NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer’s or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.
Shape Factor Modeling and Simulation

by Richard Saucier
Survivability/Lethality Analysis Directorate, ARL
The dimensionless shape factor, which relates the projected area of a fragment to its mass per unit density, plays a fundamental role in ballistic penetration. Explicit analytical formulas are given for the shape factor distributions of some common shapes with random orientations. It is straightforward to simulate these shape factor distributions with computer code, and we verify that the simulations match the plots from the analytical formulas. However, none of the simple common shapes provides an adequate simulation model for natural fragments. We show that natural fragment data can be fit with a lognormal distribution, which then provides a simulation model for Monte Carlo sampling. Laser scans of fragments can also be used to compute the fragment shape factor from any viewpoint; various methods of achieving a uniform spherical distribution are described. Finally, we show that it is possible to realize each fragment as either a yawed cylinder or a cuboid with a pitch, yaw, and roll. Thus, we have a procedure for generating all the input variables required to run THOR or FATEPEN with natural fragments.

**Shape Factor Modeling and Simulation**

Richard Saucier

**US Army Research Laboratory**

ATTN: RDRL-SLB-S

Aberdeen Proving Ground, MD 21005-5068

Approved for public release; distribution is unlimited.

The dimensionless shape factor, which relates the projected area of a fragment to its mass per unit density, plays a fundamental role in ballistic penetration. Explicit analytical formulas are given for the shape factor distributions of some common shapes with random orientations. It is straightforward to simulate these shape factor distributions with computer code, and we verify that the simulations match the plots from the analytical formulas. However, none of the simple common shapes provides an adequate simulation model for natural fragments. We show that natural fragment data can be fit with a lognormal distribution, which then provides a simulation model for Monte Carlo sampling. Laser scans of fragments can also be used to compute the fragment shape factor from any viewpoint; various methods of achieving a uniform spherical distribution are described. Finally, we show that it is possible to realize each fragment as either a yawed cylinder or a cuboid with a pitch, yaw, and roll. Thus, we have a procedure for generating all the input variables required to run THOR or FATEPEN with natural fragments.

**Shape Factor Modeling and Simulation**

Richard Saucier

**US Army Research Laboratory**

ATTN: RDRL-SLB-S

Aberdeen Proving Ground, MD 21005-5068

Approved for public release; distribution is unlimited.

The dimensionless shape factor, which relates the projected area of a fragment to its mass per unit density, plays a fundamental role in ballistic penetration. Explicit analytical formulas are given for the shape factor distributions of some common shapes with random orientations. It is straightforward to simulate these shape factor distributions with computer code, and we verify that the simulations match the plots from the analytical formulas. However, none of the simple common shapes provides an adequate simulation model for natural fragments. We show that natural fragment data can be fit with a lognormal distribution, which then provides a simulation model for Monte Carlo sampling. Laser scans of fragments can also be used to compute the fragment shape factor from any viewpoint; various methods of achieving a uniform spherical distribution are described. Finally, we show that it is possible to realize each fragment as either a yawed cylinder or a cuboid with a pitch, yaw, and roll. Thus, we have a procedure for generating all the input variables required to run THOR or FATEPEN with natural fragments.

**Shape Factor Modeling and Simulation**

Richard Saucier

**US Army Research Laboratory**

ATTN: RDRL-SLB-S

Aberdeen Proving Ground, MD 21005-5068

Approved for public release; distribution is unlimited.
Contents

List of Figures v

List of Tables vii

List of Listings viii

Acknowledgments ix

1. Introduction 1

2. Shape Factor Distributions for 5 Convex Solids 5
   2.1 Cube 6
   2.2 Cuboid 7
   2.3 Cylinder 8
   2.4 Tetrahedron 9
   2.5 Ellipsoid 10

3. Shape Factor Distributions for Natural Fragments 12
   3.1 Platonic Solids and Uniform Viewing from All Viewpoints 12
   3.2 Natural Fragments from Artillery Rounds 14
   3.3 Natural Fragments from Spall 16
   3.4 Shape Factor Computation from Laser-Scanned Fragments 18
      3.4.1 Area Contribution from Each Facet 20
      3.4.2 Volume Contribution from Each Facet 21
      3.4.3 Total Surface Area and Total Volume 22
      3.4.4 Projected Area 22

4. Shape Factor Modeling 29
   4.1 Cylinder 30
   4.2 Cuboid 32

5. Conclusions and Recommendations 34

Approved for public release; distribution is unlimited.
6. References

Appendix A. Analytical Shape Factor Formulas for 5 Convex Solids 39

Appendix B. Uniform Sampling over the Unit Sphere 67

Appendix C. Some Properties of the Lognormal Distribution 77

List of Symbols, Abbreviations, and Acronyms 87

Distribution List 88
List of Figures

Fig. 1  Contour plot of limit velocity as a function of shape factor and mass.  ...4
Fig. 2  Shape factor histograms generated by cubesim.cpp.  .........................6
Fig. 3  Shape factor histograms generated by rppsim.cpp.  .........................7
Fig. 4  Shape factor histograms generated by cylsim.cpp.  .........................8
Fig. 5  Shape factor histograms generated by tetrasim.cpp.  .....................9
Fig. 6  Shape factor histograms generated by ellipsoidsim.cpp for 3/2/1 ellipsoid. 10
Fig. 7  Shape factor histograms generated by ellipsoidsim.cpp for 8/2/1 ellipsoid. 11
Fig. 8  Raytraced image made with BRL-CAD of laser-scanned natural fragment. 12
Fig. 9  The 5 Platonic solids.  .................................................................12
Fig. 10  Mean shape factor of artillery fragments as a function of mass. .........15
Fig. 11  Shape factor histograms for artillery fragments compared to lognormal. 15
Fig. 12  Shape factor histograms for spall fragments compared to lognormal.  17
Fig. 13  Laser-scanned natural fragment showing mesh of triangles covering the surface.  .................................................................18
Fig. 14  Shape factor histograms generated by STL description of a single fragment.  .................................................................25
Fig. 15  Shape factor histograms generated by STL descriptions of 15 fragments. 25
Fig. 16  Shape factor histograms for 15 fragments accounting for hidden surfaces.  .................................................................26
Fig. A-1  Shape factor distribution for a randomly oriented cube. ...............41
Fig. A-2  Shape factor distribution for a randomly oriented cuboid. .............46
Fig. A-3  Shape factors of a cylinder as a function of $L/D$  .................52
Fig. A-4  Shape factor distribution for a randomly oriented cylinder. .........56
Fig. A-5  Fast generation of randomly oriented cylinder shape factor compared to exact plot.  .................................................................57
Fig. A-6  Shape factor distribution for a randomly oriented regular tetrahedron. 60
Fig. A-7  Shape factor distribution for a randomly oriented ellipsoid. .........62
Fig. B-1  Sampling on the circumscribed cylinder.  ....................................68
Fig. B-2  Uniform random sampling over the unit sphere ..........................73
Fig. B-3  Stratified random sampling over the unit sphere ............................ 74
Fig. B-4  Spiral distribution over the unit sphere........................................... 75
Fig. B-5  Maximal avoidance sampling over the unit sphere ......................... 76
Fig. C-1  Mass per unit area histogram compared to theoretical distribution..... 85
List of Tables

Table 1  Shape factors of some common shapes and orientations ..........3
Table 2  Properties of the 5 Platonic solids.................................. 13
Table 3  Sequence of viewing angles in Icosahedron Gage ................. 14
Table 4  Comparison of lognormal fit to artillery data ....................... 16
Table 5  Comparison of lognormal fit to spall data .......................... 17
Table C-1 Properties of the lognormal distribution .......................... 78
<table>
<thead>
<tr>
<th>Listing 1</th>
<th>cubesim.cpp</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Listing 2</td>
<td>rppsim.cpp</td>
<td>7</td>
</tr>
<tr>
<td>Listing 3</td>
<td>cytsim.cpp</td>
<td>8</td>
</tr>
<tr>
<td>Listing 4</td>
<td>tetrasim.cpp</td>
<td>9</td>
</tr>
<tr>
<td>Listing 5</td>
<td>ellipsoidsim.cpp</td>
<td>10</td>
</tr>
<tr>
<td>Listing 6</td>
<td>lognormal.cpp</td>
<td>18</td>
</tr>
<tr>
<td>Listing 7</td>
<td>stlformat.cpp</td>
<td>18</td>
</tr>
<tr>
<td>Listing 8</td>
<td>tetrahedron.stla</td>
<td>19</td>
</tr>
<tr>
<td>Listing 9</td>
<td>stl.a.cpp</td>
<td>19</td>
</tr>
<tr>
<td>Listing 10</td>
<td>stl.b.cpp</td>
<td>23</td>
</tr>
<tr>
<td>Listing 11</td>
<td>stlarea.cpp</td>
<td>26</td>
</tr>
<tr>
<td>Listing 12</td>
<td>sf-rcc.cpp</td>
<td>32</td>
</tr>
<tr>
<td>Listing 13</td>
<td>sf-rpp.cpp</td>
<td>33</td>
</tr>
<tr>
<td>Listing A-1</td>
<td>cube.cpp</td>
<td>40</td>
</tr>
<tr>
<td>Listing A-2</td>
<td>rpp.cpp</td>
<td>47</td>
</tr>
<tr>
<td>Listing A-3</td>
<td>algo.cpp</td>
<td>56</td>
</tr>
<tr>
<td>Listing A-4</td>
<td>tetrahedron.cpp</td>
<td>59</td>
</tr>
<tr>
<td>Listing A-5</td>
<td>ellipsoid.cpp</td>
<td>62</td>
</tr>
<tr>
<td>Listing A-6</td>
<td>rf.cpp</td>
<td>63</td>
</tr>
<tr>
<td>Listing A-7</td>
<td>rj.cpp</td>
<td>64</td>
</tr>
<tr>
<td>Listing A-8</td>
<td>rc.cpp</td>
<td>65</td>
</tr>
<tr>
<td>Listing B-1</td>
<td>uniform.cpp</td>
<td>69</td>
</tr>
<tr>
<td>Listing B-2</td>
<td>strat.cpp</td>
<td>69</td>
</tr>
<tr>
<td>Listing B-3</td>
<td>spiral.cpp</td>
<td>71</td>
</tr>
<tr>
<td>Listing B-4</td>
<td>avoidance.cpp</td>
<td>71</td>
</tr>
<tr>
<td>Listing C-1</td>
<td>mu.cpp</td>
<td>84</td>
</tr>
</tbody>
</table>
Acknowledgments

I would like to thank John Abell for reviewing this report, suggesting a number of improvements, and pointing out a number of errors. I also would like to thank technical editor Jessica Schultheis for her careful and thorough editing. Of course, I accept full responsibility for any errors that may remain.
INTENTIONALLY LEFT BLANK.
1. Introduction

Fragment shape factor $\gamma$ is defined by the equation

$$A_p = \gamma \left( \frac{m}{\rho} \right)^{2/3},$$

(1)

where $A_p$ is presented area, $m$ is mass, and $\rho$ is material density. The presented (or projected) area is the area that would be projected in silhouette on a screen from a distant light source, and could very well change with orientation. The mass divided by the density is the fragment’s volume. We raise it to the two-thirds power so that it has the same dimensions as the presented area (square meters in SI units). This makes the shape factor $\gamma$ dimensionless, which means that it is independent of any units. More importantly, the shape factor as defined in Eq. 1 is also independent of the fragment’s density, so that the shape factor of a steel cube has the same value as a tungsten cube or an aluminum cube. It is also a specific quantity, independent of the volume of the fragment. This will prove useful later when we compare shape factors for different masses.

It is important to recognize that there are at least 3 other shape factor definitions in common use in the ballistics community:

- **Shape factor $s$** defined by $A_p = s m^{2/3}$, where $A_p$ is in units of inch$^2$, $m$ is in units of grains, and $s$ is in units of inch$^2$/gr$^{2/3}$.

- **Shape factor $K$** defined by $A_p = K W^{2/3}$, where $A_p$ is in units of ft$^2$, $W$ is in units of pounds, and $K$ is in units of ft$^2$ gr$^{1/3}$/lb.

- **Shape factor $k$** defined by $A_p = (m/k)^{2/3}$, where $A_p$ is in units of inch$^2$, $m$ is in units of grains, and $k$ is in units of gr/inch$^3$.

Notice that these are all dimensional shape factors. Using the conversion factors 1 inch = 2.54 cm, 1 lb = 7000 gr, and 1 g = 15.4324 gr,

- the conversion between $\gamma$ and $s$ is $s = 0.025 \frac{\gamma}{\rho^{2/3}}$,

- the conversion between $\gamma$ and $K$ is $K = 0.025 \frac{7000}{144} \frac{\gamma}{\rho^{2/3}}$.
• and the conversion between $\gamma$ and $k$ is $k = 252.9 \frac{\rho}{\gamma^{3/2}}$,

where $\rho$ is the material density in units of g/cm$^3$. For example, a steel cube ($\rho = 7.83$ g/cm$^3$) with a random orientation has a mean shape factor of

• $\gamma = 3/2$,
• $s = 0.0095$ inch$^2$/gr$^{2/3}$,
• $K = 0.4623$ ft$^2$ gr$^{1/3}$/lb, and
• $k = 1077.8$ gr/inch$^3$.

And a tungsten cube ($\rho = 17.6$ g/cm$^3$) with a random orientation has a mean shape factor of

• $\gamma = 3/2$,
• $s = 0.0055$ inch$^2$/gr$^{2/3}$,
• $K = 0.2694$ ft$^2$ gr$^{1/3}$/lb, and
• $k = 2422.8$ gr/inch$^3$.

The dimensionless shape factor $\gamma$ is much simpler and less error prone than the others since it only depends upon the shape and orientation, but is completely independent of material density.

Shape factors for some common shapes and orientations can be worked out from the definition embodied in Eq. 1 and simple geometry. Some of these are listed in Table 1.
Table 1. Shape factors of some common shapes and orientations

<table>
<thead>
<tr>
<th>Shape</th>
<th>Orientation</th>
<th>Shape Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sphere</td>
<td>All</td>
<td>(3/2)^2/3(\pi/4)^1/3 ≈ 1.209</td>
</tr>
<tr>
<td>Cube</td>
<td>Face Forward</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Edge Forward</td>
<td>\sqrt{2} ≈ 1.414</td>
</tr>
<tr>
<td></td>
<td>Corner Forward</td>
<td>\sqrt{3} ≈ 1.732</td>
</tr>
<tr>
<td></td>
<td>Minimum</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Maximum</td>
<td>\sqrt{3} ≈ 1.732</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>3/2</td>
</tr>
<tr>
<td></td>
<td>Median</td>
<td>7\sqrt{3}/8 ≈ 1.516</td>
</tr>
<tr>
<td>3/2/1 Cuboid</td>
<td>Largest Face Forward</td>
<td>1.817</td>
</tr>
<tr>
<td></td>
<td>Intermediate Face Forward</td>
<td>0.909</td>
</tr>
<tr>
<td></td>
<td>Smallest Face Forward</td>
<td>0.606</td>
</tr>
<tr>
<td></td>
<td>Minimum</td>
<td>0.606</td>
</tr>
<tr>
<td></td>
<td>Maximum</td>
<td>2.120</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>5.5/6^{2/3} ≈ 1.666</td>
</tr>
<tr>
<td></td>
<td>Median</td>
<td>1.745</td>
</tr>
<tr>
<td>L/D=1 Cylinder</td>
<td>Face Forward</td>
<td>(\pi/4)^{1/3} ≈ 0.923</td>
</tr>
<tr>
<td></td>
<td>Side Forward</td>
<td>(\pi/4)^{-2/3} ≈ 1.175</td>
</tr>
<tr>
<td></td>
<td>Minimum</td>
<td>(\pi/4)^{1/3} ≈ 0.923</td>
</tr>
<tr>
<td></td>
<td>Maximum</td>
<td>(\pi/4)^{-2/3}\sqrt{(\pi/4)^2 + 1} ≈ 1.494</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>(3/2)(\pi/4)^{1/3} ≈ 1.384</td>
</tr>
<tr>
<td></td>
<td>Median</td>
<td>1.416</td>
</tr>
<tr>
<td>Regular Tetrahedron</td>
<td>Face/Corner Forward</td>
<td>1.801</td>
</tr>
<tr>
<td></td>
<td>Edge Forward</td>
<td>2.080</td>
</tr>
<tr>
<td></td>
<td>Minimum</td>
<td>1.471</td>
</tr>
<tr>
<td></td>
<td>Maximum</td>
<td>2.080</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>1.801</td>
</tr>
<tr>
<td></td>
<td>Median</td>
<td>1.775</td>
</tr>
<tr>
<td>3/2/1 Ellipsoid</td>
<td>Minimum</td>
<td>0.732</td>
</tr>
<tr>
<td></td>
<td>Mode</td>
<td>1.098</td>
</tr>
<tr>
<td></td>
<td>Maximum</td>
<td>2.197</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>1.424</td>
</tr>
<tr>
<td></td>
<td>Median</td>
<td>1.368</td>
</tr>
</tbody>
</table>

Shape factor plays a fundamental role in ballistic penetration. Two simple examples will serve to illustrate this.*

- A 725-gr steel cylinder with a shape factor \( \gamma = 0.72 \) \((L/D = 1.4506)\), when striking a 1/4-inch mild steel plate, has a limit velocity \( v_L = 1162 \) f/s. If

*All these results were obtained by running the FATEPEN model.1,2

Approved for public release; distribution is unlimited.
we reduce the shape factor by 50% ($\gamma = 0.36$ and $L/D = 4.1029$) without changing the mass, then the limit velocity becomes $v_L = 943$ f/s, a 19% reduction in limit velocity. On the other hand, if we keep the same shape factor of 0.72, then the mass would have to increase to 1385 gr, a 91% mass increase, in order to achieve the same reduction in limit velocity.

- A 25-g steel cylinder with a shape factor $\gamma = 0.86$ ($L/D = 1.11$), when striking a 16-mm face-hardened steel plate, has a limit velocity $v_L = 3621$ f/s. The same mass with a shape factor $\gamma = 0.43$ ($L/D = 3.14$) has a limit velocity $v_L = 3263$ f/s, a 10% reduction in limit velocity. If we leave the shape factor at 0.43, then the mass would have to be increased to 32 g to get a limit velocity of 3263 f/s, which represents a 28% increase in mass.

These examples illustrate that a decrease in shape factor is comparable to an increase in striking mass. In the first case, a 50% reduction in shape factor was comparable to a 91% increase in mass, and in the second, a 50% reduction in shape factor was comparable to a 28% increase in mass. So the shape factor can be more or less sensitive than the mass in influencing the limit velocity. But the important point is that to accurately determine penetration, we need to know the shape factor of the penetrator, much like the mass. This is further illustrated in Fig. 1.

![Fig. 1. Contour plot of limit velocity as a function of shape factor and mass for steel fragments impacting a 1/4-inch mild steel plate using FATEPEN](image-url)
Another point worth emphasizing is that penetration depends upon the instantaneous shape factor at impact, not the average value over all orientations. Unless the fragment is tumbling while it is penetrating—which is highly unlikely in metal—we need to use the shape factor at impact. One may argue that the THOR penetration model makes use of the average presented area, but a moment’s reflection should convince us that it is a mistake to use the average value. Consider, for example, a long thin cylinder. In face-forward orientation, it is a very effective penetrator—not so in side-forward orientation. If we average over all orientations, we may find that the average value does not perforate. It would be a mistake to conclude, therefore, that there is no perforation for any orientation—an example of averaging too soon. Of course, it is not necessarily easy to measure the impact presented area. For convex solids, Cauchy’s theorem tells us that the average presented area of a convex solid is one-fourth the total surface area. This allows us to compute the average presented area very easily from the total surface area and may have been the reason why average shape factor was used in the THOR equations.

2. Shape Factor Distributions for 5 Convex Solids

Now let us consider the 5 shapes in Table 1 that depend upon orientation (i.e., excluding the sphere), and let us consider a random viewpoint that is uniformly distributed over a sphere. We can imagine the solid fixed at the origin while we take random viewpoints on a sphere enclosing the solid, and from each viewpoint we compute the projected area on a distant screen perpendicular to that viewpoint. Analytical formulas for the projected area probability density function (PDF) and cumulative distribution function (CDF) are known for each of these shapes and are given in Appendix A, while a variety of methods of sampling over the unit sphere are described in Appendix B. Recall that the PDF is simply the derivative of the CDF, and the CDF always ranges from 0 to 1, which means that the area under the full range of the PDF must equal 1.

Some of the formulas in Appendix A are in terms of the projected area. To convert from the projected area distribution to the shape factor distribution, we make use of Eq. 1 and the chain rule to get

\[ f(\gamma) = \frac{dF}{d\gamma} = \frac{dF}{dA_p} \frac{dA_p}{d\gamma} = V^{2/3} f(A_p), \]

where \( f \) is the PDF, \( F \) is the CDF, and \( V \) is the fragment volume. So we see that it is
easy to convert a projected area distribution to a shape factor distribution by simple scaling.

Sample plots of the PDF and CDF for each of the 5 shapes are shown in Subsections 2.1–2.5. In each case we also show histograms that have been generated with the listed simulation code.

## 2.1 Cube

The shape factor distribution from a randomly oriented cube can be simulated with the code in Listing 1.

### Listing 1. cubesim.cpp

```cpp
// cubesim.cpp: simulate shape factor distribution of a cube

#include "Random.h"
#include <iostream>
#include <cstdlib>
#include <cmath>

int main( int argc, char* argv[] ) {

    int N = 1000; // default number of samples or specify on command line
    if ( argc == 2 ) N = atoi( argv[1] ); // number of samples

    rng::Random rng;
    double x, y, z, sf;
    for ( int i = 0; i < N; i++ ) {
        rng.spherical_avoidance( x, y, z );
        sf = fabs( x ) + fabs( y ) + fabs( z );
        std::cout << sf << std::endl;
    }

    return EXIT_SUCCESS;
}
```

The simulated shape factor distribution is compared to the analytical formulas in Fig. 2.

![Fig. 2. Histograms of shape factor PDF and CDF for a randomly oriented cube compared to analytical formulas, Eqs. A-1 and A-2 (black curves)](image-url)
2.2 Cuboid

The shape factor distribution from a randomly oriented cuboid—also known as a rectangular parallelepiped (RPP)—can be simulated with the code in Listing 2.

Listing 2. rppsim.cpp

```cpp
// rppsim.cpp

#include "Random.h"
#include <iostream>
#include <cstdlib>
#include <cmath>

int main( int argc, char* argv[] ) {
    int N = 1000; // default number of samples
    // default is a cube with L = W = D = 1, or specify dimensions on command line
    double L = 1., W = 1., T = 1.;
    if ( argc == 4 ) {
        L = atof( argv[1] );
        W = atof( argv[2] );
        T = atof( argv[3] );
    }
    else if ( argc == 5 ) {
        L = atof( argv[1] );
        W = atof( argv[2] );
        T = atof( argv[3] );
        N = atoi( argv[4] ); // number of samples
    }
    double a = W * T;
    double b = T * L;
    double c = L * W;
    const double V = L * W * T;
    const double F = pow( V, -2. / 3. ); // factor to convert area to shape factor
    rng::Random rng;
    double x, y, z, ap, sf;
    for ( int i = 0; i < N; i++ ) {
        rng.spherical_avoidance( x, y, z );
        ap = ( a * fabs( x ) + b * fabs( y ) + c * fabs( z ) );
        sf = ap * F;
        std::cout << sf << std::endl;
    }
    return EXIT_SUCCESS;
}
```

The simulated shape factor distribution is compared to the analytical formulas in Fig. 3.

Fig. 3. Histograms of shape factor PDF and CDF for a randomly oriented $L = 3, W = 2, T = 1$ cuboid compared to analytical formulas (black curve)

Approved for public release; distribution is unlimited.
2.3 Cylinder

The shape factor distribution from a randomly oriented cylinder can be simulated with the code in Listing 3.

Listing 3. cylsim.cpp

```cpp
#include "Random.h"
#include <iostream>
#include <cstdlib>
#include <cmath>

int main( int argc, char* argv[] ) {
  int N = 1000; // default number of samples or specify on command line
  double l_d = 1.; // default is a L/D = 1 cylinder, or specify on command line

  if ( argc == 2 )
    N = atoi( argv[1] ); // number of samples
  else if ( argc == 3 ) {
    l_d = atof( argv[1] ); // L/D
    N = atoi( argv[2] ); // number of samples
  }

  const double C = pow( M_PI_4 * l_d, -2./3. );
  rng::Random rng;
  double th, ph, sf;
  for ( int i = 0; i < N; i++ ) {
    rng.spherical_avoidance( th, ph );
    sf = C * ( l_d * sin( th ) + M_PI_4 * fabs( cos( th ) ) );
    std::cout << sf << std::endl;
  }
  return EXIT_SUCCESS;
}
```

The simulated shape factor distribution for an $L/D = 1$ cylinder is compared to the analytical formulas in Fig. 4.

Fig. 4. Histograms of shape factor PDF and CDF for a randomly oriented $L/D = 1$ cylinder compared to analytical formulas. Notice the jump in the PDF at $\gamma = (\pi/4)^{-2/3} \approx 1.175$, as predicted (see Appendix A).
2.4 Tetrahedron

The shape factor distribution from a randomly oriented regular tetrahedron can be simulated with the code in Listing 4.

```
// tetrasim.cpp

#include "Vector.h"
#include "Random.h"
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <iomanip>
using namespace std;

int main( int argc, char* argv[] ) {
    const double A = 1. / sqrt( 3. );
    const va::Vector norm[4] = {
        va::Vector( +A, -A, +A ),
        va::Vector( +A, +A, -A ),
        va::Vector( -A, +A, +A ),
        va::Vector( -A, -A, -A )
    };
    rng::Random rng;
    double x, y, z, dotprod, ap, sf;
    va::Vector u;
    const double A_FACE = sqrt( 3. ) / 2.;
    const double VOL = 1. / 3.;
    int N = 1000; // default number of samples or specify on command line
    if ( argc == 2 ) N = atoi( argv[1] );
    cout << std::setprecision(6) << std::fixed;
    for ( int n = 0; n < N; n++ ) {
        ap = 0.;
        rng.spherical_avoidance( x, y, z );
        u = va::Vector( x, y, z );
        for ( int i = 0; i < 4; i++ ) if ( ( dotprod = u * norm[i] ) > 0. ) ap += A_FACE * dotprod;
        sf = ap * pow( VOL, -2./3. );
        std::cout << sf << std::endl;
    }
    return EXIT_SUCCESS;
}
```

The simulated shape factor distribution for a regular tetrahedron is compared to the analytical formulas in Fig. 5.

![Fig. 5. Histograms of shape factor PDF and CDF for a randomly oriented regular tetrahedron compared to analytical formulas](image)

Approved for public release; distribution is unlimited.
2.5 Ellipsoid

Listing 5 is an implementation of a simulation of the shape factor for an ellipsoid.

```cpp
// ellipsoidsim.cpp: Simulate the shape factor of a randomly oriented ellipsoid

#include "Random.h"
#include <iostream>
#include <cstdlib>
#include <cmath>

#include <cmath>

inline double min( double a, double b, double c ) { return std::min( std::min( a, b ), c ); }
inline double max( double a, double b, double c ) { return std::max( std::max( a, b ), c ); }
inline double mid( double a, double b, double c ) { return std::max( std::min( a, b ), std::min( std::max( a, b ), c ) ); }

int main( int argc, char* argv[] ) {
    unsigned int N = 1000; // default number of samples or specify as 4th argument on command line
    double a = 1., b = 1., c = 1.; // default shape is a sphere

    if ( argc == 4 ) { // or specify the 3 dimensions in any order on the command line
        a = atof( argv[1] );
        b = atof( argv[2] );
        c = atof( argv[3] );
    } else if ( argc == 5 ) { // specify number of samples as 4th argument
        a = atof( argv[1] );
        b = atof( argv[2] );
        c = atof( argv[3] );
        N = atoi( argv[4] );
    }

    const double A = min( a, b, c ); // minimum value
    const double B = mid( a, b, c ); // intermediate value
    const double C = max( a, b, c ); // maximum value
    const double V = ( 4. / 3. ) * M_PI * A * B * C; // ellipsoid volume
    const double F = pow( V, -2. / 3. ); // factor to convert area to shape factor

    double x, y, z, X, Y, Z, ap, sf;
    rng::Random rng;

    for ( unsigned int n = 0; n < N; n++ ) {
        rng.spherical_avoidance( x, y, z );
        X = B * C * x;
        Y = A * C * y;
        Z = A * B * z;
        ap = M_PI * sqrt( X * X + Y * Y + Z * Z );
        sf = ap * F;
        std::cout << sf << std::endl;
    }
    return EXIT_SUCCESS;
}
```

Running this code for a 3/2/1 ellipsoid gives the results shown in Fig. 6.

![Histograms of shape factor PDF and CDF for a randomly oriented ellipsoid with a = 1, b = 2, c = 3. The solid curve is a plot of the analytical formula.](image)

Fig. 6. Histograms of shape factor PDF and CDF for a randomly oriented ellipsoid with $a = 1$, $b = 2$, $c = 3$. The solid curve is a plot of the analytical formula.
Running the code for a 8/2/1 ellipsoid gives the results shown in Fig. 7, just to show how the side length ratios shift the plots. We will see later that the distribution from a range of ellipsoid shapes begins to resemble the shape factor distribution from natural fragments.

![Fig. 7. Histograms of shape factor PDF and CDF for a randomly oriented ellipsoid with $a = 1$, $b = 2$, $c = 8$. The solid curve is the theoretical distribution.](image)

Given the semi-principal axes lengths $a$, $b$, $c$, of the ellipsoid, where $a \leq b \leq c$, the projected areas are

$$A_{\text{min}} = \pi ab, \quad A_m = \pi ac, \quad A_{\text{max}} = \pi bc, \quad \text{and} \quad V = \frac{4}{3}\pi abc. \quad (3)$$

Or, if we know $A_{\text{min}}$, $A_m$, $A_{\text{max}}$, where $A_{\text{min}} \leq A_m \leq A_{\text{max}}$, then the lengths are

$$a = \sqrt[3]{\frac{A_{\text{min}}A_m}{\pi A_{\text{max}}}}, \quad b = \sqrt[3]{\frac{A_{\text{min}}A_{\text{max}}}{\pi A_m}}, \quad c = \sqrt[3]{\frac{A_{\text{max}}A_m}{\pi A_{\text{min}}}},$$

and

$$V = \frac{4}{3}\sqrt[3]{\frac{A_{\text{min}}A_mA_{\text{max}}}{\pi}}. \quad (4)$$

Therefore, if we are given the minimum, mode, and maximum shape factors, $\gamma_{\text{min}}$, $\gamma_m$, $\gamma_{\text{max}}$, along with the ellipsoid volume, then we can compute

$$A_{\text{min}} = \gamma_{\text{min}}V^{2/3}, \quad A_m = \gamma_mV^{2/3}, \quad A_{\text{max}} = \gamma_{\text{max}}V^{2/3}, \quad (5)$$

and use Eq. 4 to compute the ellipsoid dimensions.

Notice carefully the shape of the CDF curves for each of these shapes. We will see that none of them resembles the CDF for a randomly oriented natural fragment.
which indicates that there is no simple randomly oriented shape that will suffice as a model for a natural fragment. What we will propose instead is to first characterize the probability distribution of natural fragments and find a way to sample from this distribution.

3. Shape Factor Distributions for Natural Fragments

Natural fragments resulting from artillery rounds have been collected in many munition tests over the years. Unlike the shapes we have considered so far, these fragments tend to have highly irregular shapes, an example of which is shown in Fig. 8.

![Raytraced image made with BRL-CAD of SF1290, a laser-scanned natural fragment](image)

We make the assumption that when a fragment impacts a target surface, it has a random orientation and that each orientation is equally likely. Thus, we need a way to sample over all directions in an unbiased manner. This was handled in the past by making use of Platonic solids.

3.1 Platonic Solids and Uniform Viewing from All Viewpoints

Platonic solids are 3-dimensional (3D) regular, convex polyhedra, and there are only 5, as shown in Fig. 9.

![The 5 Platonic solids, from left to right, are tetrahedron, cube or hexahedron, octahedron, dodecahedron, and icosahedron](image)
Regular in this context means that each face has the same area, which is the key property for our purposes. Convex means that if we connect any point on the inside with any point on the outside with a straight line, then it will cross the surface only once. The number of faces and vertices of the solids are listed in Table 2.

<table>
<thead>
<tr>
<th>Solid</th>
<th>Number of Faces</th>
<th>Number of Vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tetrahedron</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Cube</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>Octahedron</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>Dodecahedron</td>
<td>12</td>
<td>20</td>
</tr>
<tr>
<td>Icosahedron</td>
<td>20</td>
<td>12</td>
</tr>
</tbody>
</table>

The dual of a Platonic solid is one in which the positions of the face centers and the positions of vertices are switched—and is also a Platonic solid. The tetrahedron is self-dual, and the hexahedron and octahedron are duals of one another, as are the dodecahedron and the icosahedron. The vertices of a Platonic solid are equally spaced about a circumscribed sphere, so that makes them ideal candidates for unbiased projected area viewpoints.

The Icosahedron Gage is a measuring instrument that uses as viewpoints the vertices of both the icosahedron and the dodecahedron. This gives it 12 + 20 = 32 viewpoints, but for projected areas, half of these viewpoints are redundant, so that leaves 16 viewpoints.* Table 3 lists the angles of the Icosahedron Gage.

---

*Notice, however, that when we do this, we no longer have equal spacing between viewpoints. When a Platonic solid and its dual are combined, some points have 3 closest neighbor vertices while other points have 5. So the rotational symmetry is spoiled. There is simply no way to distribute more than 20 points over the unit sphere and maintain equal spacing between nearest neighbors. If it were possible, then we would have a new Platonic solid, but it has been proven that there are only 5.

Approved for public release; distribution is unlimited.
Table 3. Sequence of viewing angles in Icosahedron Gage

<table>
<thead>
<tr>
<th>Position</th>
<th>Platonic Solid</th>
<th>Azimuthal Angle (°)</th>
<th>Elevation Angle (°)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Icosahedron</td>
<td>0</td>
<td>90</td>
</tr>
<tr>
<td>2</td>
<td>Dodecahedron</td>
<td>0</td>
<td>52.6226</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>72</td>
<td>52.6226</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>144</td>
<td>52.6226</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>216</td>
<td>52.6226</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>288</td>
<td>52.6226</td>
</tr>
<tr>
<td>7</td>
<td>Icosahedron</td>
<td>324</td>
<td>26.5651</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>36</td>
<td>26.5651</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>108</td>
<td>26.5651</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>180</td>
<td>26.5651</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>252</td>
<td>26.5651</td>
</tr>
<tr>
<td>12</td>
<td>Dodecahedron</td>
<td>288</td>
<td>10.8123</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>0</td>
<td>10.8123</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>72</td>
<td>10.8123</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>144</td>
<td>10.8123</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>216</td>
<td>10.8123</td>
</tr>
</tbody>
</table>

Projected area measurements have been performed with early versions of the Icosahedron Gage since the 1940s. The instrument that is used today is coupled to a personal computer, which greatly automates the process.\textsuperscript{6,7}

3.2 Natural Fragments from Artillery Rounds

Close to 900 steel fragments from artillery rounds have been collected.\textsuperscript{8–11} Each fragment mass was measured along with 16 projected areas with an Icosahedron Gage, which allows us to compute 16 shape factor values for each fragment.

If we plot the mean shape factor (averaged over the 16 individual measurements) as a function of fragment mass, we find that there is essentially no correlation between fragment mass and average shape factor, as shown in Fig. 10.
Since there is essentially no correlation between shape factor and mass, we are justified in pooling all of the shape factors, which then gives us a sample size of $898 \times 16 = 14,368$ shape factors.

So, although we started out with only 16 shape factors for each irregular fragment, we are able to exploit the fact that shape factor is independent of mass to effectively come up with over 14,000 shape factor measurements. We find that the resulting shape factor distribution closely approximates a lognormal distribution, as shown in Fig. 11.

---

*For example, we might have expected that smaller fragments would be more compact than larger fragments, but that is not what we see in the data.

---

Approved for public release; distribution is unlimited.
The lognormal PDF is given by

\[ f(x) = \frac{1}{\sqrt{2\pi} \sigma x} \exp\left[ -\frac{(\ln x - \mu)^2}{2\sigma^2} \right] \] (6)

and the CDF is given by

\[ F(x) = \frac{1}{2} \left( 1 + \operatorname{erf}\left( \frac{\ln x - \mu}{\sqrt{2} \sigma} \right) \right). \] (7)

The maximum likelihood estimation* of parameters gives \( \mu = 0.596514 \) and \( \sigma = 0.340874 \). The geometric mean is \( \gamma_g = e^\mu = 1.81578 \) and the geometric standard deviation is \( \sigma_g = e^\sigma = 1.40618 \).† The comparison between the lognormal fit and the data is summarized in Table 4.

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Lognormal Fit</th>
<th>Artillery Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median</td>
<td>( e^\mu = 1.82 )</td>
<td>1.81</td>
</tr>
<tr>
<td>Mean</td>
<td>( e^{\mu + \sigma^2/2} = 1.92 )</td>
<td>1.93</td>
</tr>
<tr>
<td>Mode</td>
<td>( e^{\mu - \sigma^2} = 1.62 )</td>
<td>1.62</td>
</tr>
</tbody>
</table>

Whereas a normal distribution has the property that \( \mu \pm x \) are equally likely, a lognormal distribution has the property that \( xe^{\mu - \sigma^2} \) and \( \frac{1}{x}e^{\mu - \sigma^2} \) are equally likely, for any value \( x \neq 0 \). See Appendix C for some more properties of the lognormal distribution.

### 3.3 Natural Fragments from Spall

Spall fragments‡ were also collected in Celotex and subsequently measured for mass and projected area with an Icosahedron Gage. In this case there were 250 spall fragments, giving \( 250 \times 16 = 4000 \) shape factors. These data are displayed in Fig. 12 and again compared to a lognormal fit.

---

*Maximum likelihood estimation is easy to perform for a lognormal distribution since \( \mu \) and \( \sigma \) are respectively the mean and the standard deviation of the logs of the data.

†The geometric mean and geometric standard deviation make it easy to summarize the distribution. Thus, 68% is contained in \([\gamma_g \sigma_g^{-1}, \gamma_g \sigma_g]\) and 95% in \([\gamma_g \sigma_g^{-2}, \gamma_g \sigma_g^2]\).

‡Spall fragments are pieces of armor that are broken off during penetrator impact.
Maximum likelihood estimation gives $\mu = 0.456095$ and $\sigma = 0.479386$. The geometric mean is $\gamma_g = e^\mu = 1.5779$ and the geometric standard deviation is $\sigma_g = e^\sigma = 1.61508$. The comparison between the lognormal fit and the data is summarized in Table 5.

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Lognormal Fit</th>
<th>Spall Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median</td>
<td>$e^\mu = 1.58$</td>
<td>1.50</td>
</tr>
<tr>
<td>Mean</td>
<td>$e^{\mu + \sigma^2/2}$ = 1.77</td>
<td>1.79</td>
</tr>
<tr>
<td>Mode</td>
<td>$e^{\mu - \sigma^2}$ = 1.25</td>
<td>1.25</td>
</tr>
</tbody>
</table>

If we look back and compare the PDF for the simple shapes of cube, cuboid, cylinder, tetrahedron, and ellipsoid (Figs. 1–6) to the lognormal, it is clear that none of these shapes come close. Nor are we likely to find any simple shape that will reproduce the distribution of shape factors from natural fragments by merely randomizing the orientation. However, we have found that the lognormal distribution offers a lot of promise of simulating the shape factor. It is not necessary that we have a specific shape in order to simulate the shape factor; we only need to sample a lognormal distribution each time we need a sample of the shape factor (for example, at impact with a target).

Listing 6 is a simple program that will generate the lognormally distributed shape factors.
3.4 Shape Factor Computation from Laser-Scanned Fragments

Another technique that has been used more recently for measuring fragment shape is laser scanning. This will generate a facetized solid in stereolithography (STL) format. An example is shown in Fig. 13.

![Laser-scanned natural fragment showing mesh of 470,988 triangles covering the surface. It is obvious that the fragment is not convex, which means that the many hidden surfaces need to be accounted for when computing the projected area.](image)

Listing 7 shows the format for an STL file.

```cpp
// stlformat.cpp
solid name
facet normal n1 n2 n3
  outer loop
  vertex v1x v1y v1z
  vertex v2x v2y v2z
  vertex v3x v3y v3z
  endloop
endfacet
endsolid
```
Thus, each facet is a triangle, specified by 4 vectors:

- an outward normal vector, which follows the right-hand rule* and
- one vector for each of its 3 vertices

For example, Listing 8 is the STL file (in ASCII format†) for a regular tetrahedron.

```
Listing 8. tetrahedron.stla

solid TETRAHEDRON
facet normal 0.57735 -0.57735 0.57735
  outer loop
  vertex 0 0 1
  vertex 1 0 0
  vertex 1 1 1
  endloop
endfacet

facet normal 0.57735 0.57735 -0.57735
  outer loop
  vertex 1 1 1
  vertex 1 0 0
  vertex 0 1 0
  endloop
endfacet

facet normal -0.57735 0.57735 0.57735
  outer loop
  vertex 1 1 1
  vertex 0 1 0
  vertex 0 0 1
  endloop
endfacet

facet normal -0.57735 -0.57735 -0.57735
  outer loop
  vertex 0 0 1
  vertex 0 1 0
  vertex 1 0 0
  endloop
endfacet
endsolid TETRAHEDRON
```

The format is somewhat redundant in that the outward normal can be computed from the 3 vertices and is therefore not strictly required. Indeed, some software applications do not specify the outward normal in the STL file, so one must be careful to check the normals. Listing 9 is a program to read in an STL file in ASCII format and print summary information.

```
Listing 9. stl.a.cpp

// stl.a.cpp: reads in an ASCII STL description of a solid and computes volume and surface area

#include <iostream>
#include <cstdlib>
#include <string>
#include <cmath>
#include <iomanip>

∗The right-hand rule specifies that if the fingers of the right hand curl in the direction of the corners of the triangle from its first to second to third point, then the thumb will be pointing in the direction of the outward normal.

†There is also a binary STL format, which we shall use in Listing 10.

Approved for public release; distribution is unlimited.

19
3.4.1 Area Contribution from Each Facet

Implicit in the specification of the triangle is an origin for its vector vertices, but the origin itself is not explicitly specified in the STL file. It is here that the orientation of the triangle comes to our aid, since it turns out that the origin is not necessary to compute the triangle’s area.

Using the fact that the area of a triangle is one-half the area of the corresponding parallelogram, the area of the triangle can be written as

\[ A = \frac{1}{2} \hat{n} \cdot [(\mathbf{v}_2 - \mathbf{v}_1) \times (\mathbf{v}_3 - \mathbf{v}_1)], \]  

where \( \hat{n} \) is the outward normal, and \( \mathbf{v}_1, \mathbf{v}_2, \) and \( \mathbf{v}_3 \) are the 3 vector vertices. Expand-
ing this out provides another formula:

\[ A = \frac{1}{2} \hat{n} \cdot [\mathbf{v}_1 \times \mathbf{v}_2 + \mathbf{v}_2 \times \mathbf{v}_3 + \mathbf{v}_3 \times \mathbf{v}_1]. \]  

Interpreting this geometrically, it is easy to see that each of the 3 terms is positive if the projection of the origin lies inside the triangle, but that one of these terms will be negative if the projected origin lies outside the triangle. The net result is always the triangle area. Equation 8 is slightly more efficient in computer code than Eq. 9 since it requires 2 vector subtractions and only 1 vector cross product, as opposed to 2 additions and 3 cross products. Since the vector cross product in Eq. 8 points in the same direction as the outward normal, the area of the triangle can also be computed from the magnitude of the cross product:

\[ A = \frac{1}{2} \| (\mathbf{v}_2 - \mathbf{v}_1) \times (\mathbf{v}_3 - \mathbf{v}_1) \|. \]  

We have implemented 3D vectors and their associated algebra directly into software as a C++ class, so that Eqs. 8 and 10 can be coded directly. Alternatively, these formulas can be reduced to scalar formulas by using

\[ \mathbf{a} \times \mathbf{b} = \det \begin{bmatrix} \hat{i} & \hat{j} & \hat{k} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{bmatrix} = \hat{i}(a_yb_z - a_zb_y) + \hat{j}(a_zb_x - a_xb_z) + \hat{k}(a_xb_y - a_yb_x), \]  

where \( \hat{i}, \hat{j}, \) and \( \hat{k} \) are 3 unit vectors along the \( x, y, \) and \( z \) axes, respectively. Therefore, the area can be written as

\[ A = \frac{1}{2} \sqrt{(a_yb_z - a_zb_y)^2 + (a_zb_x - a_xb_z)^2 + (a_xb_y - a_yb_x)^2}, \]  

or, since \( \hat{n} \) is a unit vector normal to the facet,

\[ A = \frac{1}{2} [n_x(a_yb_z - a_zb_y) + n_y(a_zb_x - a_xb_z) + n_z(a_xb_y - a_yb_x)]. \]  

### 3.4.2 Volume Contribution from Each Facet

Each facet, combined with another fixed point \( \mathbf{d} \), makes up a tetrahedron. Let \( \mathbf{a} = \mathbf{v}_1 - \mathbf{d} \), \( \mathbf{b} = \mathbf{v}_2 - \mathbf{d} \), and \( \mathbf{c} = \mathbf{v}_3 - \mathbf{d} \), then the volume of this tetrahedron is the
scalar or triple product
\[ V = \frac{1}{6} \mathbf{a} \cdot (\mathbf{b} \times \mathbf{c}). \]  
(14)

The fixed point \( \mathbf{d} \) is arbitrary, so we might as well choose the origin, in which case
the formula for the volume is simply
\[ V = \frac{1}{6} \mathbf{v}_1 \cdot (\mathbf{v}_2 \times \mathbf{v}_3). \]  
(15)

This triple product is symmetric in all 3 vertices, but the order is important since this is an *oriented volume*. Once again, we can code this directly in the C++ Vector Class, or we can make use of the following formula:
\[ \mathbf{a} \times \mathbf{b} = \det \begin{bmatrix} \hat{i} & \hat{j} & \hat{k} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{bmatrix}, \]  
(16)

so that
\[ V = \frac{1}{6} \left[ v_{1x}(v_{2y}v_{3z} - v_{2z}v_{3y}) + v_{1y}(v_{2z}v_{3x} - v_{2x}v_{3z}) + v_{1z}(v_{2x}v_{3y} - v_{2y}v_{3x}) \right]. \]  
(17)

We purposely did not take the absolute value of this expression because these are *oriented* volumes. We do not really know the origin for the laser scanner. It could be inside or outside the fragment. If it is outside the fragment, then some of the facets will contribute a negative volume to the total volume. So, although the individual volume contributions depend upon the choice of the vector \( \mathbf{d} \), the total volume does not.

### 3.4.3 Total Surface Area and Total Volume

By adding up individual contributions to the surface area and volume from Eqs. 8 and 15, we can compute the total surface area and total volume of the fragment. These formulas hold whether or not the solid is convex.

### 3.4.4 Projected Area

Let the viewing direction be specified by a unit vector \( \hat{\mathbf{u}} \). Then the area of the triangle projected on a plane that is perpendicular to \( \hat{\mathbf{u}} \) is given by
\[ A_p = \frac{1}{2} \hat{\mathbf{u}} \cdot [(\mathbf{v}_2 - \mathbf{v}_1) \times (\mathbf{v}_3 - \mathbf{v}_1)] \text{ if and only if } \hat{\mathbf{u}} \cdot \hat{\mathbf{n}} > 0. \]  
(18)
The requirement that $\hat{u} \cdot \hat{n} > 0$ is necessary so that we only compute the projected area of one side of the triangle, not both sides. If the solid is convex, then the total projected area from the given viewpoint $\hat{u}$ is the simple summation from all the triangles. These fragments are not convex, however, which means that they have hidden surfaces. Not taking this into account will overestimate the presented area of the fragment. To properly account for hidden surfaces, we can use raytracing from BRL-CAD to compute the presented area from a given direction. It is useful to know how much the fragments deviate from convexity, so we also compute the presented area by simply adding the contributions according to Eq. 18.

The computation of the total projected area obtained by summing the contributions from each triangle, using Eq. 18, is fast enough that we have used 10,000 points over the unit sphere. But we also know that these fragments are not convex, so we have also used 1000 points distributed over the unit sphere to raytrace a projected area that fully accounts for hidden surfaces. Once the projected area has been computed, the (dimensionless) shape factor, $\gamma$, is then computed from $\gamma = A_p V^{-2/3}$.

As an approximation to the true projected area, we can compute the projected area of the STL solid without accounting for hidden surfaces and the fact that the solid is not convex, as in Listing 10.

**Listing 10. stl.b.cpp**

```cpp
// stl.b.cpp: reads in a binary STL description of a solid and computes volume and surface area

#include "Vector.h"
#include "Random.h"
#include <fstream>
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <iomanip>
#include <vector>

inline double square( double x ) { return x * x; }

struct facet {
    float nx, ny, nz;
    float v1x, v1y, v1z;
    float v2x, v2y, v2z;
    float v3x, v3y, v3z;
    unsigned int byte_count;
};

int main( int argc, char* argv[] ) {
    char * file;
    if ( argc == 2 ) {
        file = argv[1];
    } else {
        std::cerr << "Usage: " << argv[0] << " stl_binary_inputfile > outputfile" << std::endl;
        exit( 1 );
    }
    char header[100] = "";
```

*See Appendix B for sampling strategies.*
```cpp
unsigned long int n_facets = 0;
facet tri;

std::ifstream fin;
fin.open( file, std::ios::in | std::ios::binary );
if ( !fin ) { 
    std::cerrimeters "Error in opening file " << file << std::endl;
    exit( 1 );
}
std::cerr << "Contents of " << file << std::endl;
fin.read( char * ) header, 80;
fin.read( char * ) &n_facets, 4;
std::cerr << "There are " << n_facets << " triangles in " << file << std::endl;

double nx, ny, nz, v1x, v1y, v1z, v2x, v2y, v2z, v3x, v3y, v3z;
double ax, ay, az, bx, by, bz;
double total_area = 0., volume = 0., mean_area, a_facet;
std::vector< va::Vector > norm;
std::vector< double > area;
for ( unsigned int i = 0; i < n_facets; i++ ) { // for each facet
    fin.read( char * ) &tri, 50;
    // note that we don't need the normal vector to compute the area or volume
    nx = double( tri.nx );
    ny = double( tri.ny );
    nz = double( tri.nz );
    norm.push_back( va::Vector( nx, ny, nz ) );
    v1x = double( tri.v1x );
    v1y = double( tri.v1y );
    v1z = double( tri.v1z );
    v2x = double( tri.v2x );
    v2y = double( tri.v2y );
    v2z = double( tri.v2z );
    v3x = double( tri.v3x );
    v3y = double( tri.v3y );
    v3z = double( tri.v3z );
    // compute the area of the triangle specified by its three vertices
    ax = v2x - v1x;
    ay = v2y - v1y;
    az = v2z - v1z;
    bx = v3x - v1x;
    by = v3y - v1y;
    bz = v3z - v1z;
    a_facet = 0.5 * sqrt( square( ay * bz - az * by ) + square( az * bx - ax * bz ) + square( ax * by - ay * bx ) );
total_area += a_facet;
    area.push_back( a_facet );
    // compute six times the volume of the tetrahedron formed by the given triangle and the origin
    // note that this is an oriented volume, which could be positive or negative,
    // and the origin is completely arbitrary so might as well use Vector(0,0,0)
    volume += v1x * ( v2y * v3z - v2z * v3y ) + v1y * ( v2z * v3x - v2x * v3z ) + v1z * ( v2x * v3y - v2y * v3x );
} 
volume /= 6.;
mean_area = total_area / 4.;
const double F = pow( volume, -2./3. );

std::clog << std::setprecision(6) << std::fixed;
std::clog << "Number of facets = " << n_facets << std::endl;
std::clog << "Total surface area = " << total_area << std::endl;
std::clog << "Volume = " << volume << std::endl;
std::clog << "Mean surface area = " << total_area / 4. << " (from Cauchy's theorem)" << std::endl;
std::clog << "Mean shape factor = " << mean_area * F << " (from Cauchy's theorem)" << std::endl;

const int N = 100000;
rng::Random rng;
double dotprod, sf, ap, x, y, z;
va::Vector u;
std::cout << std::setprecision(6) << std::fixed;
for ( int n = 0; n < N; n++ ) {
    ap = 0.;
    rng::spherical_avoidance( x, y, z );
    u = va::Vector( x, y, z );
    for ( unsigned int i = 0; i < n_facets; i++ ) if ( dotprod = u * norm[i] ) > 0 ) ap += area[i] * dotprod;
    sf = ap * F;
    std::cout << sf << std::endl;
}
``
When this is applied to the fragment (SF1290) shown in Fig. 8, we get the histogram of shape factors displayed in Fig. 14.

![Fig. 14. Shape factor histograms generated by STL description of SF1290. No account has been taken for hidden surfaces so this is an overestimate of the shape factor.](image1)

Notice the similarity to the shape factor distribution from an ellipsoid (cf. Figs. 6 and 7). Now notice what happens when we sample from all the 15 fragments that were scanned7 (Fig. 15).

![Fig. 15. Shape factor histograms from 15 fragments in STL form. The routine used to generate this, stl.b.cpp, does not account for hidden surfaces, and since the fragments are not convex, this is an overestimate. The black curve is the lognormal fit. The maximum likelihood estimate is $\mu = 0.607588$ and $\sigma = 0.275688$, which gives a median $e^{\mu} = 1.71746$, mean $e^{\mu + \sigma^2/2} = 1.7949$, and mode $e^{\mu - \sigma^2} = 1.57247$.](image2)

Once again we see that the resulting distribution begins to approximate a lognormal distribution.
The next step is to take account of the hidden surfaces to come up with a more accurate computation of the true projected area. The computer code in Listing 11 was used for this purpose, and we get the results shown in Fig. 16.

![Shape Factor Histograms](image.png)

**Fig. 16.** Shape factor histograms for 15 fragments using stlarea. This computer code computes the projected area and does account for hidden surfaces, much like the Icosahedron Gage. The black curve is the lognormal fit. The maximum likelihood estimate is $\mu = 0.537$ and $\sigma = 0.297$, which gives a median $e^{\mu} = 1.71$, mean $e^{\mu+\sigma^2/2} = 1.79$, and mode $e^{\mu-\sigma^2} = 1.57$.

**Listing 11. stlarea.cpp**

```cpp
// stlarea.cpp: Read an STL binary file and compute presented areas and shape factors
// using viewpoints from the spiral distribution over the unit sphere.
// This implementation computes a true presented area, similar to raytracing,
// by imposing a grid over the bounding box from each viewpoint and then
// checking if the grid point lies in at least one of the projected facets.
// Note: Little endian is assumed in STL so may need to convert when porting to another computer.
// R. Saucier, September 2011

#include "Rotation.h"
#include <fstream>
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <string>
#include <vector>

struct facet {
    float nx, ny, nz;
    float v1x, v1y, v1z;
    float v2x, v2y, v2z;
    float v3x, v3y, v3z;
    unsigned int byte_count;
};

va::Vector normal( const va::Vector& v1, const va::Vector& v2, const va::Vector& v3 ) { // returns outward normal of oriented triangle
    return unit( ( v2 - v1 ) ^ ( v3 - v1 ) );
}

double area2( const va::Vector& v1, const va::Vector& v2, const va::Vector& v3 ) { // one way to compute the area
    return mag( ( v2 - v1 ) ^ ( v3 - v1 ) );
}

double area2( const va::Vector& n, const va::Vector& v1, const va::Vector& v2, const va::Vector& v3 ) { // alternative way to compute the area
    return n * ( ( v2 - v1 ) ^ ( v3 - v1 ) );
}
```

Approved for public release; distribution is unlimited.
// returns six times the volume of the tetrahedron formed by the given triangle and the origin
// note that this is an oriented volume, which could be positive or negative,
// and the origin is completely arbitrary so might as well use Vector(0,0,0)
double volume6( const va::Vector& v1, const va::Vector& v2, const va::Vector& v3 ) { // returns the oriented volume
  return v1 * ( v2 ^ v3 );
}

inline double min( double a, double b, double c ) { return std::min( std::min( a, b ), c ); }
inline double max( double a, double b, double c ) { return std::max( std::max( a, b ), c ); }

// fast point-in-triangle test (Ref: www.blackpawn.com/texts/pointinpoly/default.html)
bool inside( const va::Vector& p, const va::Vector& a, const va::Vector& b, const va::Vector& c ) {
  // compute vectors
  va::Vector v0 = c - a;
  va::Vector v1 = b - a;
  va::Vector v2 = p - a;
  // compute dot products
  double dot00 = v0 * v0;
  double dot01 = v0 * v1;
  double dot02 = v0 * v2;
  double dot11 = v1 * v1;
  double dot12 = v1 * v2;
  // Compute barycentric coordinates
  double w = 1. / ( dot00 * dot11 - dot01 * dot01 );
  double u = ( dot11 * dot02 - dot01 * dot12 ) * w;
  double v = ( dot00 * dot12 - dot01 * dot02 ) * w;
  // Check if point is in triangle
  return ( u > 0 ) && ( v > 0 ) && ( u + v < 1 );
}

double area_projected( const std::vector<va::Vector>& facet_v1, const std::vector<va::Vector>& facet_v2, const std::vector<va::Vector>& facet_v3, const va::Vector& u, int n_dim ) {
  const int N_FACETS = facet_v1.size();
  static const va::Vector I( 1., 0., 0. ), J( 0., 1., 0. ), K( 0., 0., 1. );
  va::Vector i = I;
  va::Vector j = J;
  va::Rotation R;
  va::Vector minus_u = -1. * u;
  if ( u != K && minus_u != K ) {
    R = va::Rotation( K, u );
    i = R * I;
    j = R * J;
  }
  // compute the bounding box in the plane perpendicular to u
  for ( int n = 0; n < N_FACETS; n++ ) {
    va::Vector v1 = facet_v1[n];
    va::Vector v2 = facet_v2[n];
    va::Vector v3 = facet_v3[n];
    v1x = v1 * i;
    v2x = v2 * i;
    v3x = v3 * i;
    v1y = v1 * j;
    v2y = v2 * j;
    v3y = v3 * j;
    xmin = min( v1x, v2x, v3x );
    xmax = max( v1x, v2x, v3x );
    ymin = min( v1y, v2y, v3y );
    ymax = max( v1y, v2y, v3y );
    x_min = xmin < x_min ? xmin : x_min;
    x_max = xmax > x_max ? xmax : x_max;
    y_min = ymin < y_min ? ymin : y_min;
    y_max = ymax > y_max ? ymax : y_max;
  }
  const double DELTA_X = ( x_max - x_min ) / double(n_dim);
  const double DELTA_Y = ( y_max - y_min ) / double(n_dim);
  const double DELTA_A = DELTA_X * DELTA_Y;
  va::Vector p, py;
  double ap = 0.;
  for ( double y = y_min; y <= y_max; y += DELTA_Y )
    for ( double x = x_min; x <= x_max; x += DELTA_X )
      p = va::Vector( x, y, 0. );
py = y * j;
for ( double x = x_min; x <= x_max; x += DELTA_Y ) {
    p = x * i + py;
    for ( register int n = 0; n < N_FACETS; n++ ) {
        v1 = facet_v1[n];
        v2 = facet_v2[n];
        v3 = facet_v3[n];
        v1 = ( v1 * i ) * i + ( v1 * j ) * j;
        v2 = ( v2 * i ) * i + ( v2 * j ) * j;
        v3 = ( v3 * i ) * i + ( v3 * j ) * j;
        if ( inside( p, v1, v2, v3 ) ) {
            ap += DELTA_A;
            break;
        }
    }
}
return ap;
}

int main( int argc, char* argv[] ) {
    char * file;
    int n_dim = 32; // grid size is n_dim x n_dim, with 32 x 32 = 1024 as default
    if ( argc == 3 ) {
        file = argv[1];
        n_dim = atoi( argv[2] );
    }
    else if ( argc == 2 )
        file = argv[1];
    else {
        std::cerr << "Usage: " << argv[0] << " stl_binary_inputfile" << std::endl;
        exit( 1 );
    }
    char header[100] = "";
    unsigned long int number = 0;
    facet tri;
    std::ifstream fin;
    fin.open( file, std::ios::in | std::ios::binary );
    if ( fin.bad() ) {
        std::cerr << "Error in opening file " << file << std::endl;
        exit( 1 );
    }
    std::cerr << "Contents of " << file << std::endl;
    fin.read( (char *) header, 80 );
    fin.read( (char *) &number, 4 );
    std::clog << "Header = " << header << std::endl;
    std::clog << "There are " << number << " triangles in " << std::endl;
    va::Vector n, v1, v2, v3;
    double surface_area = 0., vol6 = 0.;
    std::vector<va::Vector> facet_v1(number), facet_v2(number), facet_v3(number);
    for ( unsigned int i = 0; i < number; i++ ) {
        fin.read( [char *] &tri, 50 );
        n = va::Vector( (double)tri.nz, (double)tri.ny, (double)tri.nz );
        v1 = va::Vector( (double)tri.v1x, (double)tri.v1y, (double)tri.v1z );
        v2 = va::Vector( (double)tri.v2x, (double)tri.v2y, (double)tri.v2z );
        v3 = va::Vector( (double)tri.v3x, (double)tri.v3y, (double)tri.v3z );
        surface_area += area2( n, v1, v2, v3 );
        vol6 += volume6( v1, v2, v3 );
        facet_v1[i] = v1;
        facet_v2[i] = v2;
        facet_v3[i] = v3;
    }
    fin.close();
    double surface_area /= 2.;
    const double VOLUME = fabs( vol6 / 6. );
    const double F = pow( VOLUME, -2. / 3. );
    std::cerr << "Volume = " << VOLUME << std::endl;
    std::cerr << "Number of facets = " << number << std::endl;
    std::cerr << "Total surface area = " << surface_area << std::endl;
    std::cerr << "Mean shape factor (using Cauchy’s theorem for convex solid) = " << 0.25 * surface_area * F << std::endl;
    const int N = 10;
    double x, y, z, sf;
range: Random rng;
for ( int i = 0; i < N; i++ ) {
    rng.spherical_avoidance( x, y, z );
    sf = F * area_projected( facet_v1, facet_v2, facet_v3, va::Vector( x, y, z ), n_dim );
    std::cout << sf << std::endl;
}
return EXIT_SUCCESS;

Thus, to characterize the shape factor distribution of natural fragments, we can either measure many fragments using the Icosahedron Gage, or we can laser scan, possibly a fewer number. Two parameters are sufficient to characterize the lognormal distribution, in either case. Then it is a simple matter to simulate the shape factor using Listing 6.

4. Shape Factor Modeling

Finally, let us consider modeling the shape factor. Some penetration models, such as THOR, only require the presented area of the fragment at impact, in which case the shape factor is sufficient. But other penetration models, such as FATEPEN, require a specific shape and orientation. This requires a realization of the shape factor, and the simplest shapes in FATEPEN that allow for this are a cylinder and a cuboid.

The procedure we use applies to both shapes. We start with the cylinder and cuboid in standard orientation, which is with the axis of symmetry along the $z$-axis in the case of the cylinder, and with the length along the $z$-axis, width along the $x$-axis, and thickness along the $y$-axis in the case of the cuboid. The target is taken to be in the $x$-$y$ plane. Then we are given the mass $m$, material density $\rho$, and shape factor $\gamma$.

For cylinders, we randomly select a candidate $L/D$, which then allows us to compute a minimum and a maximum shape factor for this cylinder, depending upon its orientation. If the drawn shape factor falls within this range, $\gamma_{\text{min}} \leq \gamma \leq \gamma_{\text{max}}$, then we know there is some orientation that will work; in the next section we work out the formula for the appropriate yaw angle. We then have enough information to also compute the diameter and the length of the cylinder.

For cuboids, it is the same idea except that we need to select candidate $W/L$ and $T/W$ ratios, which then provide enough information to compute a minimum and a maximum shape factor, depending upon orientation. We then compute the 3D rotation that will realize this shape factor. Finally, we factor this rotation into a
The formula for the dimensionless shape factor of a right-circular cylinder (RCC) as a function of the effective yaw angle*$\phi_y$ is$^{14}$

$$\gamma(\phi_y) = a \sin \phi_y + b \cos \phi_y, \quad (19)$$

where

$$a \equiv \left( \frac{\pi}{4} \frac{L}{D} \right)^{-2/3} \frac{L}{D} \quad \text{and} \quad b \equiv \left( \frac{\pi}{4} \frac{L}{D} \right)^{-2/3} \frac{\pi}{4}. \quad (20)$$

The yaw angle that gives the maximum shape factor is obtained by setting the derivative with respect to $\phi_y$ equal to zero and solving for $\phi_y$:

$$\left( \frac{d\gamma}{d\phi_y} \right)_{\gamma=\gamma_{\max}} = a \cos \phi_y - b \sin \phi_y = 0, \quad (21)$$

which gives

$$\phi_y = \gamma_{\max} = \tan^{-1} \left( \frac{a}{b} \right). \quad (22)$$

To simplify the notation, let $\hat{\phi}_y$ denote this angle: $\hat{\phi}_y \equiv \phi_y = \gamma_{\max}$. Then,

$$\gamma_{\max} = \gamma(\hat{\phi}_y) = a \frac{a}{\sqrt{a^2 + b^2}} + b \frac{b}{\sqrt{a^2 + b^2}} = \sqrt{a^2 + b^2}, \quad (23)$$

and we can write

$$a = \gamma_{\max} \sin \hat{\phi}_y \quad \text{and} \quad b = \gamma_{\max} \cos \hat{\phi}_y, \quad (24)$$

so that Eq. 19 can be written as

$$\gamma(\phi_y) = \begin{cases} 
\gamma_{\max} \cos(\phi_y - \hat{\phi}_y) & \text{if } \phi_y > \hat{\phi}_y \\
\gamma_{\max} \cos(\hat{\phi}_y - \phi_y) & \text{if } \phi_y < \hat{\phi}_y
\end{cases} \quad (25)$$

---

$^*$Note that effective yaw includes pitch and is defined by $\phi_{y,\text{eff}} = \cos^{-1}(\cos \phi_p \cos \phi_y)$, where $\phi_p$ is pitch and $\phi_y$ is actual yaw, but we use $\phi_y$ rather than $\phi_{y,\text{eff}}$ just to simplify the notation.
where $\phi_y = \cos^{-1}(b/\gamma_{\text{max}})$. Solving for the yaw angle, we get

$$\phi_y = \begin{cases} 
\cos^{-1}(b/\gamma_{\text{max}}) + \cos^{-1}(\gamma/\gamma_{\text{max}}) & \text{if } \gamma < b \\
\cos^{-1}(b/\gamma_{\text{max}}) - \cos^{-1}(\gamma/\gamma_{\text{max}}) & \text{if } \gamma \geq b 
\end{cases} \quad (26)$$

which shows that we can easily get the orientation (yaw angle) of an RCC from the shape factor at impact.

The minimum shape factor is

$$\gamma_{\text{min}} = \begin{cases} 
\left(\frac{\pi}{4}\frac{L}{D}\right)^{-2/3} \frac{L}{D} & \text{if } \frac{L}{D} < \frac{\pi}{4} \quad (\text{disk-like}) \\
\left(\frac{\pi}{4}\frac{L}{D}\right)^{-2/3} \frac{\pi}{4} & \text{if } \frac{L}{D} > \frac{\pi}{4} \quad (\text{rod-like}) 
\end{cases} \quad (27)$$

Solving this for $L/D$ for a given minimum shape factor, we get

$$L = \begin{cases} 
\left(\frac{\pi}{4}\right)^2 \gamma_{\text{min}} & \text{if } \frac{L}{D} < \frac{\pi}{4} \quad (\text{disk-like}) \\
\left(\frac{\pi}{4}\right)^{1/2} \gamma_{\text{min}}^{-3/2} & \text{if } \frac{L}{D} > \frac{\pi}{4} \quad (\text{rod-like}) 
\end{cases} \quad (28)$$

The minimum shape factor for natural fragments is around 0.5. Inserting this value into this equation gives

$$\frac{L}{D} = \begin{cases} 
0.077 & \text{if } \frac{L}{D} < \frac{\pi}{4} \quad (\text{disk-like}) \\
2.5 & \text{if } \frac{L}{D} > \frac{\pi}{4} \quad (\text{rod-like}) 
\end{cases} \quad (29)$$

The maximum shape factor for natural fragments is around 5.5. Inserting this value into the appropriate cubic equation (see Appendix A, Eq. A-46) gives

$$\frac{L}{D} = \begin{cases} 
0.0691055 & \text{if } \frac{L}{D} < \frac{\pi}{4} \quad (\text{disk-like}) \\
102.619 & \text{if } \frac{L}{D} > \frac{\pi}{4} \quad (\text{rod-like}) 
\end{cases} \quad (30)$$

The $L/D$ value for a rod-like RCC is unrealistic, so we choose disk-like RCCs to model the maximum shape factor. Selecting disk-like RCCs with $L/D$ in the range...
from 0.069 to $\pi/4$ will span the range of shape factors from 0.5 to 5.5. The actual code that generates a yawed RCC to represent the shape factor is given in Listing 12.

**Listing 12. sf-rcc.cpp**

```cpp
// sf-rcc.cpp: Implementation of an algorithm for generating FATEPEN RCCs to represent a specified shape factor.
// Given a dimensionless shape factor, generates the L/D and yaw angle for the RCC to represent it.
// The RCCs are disk-like in order to span the range of shape factors from 0.5 to 5.5.
// R. Saucier, October 2011

#include <Random.h>
#include <iostream>
#include <cstdlib>
#include <cmath>

int main( int argc, char* argv[] ) {
    int N = 1000; // number of samples or override on command line
    const double R2D = 180. / M_PI; // to convert from radians to degrees
    const double SF_MIN = 0.5; // minimum shape factor (found from artillery fragments)
    const double SF_MAX = 4.5; // maximum shape factor (found from artillery fragments)

    if ( argc == 2 ) N = atoi( argv[1] ); // override default number of samples on command line

    // default values for the shape factor lognormal distribution from 122mm, 152mm and 155mm artillery
    double mu = 0.596514; // these two parameters characterize the lognormal shape factor distribution
    double sigma = 0.340874; // with mode = 1.62, median = 1.81 and mean = 1.93

    rng::Random rng;
    double a, b, c, th, sf_min, sf_max, sf, l_d, l, d, yaw, V = 1.; // here we use a fragment with unit volume

    for ( int n = 0; n < N; n++ ) {
        do { sf = rng.lognormal( 0., mu, sigma ); } while ( sf < SF_MIN || sf > SF_MAX );

        l_d = rng.uniform( 0.069, M_PI_4 ); // disk-like RCCs (l_d = 0.069 corresponds to sf_max = 5.5)
        c = pow( M_PI_4 * l_d, -2./3. );
        a = c * l_d;
        b = c * M_PI_4;
        sf_min = a;
        sf_max = sqrt( a * a + b * b );
        if ( sf < b ) th = acos( b / sf_max ) + acos( sf / sf_max );
        else th = acos( b / sf_max ) - acos( sf / sf_max );

        d = pow( V / ( M_PI_4 * l_d ), 1./3. );
        l = d * l_d;
        yaw = th * R2D;

        std::cout << sf << "\t" << l_d << "\t" << yaw << std::endl;
    }
    return EXIT_SUCCESS;
}
```

This algorithm is stochastic, so that for a given shape factor there will be a range of $L/D$ ratios and yaw angles that can represent the fragment. This will result in a range of residual velocities, but this range will be relatively small. For example, a 725-gr steel RCC with a shape factor of 1.93 striking a 1/4-inch mild steel plate at 3500 f/s results in a residual velocity of 1988 ± 23 f/s.

### 4.2 Cuboid

The procedure for realizing the shape factor as a cuboid with a specific pitch, yaw, and roll\(^{15}\) is implemented in the code in Listing 13.
Listing 13. sf-rpp.cpp

```
// sf-rpp.cpp: Monte Carlo shape factor from a lognormal distribution
// This code demonstrates how it is possible to make use of a lognormal shape factor distribution,
// and at the same time create RPPs for FATEPEN that have the proper volume and presented area.
// Given the fragment volume (or mass and density), it provides everything that FATEPEN requires:
// the length, width, and thickness of the RPP, as well as the pitch-yaw-roll rotation sequence
// that will take the RPP from standard orientation to the orientation that realizes the shape factor.
// R. Saucier, September 2011

#include "Random.h"
#include "Rotation.h"
#include <iostream>
#include <cstdlib>
#include <cassert>
#include <iomanip>

const double SF_MIN = 0.5; // minimum shape factor
const double SF_MAX = 4.5; // maximum shape factor
const double MU = 0.596514; // these two parameters characterize the lognormal shape factor distribution
const double SIGMA = 0.340874; // mode = 1.62, median = 1.81, mean = 1.93
const va::Vector I( 1., 0., 0. ), J( 0., 1., 0. ), K( 0., 0., 1. );

int main( void ) {
    const int N_SAMPLES = 10000;
    const double W_L_MIN = 0.185; // minimum W/L ratio (must be 0.185 or smaller to get SF_MAX = 5.5)
    const double T_W_MIN = 0.185; // minimum T/W ratio (must be 0.185 or smaller to get SF_MAX = 5.5)
    const double G2GR = 15.4324; // to convert grams to grains
    const double GR2G = 1. / G2GR; // to convert grains to grams
    const double MASS = 725. * GR2G; // mass (725 grains converted to grams)
    const double RHO = 7.83; // density of steel (g/cm^3)
    const double V = MASS / RHO; // volume (cm^3)
    const double V_23 = pow( V, 2./3. );

    double th, th_max, Amin, Amax, w_l, t_w, Ax, Ay, Az, W, L, T;
    va::Vector Ap, u_max, axis, u;
    va::Rotation R;
    va::sequence s;
    rng::Random rng;
    double cp, cy, cr, sp, sy, sr, p, y, r, sf, ap, apcalc;

    std::cout << std::setprecision(6) << std::fixed;
    for ( int n = 0; n < N_SAMPLES; n++ ) {
        // normally, the shape factor would be provided, but here we get a shape factor within bounds [SF_MIN, SF_MAX]
        do { sf = rng.lognormal( 0., MU, SIGMA ); } while ( sf < SF_MIN || sf > SF_MAX );

        sf = 1.93; // or select from a lognormal distribution
        ap = sf * V_23;

        do {
            w_l = rng.uniform( W_L_MIN, 1. ); // ratio of W./L
            t_w = rng.uniform( T_W_MIN, 1. ); // ratio of T/W
            W = pow( V * w_l / t_w, 1./3. ); // width of RPP
            L = W / w_l; // length of RPP
            T = W * t_w; // thickness of RPP
            Ax = L * T; // presented area orthogonal to x-axis (intermediate) in initial orientation
            Ay = L * W; // presented area orthogonal to y-axis (maximum)
            Az = W * T; // presented area orthogonal to z-axis (minimum)
        } while ( ap < Amin || ap > Amax );

        Ap = Ax * I + Ay * J + Az * K; // presented area vector
        va::Vector u_max = Ap / ap;
        va::Rotation R( axis, th_max ); // rotation which takes K to u_max, and will take [Ap] from Axim to Arnax
        th = th_max - acos( ap / Amax ); // rotation angle for u
        R = va::Rotation( axis, th ); // rotation which takes K to u
        va::Rotation R2( K, rng.uniform( 0., 2. * M_PI ) ); // this should have no effect upon the projected area perp to K
        s = va::factor( R2 * R, va::XYZ ); // pitch-yaw-roll rotation sequence

        th = th.first; // pitch (rad)
        y = th.second; // yaw (rad)
        r = th.third; // roll (rad)
        cp = cos( p );
        cy = cos( y );
        cr = cos( r );
        sp = sin( p );
        sy = sin( y );
    }
}
```

Approved for public release; distribution is unlimited.
This will result in a range of residual velocities. For example, a 725-gr steel cuboid with a shape factor of 1.93 striking a 1/4-inch mild steel plate at 3500 f/s results in a residual velocity of $2464 \pm 363$ f/s, much more variation than is the case with cylinders.

5. Conclusions and Recommendations

We have provided explicit analytical formulas for the shape factor distributions of some common shapes with random orientations. And we have shown that it is easy to simulate these shape factor distributions with computer code and demonstrated through plots that the simulations match the plots from the analytical formulas. When we examine natural fragment shape factor distributions, however, the only shape that comes close is an ellipsoid. But it, too, fails to provide an adequate representation by simply randomizing its orientation. We can work out the dimensions of the ellipsoid from the projected area measurements, but then the volume comes out wrong because it does not account for hidden surfaces.

A better approach to shape factor simulation was found after enough natural fragment data was processed to reveal that it could be fit with a lognormal distribution. The measurement of fragment shape factor with the Icosahedron Gage does not give any indication of a lognormal distribution with only 16 measurements, but once we combine the measurements from hundreds of fragments, the resemblance to the lognormal is striking. Rather than trying to find a shape that will will work by randomizing the orientation, it makes more sense to use the lognormal as a probability distribution in Monte Carlo sampling.

We also showed that laser scans of fragments can be used to compute the fragment shape factor from any viewpoint, and we described a variety of methods of achieving a uniform spherical distribution. Computing fragment volume and projected area is fast if we treat the fragments as convex solids. But natural fragments are not convex, and accounting for hidden surfaces uses much more computer time.
Finally, we showed that it is possible to realize each fragment mass and shape factor as either a yawed cylinder or a cuboid with a pitch, yaw, and roll. Thus, we have a procedure for generating all the input variables required to run THOR or FATEPEN from the given mass and shape factor.
6. References


Approved for public release; distribution is unlimited.


INTENTIONALLY LEFT BLANK.
Appendix A. Analytical Shape Factor Formulas for 5 Convex Solids
Here we cite explicit formulas for the probability density function (PDF) and the cumulative distribution function (CDF) for 5 convex solids that have a random orientation. For each solid, we also list code that can be used to plot the PDF and CDF.

### A-1. Cube

The PDF \( f(\gamma) \) is given by\(^1\)

\[
\begin{align*}
f(\gamma) &= \begin{cases} 
\frac{4}{\sqrt{3}} - \frac{4\sqrt{3}}{\pi} \tan^{-1} \left( \frac{\sqrt{3}}{\gamma} \sqrt{2 - \gamma^2} \right) & \text{if } 1 \leq \gamma \leq \sqrt{2} \\
\frac{4}{\sqrt{3}} & \text{if } \sqrt{2} \leq \gamma \leq \sqrt{3}
\end{cases} 
\end{align*}
\]  

(A-1)

and the CDF \( F(\gamma) \) is given by

\[
\begin{align*}
F(\gamma) &= \begin{cases} 
\frac{4\gamma}{\sqrt{3}} - \frac{4\sqrt{3}\gamma}{\pi} \tan^{-1} \left( \frac{\sqrt{3}}{\gamma} \sqrt{2 - \gamma^2} \right) + \\
\frac{6}{\pi} \left[ \tan^{-1} \left( \frac{2 - \sqrt{3}\gamma}{\sqrt{2 - \gamma^2}} \right) + \tan^{-1} \left( \frac{2 + \sqrt{3}\gamma}{\sqrt{2 - \gamma^2}} \right) \right] & \text{if } 1 \leq \gamma \leq \sqrt{2} \\
\frac{4\gamma}{\sqrt{3}} - 3 & \text{if } \sqrt{2} \leq \gamma \leq \sqrt{3}
\end{cases} 
\end{align*}
\]  

(A-2)

These formulas are implemented in the code in Listing A-1 and shown plotted in Fig. A-1.

#### Listing A-1. cube.cpp

```cpp
// cube.cpp: generates plotting points for cube pdf and cdf
// Ref: Vickers, G. T. and Brown, D. J.,
// "The distribution of projected area and perimeter of convex, solid particles,"
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <cassert>

static const double SQRT2 = M_SQRT2, SQRT3 = sqrt(3.);

double pdf(double x) {
    assert(1. <= x && x <= SQRT3);
    if (x < SQRT2) {
        double a = sqrt(2. - x * x);
        return 4. / SQRT3 - 4. * SQRT3 * atan(SQRT3 * a / x) / M_PI;
    } else
        return 4. / SQRT3;
}

double cdf(double x) {
    assert(1. <= x && x <= SQRT3);
    if (x < SQRT2) {
        double a = sqrt(2. - x * x);
        return 4. / SQRT3 - 3 - 4. * SQRT3 * atan(SQRT3 * a / x) / M_PI +
            6 / M_PI * [tan^{-1}(\frac{2 - \sqrt{3}\gamma}{\sqrt{2 - \gamma^2}}) +
                tan^{-1}(\frac{2 + \sqrt{3}\gamma}{\sqrt{2 - \gamma^2}})];
    } else
        return 4. / SQRT3;
}
```

return 4. * x / SQRT3 - 3. - ( 4. * SQRT3 * x / M_PI ) * atan( SQRT3 * a / x ) +
( 6. / M_PI ) * ( atan( ( 2. - SQRT3 * x ) / a ) + atan( ( 2. + SQRT3 * x ) / a ) );
else
return 4. * x / SQRT3 - 3.;
}

int main( int argc, char* argv[] ) { // specify number of points on commandline or use 1000
int N = 1000;
if ( argc == 2 ) N = atoi( argv[1] );
const double GMIN = 1., GMAX = SQRT3;
for ( double g = GMIN; g <= GMAX; g += ( GMAX - GMIN ) / N )
std::cout << g << "\t" << pdf( g ) << "\t" << cdf( g ) << std::endl;
return EXIT_SUCCESS;
}

The conventional way of turning this into a shape factor probability distribution is to sample the uniform distribution between 0 and 1, $P \sim U(0, 1)$, and then invert $F$ to get $\gamma = F^{-1}(P)$. But there is no simple way to solve Eq. A-2 for $\gamma$ when $1 \leq \gamma \leq \sqrt{2}$, so instead we simulate the shape factor probability distribution by uniform random sampling over the unit sphere and compute the shape factor for each orientation to build up a probability distribution.

![Plot of shape factor PDF and CDF for a randomly oriented cube](image)

**Fig. A-1.** Plot of shape factor PDF and CDF for a randomly oriented cube

## A-2. Cuboid

The source for the following formulas is Walters.²

### A-2.1. Notation

Let $L \geq W \geq T$ be the length, width, and thickness of the cuboid, so that length is always the largest dimension and thickness the smallest. By setting

$$A_x = WT, \quad A_y = TL, \quad A_z = LW,$$


Approved for public release; distribution is unlimited.
we order the areas of the 3 faces so that \( A_x \leq A_y \leq A_z \) and match the notation in Walters\(^2\). If we only know the 3 face areas, then we can get the dimensions from

\[
L = \sqrt{\frac{A_y A_z}{A_x}}, \quad W = \sqrt{\frac{A_z A_x}{A_y}}, \quad T = \sqrt{\frac{A_x A_y}{A_z}}. \tag{A-4}
\]

To simplify the notation, let \( x \) represent the variable presented area and define

\[
a \equiv A_x, \quad b \equiv A_y, \quad c \equiv A_z, \quad m^2 \equiv a^2 + b^2 + c^2, \tag{A-5}\]

\[
x_a \equiv \sqrt{m^2 - a^2}, \quad x_b \equiv \sqrt{m^2 - b^2}, \quad x_c \equiv \sqrt{m^2 - c^2}. \tag{A-6}\]

Walters\(^2\) also defines the following quantities:

\[
\Delta \equiv \frac{m^2 + x^2}{m^2 - x^2}, \quad \Delta_z \equiv \frac{m^2 + c^2}{m^2 - c^2}, \quad D = \frac{2m^2 x^2}{m^2 - x^2}, \quad D_z = \frac{2m^2 c^2}{m^2 - c^2}, \tag{A-7}\]

\[
\alpha_1 \equiv \frac{1}{c^2 + x^2}, \quad \alpha_x \equiv \frac{1}{c^2 + a^2}, \quad \alpha_y \equiv \frac{1}{c^2 + b^2}. \tag{A-8}\]

### A-2.2. Integrals

Using \( x \) as the variable presented area, we make use of the following 4 indefinite integrals:

\[
I_1(x) \equiv \int \sin^{-1}(\Delta - \alpha_1 D)\,dx = x \sin^{-1}(\Delta - \alpha_1 D) + 2m \tanh^{-1}\left(\frac{\sqrt{x_a^2 - x^2}}{m}\right) \quad \text{for} \quad a \leq x \leq x_c \tag{A-9}\]

\[
I_2(x) \equiv \int \sin^{-1}(\Delta - \alpha_x D)\,dx = x \sin^{-1}(\Delta - \alpha_x D) - 2m \tan^{-1}\left(\frac{\sqrt{x_b^2 - x^2}}{b}\right) \quad \text{for} \quad a \leq x \leq x_b \tag{A-10}\]

\[
I_3(x) \equiv \int \sin^{-1}(\Delta - \alpha_y D)\,dx = x \sin^{-1}(\Delta - \alpha_y D) - 2m \tan^{-1}\left(\frac{\sqrt{x_a^2 - x^2}}{a}\right) \quad \text{for} \quad a \leq x \leq x_a \tag{A-11}\]

\[
I_4(x) \equiv \int \sin^{-1}(\Delta_z - \alpha_1 D_z)\,dx = x \sin^{-1}(\Delta_z - \alpha_1 D_z) + 2c \tanh^{-1}\left(\frac{\sqrt{m^2 - x^2}}{m}\right) \quad \text{for} \quad a \leq x \leq x_c \tag{A-12}\]
The corresponding definite integrals are

\[ I_i(x_1, x_2) \equiv I_i(x_2) - I_i(x_1) \quad \text{for } i = 1, 2, 3, 4. \]  \hspace{1cm} (A-13)

It is also convenient to define the following 2 constants:

\[ k_1 \equiv \sin^{-1}(\Delta_x - \alpha_x D_z) \quad \text{and} \quad k_2 \equiv \sin^{-1}(\Delta_z - \alpha_y D_z). \]  \hspace{1cm} (A-14)

**A-2.3. Probability Density Function and Cumulative Distribution Function**

Let \( f(x) \) represent the probability density function and \( F(x) \) represent the cumulative distribution function. The following expressions for \( f(x) \) are taken from Walters,\(^2\) and \( F(x) \) is obtained by integrating the corresponding density function. There are 2 cases to consider, depending upon the value of \( a^2 + b^2 \) relative to \( c^2 \), and each case has 6 distinct regions.

**Case 1:** \( a^2 + b^2 < c^2 \)

- **Case 1.1:** \( a < x < b \)

\[ f(x) = \frac{1}{\pi m} \left[ \sin^{-1}(\Delta - \alpha_1 D) - \sin^{-1}(\Delta - \alpha_x D) + \sin^{-1}(\Delta_x - \alpha_1 D_z) - k_1 \right] \]  \hspace{1cm} (A-15)

\[ F(x) = \frac{1}{\pi m} \left[ I_1(a, x) - I_2(a, x) + I_4(a, x) - k_1(x - a) \right] \]  \hspace{1cm} (A-16)

- **Case 1.2:** \( b < x < x_c \)

\[ f(x) = \frac{1}{\pi m} \left[ 2 \sin^{-1}(\Delta - \alpha_1 D) - \sin^{-1}(\Delta - \alpha_x D) - \sin^{-1}(\Delta - \alpha_y D) + 2 \sin^{-1}(\Delta_x - \alpha_1 D_z) - k_1 - k_2 \right] \]  \hspace{1cm} (A-17)

\[ F(x) = \frac{1}{\pi m} \left[ I_1(a, b) - I_2(a, b) + I_4(a, b) - k_1(b - a) + 2 I_1(b, x) - I_2(b, x) - I_3(b, x) + 2 I_4(b, x) - k_1(x - b) - k_2(x - b) \right] \]  \hspace{1cm} (A-18)

- **Case 1.3:** \( x_c < x < c \)

\[ f(x) = \frac{1}{\pi m} \left[ 2 \pi - \sin^{-1}(\Delta - \alpha_x D) - \sin^{-1}(\Delta - \alpha_y D) - k_1 - k_2 \right] \]  \hspace{1cm} (A-19)

\[ F(x) = \frac{1}{\pi m} \left[ I_1(a, b) - I_2(a, b) + I_4(a, b) - k_1(b - a) + 2 I_1(b, x_c) - I_2(b, x_c) - I_3(b, x_c) + 2 I_4(b, x_c) - k_1(x_c - b) - k_2(x_c - b) + 2 \pi (x - x_c) - I_2(x_c, x) - I_3(x_c, x) - k_1(x - x_c) - k_2(x - x_c) \right] \]  \hspace{1cm} (A-20)
• Case 1.4: \( c < x < x_b \)

\[
f(x) = \frac{2}{\pi m} [\pi - \sin^{-1}(\Delta - \alpha_x D) - \sin^{-1}(\Delta - \alpha_y D)]
\]
(A-21)

\[
F(x) = \frac{1}{\pi m} [I_1(a, b) - I_2(a, b) + I_4(a, b) - k_1(b - a) + 2I_1(b, x_c) -
I_2(b, x_c) - I_3(b, x_c) + 2I_4(b, x_c) - k_1(x_c - b) - k_2(x_c - b) +
2\pi(c - x_c) - I_2(x_c, c) - I_3(x_c, c) - k_1(c - x_c) - k_2(c - x_c) +
2\pi(x - c) - 2I_2(c, x) - 2I_3(c, x)]
\]
(A-22)

• Case 1.5: \( x_b < x < x_a \)

\[
f(x) = \frac{1}{\pi m} [3\pi - 2\sin^{-1}(\Delta - \alpha_y D)]
\]
(A-23)

\[
F(x) = \frac{1}{\pi m} [I_1(a, b) - I_2(a, b) + I_4(a, b) - k_1(b - a) + 2I_1(b, x_c) -
I_2(b, x_c) - I_3(b, x_c) + 2I_4(b, x_c) - k_1(x_c - b) - k_2(x_c - b) +
2\pi(c - x_c) - I_2(x_c, c) - I_3(x_c, c) - k_1(c - x_c) - k_2(c - x_c) +
2\pi(x_b - c) - 2I_2(c, x_b) - 2I_3(c, x_b) +
3\pi(x - x_b) - 2I_3(x_b, x)]
\]
(A-24)

• Case 1.6: \( x_a < x < m \)

\[
f(x) = \frac{4}{m}
\]
(A-25)

\[
F(x) = \frac{1}{\pi m} [I_1(a, b) - I_2(a, b) + I_4(a, b) - k_1(b - a) + 2I_1(b, x_c) -
I_2(b, x_c) - I_3(b, x_c) + 2I_4(b, x_c) - k_1(x_c - b) - k_2(x_c - b) +
2\pi(c - x_c) - I_2(x_c, c) - I_3(x_c, c) - k_1(c - x_c) - k_2(c - x_c) +
2\pi(x_b - c) - 2I_2(c, x_b) - 2I_3(c, x_b) +
3\pi(x_a - x_b) - 2I_3(x_b, x_a)] + \frac{4}{m}(x - x_a)
\]
(A-26)

Case 2: \( a^2 + b^2 > c^2 \)

• Case 2.1: \( a < x < b \)

\[
f(x) = \frac{1}{\pi m} [\sin^{-1}(\Delta - \alpha_1 D) - \sin^{-1}(\Delta - \alpha_x D) + \sin^{-1}(\Delta_z - \alpha_1 D_z) - k_1]
\]
(A-27)

\[
F(x) = \frac{1}{\pi m} [I_1(a, x) - I_2(a, x) + I_4(a, x) - k_1(x - a)]
\]
(A-28)

Approved for public release; distribution is unlimited.
• Case 2.2: \( b < x < c \)

\[
f(x) = \frac{1}{\pi m} \left[ 2 \sin^{-1}(\Delta - \alpha_1 D) - \sin^{-1}(\Delta - \alpha_x D) - \sin^{-1}(\Delta - \alpha_y D) + 2 \sin^{-1}(\Delta_z - \alpha_1 D_z) - k_1 - k_2 \right]
\]

\[
F(x) = \frac{1}{\pi m} \left[ I_1(a, b) - I_2(a, b) + I_4(a, b) - k_1(b - a) + 2I_1(b, x) - I_2(b, x) - I_3(b, x) + 2I_4(b, x) - k_1(x - b) - k_2(x - b) \right]
\]

(A-29)

• Case 2.3: \( c < x < x_c \)

\[
f(x) = \frac{2}{\pi m} \left[ \sin^{-1}(\Delta - \alpha_1 D) - \sin^{-1}(\Delta - \alpha_x D) + \sin^{-1}(\Delta_z - \alpha_1 D_z) - \sin^{-1}(\Delta - \alpha_y D) \right]
\]

\[
F(x) = \frac{1}{\pi m} \left[ I_1(a, b) - I_2(a, b) + I_4(a, b) - k_1(b - a) + 2I_1(b, c) - I_2(b, c) - I_3(b, c) + 2I_4(b, c) - k_1(c - b) - k_2(c - b) + 2I_1(c, x) - 2I_2(c, x) - 2I_3(c, x) + 2I_4(c, x) \right]
\]

(A-30)

• Case 2.4: \( x_c < x < x_b \)

\[
f(x) = \frac{2}{\pi m} \left[ \pi - \sin^{-1}(\Delta - \alpha_x D) - \sin^{-1}(\Delta - \alpha_y D) \right]
\]

\[
F(x) = \frac{1}{\pi m} \left[ I_1(a, b) - I_2(a, b) + I_4(a, b) - k_1(b - a) + 2I_1(b, c) - I_2(b, c) - I_3(b, c) + 2I_4(b, c) - k_1(c - b) - k_2(c - b) + 2I_1(c, x_c) - 2I_2(c, x_c) - 2I_3(c, x_c) + 2I_4(c, x_c) + 2\pi(x - x_c) - 2I_2(x_c, x) - 2I_4(x_c, x) \right]
\]

(A-33)

• Case 2.5: \( x_b < x < x_a \)

\[
f(x) = \frac{1}{\pi m} \left[ 3\pi - 2\sin^{-1}(\Delta - \alpha_y D) \right]
\]

\[
F(x) = \frac{1}{\pi m} \left[ I_1(a, b) - I_2(a, b) + I_4(a, b) - k_1(b - a) + 2I_1(b, c) - I_2(b, c) - I_3(b, c) + 2I_4(b, c) - k_1(c - b) - k_2(c - b) + 2I_1(c, x) - 2I_2(c, x) - 2I_3(c, x) + 2I_4(c, x) + 2\pi(x_b - x_c) - 2I_2(x_c, x_b) - 2I_3(x_c, x_b) + 3\pi(x - x_b) - 2I_3(x_b, x) \right]
\]

(A-35)

(A-36)
• Case 2.6: $x_a < x < m$

\[
f(x) = \frac{4}{m} \quad (A-37)
\]

\[
F(x) = \frac{1}{\pi m} \left[ I_1(a, b) - I_2(a, b) + I_4(a, b) - k_1(b - a) + 2I_1(b, c) - \right.
\]

\[
I_2(b, c) - I_3(b, c) + 2I_4(b, c) - k_1(c - b) - k_2(c - b) + \]

\[
2I_1(c, x_c) - 2I_2(c, x_c) - 2I_3(c, x_c) + 2I_4(c, x_c) + \]

\[
2\pi(x_b - x_c) - 2I_2(x_c, x_b) - 2I_3(x_c, x_b) + \]

\[
3\pi(x_a - x_b) - 2I_3(x_b, x_a) \right] + \frac{4}{m}(x - x_a) \quad (A-38)
\]

These formulas have been implemented in Listing A-2. Example plots are displayed in Fig. A-2.

\[\text{Fig. A-2. Plot of shape factor PDF and CDF for a randomly oriented cuboid with } L = 3, W = 2, T = 1\]
Listing A-2. rpp.cpp

```
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <cassert>

inline double min( double a, double b, double c ) { return std::min( std::min( a, b ), c ); }
inline double max( double a, double b, double c ) { return std::max( std::max( a, b ), c ); }
inline double mid( double a, double b, double c ) { return std::max( std::min( a, b ), std::min( std::max( a, b ), c ) ); }

double i1( double a, double b, double c, double x ) {
    double a2 = a * a;
    double b2 = b * b;
    double c2 = c * c;
    double m2 = a2 + b2 + c2;
    double m = sqrt( m2 );
    double x2 = x * x;
    double xc2 = m2 - c2;
    if ( x2 < xc2 )
        return x * asin( ( c2 * m2 + ( c2 - m2 ) * x2 + x4 ) / ( c2 * m2 + ( m2 - c2 ) * x2 - x4 ) ) + 2. * m * atan( sqrt( xc2 - x2 ) / c ) - 2. * c * atanh( sqrt( xc2 - x2 ) / m );
    else
        return sqrt( xc2 ) * M_PI_2;
}

inline double i1( double a, double b, double c, double x1, double x2 ) { return i1( a, b, c, x2 ) - i1( a, b, c, x1 ); }

double i2( double a, double b, double c, double x ) {
    double a2 = a * a;
    double b2 = b * b;
    double c2 = c * c;
    double m2 = a2 + b2 + c2;
    double m = sqrt( m2 );
    double x2 = x * x;
    double xc2 = m2 - b2;
    if ( x2 < xc2 )
        return x * asin( ( m2 * ( a2 + c2 ) + ( a2 + c2 - m2 - m2 ) * x2 + x4 ) / ( m2 * ( a2 + c2 ) - ( a2 + c2 ) * x2 - x4 ) ) + 2. * m * atan( sqrt( xc2 - x2 ) / b ) + c * atan( sqrt( xc2 - x2 ) / m );
    else
        return sqrt( xc2 ) * M_PI_2;
}

inline double i2( double a, double b, double c, double x1, double x2 ) { return i2( a, b, c, x2 ) - i2( a, b, c, x1 ); }
```

double i3( double a, double b, double c, double x ) {
    double a2 = a * a;
    double b2 = b * b;
    double c2 = c * c;
    double m2 = a2 + b2 + c2;
    double m = sqrt( m2 );
    double x2 = x * x;
    double xa2 = m2 - a2;
    if ( x2 < xa2 )
        return x * asin( ( m2 * ( b2 + c2 ) + ( b2 + c2 - m2 - m2 ) * x2 ) / ( m2 * ( b2 + c2 ) - ( b2 + c2 ) * x2 ) ) - 2. * m * atan( sqrt( xa2 - x2 ) / a );
    else
        return -sqrt( xa2 ) * M_PI_2;
}

inline double i3( double a, double b, double c, double x1, double x2 ) { return i3( a, b, c, x2 ) - i3( a, b, c, x1 ); }

double i4( double a, double b, double c, double x ) {
    double a2 = a * a;
    double b2 = b * b;
    double c2 = c * c;
    double m2 = a2 + b2 + c2;
    double m = sqrt( m2 );
    double x2 = x * x;
    double xc2 = m2 - c2;
    if ( x2 < xc2 )
        return x * asin( ( c2 * ( c2 - m2 ) + ( c2 + m2 ) * x2 ) / ( c2 * ( m2 - c2 ) + ( m2 - c2 ) * x2 ) ) + 2. * c * atanh( sqrt( xc2 - x2 ) / m );
    else
        return sqrt( xc2 ) * M_PI_2;
}

inline double i4( double a, double b, double c, double x1, double x2 ) { return i4( a, b, c, x2 ) - i4( a, b, c, x1 ); }

int main( int argc, char* argv[] ) {
    double l = 1., w = 1., t = 1.; // default is a cube of unit side length
    if ( argc == 4 ) {  // or specify the 3 dimensions (in any order) on the command line
        l = atof( argv[1] );
        w = atof( argv[2] );
        t = atof( argv[3] );
    }
    const double T = min( l, w, t ); // smallest dimension is thickness
    const double W = mid( l, w, t ); // intermediate dimension is width
    const double L = max( l, w, t ); // largest dimension is length
    const double V = l * w * t; // volume
    const double S1 = pow( V, +2. / 3. ); // factor to convert PDF from area to shape factor
    const double S2 = pow( V, -2. / 3. ); // factor to convert area to shape factor
    double a = W * T; // smallest area, Ax in Walters' formula
    double b = T * L; // intermediate area, Ay in Walters' formula
double c = L * W; // largest area, A_ in Walters' formula

double a2 = a * a;
double b2 = b * b;
double c2 = c * c;
double m = sqrt( m2 );

double xa = sqrt( b2 + c2 );
double xb = sqrt( a2 + c2 );
double xc = sqrt( a2 + b2 );

const double DELTAZ = ( m2 + c2 ) / ( m2 - c2 );
const double DZ = 2. * m2 * c2 / ( m2 - c2 );
const double ALPHAX = 1. / ( c2 + a2 );
const double ALPHAY = 1. / ( c2 + b2 );

const double K1 = asin( DELTAZ - ALPHAX * DZ );
const double K2 = asin( DELTAZ - ALPHAY * DZ );

const double C = 1. / ( M_PI * m );
const double C1 = C * ( i1( a, b, c, a, b ) - i2( a, b, c, a, b ) - K1 * ( b - a ) );
const double C2 = C * ( 2. + i1( a, b, c, b, xc ) - i2( a, b, c, b, xc ) + 2. + i4( a, b, c, b, xc ) - K1 * ( xc - b ) - K2 * ( c - b ) );
const double C3 = 2. * C * ( 1.5 * M_PI * ( xc - xb ) - i2( a, b, c, xc, xb ) - i3( a, b, c, xc, xb ) + i4( a, b, c, xc, xb ) );
const double C4 = 2. * C * ( M_PI * ( xb - c ) - i2( a, b, c, xb ) - i3( a, b, c, xb ) );

const double C5 = 2. * C * ( 1.5 * M_PI * ( x2 - x ) - i2( a, b, c, x2, x ) - i3( a, b, c, x2, x ) + i4( a, b, c, x2, x ) );

double x, x2, delta, d, alpha1, f1, f2, f3, f4, sf, pdf = 0., cdf = 0. ;

da is the minimum projected area and m is the maximum projected area

const int N = 1000;
const double inc = ( m - a ) / double( N );

for ( x = a; x <= m; x += inc ) {
  x2 = x * x;
  delta = ( m2 + x2 ) / ( m2 - x2 );
  d = 2. * m2 * x2 / ( m2 - x2 );
  alpha1 = 1. / ( c2 + x2 );

  f1 = asin( delta - alpha1 * d );
  f2 = asin( DELTAZ - ALPHAX * DZ );
  f3 = asin( DELTAZ - ALPHAY * DZ );
  f4 = asin( DELTAZ - alpha1 * DZ );

  if ( xc <= c ) { // case: a2 + b2 <= c2
    f4 = asin( delta - alpha1 * d );
    f2 = asin( DELTAZ - ALPHAY * DZ );
  }

  if ( a <= x && x < b ) { // case a
    pdf = C * ( f1 - f2 + f4 - K1 );
    cdf = C * ( i1( a, b, c, a, x ) - i2( a, b, c, a, x ) + i4( a, b, c, a, x ) - K1 * ( x - a ) );
  } else if ( b <= x && x <= xc ) { // case b
    pdf = C * ( f1 - f2 + f4 - K1 );
    cdf = C * ( i1( a, b, c, a, x ) - i2( a, b, c, a, x ) + i4( a, b, c, a, x ) - K1 * ( x - a ) );
  }
  else if ( x == N ) { cdf = C * ( i1( a, b, c, a, x ) - i2( a, b, c, a, x ) + i4( a, b, c, a, x ) - K1 * ( x - a ) );
  }
}
\[ pdf = C \cdot (2 \cdot f_1 - f_2 - f_3 + 2 \cdot f_4 - K_1 - K_2); \]
\[ cdf = C_1 + C \cdot (2 \cdot i_1(a, b, c, x) - i_2(a, b, c, x) - i_3(a, b, c, x) + 2 \cdot i_4(a, b, c, x) - K_1 \cdot (x - b) - K_2 \cdot (x - b)); \]
\[ } \]
\[ else if (x_c <= x && x < c) \{ \text{ // case c} \]
\[ pdf = C \cdot (2 \cdot M_\Pi - f_2 - f_3 - K_1 - K_2); \]
\[ cdf = C_1 + C_2 + C \cdot (2 \cdot M_\Pi \cdot (x - x_c) - i_2(a, b, c, x_c) - i_3(a, b, c, x_c) - K_1 \cdot (x - x_c) - K_2 \cdot (x - x_c)); \]
\[ } \]
\[ else if (c <= x && x < x_b) \{ \text{ // case d} \]
\[ pdf = 2 \cdot C \cdot (M_\Pi - f_2 - f_3); \]
\[ cdf = C_1 + C_2 + C_3 + 2 \cdot C \cdot (M_\Pi \cdot (x - c) - i_2(a, b, c, x_c) - i_3(a, b, c, x_c)); \]
\[ } \]
\[ else if (x_b <= x && x < x_a) \{ \text{ // case e} \]
\[ pdf = 2 \cdot C \cdot (1.5 \cdot M_\Pi - f_3); \]
\[ cdf = C_1 + C_2 + C_3 + C_4 + 2 \cdot C \cdot (1.5 \cdot M_\Pi \cdot (x - x_b) - i_3(a, b, c, x_b)); \]
\[ } \]
\[ else if (x_a <= x && x <= m) \{ \text{ // case f} \]
\[ pdf = 4 / m; \]
\[ cdf = C_1 + C_2 + C_3 + C_4 + C_5 + (x - x_a) \cdot 4 / m; \]
\[ } \]
\[ else \{ \text{ // case: a_2 + b_2 > c_2} \]
\[ if (a <= x && x < b) \{ \text{ // case a} \]
\[ pdf = C \cdot (f_1 - f_2 + f_4 - K_1); \]
\[ cdf = C \cdot (i_1(a, b, c, a, x) - i_2(a, b, c, a, x) + i_4(a, b, c, a, x) - K_1 \cdot (x - a)); \]
\[ } \]
\[ else if (b <= x && x < c) \{ \text{ // case b} \]
\[ pdf = C \cdot (2 \cdot f_1 - f_2 - f_3 + 2 \cdot f_4 - K_1 - K_2); \]
\[ cdf = C_1 + C \cdot (2 \cdot i_1(a, b, c, b, x) - i_2(a, b, c, b, x) - i_3(a, b, c, b, x) + 2 \cdot i_4(a, b, c, b, x) - K_1 \cdot (x - b) - K_2 \cdot (x - b)); \]
\[ } \]
\[ else if (c <= x && x < x_c) \{ \text{ // case c} \]
\[ pdf = 2 \cdot C \cdot (f_1 - f_2 + f_4 - f_3); \]
\[ cdf = C_1 + D_2 + 2 \cdot C \cdot (i_1(a, b, c, c, x) - i_2(a, b, c, c, x) - i_3(a, b, c, c, x) + i_4(a, b, c, c, x)); \]
\[ } \]
\[ else if (x_c <= x && x < x_b) \{ \text{ // case d} \]
\[ pdf = 2 \cdot C \cdot (M_\Pi - f_2 - f_3); \]
\[ cdf = C_1 + D_2 + D_3 + 2 \cdot C \cdot (M_\Pi \cdot (x - x_c) - i_2(a, b, c, x_c) - i_3(a, b, c, x_c)); \]
\[ } \]
\[ else if (x_b <= x && x < x_a) \{ \text{ // case e} \]
\[ pdf = 2 \cdot C \cdot (1.5 \cdot M_\Pi - f_3); \]
\[ cdf = C_1 + D_2 + D_3 + D_4 + 2 \cdot C \cdot (1.5 \cdot M_\Pi \cdot (x - x_b) - i_3(a, b, c, x_b)); \]
\[ } \]
\[ else if (x_a <= x && x <= m) \{ \text{ // case f} \]
\[ pdf = 4 / m; \]
\[ cdf = C_1 + D_2 + D_3 + D_4 + C_5 + (x - x_a) \cdot 4 / m; \]
\[ } \]
\[ } \]
\[ sf = x \cdot S_2; \text{ // convert presented area to dimensionless shape factor} \]
\[ std::cout << sf << \t pdf * S_1 \t cdf \std::endl; \text{ // shape factor PDF and CDF} \]
\[ return EXIT_SUCCESS; \]
Consider a right-circular cylinder (RCC) with length $L$ and diameter $D$. Let $\phi_y$ be the yaw angle measured from the axis of symmetry so that $\phi_y = 0$ corresponds to a face-forward orientation of the cylinder. Then the shape factor as a function of yaw angle is

$$\gamma(\phi_y) = \left( \frac{\pi L}{4 D} \right)^{-2/3} \left( \frac{L}{D} \sin \phi_y + \frac{\pi}{4} |\cos \phi_y| \right), \quad (A-39)$$

where $0 \leq \phi_y \leq \pi$. The minimum shape factor is

$$\gamma_{\text{min}} = \left( \frac{\pi L}{4 D} \right)^{-2/3} \min \left( \frac{L}{D}, \frac{\pi}{4} \right) \quad (A-40)$$

and is realized at the yaw angle

$$\phi_y = \begin{cases} \pi/2 & \text{if } L/D < \pi/4 \\ 0 & \text{if } L/D > \pi/4 \end{cases} \quad (A-41)$$

The maximum shape factor is

$$\gamma_{\text{max}} = \left( \frac{\pi L}{4 D} \right)^{-2/3} \sqrt{\left( \frac{L}{D} \right)^2 + \left( \frac{\pi}{4} \right)^2} \quad (A-42)$$

and is realized at

$$\phi_y = \tan^{-1} \left( \frac{L/D}{\pi/4} \right) \quad (A-43)$$

The mean shape factor when averaged over all random orientations is

$$\bar{\gamma} = \left( \frac{\pi}{4} \right)^{1/3} \left( \frac{L}{D} \right)^{-2/3} \left( \frac{L}{D} + \frac{1}{2} \right) \quad (A-44)$$

Some plots of Eqs. A-39, A-42, and A-44 as a function of $L/D$ are shown in Fig. A-3.

---


Approved for public release; distribution is unlimited.
Fig. A-3. Shape factors of a cylinder as a function of $L/D$ at a fixed yaw angle from Eq. A-39 are displayed at $30^\circ$, $45^\circ$, and $60^\circ$. The maximum shape factor as a function of $L/D$ is from Eq. A-42, and the average shape factor of a randomly oriented cylinder as a function of $L/D$ is from Eq. A-44.

The minimum of the maximum shape factor curve is $(\gamma_{\text{max}})_{\text{min}} = \sqrt{3} \left( \frac{\pi}{2} \right)^{-1/3} \approx 1.49$ and occurs when $L/D = \sqrt{2} \left( \frac{\pi}{4} \right) \approx 1.11072$. The minimum of the mean shape factor curve is $\bar{\gamma}_{\text{min}} = \frac{3}{2} \left( \frac{\pi}{4} \right)^{1/3} \approx 1.38395$ and occurs when $L/D = 1$.

In Eq. A-39, let $x \equiv (L/D)^{-1/3}$; then it can be written as

$$x^3 - \left( \frac{\pi}{4} \right)^{-1/3} \frac{\gamma}{\cos \phi_y} x + \left( \frac{\pi}{4} \right)^{-1} \tan \phi_y = 0. \quad (A-45)$$

Square Eq. A-42 and let $x \equiv (L/D)^{-2/3}$; then it can be written as

$$x^3 - \left( \frac{\pi}{4} \right)^{-2/3} \gamma_{\text{max}}^2 x + \left( \frac{\pi}{4} \right)^{-2} = 0. \quad (A-46)$$

And in Eq. A-44, let $x \equiv (L/D)^{-1/3}$; then it can be written as

$$x^3 - 2 \left( \frac{\pi}{4} \right)^{-1/3} \bar{\gamma} x + 2 = 0. \quad (A-47)$$

So we see that these 3 equations all have the same form:

$$x^3 + px + q = 0, \quad (A-48)$$
where

- \( x = \left( \frac{L}{D} \right)^{-1/3}, \ p = -\left( \frac{\pi}{4} \right)^{-1/3} \frac{\gamma}{\cos \phi_y}, \ q = \left( \frac{\pi}{4} \right)^{-1} \tan \phi_y \) in Eq. A-45;
- \( x = \left( \frac{L}{D} \right)^{-2/3}, \ p = -\left( \frac{\pi}{4} \right)^{-2/3} \gamma_{\text{max}}^2, \ q = \left( \frac{\pi}{4} \right)^{-2} \) in Eq. A-46;
- \( x = \left( \frac{L}{D} \right)^{-1/3}, \ p = -2 \left( \frac{\pi}{4} \right)^{-1/3} \bar{\gamma}, \ q = 2 \) in Eq. A-47.

### A-3.1. Diversion: Solution to Cubic Equation in the Case of Real Roots

Without loss of generality, the general cubic can be written as

\[
x^3 + ax^2 + bx + c = 0.
\]

Setting \( x = y - a/3 \) eliminates the quadratic term and puts it in the form

\[
y^3 + py + q = 0,
\]

with

\[
p = b - \frac{a^2}{3} \quad \text{and} \quad q = c - \frac{ab}{3} + \frac{2a^3}{27}.
\]

We note in passing that the absence of the quadratic term implies that the roots must sum to zero. For if \( y_1, y_2, \) and \( y_3 \) are the roots of Eq. A-50, then

\[
(y - y_1)(y - y_2)(y - y_3) = y^3 - (y_1 + y_2 + y_3)y^2 + (y_1y_2 + y_1y_3 + y_2y_3)y - y_1y_2y_3 = 0,
\]

and eliminating the \( y^2 \) term means that \( y_1 + y_2 + y_3 = 0 \).

A useful trick\(^4\) for solving the cubic without the quadratic term (Eq. A-50) is to make use of the trigonometric identity

\[
4 \cos^3 \theta - 3 \cos \theta - \cos(3\theta) = 0
\]

which can be easily derived by using the double angle formulas in the expansion of \( \cos(3\theta) \). Let us return then to Eq. A-48 to see if we can transform it into this form.

---


Approved for public release; distribution is unlimited.
Begin by setting

\[ x = t \cos \theta, \]  

which gives

\[ t^3 \cos^3 \theta + pt \cos \theta + q = 0, \]  

or, multiplying through by \(4/t^3\),

\[ 4 \cos^3 \theta + \frac{4p}{t^2} \cos \theta + \frac{4q}{t^3} = 0. \]  

Choosing

\[ t = \sqrt{-\frac{4p}{3}} \]  

then gives

\[ 4 \cos^3 \theta - 3 \cos \theta - \frac{3q}{p} \sqrt{-\frac{3}{4p}} = 0. \]  

Now we see that if we choose \(\theta\) such that

\[ \cos(3\theta) = \frac{3q}{p} \sqrt{-\frac{3}{4p}}, \]  

then the cubic is automatically satisfied, guaranteed by the trigonometric identity Eq. A-53. Therefore,

\[ 3\theta = \cos^{-1} \left( \frac{3q}{p} \sqrt{-\frac{3}{4p}} \right) + 2\pi k \quad \text{where} \quad k = 0, \pm 1, \]  

and from Eqs. A-54 and A-57, the solutions for \(x\) are

\[ x_k = \sqrt{-\frac{4p}{3}} \cos \left( \frac{1}{3} \cos^{-1} \left( \frac{3q}{p} \sqrt{-\frac{3}{4p}} \right) + \frac{k2\pi}{3} \right) \quad \text{for} \quad k = 0, \pm 1. \]  

It is easy enough to check that the 3 solutions do indeed sum to zero as promised, since

\[ \cos \theta + \cos \left( \theta + \frac{2\pi}{3} \right) + \cos \left( \theta - \frac{2\pi}{3} \right) = 0 \]  

for arbitrary \(\theta\). Two of the solutions will be positive, corresponding to a disk-like and a rod-like cylinder, and the third solution will be negative, which is of no physical interest.
The solutions for $L/D$ are

- $\frac{L}{D} = x_0^{-3}$ for disk-like cylinders and $\frac{L}{D} = x_1^{-3}$ for rod-like cylinders in Eq. A-45;
- $\frac{L}{D} = x_0^{-3/2}$ for disk-like cylinders and $\frac{L}{D} = x_1^{-3/2}$ for rod-like cylinders in Eq. A-46;
- $\frac{L}{D} = x_0^{-3}$ for disk-like cylinders and $\frac{L}{D} = x_1^{-3}$ for rod-like cylinders in Eq. A-47.

### A-3.2. Cylinder Probability Distributions

There are 2 cases to consider: rod-like with $\frac{L}{D} > \frac{\pi}{4}$ and disk-like with $\frac{L}{D} < \frac{\pi}{4}$.

For the following formulas, define $a \equiv \left(\frac{\pi L}{4 D}\right)^{-2/3} \frac{L}{D}$ and $b \equiv \left(\frac{\pi L}{4 D}\right)^{-2/3} \frac{\pi}{4}$.

- **$L/D > \pi/4$:**
  
  The PDF is given by
  \[
  f(\gamma) = \begin{cases} 
  a\gamma - b\sqrt{\gamma_{\text{max}}^2 - \gamma^2} & \text{if } b \leq \gamma < a \\
  \frac{\gamma_{\text{max}}^2}{\gamma_{\text{max}}^2 - \gamma^2} & \text{if } a \leq \gamma < \gamma_{\text{max}} \\
  2a\gamma & \text{if } \gamma \leq \gamma_{\text{max}} 
  \end{cases} \tag{A-63}
  \]
  
  and the CDF is given by
  \[
  F(\gamma) = \begin{cases} 
  1 - \frac{b\gamma + a\sqrt{\gamma_{\text{max}}^2 - \gamma^2}}{\gamma_{\text{max}}} & \text{if } b \leq \gamma < a \\
  1 - \frac{2a\sqrt{\gamma_{\text{max}}^2 - \gamma^2}}{\gamma_{\text{max}}^2} & \text{if } a \leq \gamma \leq \gamma_{\text{max}} 
  \end{cases} \tag{A-64}
  \]

- **$L/D < \pi/4$:**
  
  The PDF is given by
  \[
  f(\gamma) = \begin{cases} 
  a\gamma + b\sqrt{\gamma_{\text{max}}^2 - \gamma^2} & \text{if } a \leq \gamma < b \\
  \frac{\gamma_{\text{max}}^2}{\gamma_{\text{max}}^2 - \gamma^2} & \text{if } b \leq \gamma \leq \gamma_{\text{max}} \\
  2a\gamma & \text{if } \gamma \leq \gamma_{\text{max}} 
  \end{cases} \tag{A-65}
  \]
and the CDF is given by

\[
F(\gamma) = \begin{cases} 
\frac{b\gamma - a\sqrt{\gamma^2_{\text{max}} - \gamma^2}}{\gamma^2_{\text{max}}} & \text{if } a \leq \gamma < b \\
1 - \frac{2a\sqrt{\gamma^2_{\text{max}} - \gamma^2}}{\gamma^2_{\text{max}}} & \text{if } b \leq \gamma \leq \gamma_{\text{max}}
\end{cases}
\]  
(A-66)

These functions are plotted in Fig. A-4 for the case when \(L/D = 1\).

Fig. A-4. Plot of shape factor PDF and CDF for a randomly oriented cylinder with \(L/D = 1\). Notice the discontinuity at \(\gamma = (\pi/4)^{-2/3}\) where the shape factor changes from being single-valued to being multivalued.

Equations A-64 and A-66 can be solved for \(\gamma\) for a given value of \(F\)—and in this way turn this into a direct and fast method for computing the shape factor probability distribution of a random tumbling cylinder. The resulting algorithm has been implemented into the C++ code in Listing A-3.

**Listing A-3. algo.cpp**

```cpp
#include <iostream>
#include <cmath>
#include <chrono>
#include <random>
using namespace std;

int main( int argc, char* argv[] ) {
  double l_d = 1.;
  if ( argc == 2 ) l_d = atof( argv[1] ); // L/D = 1 is default or override with 1st arg
  int N = 1000;
  if ( argc == 3 ) N = atoi( argv[2] ); // 1000 samples is default or override with 2nd arg
  double C = pow( M_PI_4 * l_d, -2. / 3. );
  const double A = C * l_d;
  const double B = C * M_PI_4;
  const double G_MAX2 = A * A + B * B;
  const double G_MAX = sqrt( G_MAX2 );
  const double K = G_MAX / ( 2. * A );
  const double P1 = 1. - 2. * A * B / G_MAX2;
}```
This code generates over 28 million shape factors per second on a Mac with a 2.4-GHz Intel Xeon processor. Running the code for an $L/D = 1$ cylinder gives the results shown in Fig. A-5.

![Fig. A-5. Histograms of shape factor PDF and CDF for a randomly oriented $L/D = 1$ cylinder compared to plots from analytical formulas, Eqs. A-65 and A-66. Notice the jump in the PDF at $\gamma = (\pi/4)^{-2/3} \approx 1.175$, as predicted (compare to Fig. A-4).](image-url)
A-4. Tetrahedron

Consider a regular tetrahedron of unit side length, which has a volume of \( \frac{1}{3} \left( \frac{1}{\sqrt{2}} \right)^3 \). Its presented area ranges from \( \frac{1}{\sqrt{8}} \) to \( \frac{1}{2} \). The PDF as a function of area is given by \(^1\)

\[
f(A) = \begin{cases} 
\frac{12}{\pi} \sin^{-1} \left( \frac{8A^2 - 1}{1 - 4A^2} \right) + \frac{8\sqrt{3}}{\pi} \cos^{-1} \left( \frac{3 - 20A^2}{3 - 16A^2} \right) - \frac{8\sqrt{3}}{\pi} A^+ & \text{if } \frac{1}{\sqrt{8}} \leq A \leq \frac{1}{\sqrt{6}} \\
6 + \frac{16}{\sqrt{3}} & \text{if } \frac{1}{\sqrt{6}} \leq A \leq \frac{\sqrt{3}}{4} \\
6 & \text{if } \frac{\sqrt{3}}{4} \leq A \leq \frac{1}{2} 
\end{cases}
\]

(A-67)

Since presented area, \( A \), and dimensionless shape factor, \( \gamma \), are related by

\[ A = \gamma V^{2/3}, \]

where \( V \) is the volume, we can get the PDF as a function of \( \gamma \) by simply scaling the area by \( V^{-2/3} = 2 \cdot 3^{2/3} \).

The CDF as a function of area is given by

\[
F(A) = \begin{cases} 
\frac{12}{\pi} A \sin^{-1} \left( \frac{8A^2 - 1}{1 - 4A^2} \right) + \frac{8\sqrt{3}}{\pi} A \cos^{-1} \left( \frac{3 - 20A^2}{3 - 16A^2} \right) - \frac{8\sqrt{3}}{\pi} A^+ & \text{if } \frac{1}{\sqrt{8}} \leq A \leq \frac{1}{\sqrt{6}} \\
\frac{6}{\pi} \left[ \tan^{-1} \left( \frac{\sqrt{2}(1 - 3A)}{\sqrt{1 - 6A^2}} \right) + \tan^{-1} \left( \frac{\sqrt{2}(1 + 3A)}{\sqrt{1 - 6A^2}} \right) \right] & \text{if } \frac{1}{\sqrt{6}} \leq A \leq \frac{\sqrt{3}}{4} \\
\frac{6}{\pi} \left[ \tan^{-1} \left( \frac{\sqrt{2} - 3\sqrt{3}A}{\sqrt{1 - 6A^2}} \right) + \tan^{-1} \left( \frac{\sqrt{2} + 3\sqrt{3}A}{\sqrt{1 - 6A^2}} \right) \right] & \text{if } \frac{\sqrt{3}}{4} \leq A \leq \frac{1}{2} \\
\left( 6 + \frac{16}{\sqrt{3}} \right) A + \frac{6}{\pi} \left[ \tan^{-1} (3 - 2\sqrt{2}) - \tan^{-1} (3 + 2\sqrt{2}) \right] - \\
\frac{6}{\pi} \left[ \tan^{-1} (4\sqrt{2} - 3\sqrt{3}) + \tan^{-1} (4\sqrt{2} + 3\sqrt{3}) \right] & \text{if } \frac{1}{\sqrt{8}} \leq A \leq \frac{1}{\sqrt{6}} \\
\left( 6 + \frac{16}{\sqrt{3}} \right) A + \frac{6}{\pi} \left[ \tan^{-1} (3 - 2\sqrt{2}) - \tan^{-1} (3 + 2\sqrt{2}) \right] - \\
\frac{6}{\pi} \left[ \tan^{-1} (4\sqrt{2} - 3\sqrt{3}) + \tan^{-1} (4\sqrt{2} + 3\sqrt{3}) \right] & \text{if } \frac{1}{\sqrt{6}} \leq A \leq \frac{\sqrt{3}}{4} \\
6A - 2 & \text{if } \frac{\sqrt{3}}{4} \leq A \leq \frac{1}{2} 
\end{cases}
\]

(A-69)

Listing A-4. tetrahedron.cpp

// tetrahedron.cpp: generates plotting points for tetrahedron pdf and cdf
// Ref: Vickers, G. T. and Brown, D. J.,
// "The distribution of projected area and perimeter of convex, solid particles,"
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <cassert>

const double CUBE_VOLUME = 1. / ( M_SQRT2 * M_SQRT2 * M_SQRT2 ); // volume of cube that encloses the tetrahedron with unit side length
const double TETRAHEDRON_VOLUME = CUBE_VOLUME / 3.; // volume of tetrahedron with unit side length
const double FACTOR = pow( TETRAHEDRON_VOLUME, -2. / 3. ); // factor that converts presented area to dimensionless shape factor
const double SQRT3 = sqrt( 3. );
const double SQRT6 = M_SQRT2 * SQRT3;
const double SQRT8 = sqrt( 8. );
const double AMIN = 1. / SQRT8;
const double AMAX = 1. / 2.;

double pdf( double x ) {
    assert( AMIN <= x && x <= AMAX );
    if ( x < 1. / SQRT6 ) {
        double x2 = x * x;
    } else if ( x < SQRT3 / 4. )
        return 6. + 16. / SQRT3;
    else
        return 6.;
}

double cdf( double x ) {
    assert( AMIN <= x && x <= AMAX );
    if ( x < 1. / SQRT6 ) {
        double x2 = x * x;
        double x4 = x2 * x2;
    } else if ( x < SQRT3 / 4. ) {
    } else
        return 6. * x - 2.;
}

int main( int argc, char** argv ) {
    // specify number of points on commandline or use 1000
    int N = 1000;
    if ( argc == 2 ) N = atof( argv[1] );
    for ( double a = AMIN; a <= AMAX; a += ( AMAX - AMIN ) / double( N ) ) {
        double g = FACTOR * a;
        std::cout << g << "\\t" << pdf( a ) / FACTOR << "\\t" << cdf( a ) << std::endl;
    }
    return EXIT_SUCCESS;
}
This is plotted in Fig. A-6.

Fig. A-6. Plot of shape factor PDF and CDF for a randomly oriented regular tetrahedron

A-5. Ellipsoid

The equation of an ellipsoid, in which its principal axes are aligned with the coordinate axes, is given by

\[
\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1, \tag{A-70}
\]

where \(a, b,\) and \(c\) are the 3 semi-principal axes, and its volume is

\[
V = \frac{4}{3} \pi abc. \tag{A-71}
\]

Let the viewing angle be specified by the polar angle \(\theta\) and the azimuthal angle \(\phi\). The presented area of the ellipsoid will be in the shape of an ellipse of area\(^5\)

\[
A = \pi \sqrt{b^2 c^2 \sin^2 \theta \cos^2 \phi + a^2 c^2 \sin^2 \theta \sin^2 \phi + a^2 b^2 \cos^2 \theta}. \tag{A-72}
\]

Let the dimensions be ordered so that \(a \leq b \leq c\). The CDF as a function of area is given by\(^1\)

\[
F(x) = \begin{cases} 
1 - \frac{2}{\pi} \sqrt{\xi \eta} \left[ R_F(0, \xi - 1, \xi - \eta) - \frac{1}{3} R_J(0, \xi - 1, \xi - \eta, \xi) \right] & \text{if } A_{\min} \leq x \leq A_m \\
\frac{2}{\pi} \sin^{-1} \sqrt{(A_m^2 - A_{\min}^2)/(A_{\max}^2 - A_{\min}^2)} & \text{if } x = A_m \\
1 - \frac{2}{\pi} \sqrt{\xi \eta} \left[ R_F(0, 1 - \xi, 1 - \eta) - \frac{1}{3} R_J(0, 1 - \xi, 1 - \eta, 1) \right] & \text{if } A_m < x \leq A_{\max}
\end{cases} \tag{A-73}
\]

where
\[ \xi = \frac{A_{\text{max}}^2 - x^2}{A_{\text{max}}^2 - A_m^2}, \quad \eta = \frac{A_{\text{max}}^2 - x^2}{A_{\text{max}}^2 - A_{\text{min}}^2}, \quad A_{\text{min}} = \pi a b, \quad A_m = \pi a c, \quad A_{\text{max}} = \pi b c, \] (A-74)
and the functions \( R_F \) and \( R_J \) are the Carlson symmetrized form of the classic elliptic integrals:
\[ R_F(x, y, z) = \frac{1}{2} \int_0^\infty \frac{dt}{\sqrt{(t + x)(t + y)(t + z)}} \] (A-75)
and
\[ R_J(x, y, z, \rho) = \frac{3}{2} \int_0^\infty \frac{dt}{(t + \rho)\sqrt{(t + x)(t + y)(t + z)}}. \] (A-76)

It is shown in *Numerical Recipes*\(^6\) that the PDF as a function of area is given by
\[
f(x) = \begin{cases} \frac{2x}{\pi} R_F(0, (A_m^2 - x^2)(A_{\text{max}}^2 - A_m^2), (A_{\text{max}}^2 - x^2)(A_m^2 - A_{\text{min}}^2)) & \text{if } A_{\text{min}} \leq x \leq A_m \\ \frac{2x}{\pi} R_F(0, (x^2 - A_m^2)(A_{\text{max}}^2 - A_m^2), (x^2 - A_{\text{min}}^2)(A_{\text{max}}^2 - A_m^2)) & \text{if } A_m < x \leq A_{\text{max}} \end{cases} \] (A-77)

There is a logarithmic singularity in \( f \) at \( x = A_m \) when \( a, b, \) and \( c \) are all different.

The total surface area of the ellipsoid is given by
\[
S = 2\pi a^2 + 2\pi b c a R_F\left(\frac{1}{a^2}, \frac{1}{b^2}, \frac{1}{c^2}\right) - \frac{2\pi a b c}{3} \left(\frac{1}{a^2} - \frac{1}{c^2}\right) R_F\left(\frac{1}{c^2}, \frac{1}{b^2}, \frac{1}{a^2}\right), \] (A-78)

and from this we can get the mean presented area as \( S/4 \) from Cauchy’s theorem.

The presented areas in these formulas are easily converted to dimensionless shape factors, \( \gamma \), via the relationship
\[ A = \gamma V^{2/3}, \] (A-79)
where \( V \) is the ellipsoid volume. To convert the PDF from a function of area to a function of shape factor, use
\[
f(\gamma) = \frac{dF}{d\gamma} = \frac{dF}{dA} \frac{dA}{d\gamma} = V^{2/3} f(x). \] (A-80)

The formulas in Eqs. A-77 and A-74 are plotted in Fig. A-7.

---

Fig. A-7. Plot of shape factor PDF and CDF for an ellipsoid with $a = 1$, $b = 2$, $c = 3$. There is a logarithmic singularity in the PDF when $a$, $b$, and $c$ are all different and is located at

$$\gamma = \left(\frac{4}{3}\right)^{-2/3}\left(\frac{\pi ac}{b^2}\right)^{1/3},$$

which in this case is at $3\pi^{1/3}/4 \approx 1.098$.

Listings A-5 through A-8 can be used for printing out the solid curves in Fig. A-7.

---

**Listing A-5. ellipsoid.cpp**

```cpp
// ellipsoid.cpp: Prints out the PDF and CDF as a function of shape factor for an ellipsoid
// Specify the dimension of the ellipsoid on the commandline (or use default 3/2/1)
// R. Saucier, Feb 2016

#include <iostream>
#include <cstdlib>
#include <cmath>

double rf(double x, double y, double z);

double rj(double x, double y, double z, double p);

inline double min(double a, double b, double c) { return std::min(std::min(a, b), c); }
inline double max(double a, double b, double c) { return std::max(std::max(a, b), c); }
inline double mid(double a, double b, double c) { return std::max(std::min(a, b), std::min(std::max(a, b), c)); }

class PDF { // functor for probability density function
public:
    PDF(double amin, double am, double amax) : _amin(amin), _am(am), _amax(amax) {} 
    double operator()(double x) {
        double y, z;
        if (_amin <= x && x <= _am) {
            y = (_am - x) * (_am + x) * (_amax - _amin) * (_amax + _amin);
            z = (_amax - x) * (_amax + x) * (_am - _amin) * (_am + _amin);
        } else {
            y = (x - _am) * (x + _am) * (_amax - _amin) * (_amax + _amin);
            z = (x - _amin) * (x + _amin) * (_amax - _am) * (_amax + _am);
        }
        return (x / M_PI_2) * rf(0, y, z);
    }
private:
    double _amin, _am, _amax;
};

class CDF { // functor for cumulative distribution function
public:
    CDF(double amin, double am, double amax) : _amin(amin), _am(am), _amax(amax) {} 
    double operator()(double x) {
        return asin(sqrt((_am - _amin) * (_am + _amin) / ((_amax - _amin) * (_amax + _amin)))) / M_PI_2;
    }
private:
    double _amin, _am, _amax;
};
```

Approved for public release; distribution is unlimited.

62
55  p = 1;
56  }
57  return 1. - ( sqrt( xi * eta ) / M_PI_2 ) * ( rf( 0, y, z ) - rj( 0, y, z, p ) / 3. );
58 }
59
60 private:
61  double _amin, _am, _amax;
62
63 int main( int argc, char* argv[] ) {
64
65  double a = 1., b = 2., c = 3.;
66  if ( argc == 4 ) { // or specify the 3 dimensions (in any order) on the command line
67    a = atof( argv[1] );
68    b = atof( argv[2] );
69    c = atof( argv[3] );
70  }
71  const double A = min( a, b, c ); // minimum value
72  const double B = mid( a, b, c ); // intermediate value
73  const double C = max( a, b, c ); // maximum value
74  const double V = ( 4. / 3. ) * M_PI * A * B * C; // ellipsoid volume
75  const double S1 = pow( V, +2. / 3. ); // factor to convert PDF from area to shape factor
76  const double S2 = pow( V, -2. / 3. ); // factor to convert area to shape factor
77  const double AMIN = M_PI * A * B;
78  const double AM = M_PI * A * C;
79  const double AMAX = M_PI * B * C;
80
81  PDF pdf( AMIN, AM, AMAX );
82  CDF cdf( AMIN, AM, AMAX );
83  const int N = 1000;
84  for ( double x = AMIN; x <= AMAX; x += ( AMAX - AMIN ) / double( N ) )
85    std::cout << S2 * x << "\t" << S1 * pdf( x ) << "\t" << cdf( x ) << std::endl;
86
87  return EXIT_SUCCESS;
88
Listing A-6. rf.cpp

// rf.cpp: Computes Carlson's elliptic integral of the first kind, Rf(x,y,z),
// where x, y, and z must be nonnegative and at most one can be zero. TINY
// must be at least 5 times the machine underflow limit and BIG at most
// one fifth the machine overflow limit.
// Ref: Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.,

#include <cmath>
#include <cstdlib>
#include <iostream>

inline double FMIN3( double a, double b, double c ) { return std::min( std::min( a, b ), c ); }
inline double FMAX3( double a, double b, double c ) { return std::max( std::max( a, b ), c ); }

double rf( double x, double y, double z ) {

const double ERRTOL = 0.08;
const double TINY = 5.7e-103; // at least 2(DBL_MIN)^(1/3), where DBL_MIN = 2.22507e-308
const double BIG = 1.1e+102; // at most (1/5)(DBL_MAX)^(1/3), where DBL_MAX = 1.79769e+308

const double THIRD = 1. / 3.;
const double C1 = 1. / 24.;
const double C2 = 0.1;
const double C3 = 3. / 44.;
const double C4 = 1. / 14.;

double alamb, ave, delx, dely, delz, e2, e3, sqrtx, sqrty, sqrtz, xt, yt, zt;

if ( FMIN3( x, y, z ) < 0.0 ||
FMIN3( x + y, x + z, y + z ) < TINY ||
FMAX3( x, y, z ) > BIG ) {
std::cerr << "Invalid arguments in rf: " << std::endl;
std::cerr << " x = " << x << std::endl;
std::cerr << " y = " << y << std::endl;
std::cerr << " z = " << z << std::endl;
exit( EXIT_FAILURE );
}

xt = x;
yt = y;
zt = z;

do {
  sqrtx = sqrt( xt );
  sqrty = sqrt( yt );
  sqrtz = sqrt( zt );
  alamb = sqrtx * ( sqrty + sqrtz ) + sqrty * sqrtz;
  xt = 0.25 * ( xt + alamb );
  yt = 0.25 * ( yt + alamb );
  zt = 0.25 * ( zt + alamb );
}

}
47  \[ yt = 0.25 \times ( yt + \alpha \lambda ); \]
48  \[ zt = 0.25 \times ( zt + \alpha \lambda ); \]
49  \[ ave = THIRD \times ( xt + yt + zt ); \]
50  \[ delx = ( ave - xt ) / ave; \]
51  \[ dely = ( ave - yt ) / ave; \]
52  \[ delz = ( ave - zt ) / ave; \]
53  \[
54  \]
55  \[
56  \]  
57  \}

Listing A-7. rj.cpp
Listing A-8. rc.cpp

// rc.cpp: Computes Carlson's degenerate elliptic integral, Rc(x,y), where x must
// be nonnegative and y must be nonzero. If y < 0, the Cauchy principal
// value is returned. TINY must be at least 5 times the machine underflow
// limit and BIG must be at most one fifth the machine overflow limit.
// Ref: Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.,

#include <cmath>
#include <cstdlib>
#include <iostream>

double rc( double x, double y ) {
    // Compute Carlson's degenerate elliptic integral, Rc(x,y).
    // x must be nonnegative and y must be nonzero. If y < 0, the Cauchy principal
    // value is returned. TINY must be at least 5 times the machine underflow
    // limit and BIG must be at most one fifth the machine overflow limit.
    // Ref: Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.,

    const double ERRTOL = 0.04;
    const double TINY = 5.7e-103; // at least 2(DBL_MIN)^1/3, where DBL_MIN = 2.22507e-308
    const double BIG = 1.1e+102; // at most (1/5)(DBL_MAX)^1/3, where DBL_MAX = 1.79769e+308
    const double SQRTNY = sqrt( TINY );
    const double TNBG = TINY * BIG;
    const double COMP1 = 2.236 / SQRTNY;
    const double COMP2 = TNBG * TNBG / 25.;
    const double THIRD = 1. / 3.;
    const double C1 = 0.3;
    const double C2 = 1. / 7.;
    const double C3 = 0.375;
    const double C4 = 9. / 22.;
    const double C5 = 2.236 / SQRTNY;
    const double C6 = COMP1 * C5;
    const double C7 = 2. ;
    const double C8 = 1. / 7.;
    const double C9 = 1. / 3.;

    double alamb, ave, s, w, xt, yt;
    if ( x <= 0.0 || y <= 0.0 )
        return -0.0;
    if ( x + fabs( y ) ) < TINY )
        return -0.0;
    if ( y < -COMP1 && x > 0.0 && x < COMP2 )
        return -0.0;

    xt = x;
    yt = y;
    w = 1.0;
    } else {
        xt = x - y;
        yt = y;
        w = sqrt( xt ) / sqrt( yt );
    }

    do {
        sqrtx = sqrt( xt );
        sqrtz = sqrt( xt );
        alamb = sqrtx * ( sqrtz + sqrtz ) + sqrtz * sqrtz;
        beta = pt * SQR( pt + alamb );
        sum += fac * rc( alpha, beta );
        fac *= 0.25;
        xt = 0.25 * ( xt + alamb );
        yt = 0.25 * ( yt + alamb );
        pt = 0.25 * ( pt + alamb );
        ave = 0.25 * ( xt + yt + zt + pt + pt );
        delx = ( ave - xt ) / ave;
        dely = ( ave - yt ) / ave;
        delz = ( ave - zt ) / ave;
        delp = ( ave - pt ) / ave;
    }
    } while( FMAX4( fabs( delx ), fabs( dely ), fabs( delz ), fabs( delp ) ) > ERRTOL );

    if ( p <= 0.0 ) ans = a * ( b * ans + 3.0 * ( rcx - rf( xt, yt, zt ) ) );
    return ans;
}
\begin{verbatim}
48  alamb = 2. * sqrt( xt ) * sqrt( yt ) + yt;
49  xt = 0.25 * ( xt + alamb );
50  yt = 0.25 * ( yt + alamb );
51  ave = THIRD * ( xt + yt + yt );
52  s = ( yt - ave ) / ave;
53  while ( fabs( s ) > ERRTOL );
54  return w * ( 1. + s * s * ( C1 + s * ( C2 + s * ( C3 + s * C4 ) ) ) ) / sqrt( ave );
\end{verbatim}
Appendix B. Uniform Sampling over the Unit Sphere
The global version of Archimedes’ theorem\(^1\) states that the area of a sphere is equal to the area of a cylinder circumscribed about the sphere, excluding the bases. The area of a unit sphere is \(4\pi\). The area of the circumscribed cylinder is the circumference times the height: \(2\pi \times 2 = 4\pi\). The local version of the theorem states further that any region on the sphere is equal to the axial projection on the cylinder. This is a very powerful theorem for our purposes since it is much easier to define a sampling strategy on the cylinder, which we can lay out flat and independently sample \(\phi\) and \(z\), and then use Archimedes’ theorem to map onto the unit sphere.

Let \(\theta\) and \(\phi\) be the polar and azimuthal angles, respectively, on the unit sphere, and let \(\phi\) and \(z\) be coordinates on the circumscribed cylinder, where \(\theta \in [0, \pi]\), \(\phi \in [0, 2\pi]\), and \(z \in [-1, 1]\). Then the mapping from the cylinder to the sphere \([0, 2\pi] \times [-1, 1]\) \(\Rightarrow S^2(\theta, \phi)\) is simply

\[\theta = \cos^{-1} z\]  \hspace{1cm} (B-1)

while the \(\phi\) value remains the same. Now that we know the mapping from the cylinder to the sphere, we focus on the sampling strategy on the unwrapped cylinder, the aspect ratio of which is depicted in Fig. B-1.

![Fig. B-1. Sampling on the \([0, 2\pi] \times [-1, 1]\) circumscribed cylinder allows us to sample both \(\phi\) and \(z\) uniformly and independently over their entire range](image)

We describe 4 sampling strategies, 2 randomized and 2 deterministic.

---


Approved for public release; distribution is unlimited.
The first is independent uniform random sampling on both $\phi \sim U(0, 2\pi)$ and $z \sim U(-1, 1)$. The C++ code is given in Listing B-1.

Listing B-1. uniform.cpp

```cpp
#include "Random.h"
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <iomanip>

int main( int argc, char* argv[] ) {
    unsigned int N = 1000;
    if ( argc == 2 ) N = atoi( argv[1] );

    std::cout << std::setprecision(6) << std::fixed;
    rng::Random rng;
    double th, ph, x, y, z;
    for ( unsigned int n = 0; n < N; n++ ) {
        x = rng.uniform( 0., 2. * M_PI );
        z = rng.uniform( -1., 1 );
        //std::cout << x << "\t" << z << std::endl;
        ph = x;
        th = acos( z );
        //std::cout << ph << "\t" << th << std::endl;
        x = sin( th ) * cos( ph );
        y = sin( th ) * sin( ph );
        std::cout << x << "\t" << y << "\t" << z << std::endl;
    }
    return EXIT_SUCCESS;
}
```

This is the simplest strategy, and it has the advantage that we do not need to know the total number of sample points beforehand; we can simply continue until we meet some convergence criterion. The biggest disadvantage is that it produces a pattern that contains clustering of points and is not very uniform, as we see in Fig. B-2.

**B-2. Stratified Random**

We can improve upon the clustering problem that we see with uniform sampling by using stratified random sampling. This can be achieved by imposing a grid on the cylinder and drawing a random sample within each grid cell. The C++ code is given in Listing B-2.

Listing B-2. strat.cpp

```cpp
#include "Random.h"

int main( int argc, char* argv[] ) {
    unsigned int N = 1000;
    if ( argc == 2 ) N = atoi( argv[1] );

    std::cout << std::setprecision(6) << std::fixed;
    rng::Random rng;
    double th, ph, x, y, z;
    for ( unsigned int n = 0; n < N; n++ ) {
        x = rng.uniform( 0., 2. * M_PI );
        z = rng.uniform( -1., 1 );
        //std::cout << x << "\t" << z << std::endl;
        ph = x;
        th = acos( z );
        //std::cout << ph << "\t" << th << std::endl;
        x = sin( th ) * cos( ph );
        y = sin( th ) * sin( ph );
        std::cout << x << "\t" << y << "\t" << z << std::endl;
    }
    return EXIT_SUCCESS;
}
```

---

#include <iostream>
#include <cstdlib>
#include <cmath>

int main(int argc, char* argv[]) {
    // override default N on commandline
    int N = 32; // number of points is N^2, so default is 32^2 = 1024
    if (argc == 2) N = atoi(argv[1]);
    const double DEL_X = 2. * M_PI / double(N);
    const double DEL_Z = 2. / double(N);

    rng::Random rng;

double x, y, ph1, ph2, z, z1, z2, ph, th;
    for (int i = 1; i <= N; i++) {
        z2 = i * DEL_Z;
        z1 = z2 - DEL_Z;
        for (int j = 1; j <= N; j++) {
            ph2 = j * DEL_X;
            ph1 = ph2 - DEL_X;
            ph = rng.uniform(ph1, ph2);
            z = rng.uniform(z1, z2) - 1.;

            //std::cout << ph << "\t" << z << std::endl;
            th = acos(z);
            //std::cout << ph << "\t" << th << std::endl;
            x = sin(th) * cos(ph);
            y = sin(th) * sin(ph);
            //std::cout << x << "\t" << y << "\t" << z << std::endl;
        }
    }
    return EXIT_SUCCESS;
}

This does a lot to remove the clustering as shown in Fig. B-3, but the disadvantage is that we need to know the total number of sample points beforehand to impose the grid. There is another issue with this stratified sampling: to get uniform sampling, the grid cells must be rectangles rather than squares. This results in a different density of points along the 2 dimensions.

### B-3. Spiral Distribution

A good discussion of the general problem of distributing points uniformly over the unit sphere is contained in the paper by Saff and Kuijlaars. They show that a distribution of points on the sphere spiraling from the north pole to south pole provides a good compromise that keeps the spacing between points about the same. Their formulation is implemented in Listing B-3. This is not randomized, and we need to know beforehand the total number of sample points. The pattern it produces is shown in Fig. B-4.

---


Approved for public release; distribution is unlimited.
Listing B-3. spiral.cpp

```cpp
#include <iostream>
#include <cstdlib>
#include <cmath>
using namespace std;

int main( int argc, char* argv[] ) {
    const double TWO_PI = 2. * M_PI;
    int N = 1000;
    if ( argc == 2 ) N = atoi( argv[1] );

    double x, y, z, th, ph = 0.;
    for ( int i = 1; i <= N; i++ ) {
        if ( i == 1 ) {
            z = -1.;
            ph = 0.;
        } else if ( i == N ) {
            z = 1.;
            ph = 0.;
        } else {
            z = -1. + 2. * ( i - 1 ) / double( N - 1 );
            ph += 3.6 / sqrt( double( N ) * ( 1. - z ) * ( 1. + z ) );
        }
        th = acos( z );
        while ( ph > TWO_PI ) ph -= TWO_PI;
        //cout << ph << "\t" << z << endl;
        x = sin( th ) * cos( ph );
        y = sin( th ) * sin( ph );
        cout << x << "\t" << y << "\t" << z << endl;
    }
    return EXIT_SUCCESS;
}
```

We call this the **spiral distribution** since it starts at the north pole and spirals points around the sphere until it reaches the south pole.

### B-4. Maximal Avoidance

The last method we consider is based upon number theory. The code listing is in Listing B-4 and is based upon the code in *Numerical Recipes.*

Listing B-4. avoidance.cpp

```cpp
#include "Random.h"
#include "Vector.h"
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <iomanip>

va::Vector spherical( rng::Random& rng ) { // returns a random unit vector uniformly distributed over the unit sphere
    double x, y, z;
    rng.spherical_avoidance( x, y, z );
    return va::Vector( x, y, z );
}
```

---


Approved for public release; distribution is unlimited.
int main( int argc, char* argv[] ) {
    unsigned int N = 1024;
    if ( argc == 2 ) N = atoi( argv[1] );
    std::cout << std::setprecision(6) << std::fixed;
    std::cout << std::setprecision(6) << std::fixed;
    rng::Random rng;
    for ( unsigned int n = 0; n < N; n++ ) std::cout << spherical( rng ) << std::endl;
    double th, ph, xy[2];
    for ( unsigned int n = 0; n < N; n++ ) {
        //rng.avoidance( xy, 2 );
        rng.spherical_avoidance( th, ph );
        std::cout << ph << "\t" << th << std::endl;
    }
    return EXIT_SUCCESS;
}

This is also deterministic, not random, and one requires to know the number of sample points ahead of time. However, it does a very nice job of distributing the points, as shown in Fig. B-5. The points are computed sequentially, and number theory is used to avoid previous points.
Fig. B-2. Using uniform sampling on the cylinder with $x \in [0, 2\pi]$ and $z \in [-1, 1]$ (top plot) and Archimedes’ theorem to map onto the sphere with $\phi = x$ and $\theta = \cos^{-1} z$. 
Fig. B-3. Using *stratified sampling* on the cylinder with $x \in [0, 2\pi]$ and $z \in [-1, 1]$ (top plot) and Archimedes’ theorem to map onto the sphere with $\phi = x$ and $\theta = \cos^{-1} z$. 
Fig. B-4. Using the spiral distribution on the cylinder with $x \in [0, 2\pi]$ and $z \in [-1, 1]$ (top plot) and Archimedes’ theorem to map onto the sphere with $\phi = x$ and $\theta = \cos^{-1} z$.
Fig. B-5. Using maximal avoidance on the cylinder with $x \in [0, 2\pi]$ and $z \in [-1, 1]$ (top plot) and Archimedes’ theorem to map onto the sphere with $\phi = x$ and $\theta = \cos^{-1} z$. 

Appendix C. Some Properties of the Lognormal Distribution
The probability density function (PDF) is
\[
f(x | \mu, \sigma^2) = \frac{1}{\sqrt{2\pi \sigma x}} \exp \left[ -\frac{(\ln x - \mu)^2}{2\sigma^2} \right]
\] (C-1)
and the cumulative distribution function (CDF) is
\[
F(x | \mu, \sigma^2) = \frac{1}{2} \left( 1 + \text{erf} \left[ \frac{\ln x - \mu}{\sqrt{2} \sigma} \right] \right),
\] (C-2)
where \( \mu \) is the location parameter and \( \sigma \) is the scale parameter. Expressions for the usual metrics are listed in Table C-1.

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometric Mean</td>
<td>( x_g = e^\mu )</td>
</tr>
<tr>
<td>Geometric Standard Deviation</td>
<td>( \sigma_g = e^\sigma )</td>
</tr>
<tr>
<td>Median</td>
<td>( x_{50} = e^\mu )</td>
</tr>
<tr>
<td>Mean</td>
<td>( \bar{x} = e^{\mu + \sigma^2 / 2} )</td>
</tr>
<tr>
<td>Mode</td>
<td>( \hat{x} = e^{\mu - \sigma^2} )</td>
</tr>
</tbody>
</table>

- The values \( ax \) and \( \hat{x}/a \) are equally likely for any value \( a \neq 0 \). That is, \( f(ax) = f(\hat{x}/a) \).
- 68% of the distribution is contained in the interval \([x_g^{-1} \sigma_g, x_g \sigma_g]\).
- 95% of the distribution is contained in the interval \([x_g^{-2} \sigma_g, x_g \sigma_g^2]\).

The \( n \)-th moment about the origin is
\[
\lambda_n \equiv \int_0^\infty x^n f(x) dx = \int_0^\infty e^{n \ln x} f(x) dx
\]
\[
= \frac{1}{\sqrt{2\pi \sigma}} \int_0^\infty \exp \left[ -\frac{(\ln x - \mu)^2}{2\sigma^2} + n \ln x \right] d \ln x
\]
\[
= \frac{1}{\sqrt{2\pi \sigma}} \int_0^\infty \exp \left[ -\frac{(\ln x - \mu - n\sigma^2)^2}{2\sigma^2} + n\mu + \frac{1}{2}n^2\sigma^2 \right] d \ln x
\]
\[
= \exp \left( n\mu + \frac{1}{2}n^2\sigma^2 \right)
\]
\[
= x_g^n \exp \left( \frac{1}{2}n^2\sigma^2 \right)
\] (C-3)
The $n$-th moment distribution function of $f(x \mid \mu, \sigma^2)$ is defined by

$$f_n(x \mid \mu, \sigma^2) \equiv \frac{1}{\lambda_n} x^n f(x \mid \mu, \sigma^2). \quad (C-4)$$

Using Eqs. C-3 and C-1,

$$f_n(x \mid \mu, \sigma^2) = \exp \left( -n\mu - \frac{1}{2} \frac{n^2\sigma^2}{\sigma^2} \right) \frac{1}{\sqrt{2\pi\sigma x}} \exp \left[ -\frac{(\ln x - \mu)^2}{2\sigma^2} + n\ln x \right]$$

$$= \exp \left( -n\mu - \frac{1}{2} \frac{n^2\sigma^2}{\sigma^2} \right) \frac{1}{\sqrt{2\pi\sigma x}} \exp \left[ -\frac{(\ln x - \mu - n\sigma^2)^2}{2\sigma^2} + n\mu + \frac{1}{2} n^2\sigma^2 \right]$$

$$= \frac{1}{\sqrt{2\pi\sigma x}} \exp \left[ -\frac{(\ln x - \mu - n\sigma^2)^2}{2\sigma^2} \right]. \quad (C-5)$$

And thus we have derived the **Fundamental Theorem of the Moment Distribution**:

The $n$-th moment distribution of a lognormal distribution with parameters $\mu$ and $\sigma^2$ is also a lognormal distribution with parameters $\mu + n\sigma^2$ and $\sigma^2$, respectively,

$$f_n(x \mid \mu, \sigma^2) = f(x \mid \mu + n\sigma^2, \sigma^2). \quad (C-6)$$

We can also show that the product and quotient of lognormal distributions are also lognormal. Using the notation of Aitchison and Brown\(^1\), if $X_1 \sim \Lambda(\mu_1, \sigma_1^2)$ and $X_2 \sim \Lambda(\mu_2, \sigma_2^2)$, then the product $X_1X_2$ is also lognormal with

$$X_1X_2 \sim \Lambda(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2), \quad (C-7)$$

and the quotient $X_1/X_2$ is also lognormal with

$$X_1/X_2 \sim \Lambda(\mu_1 - \mu_2, \sigma_1^2 + \sigma_2^2). \quad (C-8)$$

We can derive these results as follows.

---


Approved for public release; distribution is unlimited.
C-1. Product of Lognormals

Let $X_1 \sim \Lambda(\mu_1, \sigma_1^2)$ and $X_2 \sim \Lambda(\mu_2, \sigma_2^2)$ be lognormal distributions. Then the cumulative distribution of their product is

$$F_{X_1X_2}(u) = \int\int_{X_1X_2 \leq u} f(x_1, x_2) \, dx_1 \, dx_2 = \int_0^\infty \left( \int_{0}^{u/x_1} f(x_1, x_2) \, dx_2 \right) \, dx_1. \quad \text{(C-9)}$$

The density is obtained by differentiating with respect to $u$, so that

$$f_{X_1X_2}(u) = \int_0^\infty f(x_1, u/x_1) \frac{1}{x_1} \, dx_1$$

$$= \int_0^\infty f(x) f \left( \frac{u}{x} \right) \, d\ln x$$

$$= \int_0^\infty \frac{1}{\sqrt{2\pi} \sigma_1 x} \exp \left[ -\frac{(\ln x - \mu_1)^2}{2\sigma_1^2} \right] \frac{1}{\sqrt{2\pi} \sigma_2(u/x)} \exp \left[ -\frac{(\ln(u/x) - \mu_2)^2}{2\sigma_2^2} \right] \, d\ln x$$

$$= \frac{1}{2\pi\sigma_1\sigma_2 u} \int_0^\infty \exp \left[ -\frac{(\ln x - \mu_1)^2}{2\sigma_1^2} - \frac{(\ln u - \ln x - \mu_2)^2}{2\sigma_2^2} \right] \, d\ln x$$

$$= \frac{1}{2\pi\sigma_1\sigma_2 u} \int_0^\infty \exp \left[ -\left( \frac{(\ln x - \mu_1)^2}{2\sigma_1^2} + \frac{(\ln x + \mu_2 - \ln u)^2}{2\sigma_2^2} \right) \right] \, d\ln x$$

$$= \frac{1}{2\pi\sigma_1\sigma_2 u} \int_0^\infty \exp \left[ -\frac{(x - \mu_1)^2}{2\sigma_1^2} + \frac{(x + \mu_2 - \ln u)^2}{2\sigma_2^2} \right] \, dx$$

$$= \frac{1}{2\pi\sigma_1\sigma_2 u} \int_{-\infty}^\infty \exp \left[ -g(x) \right] \, dx, \quad \text{(C-10)}$$

where

$$g(x) = \frac{(x - \mu_1)^2}{2\sigma_1^2} + \frac{(x + \mu_2 - \ln u)^2}{2\sigma_2^2} \equiv a(x - \mu_1)^2 + b(x + \mu_2 - c)^2, \quad \text{(C-11)}$$

and

$$a \equiv \frac{1}{2\sigma_1^2}, \quad b \equiv \frac{1}{2\sigma_2^2}, \quad c \equiv \ln u. \quad \text{(C-12)}$$

Completing the square in $x$ gives, after some messy algebra,

$$g(x) = (a + b) \left( x - \frac{a\mu_1 - b\mu_2 + bc}{a + b} \right)^2 + \frac{ab}{a + b} \left[ c - (\mu_1 + \mu_2) \right]^2 \quad \text{(C-13)}$$
or, in terms of $\sigma_1$, $\sigma_2$, and $\ln u$,

$$g(x) = \frac{\sigma_1^2 + \sigma_2^2}{2\sigma_1^2\sigma_2^2} \left( x - \frac{\sigma_2^2\mu_1 - \sigma_1^2\mu_2 + \sigma_1^2\ln u}{\sigma_1^2 + \sigma_2^2} \right)^2 + \frac{(\ln u - \mu_1 - \mu_2)^2}{2(\sigma_1^2 + \sigma_2^2)}. \quad (C-14)$$

Therefore, returning to Eq. C-10,

$$f_{X_1X_2}(u) = \exp \left[ -\frac{(\ln u - \mu_1 - \mu_2)^2}{2(\sigma_1^2 + \sigma_2^2)} \right] \times$$

$$\frac{1}{2\pi\sigma_1\sigma_2 u} \int_{-\infty}^{\infty} \exp \left[ -\frac{\sigma_1^2 + \sigma_2^2}{2\sigma_1^2\sigma_2^2} \left( x - \frac{\sigma_2^2\mu_1 - \sigma_1^2\mu_2 + \sigma_1^2\ln u}{\sigma_1^2 + \sigma_2^2} \right)^2 \right] dx \quad (C-15)$$

The integral is now easily evaluated with the substitution

$$\xi = \sqrt{\frac{\sigma_1^2 + \sigma_2^2}{2\sigma_1\sigma_2}} \left( x - \frac{\sigma_2^2\mu_1 - \sigma_1^2\mu_2 + \sigma_1^2\ln u}{\sigma_1^2 + \sigma_2^2} \right) \quad \text{and} \quad d\xi = \frac{\sqrt{\sigma_1^2 + \sigma_2^2}}{\sqrt{2\pi}\sigma_1\sigma_2} dx \quad (C-16)$$

and we get

$$f_{X_1X_2}(u) = \exp \left[ -\frac{(\ln u - \mu_1 - \mu_2)^2}{2(\sigma_1^2 + \sigma_2^2)} \right] \frac{1}{\sqrt{2\pi\sigma_1^2 + \sigma_2^2} u} \int_{-\infty}^{\infty} e^{-\xi^2} d\xi$$

$$= \frac{1}{\sqrt{2\pi(\sigma_1^2 + \sigma_2^2)}} \exp \left[ -\frac{(\ln u - \mu_1 - \mu_2)^2}{2(\sigma_1^2 + \sigma_2^2)} \right]$$

$$= \Lambda(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2). \quad \text{(C-17)}$$

Thus,

$$X_1X_2 \sim \Lambda(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2) \quad \text{(C-18)}$$

as was to be shown.

### C-2. Quotient of Lognormals

Let $X_1 \sim \Lambda(\mu_1, \sigma_1^2)$ and $X_2 \sim \Lambda(\mu_2, \sigma_2^2)$ be lognormal distributions. Then the cumulative distribution of their quotient is

$$F_{X_1/X_2}(u) = \int_{X_1/X_2 \leq u} f(x_1, x_2) \, dx_1 \, dx_2 = \int_{0}^{\infty} \left( \int_{0}^{u\sqrt{x_2}} f(x_1, x_2) \, dx_1 \right) \, dx_2. \quad (C-19)$$
The density is obtained by differentiating with respect to \( u \), so that

\[
f_{X_1/X_2}(u) = \int_0^\infty f(u x_2, x_2) x_2 \, dx_2
\]

\[
= \int_0^\infty f(u x) f(x) \, dx
\]

\[
= \int_0^\infty \frac{1}{\sqrt{2\pi \sigma_1 x}} \exp \left[ -\frac{(\ln(u x) - \mu_1)^2}{2\sigma_1^2} \right] \frac{1}{\sqrt{2\pi \sigma_2 x}} \exp \left[ -\frac{(\ln x - \mu_2)^2}{2\sigma_2^2} \right] x \, dx
\]

\[
= \frac{1}{2\pi \sigma_1 \sigma_2 u} \int_0^\infty \exp \left[ -\frac{(\ln u + \ln x - \mu_1)^2}{2\sigma_1^2} - \frac{(\ln x - \mu_2)^2}{2\sigma_2^2} \right] d\ln x
\]

\[
= \frac{1}{2\pi \sigma_1 \sigma_2 u} \int_0^\infty \exp \left[ -\left( \frac{(\ln x - \mu_1 + \ln u)^2}{2\sigma_1^2} + \frac{(\ln x - \mu_2)^2}{2\sigma_2^2} \right) \right] d\ln x
\]

\[
= \frac{1}{2\pi \sigma_1 \sigma_2 u} \int_{-\infty}^\infty \exp \left[ -h(x) \right] \, dx,
\]

where

\[
h(x) \equiv \frac{(x - \mu_1 + \ln u)^2}{2\sigma_1^2} + \frac{(x - \mu_2)^2}{2\sigma_2^2} \equiv a(x - \mu_1 + c)^2 + b(x - \mu_2)^2,
\]

\[
a \equiv \frac{1}{2\sigma_1^2}, \quad b \equiv \frac{1}{2\sigma_2^2}, \quad c \equiv \ln u.
\]

Completing the square in \( x \) gives

\[
h(x) = (a + b) \left( x - \frac{a\mu_1 + b\mu_2 - ac}{a + b} \right)^2 + \frac{ab}{a + b} [c - (\mu_1 - \mu_2)]^2
\]

or, in terms of \( \sigma_1, \sigma_2, \) and \( \ln u, \)

\[
h(x) = \frac{\sigma_1^2 + \sigma_2^2}{2\sigma_1^2 \sigma_2^2} \left( x - \frac{\sigma_1^2 \mu_1 + \sigma_2^2 \mu_2 - \sigma_1^2 \ln u}{\sigma_1^2 + \sigma_2^2} \right)^2 + \frac{[\ln u - (\mu_1 - \mu_2)]^2}{2(\sigma_1^2 + \sigma_2^2)}.
\]
Therefore, returning to Eq. C-20,

$$f_{X_2/X_1}(u) = \exp \left[ -\frac{[\ln u - (\mu_1 - \mu_2)]^2}{2(\sigma_1^2 + \sigma_2^2)} \right] \times$$

$$\frac{1}{2\pi \sigma_1 \sigma_2 u} \int_{-\infty}^{\infty} \exp \left[ -\frac{\sigma_1^2 + \sigma_2^2}{2\sigma_1^2 \sigma_2^2} \left( x - \frac{\sigma_2^2 \mu_1 + \sigma_1^2 \mu_2 - \sigma_2^2 \ln u}{\sigma_1^2 + \sigma_2^2} \right)^2 \right] dx$$  \hspace{1cm} (C-25)

The integral is now easily evaluated with the substitution

$$\xi = \frac{\sqrt{\sigma_1^2 + \sigma_2^2}}{\sqrt{2} \sigma_1 \sigma_2} \left( x - \frac{\sigma_2^2 \mu_1 + \sigma_1^2 \mu_2 - \sigma_2^2 \ln u}{\sigma_1^2 + \sigma_2^2} \right)$$  \hspace{1cm} (C-26)

and we get

$$f_{X_2/X_1}(u) = \exp \left[ -\frac{[\ln u - (\mu_1 - \mu_2)]^2}{2(\sigma_1^2 + \sigma_2^2)} \right] \frac{1}{\sqrt{2\pi(\sigma_1^2 + \sigma_2^2)u}} \int_{-\infty}^{\infty} e^{-\xi^2} d\xi$$

$$= \frac{1}{\sqrt{2\pi(\sigma_1^2 + \sigma_2^2)u}} \exp \left[ -\frac{[\ln u - (\mu_1 - \mu_2)]^2}{2(\sigma_1^2 + \sigma_2^2)} \right]$$

$$= \Lambda(\mu_1 - \mu_2, \sigma_1^2 + \sigma_2^2).$$  \hspace{1cm} (C-27)

Thus,

$$X_1/X_2 \sim \Lambda(\mu_1 - \mu_2, \sigma_1^2 + \sigma_2^2)$$  \hspace{1cm} (C-28)

as was to be shown.

In general, if $X_i, i = 1, 2, ..., n$ are independent random variables with $X_i \sim \Lambda(\mu_i, \sigma_i^2)$ and $a$ and $p_i$ are constants, then

$$\prod_{i=1}^{n} a X_i^{p_i} \sim \Lambda(\ln a + \sum_{i=1}^{n} p_i \mu_i, \sum_{i=1}^{n} p_i^2 \sigma_i^2).$$  \hspace{1cm} (C-29)

Notice that the variance can only increase, never decrease. The next section gives an application of this result.

---

C-3. Mass per Unit Area Distribution

Suppose that the dimensionless shape factor $\gamma$ is lognormally distributed with $\gamma \sim \Lambda(\mu_1, \sigma_1^2)$ and that fragment mass is also lognormally distributed with $m \sim \Lambda(\mu_2, \sigma_2^2)$.

Then, since the fragment presented area, $A_p$, equals $\gamma(m/\rho)^{2/3}$, where $\rho$ is the material density, it follows from Eq. C-29 that mass per unit area is also lognormally distributed:

$$\frac{m}{A_p} = \frac{1}{\gamma} \rho^{2/3} m^{1/3} \sim \Lambda(\mu, \sigma^2),$$

(C-30)

where

$$\mu = \ln \rho^{2/3} - \mu_1 + \frac{\mu_2}{3} \quad \text{and} \quad \sigma^2 = \sigma_1^2 + \left(\frac{\sigma_2}{3}\right)^2.$$  

(C-31)

To illustrate this, let us return to the artillery fragments described in Section 3.2.

A lognormal distribution fits the fragment masses (in grams) with $\mu_2 = 1.690$ and $\sigma_2 = 1.323$. We also found that the shape factor distribution was lognormal with $\mu_1 = 0.597$ and $\sigma_1 = 0.341$. From Eqs. C-29, C-30, and C-31, it then follows that the mass per unit area distribution (in g/cm$^2$) should also be lognormal with

$$\mu = \ln 7.83^{2/3} - \mu_1 + \frac{\mu_2}{3} = 1.338 \quad \text{and} \quad \sigma = \sqrt{\sigma_1^2 + (\sigma_2/3)^2} = 0.557.$$  

(C-32)

The program in Listing C-1 generates the shape factor and mass independently and outputs the mass per unit area.

Listing C-1. mu.cpp

```cpp
// mu.cpp: generate shape factor and mass independently
// and output mass per unit area to compare to the theoretical distribution

#include "Random.h"
#include <iostream>
#include <cstdlib>
#include <cmath>

int main( void ) {
    const int N = 10000;
    const double MU1 = 0.597, SIGMA1 = 0.341, RHO = 7.83, C = pow( RHO, 2./3. );
    const double MU2 = 1.690, SIGMA2 = 1.323;
    rng::Random rng;
    double sf, m, mu;
    for ( int i = 0; i < N; i++ ) {
        sf = rng.lognormal( 0., MU1, SIGMA1 ); // shape factor (dimensionless)
        m = rng.lognormal( 0., MU2, SIGMA2 ); // mass (grams)
        mu = C * pow( m, 1./3. ) / sf; // mass per unit area (g/cm^2)
        std::cout << mu << std::endl;
    }
    return EXIT_SUCCESS;
}
```

These values are compared to the theoretical distribution in Fig. C-1.
Fig. C-1. The program in Listing C-1 was used to generate independent samples of shape factor and mass and output mass per unit area in order to compare to the theoretical distribution (black curve).

This shows that the resulting distribution is indeed lognormal with $\mu$ and $\sigma$ as given by Eq. C-31. Thus, the mass per unit area samples could be generated from a single lognormal distribution.
INTENTIONALLY LEFT BLANK.
List of Symbols, Abbreviations, and Acronyms

TERMS:

3D: 3 dimensional

FATEPEN: Fast Air Target Encounter Penetration

RCC: right-circular cylinder

RPP: rectangular parallelepiped (also known as cuboid)

STL: stereolithography

MATHEMATICAL SYMBOLS:

\( f \), PDF: probability density function

\( F \), CDF: cumulative distribution function

\( \gamma \): shape factor

\( \rho \): material density

\( \Lambda(\mu, \sigma^2) \): lognormal distribution with mean \( \mu \) and variance \( \sigma^2 \)