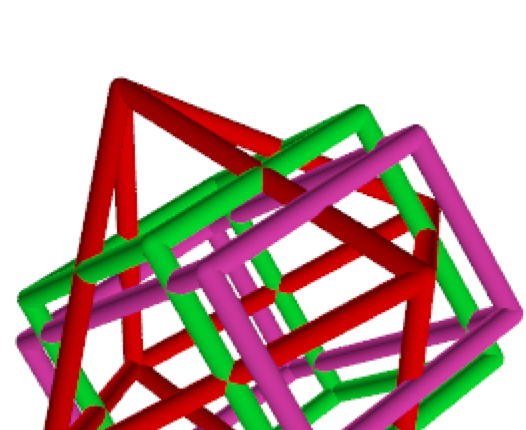
The Rootful Cuboid

by Kirby Urner and David Koski



A round of applause for this David Koski discovery, perhaps known to nomads, but missing from contemporary popularizations:

The cuboid of height, width, depth $\sqrt{1}\ \sqrt{2}\ \sqrt{4}$, and face diagonals $\sqrt{3}\ \sqrt{5}\ \sqrt{6}$ and body diagonal $\sqrt{7}$.

That's a parellelopiped with an inscribed tetrahedron of only face diagonals, and four right tetrahedrons each involving a corner and three right angles.

The inscribed tetrahedron will have a volume 1/3rd that of the paralleopiped, with the corner tetrahedrons each having 1/6th that total volume. This is a generic truth for parallelopipeds.

Lets label the two ends of the cuboic (brick) ABCD counter-clockwise with EFGH at the other end, and matching A with E, B with F and so on, such that the six faces of the cuboid are:

- ABCD, EFGH (parallel front/back)
- AEFB, DHGF (parallel sides)
- ADHE, BCGF (parallel top/bottom)

The convention for naming edges is to write the two corner points alphabetically, to keep them unique. We write AB, but not BA, as these describe the same edge and we want a unique canonical identifier for each one.

In [1]:
```python
from math import sqrt as rt2

# cuboid
AB = EF = CD = GH = rt2(1)/2
AD = BC = FG = EH = rt2(2)/2
AC = BD = EG = FH = rt2(3)/2
AE = BF = CG = DH = rt2(4)/2
AF = BE = CH = DG = rt2(5)/2
AH = BG = CF = DE = rt2(6)/2
AG = BH = CE = DF = rt2(7)/2
```

Enter our frame of reference made of unit radius spheres packed in the CCP interconnected by IVM vectors. Four such spheres packed together define a tetrahedron of edges one sphere diameter (D), or twice the radius (R).

If we wish our home base tetrahedron of edges one, to have volume one, then set we set our unit length to D (= 2 R).

The reason for scaling the entire cuboid describe above by 1/2 is to imagine we're going from R as unit length to D as unit length. Every length gets halved, and the result is an inscribed tetradron of unit tetravolume, the same as our reference tetrahedron's, even though this one is irregular.

The transformation from XYZ volume to IVM volume is baked into the constant S3, such that we sometimes say "XYZ volume times S3 gives IVM volume" with 1/S3 used to go the other direction. S3 = $\sqrt{9/8}$ or $(3\sqrt{2})/4$.

In [2]:
```python
import tetravolume as tv
```

The volume method used below takes the six lengths of a tetrahedron as input.

Order matters.

Pick a corner, any corner, of the tetrahedron, and start with the three lengths stemming from that corner, to the corners of the opposite face.

Then provide the segments around the opposite face, in the same order mentioned.

For example, given tetrahedron ABCD, lets pick A and go AB, AC, AD.

The opposite face is BCD and the lengths should be in the order mentioned the first time i.e. B first, so B-to-C, C-to-D and D-back-to-B i.e. BC, CD, BD.

Remember how edges have a unique representation and we write BD versus DB, because B is before D alphabetically.

In [3]:
```python
unit_vol = tv.Tetrahedron(1, 1, 1, 1, 1, 1) # reference tetrahedron
```

In [4]:
```python
unit_vol.ivm_volume()
```

Out[4]: 1.0

In [5]:
```python
ACHF = tv.Tetrahedron(AC, AH, AF, CH, FH, CF) # inscribed tetrahedron, all face diago
ACHF.ivm_volume()
```

Out[5]: 1.0

In [6]:
```python
S3 = rt2(9/8)
```

In [7]:
```python
ACHF.xyz_volume()
```

Out[7]: 0.9428090415820635

In [8]:
```python
AEDB = tv.Tetrahedron(AB, AE, AD, BE, DE, BD) # right angles at A
AEDB.ivm_volume()
```

Out[8]: 0.5

In [9]:
```python
ABCE = tv.Tetrahedron(AB, AC, AE, BC, CE, BE) # right angles at B
ABCE.ivm_volume()
```

Out[9]: 0.49999999999999956

In [10]:
```python
ACDH = tv.Tetrahedron(AD, CD, DH, AC, CH, AH)  # right angles at H
ACDH.ivm_volume()
```

Out[10]: 0.5

In [11]:
```python
FGHC = tv.Tetrahedron(FG, GH, CG, FH, CH, CF) # right angles at G
FGHC.ivm_volume()
```

Out[11]: 0.5

In [12]:
```python
ABCF = tv.Tetrahedron(AB, AC, AF, BC, CF, BF) # includes body diagonal
ABCF.ivm_volume()
```

Out[12]: 0.5

Generating the POV-Ray Graphic

The `flextegrity.py` and `qrays.py` files provide the necessary dependencies for generating the requiste scene description language. Here's the code:

In [13]:
```python
# %load cuboid.py
#!/usr/bin/env python3
"""
Created on Sun Sep 19 10:46:45 2021

@author: Kirby Urner
"""

from flextegrity import pov_header, Cuboid, \
    draw_poly, Octahedron, Cube

with open("cuboid.pov", "w") as target:
    target.write(pov_header)
    cuboid = Cuboid() * 0.5
    octa = Octahedron()
    cube = Cube()
    draw_poly(cuboid, target)
    draw_poly(octa, target)
    draw_poly(cube, target)
```

But what does Cuboid, the class, look like internally. We can check it out here:

```python
class Cuboid (Polyhedron):
        """
        Cuboid with height, width, depth = sqrt(1), sqrt(2), sqrt(4)
        """

        def __init__(self):
            # POV-Ray
            self.edge_color = "rgb <255/255, 20/255, 147/255>"
            self.edge_radius= 0.03
            self.vert_color = "rgb <255/255, 20/255, 147/255>"
            self.vert_radius= 0.03
            self.face_color = "rgb <0, 0, 0>"

            verts = {}
            verts['A'] = Vector(( 1,  0.5,  math.sqrt(2)/2))
            verts['B'] = Vector(( 1, -0.5,  math.sqrt(2)/2))
            verts['C'] = Vector(( 1, -0.5, -math.sqrt(2)/2))
            verts['D'] = Vector(( 1,  0.5, -math.sqrt(2)/2))
            verts['E'] = Vector((-1,  0.5,  math.sqrt(2)/2))
            verts['F'] = Vector((-1, -0.5,  math.sqrt(2)/2))
            verts['G'] = Vector((-1, -0.5, -math.sqrt(2)/2))
            verts['H'] = Vector((-1,  0.5, -math.sqrt(2)/2))

            self.name = "Cuboid"
            self.volume = 8  # per Concentric Hierarchy
            self.center = ORIGIN

            # 8 vertices
            self.vertexes = verts

            # 6 faces
            self.faces = (('A','B','C','D'),('E','F','G','H'),
                          ('A','E','F','B'),('D','H','G','C'),
                          ('A','E','H','D'),('B','F','G','C'))

        self.edges = self._distill()
```

This time we're using ordinary XYZ Vectors, not Quadrays or Qvectors. Per the cuboid program, the final Polyhedron is scaled down by 1/2 to have it fit more neatly into our canonical hierarchy of polyhedrons.

After the target file is written, a .pov file, we may then render it using the povray raytracing engine.

In [14]:
```
! /usr/local/bin/povray +A +H768 +W1024 cuboid.pov
```

```
Persistence of Vision(tm) Ray Tracer Version 3.7.0.8.unofficial (clang++ 4.2.1 @
 x86_64-apple-darwin14.5.0)
This is an unofficial version compiled by:
 homebrew
 The POV-Ray Team is not responsible for supporting this version.

POV-Ray is based on DKBTrace 2.12 by David K. Buck & Aaron A. Collins
Copyright 1991-2013 Persistence of Vision Raytracer Pty. Ltd.

Primary POV-Ray 3.7 Architects/Developers: (Alphabetically)
  Chris Cason         Thorsten Froehlich  Christoph Lipka

With Assistance From: (Alphabetically)
  Nicolas Calimet     Jerome Grimbert     James Holsenback    Christoph Hormann
  Nathan Kopp         Juha Nieminen

Past Contributors: (Alphabetically)
  Steve Anger         Eric Barish         Dieter Bayer        David K. Buck
  Nicolas Calimet     Chris Cason         Aaron A. Collins    Chris Dailey
  Steve Demlow        Andreas Dilger      Alexander Enzmann   Dan Farmer
  Thorsten Froehlich  Mark Gordon         James Holsenback    Christoph Hormann
  Mike Hough          Chris Huff          Kari Kivisalo       Nathan Kopp
  Lutz Kretzschmar    Christoph Lipka     Jochen Lippert      Pascal Massimino
  Jim McElhiney       Douglas Muir        Juha Nieminen       Ron Parker
  Bill Pulver         Eduard Schwan       Wlodzimierz Skiba   Robert Skinner
  Yvo Smellenbergh    Zsolt Szalavari     Scott Taylor        Massimo Valentini
  Timothy Wegner      Drew Wells          Chris Young

Other contributors are listed in the documentation.

Support libraries used by POV-Ray:
  ZLib 1.2.5, Copyright 1995-2012 Jean-loup Gailly and Mark Adler
  LibPNG 1.6.37, Copyright 1998-2012 Glenn Randers-Pehrson
  LibJPEG 90, Copyright 1991-2013 Thomas G. Lane, Guido Vollbeding
  LibTIFF 4.3.0, Copyright 1988-1997 Sam Leffler, 1991-1997 SGI
  Boost 1.70, http://www.boost.org/

Parser Options
  Input file: cuboid.pov
  Remove bounds........On
  Split unions.........Off
  Library paths:
    /usr/local/Cellar/povray/3.7.0.8_1/share/povray-3.7
    /usr/local/Cellar/povray/3.7.0.8_1/share/povray-3.7/ini
    /usr/local/Cellar/povray/3.7.0.8_1/share/povray-3.7/include
  Clock value:    0.000  (Animation off)
Image Output Options
  Image resolution.....1024 by 768 (rows 1 to 768, columns 1 to 1024).
  Output file..........cuboid.png, 24 bpp PNG
  Dithering............Off
  Graphic display......On  (gamma: sRGB)
  Mosaic preview.......Off
  Continued trace......Off
Information Output Options
  All Streams to console..........On
  Debug Stream to console.........On
  Fatal Stream to console.........On
  Render Stream to console........On
  Statistics Stream to console....On
  Warning Stream to console.......On
==== [Parsing...] ==========================================================
Parse Warning: This scene had other declarations preceding the first #version
 directive. Please be aware that as of POV-Ray 3.7, unless already specified via
 an INI option, a #version is expected as the first declaration in a scene file.
 If this is not done, POV-Ray may apply compatibility settings to some features
 that are intended to make pre-3.7 scenes render as designed. You are strongly
 encouraged to add a #version statement to the scene to make your intent clear.
 Future versions of POV-Ray may make the presence of a #version mandatory.
----------------------------------------------------------------------------
Parser Statistics
----------------------------------------------------------------------------
Finite Objects:           58
Infinite Objects:          0
Light Sources:             2
Total:                    60
----------------------------------------------------------------------------
Parser Time
  Parse Time:       0 hours  0 minutes  0 seconds (0.031 seconds)
              using 1 thread(s) with 0.-01 CPU-seconds total
  Bounding Time:    0 hours  0 minutes  0 seconds (0.000 seconds)
              using 1 thread(s) with 0.-01 CPU-seconds total
----------------------------------------------------------------------------
Render Options
  Quality:  9
  Bounding boxes.......On   Bounding threshold: 3
  Antialiasing.........On  (Method 1, Threshold 0.300, Depth 3, Jitter 1.00,
 Gamma 2.50)
==== [Rendering...] ========================================================
Rendered 786432 of 786432 pixels (100%)
----------------------------------------------------------------------------
Render Statistics
Image Resolution 1024 x 768
----------------------------------------------------------------------------
Pixels:           835584   Samples:           94086   Smpls/Pxl: 0.11
Rays:             929670   Saved:                 0   Max Level: 1/5
----------------------------------------------------------------------------
Ray->Shape Intersection          Tests       Succeeded  Percentage
----------------------------------------------------------------------------
Cone/Cylinder                   372085          109666     29.47
Sphere                            8128            3112     38.29
Bounding Box                   9922847         2508100     25.28
----------------------------------------------------------------------------
Shadow Ray Tests:            102260   Succeeded:                     0
----------------------------------------------------------------------------
----------------------------------------------------------------------------
Render Time:
  Photon Time:      No photons
  Radiosity Time:   No radiosity
  Trace Time:       0 hours  0 minutes  2 seconds (2.468 seconds)
              using 2 thread(s) with 0.-02 CPU-seconds total
POV-Ray finished
```