two perpendicular lines cross.

* https://www.mpia-hd.mpg.de/~mathar

We calculate only the case where the direction from the (projected) observer's position to the circle center is aligned with one of the straight edges of the sector, i.e., the reference angle $\alpha = 0$ in Fig. 1 is defined by the direction of the dashed arrow of length $D$. In general, the sector will be rotated with respect to this aligned geometry. The solid angle is then obtained by subtraction of the two solid angles of two circular sectors that both have a common edge aligned with the viewing direction and a difference or sum of their interior angles equaling the actual interior angle of the desired sector. In the language of analysis this means we calculate a first integral over the angle $\alpha$ with lower limit at zero, defined by the line of intersection between (i) the plane of the circle and (ii) the orthogonal plane which contains the observer and the center of the circle. The other integrals are then differences between these first integrals.

## II. TRANSFORMATION TO OFF-CENTER CIRCLE COORDINATES

In a spherical coordinate system centered at the observer, with polar angle $\theta$ ($0 \leq \theta \leq \pi$) and azimuth angle $\phi$ ($0 \leq \phi \leq 2\pi$), the solid angle of the object is

$$\Omega = \int \sin\theta \, d\theta \, d\phi, \tag{1}$$

where the two angular coordinates scan the surface of the object. We transform Cartesian coordinates to spherical coordinates and perform the double integral, using the generic

$$x = r \sin\theta \cos\phi, \tag{2}$$
$$y = r \sin\theta \sin\phi, \tag{3}$$
$$z = r \cos\theta, \tag{4}$$

and the inverse transformation

$$\phi = \arctan\frac{y}{x}, \quad \theta = \arccos\frac{z}{\sqrt{x^2+y^2+z^2}}. \tag{5}$$

If the coordinate system is oriented such that the azimuth angle $\phi$ starts at zero in the direction of the circle center seen from the projected observer's position (Figure 1), the equation of the circle is

$$(x-D)^2 + y^2 = R^2; \quad z = h. \tag{6}$$

Because $z$ is fixed inside the circle plane, so $r = h/\cos\theta$, the relevant 2D coordinates for the computation of the solid angle in (2) and (3) are

$$x = h \tan\theta \cos\phi; \quad y = h \tan\theta \sin\phi. \tag{7}$$

The same point $(x, y)$ is characterized in a a circular coordinate system of radial $\rho$ and azimuthal $\gamma$ centered at the circle center as

$$x - D = \rho \cos\gamma; \quad y = \rho \sin\gamma. \tag{8}$$

Eliminating $x$ and $y$ from the previous equations, the transformation between the $(\theta, \phi)$ and the $(\rho, \gamma)$ system is

$$h \tan\theta \cos\phi - D = \rho \cos\gamma; \tag{9}$$
$$h \tan\theta \sin\phi = \rho \sin\gamma. \tag{10}$$

Adding the squares and building the ratio of these two equations yields the transformation from $(\theta, \phi)$ to $(\rho, \gamma)$ coordinates:

$$\rho = \sqrt{(h\tan\theta\cos\phi - D)^2 + (h\tan\theta\sin\phi)^2}; \quad \tan\gamma = \frac{h\tan\theta\sin\phi}{h\tan\theta\cos\phi - D} \tag{11}$$

and

$$h^2\tan^2\theta = (D + \rho\cos\gamma)^2 + (\rho\sin\gamma)^2 = D^2 + 2D\rho\cos\gamma + \rho^2 > 0, \tag{12}$$

so the transformation from $(\rho, \gamma)$ to $(\theta, \phi)$ coordinates is

$$\theta = \arctan\frac{\sqrt{D^2 + 2D\rho\cos\gamma + \rho^2}}{h}; \quad \phi = \arctan\frac{\rho\sin\gamma}{D + \rho\cos\gamma}. \tag{13}$$

The Jacobi determinant is derived from the four partial derivatives of these representations of $\theta$ and $\phi$:

$$\left|\begin{array}{cc}\partial\theta/\partial\rho & \partial\theta/\partial\gamma\\ \partial\phi/\partial\rho & \partial\phi/\partial\gamma\end{array}\right| = h\rho\frac{1}{\sqrt{D^2+2D\rho\cos\gamma+\rho^2}(h^2+D^2+2D\rho\cos\gamma+\rho^2)}. \tag{14}$$

Inspecting (12), all factors in this equation are positive. Eq. (1) transforms into $(\rho,\gamma)$ coordinates,

$$\Omega = \int\sin\theta d\theta\int d\phi = \int_0^R d\rho\int_0^\alpha d\gamma\left|\begin{array}{cc}\partial\theta/\partial\rho & \partial\theta/\partial\gamma\\ \partial\phi/\partial\rho & \partial\phi/\partial\gamma\end{array}\right|\sin\theta. \tag{15}$$

Substitution of (14) as well as the sine-factor [12, (4.3.45)]

$$\sin\theta = \frac{\tan\theta}{\sqrt{1+\tan^2\theta}} = \frac{\frac{\sqrt{D^2+2D\rho\cos\gamma+\rho^2}}{h}}{\sqrt{1+\frac{D^2+2D\rho\cos\gamma+\rho^2}{h^2}}} = \frac{\sqrt{D^2+2D\rho\cos\gamma+\rho^2}}{\sqrt{h^2+D^2+2D\rho\cos\gamma+\rho^2}}, \tag{16}$$

cancels a square root [6, 13]:

$$\Omega = \int_0^R d\rho\int_0^\alpha d\gamma\frac{h\rho}{(h^2+D^2+2D\rho\cos\gamma+\rho^2)^{3/2}}. \tag{17}$$

We may define the two unitless positive variables

$$\delta\equiv D/h\geq 0,\quad \beta\equiv R/h>0. \tag{18}$$

As expected from a homogeneous scaling of all lengths, the solid angle depends only on these length ratios:

$$\Omega(\beta,\delta,\alpha) = \beta^2\int_0^1 d\xi\int_0^\alpha d\gamma\frac{\xi}{(1+\delta^2+2\delta\beta\xi\cos\gamma+\beta^2\xi^2)^{3/2}}. \tag{19}$$

## III. RADIAL INTEGRAL

Up to here, the solid angle has merely been rephrased in an off-center circular coordinate system of the planar target. The following calculation works out the double integral supposing that the target shape is a circular sector. To eliminate roots in the denominator of (19), an Euler substitution [14, 2.251] is engaged:

$$\sqrt{1+\delta^2+2\delta\beta\xi\cos\gamma+\beta^2\xi^2} = t+\xi\beta, \tag{20}$$

which is a quadratic equation for $t(\xi)$

$$1+\delta^2+2\delta\beta\xi\cos\gamma = t^2+2t\xi\beta, \tag{21}$$

and a linear equation for $\xi(t)$

$$\xi = \frac{t^2-1-\delta^2}{2\beta(\delta\cos\gamma-t)}. \tag{22}$$

The proper branch of the solution of the quadratic equation for $t$ follows from (20):

$$t = -\beta\xi+\sqrt{\beta^2\xi^2+1+\delta^2+2\delta\beta\cos\gamma\xi}. \tag{23}$$

The differential in the $\xi$-integral is replaced via the derivative of (22):

$$\frac{d\xi}{dt} = \frac{2t}{2\beta(\delta\cos\gamma-t)}+\frac{t^2-1-\delta^2}{2\beta(\delta\cos\gamma-t)^2}. \tag{24}$$

Executing the Euler substitution in (19)

$$\begin{aligned}\Omega(\beta,\delta,\alpha) &= \beta^2\int_{\sqrt{1+\delta^2}}^{-\beta+\sqrt{\beta^2+1+\delta^2+2\delta\beta\cos\gamma}}dt\int_0^\alpha d\gamma\frac{t^2-1-\delta^2}{2\beta(\delta\cos\gamma-t)}\frac{1}{(t+\xi\beta)^3}\Big[\frac{2t}{2\beta(\delta\cos\gamma-t)}+\frac{t^2-1-\delta^2}{2\beta(\delta\cos\gamma-t)^2}\Big]\\ &= 2\int_0^\alpha d\gamma\int_{-\beta+\sqrt{\beta^2+1+\delta^2+2\delta\beta\cos\gamma}}^{\sqrt{1+\delta^2}}dt\frac{1+\delta^2-t^2}{(-2\delta\cos\gamma t+t^2+1+\delta^2)^2}.\end{aligned} \tag{25}$$

Evaluation of the $t$-integral in Appendix B gives

$$\Omega(\beta,\delta,\alpha) = 2\int_0^\alpha d\gamma \frac{t}{-2\delta\cos\gamma t + t^2 + 1 + \delta^2}\,|_{-\beta+\sqrt{\beta^2+1+\delta^2+2\delta\beta\cos\gamma}}^{\sqrt{1+\delta^2}}$$
$$= 2\int_0^\alpha d\gamma\Big[\frac{1}{2}\frac{1}{\sqrt{1+\delta^2}-\delta\cos\gamma}$$
$$-\frac{-\beta+\sqrt{1+\delta^2+\beta^2+2\delta\cos\gamma\beta}}{2+2\delta^2+2\beta^2+2\delta\cos\gamma\beta-2\beta\sqrt{1+\delta^2+\beta^2+2\delta\cos\gamma\beta}-2\delta\cos\gamma[-\beta+\sqrt{1+\delta^2+\beta^2+2\delta\cos\gamma\beta}]}\Big]$$
$$= \int_0^\alpha d\gamma\left[\frac{1}{\sqrt{1+\delta^2}-\delta\cos\gamma} - \frac{-\beta+\sqrt{1+\delta^2+\beta^2+2\delta\cos\gamma\beta}}{1+\delta^2+\beta^2+2\delta\cos\gamma\beta-(\beta+\delta\cos\gamma)\sqrt{1+\delta^2+\beta^2+2\delta\cos\gamma\beta}}\right]. \quad (26)$$

The kernel is an even function of $\gamma$, so

$$\Omega(\beta,\delta,-\alpha) = -\Omega(\beta,\delta,\alpha). \quad (27)$$

## IV. INTEGRATING OVER THE AZIMUTH

The first term in the previous equation is a tabulated standard [14, 2.553.4]

$$\Omega_1 = \int_0^\alpha d\gamma\frac{1}{\sqrt{1+\delta^2}-\delta\cos\gamma} = 2\,\mathrm{Arctan}\,\frac{\tan\frac{\alpha}{2}}{\sqrt{1+\delta^2}-\delta}. \quad (28)$$

The notation Arctan indicates that the same branch of the inverse function is to be taken as picked up by $\alpha/2$.

In the second term, the denominator is made square-root-free by multiplying numerator and denominator by its conjugate:

$$\frac{-\beta+\sqrt{1+\delta^2+\beta^2+2\delta\cos\gamma\beta}}{1+\delta^2+\beta^2+2\delta\cos\gamma\beta-(\beta+\delta\cos\gamma)\sqrt{1+\delta^2+\beta^2+2\delta\cos\gamma\beta}}$$
$$= \frac{(-\beta+\sqrt{1+\delta^2+\beta^2+2\delta\cos\gamma\beta})[1+\delta^2+\beta^2+2\delta\cos\gamma\beta+(\beta+\delta\cos\gamma)\sqrt{1+\delta^2+\beta^2+2\delta\cos\gamma\beta}]}{(1+\delta^2+\beta^2+2\delta\cos\gamma\beta)^2-(\beta+\delta\cos\gamma)^2(1+\delta^2+\beta^2+2\delta\cos\gamma\beta)}$$
$$= \frac{\delta\cos\gamma(1+\delta^2+\beta^2+2\delta\cos\gamma\beta)+[1+\delta^2+\delta\cos\gamma\beta]\sqrt{1+\delta^2+\beta^2+2\delta\cos\gamma\beta}}{(1+\delta^2+\beta^2+2\delta\cos\gamma\beta)[1+\delta^2\sin^2\gamma]}$$
$$= \frac{\delta\cos\gamma}{1+\delta^2\sin^2\gamma} + \frac{1+\delta^2+\delta\cos\gamma\beta}{\sqrt{1+\delta^2+\beta^2+2\delta\cos\gamma\beta}(1+\delta^2\sin^2\gamma)}. \quad (29)$$

The first term is easy to integrate with the substitution $\sin\gamma = u$, $du/d\gamma = \cos\gamma$ [14, 2.124.1]

$$\Omega_2 \equiv \int_0^\alpha d\gamma\frac{\delta\cos\gamma}{1+\delta^2\sin^2\gamma} = \delta\int_0^{\sin\alpha} du\frac{1}{1+\delta^2u^2} = \arctan(\delta u)\,|_0^{\sin\alpha} = \arctan(\delta\sin\alpha). \quad (30)$$

The second term is split into partial fractions with respect to $\cos\gamma$,

$$\frac{1+\delta^2+\delta\cos\gamma\beta}{\sqrt{1+\delta^2+\beta^2+2\delta\cos\gamma\beta}(1+\delta^2\sin^2\gamma)}$$
$$= \frac{1+\delta^2+\delta\cos\gamma\beta}{\sqrt{1+\delta^2+\beta^2+2\delta\cos\gamma\beta}(1+\delta^2-\delta^2\cos^2\gamma)}$$
$$= \frac{1}{2}\frac{1}{\sqrt{1+\delta^2+\beta^2+2\delta\cos\gamma\beta}}\left[-\frac{\sqrt{1+\delta^2}+\beta}{\delta\cos\gamma-\sqrt{1+\delta^2}} + \frac{\sqrt{1+\delta^2}-\beta}{\delta\cos\gamma+\sqrt{1+\delta^2}}\right]. \quad (31)$$

To integrate the second term is reduced to evaluating two integrals which differ only by a sign:

$$J^\pm \equiv \frac{1}{2}\int_0^\alpha d\gamma\frac{1}{(\delta\cos\gamma\pm\sqrt{1+\delta^2})\sqrt{1+\delta^2+\beta^2+2\delta\cos\gamma\beta}}. \quad (32)$$

The substitution $\gamma = 2\varphi$ and the duplication formula of the cosine [12, 4.3.25] generates a shape that can be recognized as an Elliptic Integral [15, 906.02]:

$$\begin{aligned}
J^{\pm} &= \int_0^{\alpha/2} d\varphi \frac{1}{(\delta\cos(2\varphi) \pm \sqrt{1+\delta^2})\sqrt{1+\delta^2+\beta^2+2\delta\cos(2\varphi)\beta}} \\
&= \int_0^{\alpha/2} d\varphi \frac{1}{(\delta[1-2\sin^2\varphi] \pm \sqrt{1+\delta^2})\sqrt{1+\delta^2+\beta^2+2\delta\beta(1-2\sin^2\varphi)}} \\
&= \int d\varphi \frac{1}{(\pm\sqrt{1+\delta^2}+\delta-2\delta\sin^2\varphi)\sqrt{1+(\delta+\beta)^2-4\delta\beta\sin^2\varphi)}} \\
&= \frac{1}{(\delta \pm \sqrt{1+\delta^2})\sqrt{1+(\beta+\delta)^2}} \int d\varphi \frac{1}{(1-\frac{2\delta}{\delta\pm\sqrt{1+\delta^2}}\sin^2\varphi)\sqrt{1-\frac{4\delta\beta}{1+(\beta+\delta)^2}\sin^2\varphi)}} \\
&= \frac{1}{(\delta \pm \sqrt{1+\delta^2})\sqrt{1+(\beta+\delta)^2}} \Pi\left(\frac{\alpha}{2}, \frac{2\delta}{\delta\pm\sqrt{1+\delta^2}}, \frac{2\sqrt{\delta\beta}}{\sqrt{1+(\delta+\beta)^2}}\right). \qquad (33)
\end{aligned}$$

Note that in this notation the second argument of the Incomplete Elliptic Integral of the Third Kind for $J^-$ is always negative.

Superimposing $J^+$ and $J^-$ in (31), inserting this and (30) back into (29), and this (noticing signs) back into (26) yields

$$\Omega(\beta,\delta,\alpha) = \Omega_1(\delta,\alpha) - \Omega_2(\delta,\alpha) - (\sqrt{1+\delta^2}-\beta)J^+(\beta,\delta,\alpha) + (\sqrt{1+\delta^2}+\beta)J^-(\beta,\delta,\alpha). \qquad (34)$$

## V. SUMMARY

We have written the solid angle $\Omega(\beta,\delta,\alpha)$ of the circle sector as a function of the scaled lengths $\delta$ and $\beta$ defined in (18) for a fixed reference direction $\alpha$ defined in Figure 1 relative to the observers azimuth as a sum of two (inverse) trigonometric functions and two incomplete elliptic integrals of the third kind.

### Appendix A: Geometric Reduction to $h$ and $D$

Let the observer be at position $\vec{o}$, the circle center be at position $\vec{c}$ and the surface normal point into the direction $\vec{n}$. The vector from $\vec{o}$ to $\vec{c}$ is decomposed into a vector $\vec{h}$ and a vector $\vec{D}$ as $\vec{c}-\vec{o} = \vec{h}+\vec{D}$, where $\vec{D} \perp \vec{n}$ and $\vec{h} \parallel \vec{n}$. Therefore $(\vec{c}-\vec{o})\cdot\vec{n} = \vec{h}\cdot\vec{n}$ and $h$ is computed as

$$h = |\vec{h}| = |\frac{(\vec{c}-\vec{o})\cdot\vec{n}}{|\vec{n}|}| \qquad (A1)$$

where $|.|$ is the length of vectors and the absolute value of numbers, and $\cdot$ the dot product of vectors. This suffices to compute $\beta$ in (18).

In the degenerate case $h = 0$, the observer is *in* the plane of the circle, and the solid angle is either $2\pi$ if the distance to the center is smaller than $R$, or 0 if it is larger, i.e., if the observer sees the circle edge-on.

$\vec{h}$ is $h$ times a unit vector into the direction of $\vec{n}$ if $\vec{n}$ points into the half space away from the observer [i.e., if $(\vec{c}-\vec{o})\cdot\vec{n} > 0$], or $h$ times a unit vector in the opposite direction if $\vec{n}$ points into the half space that contains the observer:

$$\vec{h} = h\frac{\vec{n}}{|\vec{n}|}\,\mathrm{sgn}[(\vec{c}-\vec{o})\cdot\vec{n}]. \qquad (A2)$$

$D$ follows then as the length of $\vec{D}$:

$$D = |\vec{D}| = |\vec{c}-\vec{o}-\vec{h}| \qquad (A3)$$

or pulling the positive root in Pythagoras' $D^2+h^2 = |\vec{c}-\vec{o}|^2$.

### Appendix B: Auxiliary Integral of a Rational Function

With the shortcut $g \equiv \delta\cos\gamma$ the radial integral needs

$$I \equiv \int dt \frac{1+\delta^2-t^2}{(-2\delta\cos\gamma t + t^2 + 1 + \delta^2)^2} = \int dt \frac{1+\delta^2-t^2}{(t^2 - 2gt + 1 + \delta^2)^2} \tag{B1}$$

with partial fractions

$$= -2g\int dt \frac{t}{(t^2-2gt+1+\delta^2)^2} - \int dt \frac{1}{t^2-2gt+1+\delta^2} + 2(1+\delta^2)\int dt \frac{1}{(t^2-2gt+1+\delta^2)^2}. \tag{B2}$$

The integral with the simple quadratic in the denominator is [14, 2.172]

$$I_1 \equiv \int dt \frac{1}{t^2-2gt+1+\delta^2} = \frac{1}{\sqrt{1+\delta^2\sin^2\gamma}} \arctan \frac{t-g}{\sqrt{1+\delta^2\sin^2\gamma}}. \tag{B3}$$

The integral with the $t$-numerator is [14, 2.175.2]

$$\int dt \frac{t}{(t^2-2gt+1+\delta^2)^2} = -\frac{1+\delta^2-gt}{2(1+\delta^2\sin^2\gamma)(t^2-2gt+1+\delta^2)} + \frac{g}{2(1+\delta^2\sin^2\gamma)} I_1. \tag{B4}$$

The other integral with the quartic polynomial in the denominator is [14, 2.173.1]

$$\int dt \frac{1}{(t^2-2gt+1+\delta^2)^2} = \frac{t-g}{2(1+\delta^2\sin^2\gamma)(t^2-2gt+1+\delta^2)} + \frac{1}{2(1+\delta^2\sin^2\gamma)} I_1. \tag{B5}$$

Substitution of these three intermediate results into the partial fraction yields

$$I = \frac{t}{1+\delta^2+t^2-2\delta\cos\gamma t}. \tag{B6}$$

### Appendix C: (Nearly) On-axis

The on-axis limit of (26) is $\delta = 0$ and the well-known case of the spherical cap:

$$\Omega(\beta, 0, \alpha) = \int_0^\alpha d\gamma \left[1 - \frac{-\beta + \sqrt{1+\beta^2}}{1+\beta^2-\beta\sqrt{1+\beta^2}}\right] = \alpha(1-\cos\varphi), \tag{C1}$$

where $\beta \equiv \tan\varphi$, $\cos^2\varphi = 1/(1+\beta^2)$ and $\varphi$ is a kind of parallax of the circle measured by (18). This is the leading term of a Taylor expansion of (26) in powers of $\delta$:

$$\frac{1}{\sqrt{1+\delta^2}-\delta\cos\gamma} - \frac{-\beta+\sqrt{1+\delta^2+\beta^2+2\delta\cos\gamma\beta}}{1+\delta^2+\beta^2+2\delta\cos\gamma\beta-(\beta+\delta\cos\gamma)\sqrt{1+\delta^2+\beta^2+2\delta\cos\gamma\beta}} \equiv \sum_{i,j\geq 0} \omega_{i,j}\delta^i\cos^j\gamma. \tag{C2}$$

The nonzero coefficients for $i \leq 4$ are

$$\omega_{0,0} = 1 - \cos\varphi, \tag{C3}$$

$$\omega_{1,1} = -\sin^3\varphi, \tag{C4}$$

$$\omega_{2,0} = \frac{1}{2}(1+\cos^2\varphi+\cos\varphi)(\cos\varphi-1), \tag{C5}$$

$$\omega_{2,2} = \frac{1}{2}(3\cos^3\varphi+6\cos^2\varphi+4\cos\varphi+2)(\cos\varphi-1)^2, \tag{C6}$$

$$\omega_{3,1} = \frac{1}{2}(3\cos^2\varphi + 2)\sin^3\varphi, \tag{C7}$$

$$\omega_{3,3} = -\frac{1}{2}(5\cos^2\varphi + 2)\sin^5\varphi, \tag{C8}$$

$$\omega_{4,0} = -\frac{3}{8}(\cos^4\varphi + \cos^3\varphi + \cos^2\varphi + \cos\varphi + 1)(\cos\varphi - 1), \tag{C9}$$

$$\omega_{4,2} = -\frac{3}{4}(5\cos^5\varphi + 10\cos^4\varphi + 8\cos^3\varphi + 6\cos^2\varphi + 4\cos\varphi + 2)(\cos\varphi - 1)^2, \tag{C10}$$

$$\omega_{4,4} = -\frac{1}{8}(35\cos^6\varphi + 105\cos^5\varphi + 120\cos^4\varphi + 80\cos^3\varphi + 48\cos^2\varphi + 24\cos\varphi + 8)(\cos\varphi - 1)^3. \tag{C11}$$

The integration over $\gamma$ then leads to an expansion of $\Omega$ for small $\delta$:

$$\Omega = \alpha\sum_{i\geq 0}\omega_{i,0}\delta^i \ + \sin\alpha\sum_{i\geq 0}\omega_{i,1}\delta^i \ + \left[\frac{1}{4}\sin(2\alpha) + \frac{\alpha}{2}\right]\sum_{i\geq 0}\omega_{i,2}\delta^i + \left[\frac{1}{12}\sin(3\alpha) + \frac{3}{4}\sin\alpha\right]\sum_{i\geq 0}\omega_{i,3}\delta^i$$
$$+ \left[\frac{1}{32}\sin(4\alpha) + \frac{1}{4}\sin(2\alpha) + \frac{3}{4}\alpha\right]\sum_{i\geq 0}\omega_{i,4}\delta^i + \cdots \tag{C12}$$

## Appendix D: Small Radii

For small cross sections (i.e., small radii or distant circles), a Taylor expansion of (26) in powers of $\beta$ is:

$$\frac{1}{\sqrt{1+\delta^2} - \delta\cos\gamma} - \frac{-\beta + \sqrt{1+\delta^2+\beta^2+2\delta\cos\gamma\beta}}{1+\delta^2+\beta^2+2\delta\cos\gamma\beta - (\beta+\delta\cos\gamma)\sqrt{1+\delta^2+\beta^2+2\delta\cos\gamma\beta}} \equiv \sum_{i,j\geq 0}\sigma_{i,j}\beta^i\cos^j\gamma. \tag{D1}$$

$$\sigma_{2,0} = \frac{1}{2}\frac{1}{(1+\delta^2)^{3/2}}, \tag{D2}$$

$$\sigma_{3,1} = -\frac{\delta}{(1+\delta^2)^{5/2}}, \tag{D3}$$

$$\sigma_{4,0} = -\frac{3}{8}\frac{1}{(1+\delta^2)^{5/2}}, \tag{D4}$$

$$\sigma_{4,2} = \frac{15}{8}\frac{\delta^2}{(1+\delta^2)^{7/2}}. \tag{D5}$$

And (C12) remains valid with the substitution $\omega \to \sigma$, $\delta^i \to \beta^i$. The leading term is $\Omega \approx \alpha\sigma_{2,0}\beta^2 = \frac{A}{h^2}\frac{1}{(1+\delta^2)^{3/2}}$ where $A = \alpha R^2/2$ is the area of the circle section and $1/(1+\delta^2)^{3/2}$ is a diminishing factor characterizing how much the circle surface is inclined to the line of sight.

## Appendix E: C++ Source Code

The following C++ source code evaluates (34) with the aid of the elliptic integrals implemented in the GNU Scientific Library (GSL) [16]. (Obtained under openSUSE with `zypper in gsl-devel`, under Ubuntu with `apt install libgsl-dev`, under Fedora with `dnf install gsl-devel`, under CentOs with `yum install gsl-devel`...) There are also two alternative test implementations that (i) integrate (26) by a Simpson numerical integration for a variable number of angular abscissae points or (ii) dissect the circular section into a variable number of triangles with common apex at the circle center and the other vertices on the circle rim, accumulating the solid angle with the Oosterom-Strackee formula applied to each triangle.

Examples:

- If the observer is at $(0,0,0)$ and the circle at $(0,10,0)$ with radius 1, the angle $\alpha = 180°$, a Simpson integration with 10 points, the Oosterom-Strackee dissection with 5 triangles, and the analytical results are:

```
solidang -y 10 -R 1. -N 10 -a 180
0.0015646496458735781
solidang -y 10 -R 1. -O 5 -a 180
0.0014626836558120932
solidang -y 10 -R 1. -a 180
0.0015646496458747716
```

- If the observer is at $(0,0,0)$ and the circle at $(0,3,0)$ with radius 10, the angle $\alpha = 180°$, a Simpson integration with 8 points, the Oosterom-Strackee dissection with 3 triangles, and the analytical results are:

```
solidang -y 3 -R 10. -N 8 -a 180
2.6866258325155066
solidang -y 3 -R 10. -O 3 -a 180
2.766566865243071
solidang -y 3 -R 10. -a 180
2.8064659811454451
```

  Increasing the parameters for the numerical integrations puts them closer to the analytical result:

```
solidang -y 3 -R 10. -N 20 -a 180
2.8034806364203759
solidang -y 3 -R 10. -O 20 -a 180
2.8056826435280899
```

One can also select random positions of the circle, of the surface normal, radius and angle with a bash-script as follows:

```
#!/usr/bin/env bash

make solidang

# generate 3 integer random numbers for the position of the circle center
xrand=$(( ( $RANDOM - 16300 ) / 1000 ))
yrand=$(( ( $RANDOM - 16300 ) / 1000 ))
zrand=$(( ( $RANDOM - 16300 ) / 1000 ))

# generate 3 integer random numbers for the direction of the circle normal
urand=$(( ( $RANDOM - 16300 ) / 1000 ))
vrand=$(( ( $RANDOM - 16300 ) / 1000 ))
wrand=$(( ( $RANDOM - 16300 ) / 1000 ))

# generate a positive integer random number for the circle radius
Rrand=$(( 1 + $RANDOM / 1000 ))

# generate a integer random number for the section angle in degrees
arand=$(( ( $RANDOM - 16300 ) / 100 ))

# run the two test evaluations and the analytic computation for comparison
# taking 50 triangles in the van-Oosterom-Strackee triangulation and
# 100 points for the Simpson rule.
echo "circ=(" $xrand $yrand $zrand ") normal=(" $urand $vrand $wrand ") R=" $Rrand " ang =" $arand

echo "Oosterom-Strackee"
solidang -x $xrand -y $yrand -z $zrand -u $urand -v $vrand -w $wrand -R $Rrand -a $arand -O 50

echo "Simpson"
solidang -x $xrand -y $yrand -z $zrand -u $urand -v $vrand -w $wrand -R $Rrand -a $arand -N 100

echo "analytic"
solidang -x $xrand -y $yrand -z $zrand -u $urand -v $vrand -w $wrand -R $Rrand -a $arand
```

The transcription of Paxton's variables [3] to our variables is $r_m \to R$, $r_0 \to D$, $L \to h$. Our results differs from [3, Table 1] generally in the last two digits. To reproduce his table we can simply scale all lengths as $L = h = 1$, set $R$ to the reciprocal of the $L/r_m$ header numbers, then set $D = (r_0/r_m)R$ as the $x$ coordinate of the circular center. Example: for the column $L/r_m = 2$ we have $R = 0.5$, and for the row $r_0/r_m = 0.4$ we have $D = 0.4 \times 0.5 = 0.2$. Our program gives

```
solidang -x 0.2 -a 360 -R 0.5
0.637048888347640
```

where Paxton's value is $\Omega = 0.6370508$, $\Omega^a = 0.637049$. Another example: for the column $L/r_m = 0.5$ we have $R = 2$, and for the row $r_0/r_m = 1.8$ we have $D = 1.8 \times 2 = 3.6$. Our program gives

```
solidang -x 3.6 -a 360 -R 2.
0.32870136091572455
```

where Paxton's value is $\Omega = 0.3287007$, $\Omega^a = 0.328702$. The values of Paxton's external reference $\Omega^a$ appear to match our results better.

## 1. Makefile

```
# compile the main program in solidang.cxx and the three classes
# into a single executable named solidang.
solidang: solidang.cxx Circ3D.cxx Circ3D.h Triang3D.cxx Triang3D.h Point3D.cxx Point3D.h
    $(CXX) -o $@ -O3 -I. solidang.cxx Circ3D.cxx Triang3D.cxx Point3D.cxx -lgsl -lgslcblas
```

## 2. Point3D.h

```
#pragma once

/*!************************************************
* @file
* @brief API declaration of the Point3D class.
*/

using namespace std ;

/*!************************************
* @brief A point characterized by 3 Cartesian coordinates.
* @since 2022-02-02
* @author Richard J. Mathar
*/
class Point3D
{

public:
    /**
    * @brief The three cartesian coordinates (in some common lengths of units)
    */
    double car[3] ;

    Point3D() ;
    Point3D(const Point3D & org) ;
    Point3D(const double x, const double y, const double z) ;
    Point3D(const double cart[3]) ;

    double len() const ;
    double dotprod(const Point3D & oth) const ;
    Point3D crossprod(const Point3D & oth) const ;
    Point3D to(const Point3D & oth) const ;
    Point3D multiplyAdd(const double fact, const Point3D & oth) const ;

    Point3D & operator=(const Point3D & org) ;

protected:
private:

} ; /* Point3D */
```

## 3. Point3D.cxx

```
/*!************************************************
* @file
* @brief Point3D is a point in 3D space characterized by 3 Cartesian coordinates.
*/

#include <cstring>
#include <cmath>
#include "Point3D.h"
```

```
using namespace std ;

/*!*****************************************
* @brief Default constructor.
* This defines a point at the origin of coordinates.
* @since 2022-02-02
* @author Richard J. Mathar
*/
Point3D::Point3D()
{
    car[0]=car[1]=car[2]= 0.0 ;
} /* Point3D::Point3D */

/*!*****************************************
* @brief Constructor with explicit coordinate triple
* @param[in] x  The coordinate projected to first axis.
* @param[in] y  projected to 2nd axis.
* @param[in] z  projected to 3rd axis.
* @since 2022-02-02
* @author Richard J. Mathar
*/
Point3D::Point3D(const double x, const double y, const double z)
{
    car[0]= x ;
    car[1]= x ;
    car[2]= z ;
}

/*!*****************************************
* @brief Constructor with explicit coordinate triple
* @param[in] xyx the first 3 elements are the x, y and z coordinate.
* @since 2022-02-02
* @author Richard J. Mathar
*/
Point3D::Point3D(const double xyz[3])
{
    memmove(car, xyz, sizeof(double[3])) ;
}

/*!*****************************************
* @brief Copy constructor.
* @param[in] org the original value to be duplicated/copied.
* @since 2022-02-02
* @author Richard J. Mathar
*/
Point3D::Point3D(const Point3D & org)
{
    memmove(car, org.car, sizeof(double[3])) ;
}

/*!*************************************
* @brief Assignment operator
* @param[in] org the original value to be duplicated/copied.
* @since 2022-02-02
* @author Richard J. Mathar
*/
Point3D & Point3D::operator=(const Point3D & org)
{
    if ( this != & org)
        memmove(car, org.car, sizeof(double[3])) ;
    return *this ;
}

/*!*****************************************
* @brief length. Euclidean distance to origin.
* @return the non-negative distance of this point to the origin.
* @since 2022-02-02
* @author Richard J. Mathar
*/
double Point3D::len() const
{
    return hypot(hypot(car[0],car[1]),car[2]) ;
}

/*!*****************************************
* @brief dot product with another point (both interpreted as vector)
* @param[in] oth the vector this is multiplied with.
* @return sum of the 3 products of the x, y and z coordinates.
* @since 2022-02-02
* @author Richard J. Mathar
*/
double Point3D::dotprod( const Point3D & oth) const
```

```
{
    return car[0]*oth.car[0] + car[1]*oth.car[1] + car[2]*oth.car[2] ;
}

/*!****************************************
* @brief vector cross with another point (both interpreted as vector)
* @param[in] oth the vector on the right hand side this is multiplied with.
* @return A new vector equaling this cross oth.
* @since 2022-02-02
* @author Richard J. Mathar
*/
Point3D Point3D::crossprod( const Point3D & oth) const
{
    double resul[3] ;
    resul[0] = car[1]*oth.car[2] - car[2]*oth.car[1] ;
    resul[1] = car[2]*oth.car[0] - car[0]*oth.car[2] ;
    resul[2] = car[0]*oth.car[1] - car[1]*oth.car[0] ;
    return Point3D(resul) ;
} /* crossprod */

/*!****************************************
* @brief compute the vector from this point to another.
* @param[in] oth the vector on the right hand side this is multiplied with.
* @return oth minus this.
* @since 2024-05-04
* @author Richard J. Mathar
*/
Point3D Point3D::to( const Point3D & oth) const
{
    double resul[3] ;
    for(int c=0 ; c < 3 ; c++)
        resul[c] = oth.car[c] - car[c] ;
    return Point3D(resul) ;
} /* to */

/*!****************************************
* @brief compute the vector this plus fact times other.
* @param[in] fact the multiplier /stretch factor for oth
* @param[in] oth the vector on the right hand side to be added.
* @return this + fact * oth
* @since 2024-05-04
* @author Richard J. Mathar
*/
Point3D Point3D::multiplyAdd(const double fact, const Point3D & oth) const
{
    double resul[3] ;
    for(int c=0 ; c < 3 ; c++)
        resul[c] = car[c] + fact*oth.car[c] ;
    return Point3D(resul) ;
} /* multiplyAdd */
```

## 4. Triang3D.h

```
#pragma once

/*!************************************************
* @file
* @brief API declaration of the Triang3D class.
*/

using namespace std ;

#include <Point3D.h>

/*!*************************************
* @brief A triangle characterized by 3 points in space.
* @since 2022-02-02
* @author Richard J. Mathar
*/
class Triang3D
{

public:
    /**
    * @brief The three points.
    * Note that the order defines a surface normal (orientation) by the right-hand rule.
    */
    Point3D corn[3] ;

        Triang3D(const Point3D & A, const Point3D & B, const Point3D & C) ;
```

```
        Triang3D(const Triang3D & org) ;
    Triang3D & operator=(const Triang3D & org) ;

    double solidAng() const ;

protected:
private:

} ; /* Point3D */
```

## 5. Triang3D.cxx

```
/*!***************************************************
* @file
* @brief Triang3D is an oriented planar triangle in 3D.
*/

#include <cmath>
#include "Triang3D.h"

using namespace std ;

/*!****************************************
* @brief Constructor with 3 explicit corner points.
* @param[in] A  The first corner.
* @param[in] B  The second corner.
* @param[in] C  The third corner , walking ccw around the triangle interior.
* @since 2022-02-02
* @author Richard J. Mathar
*/
Triang3D::Triang3D(const Point3D & A, const Point3D & B, const Point3D & C)
{
    corn[0]= A ;
    corn[1]= B ;
    corn[2]= C ;
} /* ctor */

/*!****************************************
* @brief Copy ctor.
* @param[in] org  The triangle to be cloned/copied.
* @since 2022-02-02
* @author Richard J. Mathar
*/
Triang3D::Triang3D(const Triang3D & org)
{
    for(int i=0 ; i < 3 ; i++)
        corn[i]= org.corn[i] ;
} /* copy ctor */

/*!****************************************
* @brief Assignment operator
* @param[in] org  The triangle to be cloned/copied.
* @since 2022-02-02
* @author Richard J. Mathar
*/
Triang3D & Triang3D::operator=(const Triang3D & org)
{
    if ( this != & org)
        for(int i=0 ; i < 3 ; i++)
            corn[i]= org.corn[i] ;
    return *this ;
} /* operator= */

/*!****************************************
* @brief Compute the solid Angle in ster-radians.
*  Implementation of the Oosterom-Strackee IEEE Trans. Biomed Eng. BME-30 (2) 125-125
* @return the solid angle in srad.
* @since 2022-02-02
* @author Richard J. Mathar
*/
double Triang3D::solidAng() const
{
    /* numerator of the tan(omega/2) is triple product of the three points
    */
    const double num = corn[0].dotprod( corn[1].crossprod(corn[2])) ;

    /* denominator */
    double len[3] ;
    for(int i=0 ; i < 3 ; i++)
    {
```

```
        len[i] = corn[i].len() ;
    }

    const double den = len[0]*len[1]*len[2]
        + corn[0].dotprod(corn[1]) * len[2]
        + corn[1].dotprod(corn[2]) * len[0]
        + corn[2].dotprod(corn[0]) * len[1] ;


    return 2.*atan2(num,den) ;
} /* solidAng */
```

## 6. Circ3D.h

```
#pragma once

/*!***********************************************
* @file
* @brief API declaration of the Circ3D class.
*/

#include <Point3D.h>
using namespace std ;

/*!*************************************
* @brief A planar circle embedded into 3D space.
*  Specified by the location of the center (a 3D cartesian vector),
*  the radius (positive), and the direction of the surface normal
*  (a 3D cartesian vector, in practise of unit length).
* @since 2024-05-04
* @author Richard J. Mathar
*/
class Circ3D
{

public:
    /**
    * @brief The three cartesian coordinates of the circle center
    */
    Point3D cntr ;

    /**
    * @brief The direction of the surface normal, three
    *  Cartesian components
    */
    Point3D normal ;

    /**
    * @brief the circle radius
    */
    double R ;

    Circ3D() ;
    Circ3D(Point3D & cntrCart, Point3D normCart, double radius) ;

    double solidAng(double alpha, int Ooster, int NRiemann) ;
    double solidAng(Point3D & obs, double alpha, int Ooster, int NRiemann) ;

protected:
    double solidAngRie(double alpha, int NRiemann) ;
    double solidAngEllip(double alpha) ;
    double solidAngOost(double alpha, int Ooster) ;
    void normalize(double &delta, double & beta) ;
private:
    static double Irad(const double delta, const double cosgam, const double t) ;

} ; /* Circ3D */
```

## 7. Circ3D.cxx

```
/*!***************************************************
* @file
* @brief Circ3D is a planar circle in 3D space characterized by
*   position of the center, radius and direction of the surface normal
*/

#include <iostream>
```

```
#include <cstring>
#include <cmath>

/* headers of the GNU scientific library
* of https://www.gnu.org/software/gsl
*/
#include <gsl/gsl_sf_elljac.h>
#include <gsl/gsl_sf_ellint.h>
#include <gsl/gsl_const_num.h>


#include "Circ3D.h"
#include "Triang3D.h"

using namespace std ;

/*!***************************************
* @brief Default constructor.
* Defines a circle with center at (0,0,1) and radius 1.
* @since 2024-05-04
* @author Richard J. Mathar
*/
Circ3D::Circ3D()
    : cntr(0,0,1), normal(0,0,1), R(1.0)
{
} /* Circ3D::Circ3D */

/*!***************************************
* @brief Constructor by center, normal and radius.
* @param[in] cntrCart The center in cartesian coordinates
* @param[in] normCart The direction of the normal vector
* @param[in] radius The radius
* @since 2024-05-04
* @author Richard J. Mathar
*/
Circ3D::Circ3D(Point3D & cntrCart, Point3D normCart, double radius)
    : cntr(cntrCart), normal(normCart), R(radius)
{
} /* Circ3D::Circ3D */

/*!*****************************
* @brief extract the length ratios beta and delta relevant to the solid angle
*  Note that this is only called as the observer and circle relative position
*  have been shifted such that the observer is at the origin here.
* @param[out] delta ratio of the distance of circle center to the footer and distance of observer to footer
*    Where the footer is the observer's position projected onto the circle plane
* @param[out] beta ratio of the circle radius and distance of observer to footer
*    Where the footer is the observer's position projected onto the circle plane
*/
void Circ3D::normalize(double & delta, double & beta)
{

    /* vect(circlcenter) = h*vect(unitfooter)+ D*vect(unitinplane)
    * where vect(unitfooter) is parallel or antiparallel to vect(normal)
    * and perpendicular to vect(unitinplane). Both unitfooter and unitinplane have length 1,
    * such that the length of h and D are defined.
    * vect(circlecnter).vect(normal) = h* vect(footer).vect(normal)
    */
    double lhs = cntr.dotprod(normal) ;
    const double h = fabs(lhs/normal.len()) ;



    const double distC = cntr.len() ;
    const double D = sqrt(distC*distC-h*h) ;


    delta = D/h ;
    beta = R/h ;

} /* Circ3D::normalize */

/*!***************************************
* @brief solid angle by various methods of approximation.
*  The observer is at the center of the coordinate system.
*  If the NRiemann integer parameter is positive, a Simpson
*  integration of the angular integral with that many number
*  of abscissae along alpha is performed.
*  If the NRiemann parameter is zero or negative and the
*  Ooseter paremter is positive, the ciruclar section is
*  approximated by as many tringles inside the circle as
*  given by the parameter and the Oosterom-Strackee formula
*  for the solid angle is called for each of the triangles,
```

```
*  all added up.
*  If Ooster and NRiemann are both zero or negative,
*  an integration of analytical terms based on the
*  Elliptic Integrals of the third kind via the
*  GNU Scientific library is performed.
* @param[in] alpha The angle of the circular section, radians
* @param[in] Ooster The number of triangles inside the section for an Oosterom-Strackee triangulation
* @param[in] NRiemann The number of points of numerical integration along the perimeter of the Simpson integration
* @return the solid angle in srad
* @since 2024-05-04
* @author Richard J. Mathar
*/
double Circ3D::solidAng(double alpha, int Ooster, int NRiemann)
{
    double omega(0.) ;
    if ( NRiemann > 0)
        omega = solidAngRie(alpha,NRiemann) ;
    else if ( Ooster > 0)
        omega = solidAngOost(alpha,Ooster) ;
    else
    {
        omega = solidAngEllip(alpha) ;
    }
    return omega ;
} /* Circ3D::solidAng */

/*!*****************************************
* @brief solid angle by various methods of approximation.
* @param[in] obs The coordinates of the observer of the solid angle
* @param[in] alpha The angle of the circular sections, radians
* @param[in] Ooster The number of triangles inside the section for an Oosterom-Strackee triangulation
* @param[in] NRiemann The number of angular points of numerical integration along the perimeter
* @since 2024-05-04
* @author Richard J. Mathar
*/
double Circ3D::solidAng(Point3D & obs, double alpha, int Ooster, int NRiemann)
{
    /* shift the circle center such that the observer would be
    * at the center of coordinates
    */
    Point3D shftCircCntr = obs.to(cntr) ;
    Circ3D shftCirc = Circ3D(shftCircCntr, normal, R) ;


    return shftCirc.solidAng(alpha, Ooster, NRiemann) ;
} /* Circ3D::solidAng */

/*!*****************************************
* @brief solid angle by using the analytical solution with elliptic integrals
* @param[in] alpha The angle of the circular sections, radians
* @since 2024-05-05
* @author Richard J. Mathar
*/
double Circ3D::solidAngEllip(double alpha)
{
    /* The two dimensionsless length ratios,
    * circle distance to footter over distance observer to plane,
    * and radius over distance observer to plane.
    */
    double delta, beta ;
    normalize(delta, beta) ;

    double deltaprime = sqrt(1+delta*delta) ;

    /* decide on the branch of the arctan.
    * If alpha/2 is in the range -90deg..+90deg the branch is 0.
    * If alpha/2 is in the range +90deg..+270deg the branch is 1.
    * If alpha/2 is in the range +270deg..+450deg the branch is 2.
    * If alpha/2 is in the range -90deg..-270deg the branch is -1.
    */
    int branch = (alpha/2.+M_PI_2)/M_PI ;
    /* integer promotion of floating point values in C++ rounds to zero, but
    * we want to round towards minus infinity
    */
    if ( alpha/2.+M_PI_2 < 0.)
        branch-- ;

    double omega = 2* ( atan(tan(alpha/2.)/(deltaprime-delta)) +branch*M_PI) ;
    omega -= atan(delta*sin(alpha)) ;

    /* first the contribution from J+
    */
    double alphsq = 2*delta/(delta+deltaprime) ;
```

```
    double k = 2*sqrt( delta*beta/(1+pow(delta+beta,2.)) ) ;
    double J = gsl_sf_ellint_P(alpha/2.,k,-alphsq,GSL_PREC_DOUBLE) ;
    J /= (delta+deltaprime)*sqrt( 1+pow(delta+beta,2.) ) ;
    omega -= (deltaprime-beta)*J ;

    /* then the contribution form J-
    */
    alphsq = 2*delta/(delta-deltaprime) ;
    J = gsl_sf_ellint_P(alpha/2.,k,-alphsq,GSL_PREC_DOUBLE) ;
    J /= (delta-deltaprime)*sqrt( 1+pow(delta+beta,2.) ) ;
    omega += (deltaprime+beta)*J ;

    return omega ;
} /* Circ3D::solidAngEllip */

/*!****************************************
* @brief solid angle by Riemann integration along the angle
* @param[in] alpha The angle of the circular sections, radians
* @param[in] NRiemann The number of angular points of numerical integration along the perimeter
* @since 2024-05-04
* @author Richard J. Mathar
*/
double Circ3D::solidAngRie(double alpha, int NRiemann)
{
    /* the two dimensionsless length ratios,
    * circle distance to footter over distance observer to plane,
    * and radius over distance observer to plane.
    */
    double delta, beta ;
    normalize(delta, beta) ;

    /* upper and lower limits of the radial integral
    */
    double t[2] ;
    t[1] = sqrt(1+delta*delta) ;

    double omega(0.) ;
#if 0
    /* interpret NRiemann as the number of intervals.
    */
    const double deltaGamma = alpha/NRiemann ;
    /* the Riemann simple rule
    */
    for(int i=0 ; i < NRiemann ; i++)
    {
        const double gamma = (i+0.5)*deltaGamma ;
        const double cosgam = cos(gamma) ;
        t[0] = -beta+sqrt(beta*beta+1+delta*delta+2*delta*beta*cosgam) ;
        /* might obviously be sped up by moving the term with t[0] outside
        * the loop...
        */
        const double Idiff = Irad(delta,cosgam,t[1]) - Irad(delta,cosgam,t[0]) ;
        omega += 2.*Idiff ;
    }
    omega *= deltaGamma ;
#else
    /* interpret NRiemann as the number of points, including start and end.
    */
    const double deltaGamma = alpha/(NRiemann-1) ;
    /* Simpson rule
    */
    for(int i=0 ; i < NRiemann ; i++)
    {
        const double gamma = i*deltaGamma ;
        const double cosgam = cos(gamma) ;
        t[0] = -beta+sqrt(beta*beta+1+delta*delta+2*delta*beta*cosgam) ;
        /* might obviously be sped up by moving the term with t[0] outside
        * the loop...
        */
        const double Idiff = 2.*(Irad(delta,cosgam,t[1]) - Irad(delta,cosgam,t[0])) ;
        if ( i==0 || i == NRiemann-1)
            omega += Idiff ;
        else if ( i %2)
            omega += 4*Idiff ;
        else
            omega += 2*Idiff ;
    }
    omega *= deltaGamma/3. ;
#endif

    return omega ;
} /* Circ3D::solidAngRie */
```

```
/*!****************************************
* @brief solid angle by triangulation of the sector.
* @param[in] alpha The angle of the circular sections, radians
* @param[in] Ooster The number of triangles of the triangulation
* @since 2024-05-04
* @author Richard J. Mathar
*/
double Circ3D::solidAngOost(double alpha, int Ooster)
{


    /* defined the +x axis in the local plane circle coordinate system
    * where (x=0,y=0) is a the circle center and the z coordinates
    * of all points in the circle plane are zero.
    * Reduce normal vector to unit length for convenience with
    * reduction further down
    */
    double nlen = normal.len() ;
    for(int c=0 ; c< 3; c++)
        normal.car[c] /= nlen ;

    /* vect(circlcenter) = h*vect(unitfooter)+ D*vect(unitinplane)
    * where vect(unitfooter) is parallel or antiparallel to vect(normal)
    * and perpendicular to vect(unitinplane). Both unitfooter and unitinplane have length 1,
    * such that the length of h and D are defined.
    * vect(circlecnter).vect(normal) = h* vect(footer).vect(normal)
    */
    double lhs = cntr.dotprod(normal) ;
    /* two situations: the normal points into the half space without the
    * observer, so lhs >0, or the normal points into the half space that
    * contains the observer lhs<0. We want the latter and h>0.
    */
    if ( lhs >0.)
    {
        for(int c=0 ; c< 3; c++)
            normal.car[c] *= -1. ;
        lhs *= -1 ;
    }
    else if ( lhs==0.)
    {
        /* degenerate case where the circle plaen contains the
        * observer (circle looke at edge on)
        */
    }
    const double h = -lhs ;

    /* point in the circle plane onto which the observer is projected
    */
    double foot[3] ;
    for(int c=0 ; c< 3; c++)
        foot[c] = -h*normal.car[c] ;

    const Point3D obsProj(foot) ;
    /* vect(circleenter minus h*vect(unitfooter), pointing in the plane to circle center
    */
    Point3D xunit = obsProj.to(cntr) ;

    /* normalize this to unit length. This may not work if footer and circle center
    * are the same, but then we can take any other reference direction.
    * For numerical stablity better to use relation with h....
    */
    if ( xunit.len() == 0.)
    {
        xunit.car[0] = 1. ; xunit.car[1]=xunit.car[2]=0. ;
    }
    else
    {
        double xlen =xunit.len() ;
        for(int c=0 ; c< 3; c++)
            xunit.car[c] /= xlen ;
    }

    /* x cross y = normal. y= normal cross x
    */
    Point3D yunit = normal.crossprod(xunit) ;

    double omega(0.) ;
    /* sector angle of a single triangle
    */
    double deltaalpha = alpha/Ooster ;

    /* triangles enumerated 0 to Oooster
```

```
    */
    for(int t=0 ; t < Ooster ; t++)
    {
        /* angles at the two vertices on the rim
        */
        double altri[2] ;
        altri[0] = t*deltaalpha ;
        Point3D rim0 = cntr.multiplyAdd(R*cos(altri[0]),xunit)
                .multiplyAdd(R*sin(altri[0]),yunit) ;

        altri[1] = (t+1)*deltaalpha ;
        Point3D rim1 = cntr.multiplyAdd(R*cos(altri[1]),xunit)
                .multiplyAdd(R*sin(altri[1]),yunit) ;
        Triang3D sec(cntr,rim1,rim0) ;


        omega += sec.solidAng() ;
    }

    return omega ;
} /* Circ3D::solidAngOoster */

/*!****************************************
* @brief auxiliary integral over the radial part
* @param[in] delta length ratio of the normalized view
* @param[in] cosgam cosine of thte azimuth angle
* @param[in] t variable resulting from the Euler substitution.
* @since 2024-05-04
* @author Richard J. Mathar
*/
double Circ3D::Irad(const double delta, const double cosgam, const double t)
{
    return t/(1.+t*t+delta*delta-2*delta*cosgam*t) ;
} /* Circ3D::Irad */
```

## 8. solidang.cxx

This main program has the syntax

`solidang` [-x *CcntrX]* [-y *CcntrY]* [-z *CcntrZ]* [-a *alphaDegrees]* [-R *Cradius] [*`-O` *NOoster* | `-N` *NRiem]* [-u *normX]* [-v *norm]* [-w *normZ]* [-X *obsrvX]* [-Y *obsrvY]* [-Z *obsrvZ]*

The lower-case `x`, `y` and `z` options specify the Cartesian coordinates of the circle center. If not provided, the circle center is at $(0, 0, 1)$.

The upper-case `X`, `Y` and `Z` options specify the Cartesian coordinates of the observer. If not provided, the observer is at $(0, 0, 0)$.

The option `R` specifies the positive circle radius. If not provided, the value is 1.

The option `a` specifies the section angle $\alpha$ in degrees. If not provided, the value is 180. Flipping the sign of $\alpha$ also flips the sign of the result, Eq. 27.

The three options `u`, `v` and `w` specify the Cartesian coordinates of the direction of the normal of the surface vector of the circle. If not provided the values are $(0, 0, 1)$. The three values do not need to be normalized to $u^2 + v^2 + w^2 = 1$, but the length must not be zero.

The option capital-`O` specifies that the solid angle will be computed by triangulating the circle section with `NOoster` triangles (a value $\geq 1$) and their solid angles be calculated by the Oosterom-Strackee formula [9, (8)] and summed up. The `NOoster` triangles are defined by having one corner at the circle center, and the other two corners at the circle rim with an angle $\alpha/NOoster$ at that center. Two of their side lengths equal the circle radius. Because that approximation misses small circular segments between the circle rim and the straight nearby sides of the triangles, the result is a lower estimate to the full solid angle of the circular section.

The option `N` specifies that a Simpson rule with `NRiem` points along the $\gamma$ coordinate will be used to integrate (26) numerically.

The options `N` and `O` are mutually exclusive; only one can be specified per call.

```
#include <cmath>
#include <iostream>
#include <unistd.h>
#include <iomanip>

#include "Point3D.h"
#include "Circ3D.h"

void usage(char *argv0)
{
    std::cout << "usage: " << argv0 << "-x circentX -y circentY -z circentZ -X observX" ;
```

```
    std::cout << " -Y observY -Z observZ -a alpha-deg -R circrad -u circnnormX -v circnormY -w circnormZ [-O #oosterom | -N Riemann]" << std::endl  ;
}

/**
* @brief Compute the solid angle of a circular section.
* viXra:1404.0977
* @param argc Number of command line arguments, including the program name and options.
* @param argv Command line arguments.
*   Usage: solidang -x circentX -y circentY -z circentZ -X observX -Y observY -Z observZ \
*                   -a alpha-deg -R circrad -u circnnormX -v circnormY -w circnormZ [-O #oosterom | -N #Simpson]
* @author Richard J. Mathar
* @since 2024-05-04
*/
int main(int argc, char * argv[])
{
    /* option character
    */
    int oc ;

    /* number of abscissae angles for a Simpson integration along azimuth
    */
    int NRiemann(0) ;

    /* number of triangles for a Oosteroom-Strackee approximation
    */
    int Ooster(0) ;

    /* circle Radius
    */
    double R(1.0) ;

    /* observer coordinates , default values */
    Point3D obs ;

    /* circle center coordinates, default values  */
    Circ3D circ ;

    /* default angle of the sector is 180 degrees, half the circle
    */
    double alpha(M_PI) ;

    while (  (oc=getopt(argc,argv,"x:y:z:X:Y:Z:a:R:u:v:w:O:N:h")) != -1 )
    {
        switch(oc)
        {
        case 'x':
            circ.cntr.car[0] = atof(optarg) ;
            break;
        case 'y':
            circ.cntr.car[1] = atof(optarg) ;
            break;
        case 'z':
            circ.cntr.car[2] = atof(optarg) ;
            break;
        case 'u':
            circ.normal.car[0] = atof(optarg) ;
            break;
        case 'v':
                  circ.normal.car[1] = atof(optarg) ;
            break;
        case 'w':
            circ.normal.car[2] = atof(optarg) ;
            break;
        case 'X':
            obs.car[0] = atof(optarg) ;
            break;
        case 'Y':
            obs.car[1] = atof(optarg) ;
            break;
        case 'Z':
            obs.car[2] = atof(optarg) ;
            break;
        case 'O':
            Ooster = atoi(optarg) ;
            break;
        case 'N':
            NRiemann = atoi(optarg) ;
            /* upgrade to an odd number so we can use Simpson's rule
            */
            if ( NRiemann %2 ==0)
                NRiemann++ ;
            break;
        case 'a':
```

```
            /* user angle argument in degrees, but internally
            * only in radians
            */
            alpha = atof(optarg)*M_PI/180.0 ;
            break;
        case 'R':
            circ.R = R = atof(optarg) ;
            break;
        case 'h':
            usage(argv[0]) ;
            return 0 ;
          case '?' :
            cerr << "Invalid command line option " << optopt << " ... ignored "<< endl ;
            break ;
        }
    }

    /* check compliance with obvious restrictions on the geometry
    */
    if ( Ooster < 0)
    {
        cerr << "negative value " << Ooster << " of Oosterom-Strackee triangles" << endl ;
        usage(argv[0]) ;
        return 1 ;
    }

    if ( NRiemann < 0)
    {
        cerr << "negative value " << NRiemann << " of Riemann abscisae" << endl ;
        usage(argv[0]) ;
        return 1 ;
    }

    if ( R <= 0)
    {
        cerr << "negative radius " << R << endl ;
        usage(argv[0]) ;
        return 1 ;
    }

    if ( NRiemann *Ooster != 0)
    {
        cerr << "cannot use Oosterom-Str and Simpson integration at the same time" << endl ;
        usage(argv[0]) ;
        return 1 ;
    }

    double omeg = circ.solidAng(obs,alpha,Ooster,NRiemann) ;
    std::cout << std::setprecision(17) << omeg << std::endl;

    return 0 ;
} /* main */
```


[1] A. H. Jaffey, Solid angle subtended by a circular aperture at point and spread sources: formulas and some tables, Rev. Sci. Instr. **25**, 349 (1954).

[2] A. V. Masket, Solid angle contour integrals, series, and tables, Rev. Sci. Instr. **28**, 191 (1957), a square root seems to be missing in (34).

[3] F. Paxton, Solid angle calculation for a circular disk, Rev. Sci. Instr. **30**, 254 (1959).

[4] M. Naitō, A method of calculating the solid angle subtended by a circular aperture, J. Phys. Soc. Jpn. **12**, 1122 (1957).

[5] R. P. Gardner and A. Carnesale, The solid angle subtended at a point by a circular disk, Nucl. Instr. Meth. **73**, 228 (1969).

[6] D. M. Timus, M. J. Prata, S. L. Kalla, M. I. Abbas, F. Oner, and E. Galiano, Some further analytical results on the solid angle subtended at a point by a circular disk using elliptic integrals, Nucl. Instr. Meth. A. **580**, 149 (2007).

[7] S. Tryka, Angular distribution of the solid angle at a point subtended by a circular disk, Opt. Commun. **137**, 317 (1997).

[8] M. J. Prata, Analytical calculation of the solid angle defined by a circular disc detector at a point cosine source, Nucl. Instr. Meth. A **521**, 576 (2004).

[9] A. van Oosterom and J. Strackee, The solid angle of a plane triangle, IEEE Trans. Biomed. Engineer. **30**, 125 (1983).

[10] J. S. Asvestas and D. C. Englund, Computing the solid angle subtended by a planar figure, Opt. Eng. **33**, 4055 (1994), e: [17].

[11] R. P. Gardner and K. Verghese, On the solid angle subtended by a circular disk, Nucl. Instr. Meth. **93**, 163 (1971).

[12] M. Abramowitz and I. A. Stegun, eds., *Handbook of Mathematical Functions*, 9th ed. (Dover Publications, New York, 1972).

[13] V. A. Shelyuto, Exact analytic results for the solid angle in systems with axial symmetry, J. Appl. Math. Phys. (ZAMP) **40**, 608 (1989).
[14] I. Gradstein and I. Ryshik, *Summen-, Produkt- und Integraltafeln*, 1st ed. (Harri Deutsch, Thun, 1981).
[15] P. F. Byrd and M. D. Friedman, *Handbook of Elliptical Integrals for Engineers and Physicists*, 2nd ed., Die Grundlehren der mathematischen Wissenschaften in Einzeldarstellungen, Vol. 67 (Springer, Berlin, Göttingen, 1971).
[16] GNU scientific library (2024), https://www.gnu.org/software/gsl/.
[17] J. S. Asvestas, Errata: Computing the solid angle subtended by a planar figure, Opt. Eng. **50**, 059801 (2011).