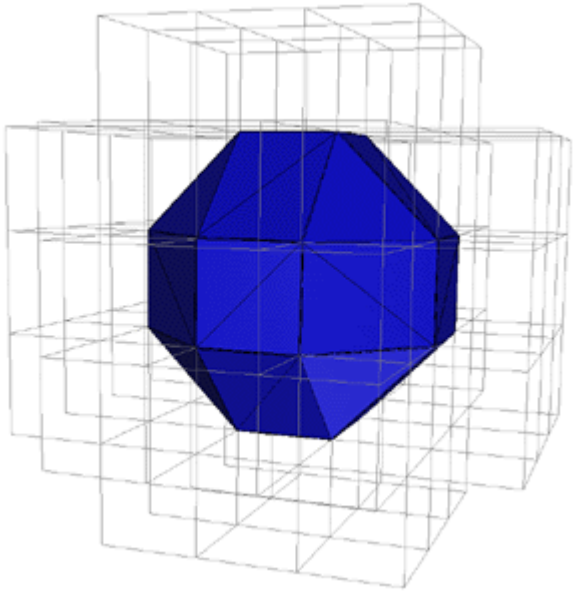
Marching Cubes

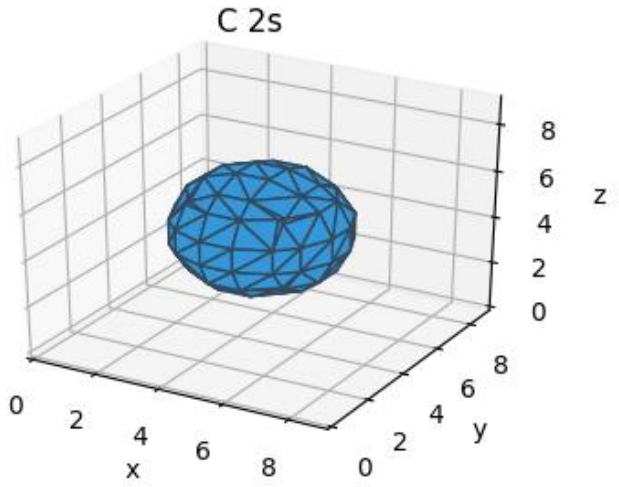
Yubo “Paul” Yang, Algorithm Group, 2017/09/12

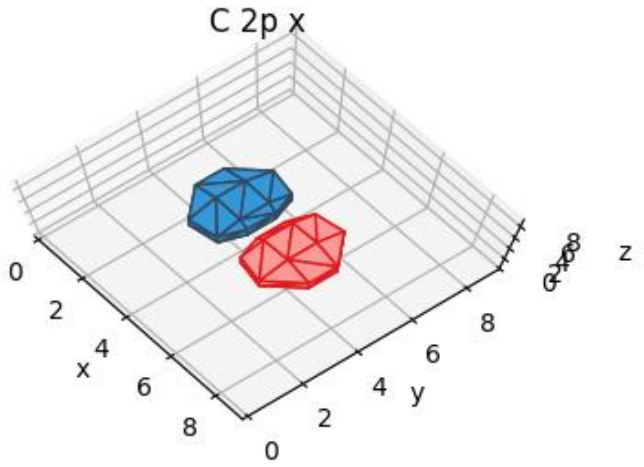


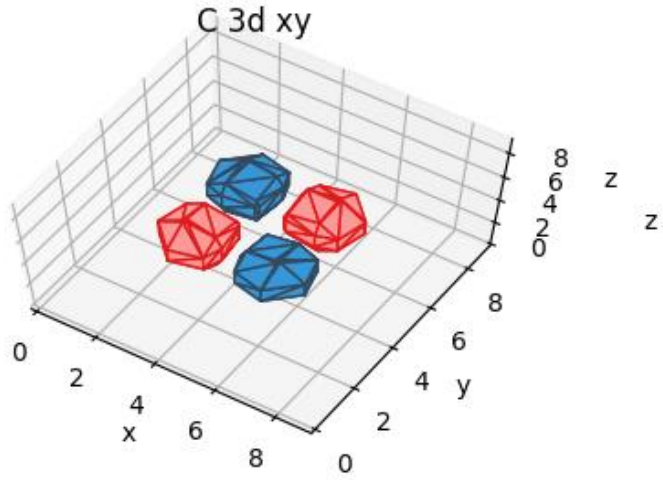
Ref: Paul Bourke, “Polygonising a scalar field,” http://paulbourke.net/geometry/polygonise/

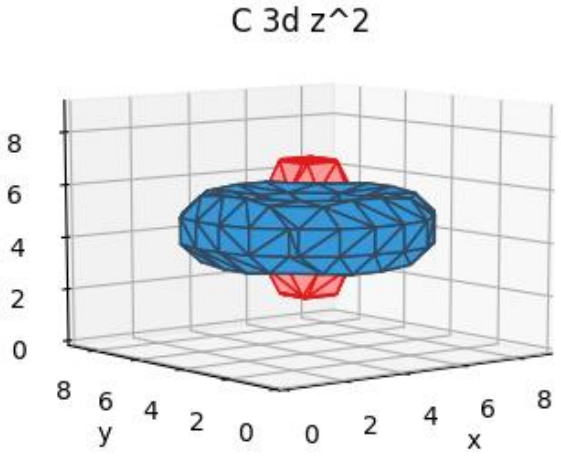
Rawkstar, “Game Art, Necromancer General,” ZBrushCentral

Showcase I: s,p,d basis function

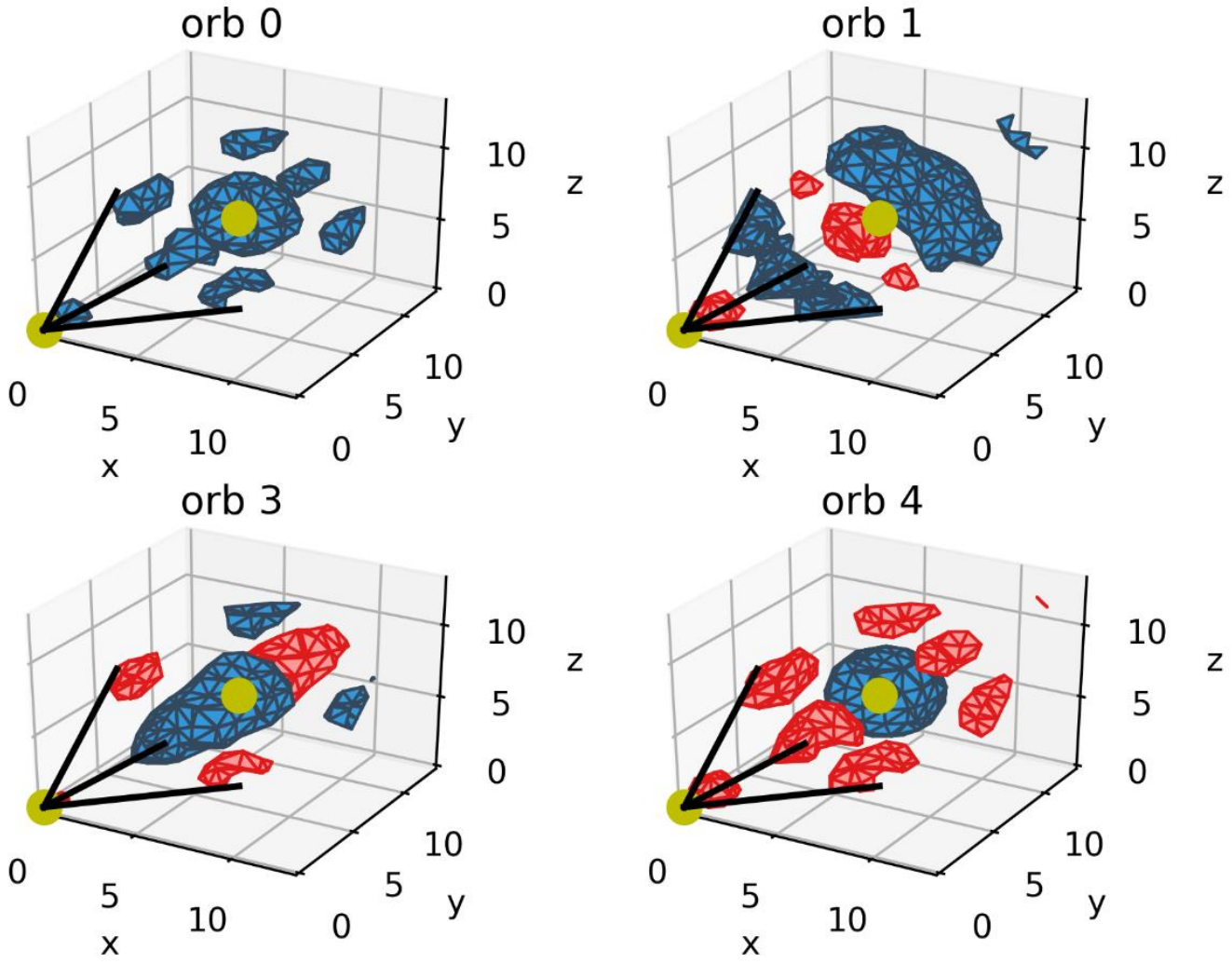








Showcase II: carbon diamond Hatree-Fock orbitals



“The” Problem: Isosurface Extraction

Find N-1 representation of the zero-crossings of an N dimensional scalar field $f(\vec{x}) = 0$.

To simplify discussion:

1. restrict to 3 dimensions

2. assume scalar field is sampled on a regular grid

Ref: Paul Bourke, “Polygonising a scalar field,” http://paulbourke.net/geometry/polygonise/

“The” Problem: Polygonising a Scalar Field

Form a facet approximation for an isosurface of a scalar field sampled on a rectangular 3D grid.

3x3x3 samples of e^{-r^2}

```
3D volumetric data:
[[[ 1.10363832  1.73575888  1.10363832]
  [ 1.73575888  2.36787944  1.73575888]
  [ 1.10363832  1.73575888  1.10363832]]

 [[ 1.73575888  2.36787944  1.73575888]
  [ 2.36787944  3.          2.36787944]
  [ 1.73575888  2.36787944  1.73575888]]

 [[ 1.10363832  1.73575888  1.10363832]
  [ 1.73575888  2.36787944  1.73575888]
  [ 1.10363832  1.73575888  1.10363832]]]
isovalue: 2.620728
```

```
vert[faces]
[[[ 1.          0.40000004  1.        ]
  [ 1.          1.          0.40000004]
  [ 0.40000004  1.          1.        ]]

 [[ 0.40000004  1.          1.        ]
  [ 1.          1.          1.5999999 ]
  [ 1.          0.40000004  1.        ]]

 [[ 1.          1.          0.40000004]
  [ 1.          1.5999999   1.        ]
  [ 0.40000004  1.          1.        ]]

 [[ 0.40000004  1.          1.        ]
  [ 1.          1.5999999   1.        ]
  [ 1.          1.          1.5999999 ]]

 [[ 1.5999999   1.          1.        ]
  [ 1.          1.          0.40000004]
  [ 1.          0.40000004  1.        ]]

 [[ 1.          1.          1.5999999 ]
  [ 1.5999999   1.          1.        ]
  [ 1.          0.40000004  1.        ]]

 [[ 1.5999999   1.          1.        ]
  [ 1.          1.5999999   1.        ]
  [ 1.          1.          0.40000004]]

 [[ 1.          1.5999999   1.        ]
  [ 1.5999999   1.          1.        ]
  [ 1.          1.          1.5999999 ]]]
```

Triangle 1

Triangle 2

Triangle 3

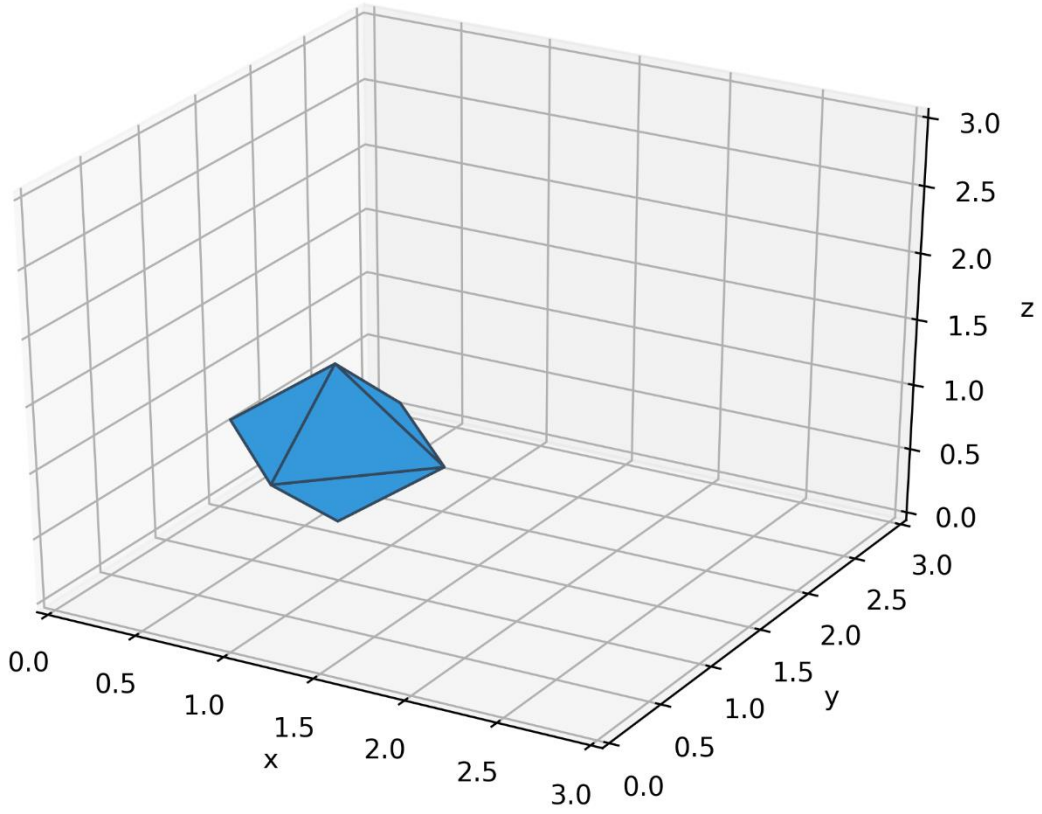
⋮

Triangle 8

Ref: Paul Bourke, “Polygonising a scalar field,” http://paulbourke.net/geometry/polygonise/

Solution: March through the voxels (cubes) and make polygons (cubes)

This is what a Gaussian looks like!

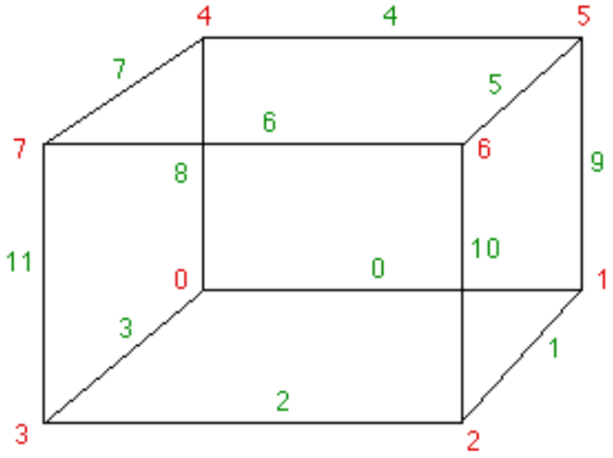


Ref: Paul Bourke, “Polygonising a scalar field,” http://paulbourke.net/geometry/polygonise/

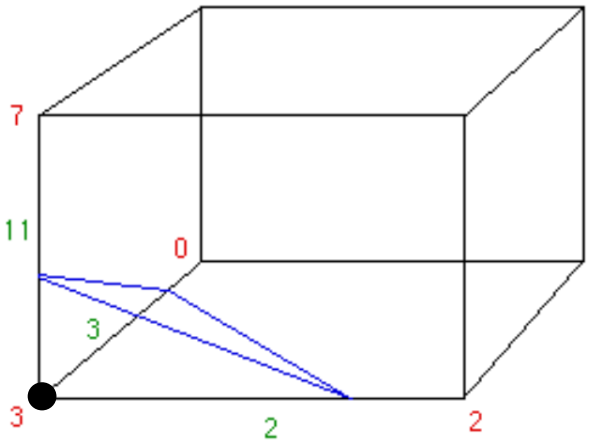
“The” Implementation of Marching Cubes

by Paul Bourke

Ref: “Polygonising a scalar field”, http://paulbourke.net/geometry/polygonise/



Edge index
Vertex index



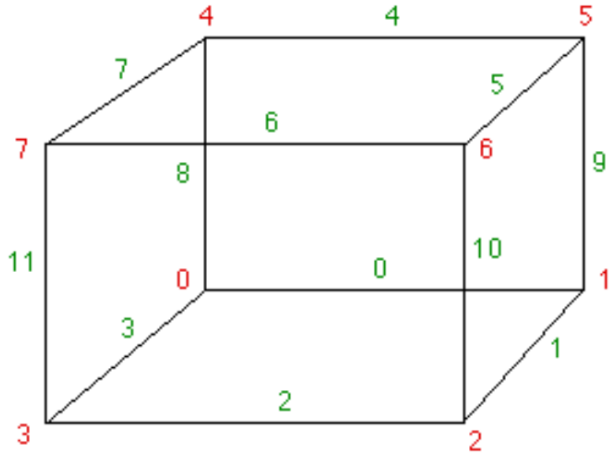
Vertex 3 inside (or outside) the volume

Isosurface facet

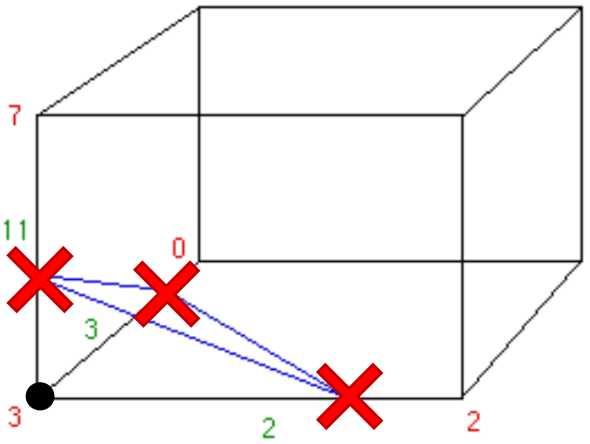
“The” Implementation: Step 1 Find Intersecting Edges

by Paul Bourke

Ref: “Polygonising a scalar field”, http://paulbourke.net/geometry/polygonise/



Edge index
Vertex index



Vertex 3 inside (or outside) the volume

Isosurface facet

vertex	< iso.
7	0
6	0
5	0
4	0
3	1
2	0
1	0
0	0

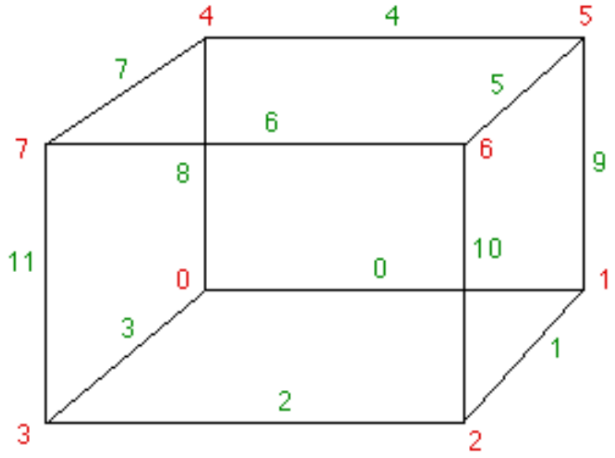
→ Edge table →

intersect	edge
1	11
0	10
0	9
0	8
0	7
0	6
0	5
0	4
1	3
1	2
0	1
0	0

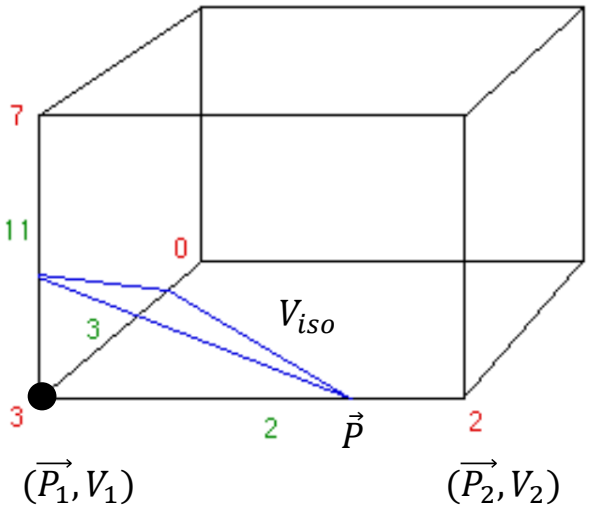
“The” Implementation: Step 2 Find Intersection Locations

by Paul Bourke

Ref: “Polygonising a scalar field”, http://paulbourke.net/geometry/polygonise/



Edge index
Vertex index



Vertex 3 inside (or outside) the volume

Isosurface facet

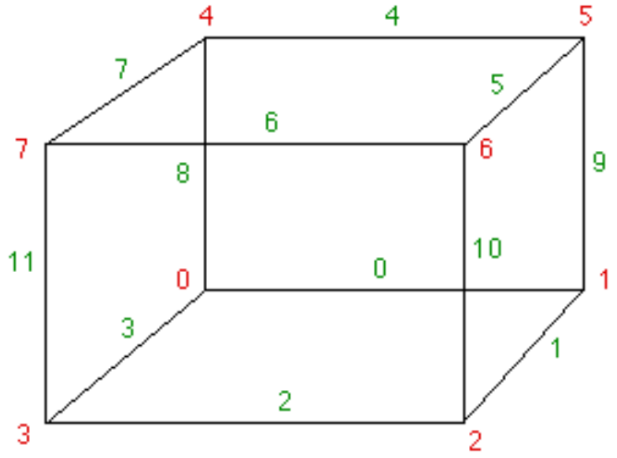
Linear Interpolation:

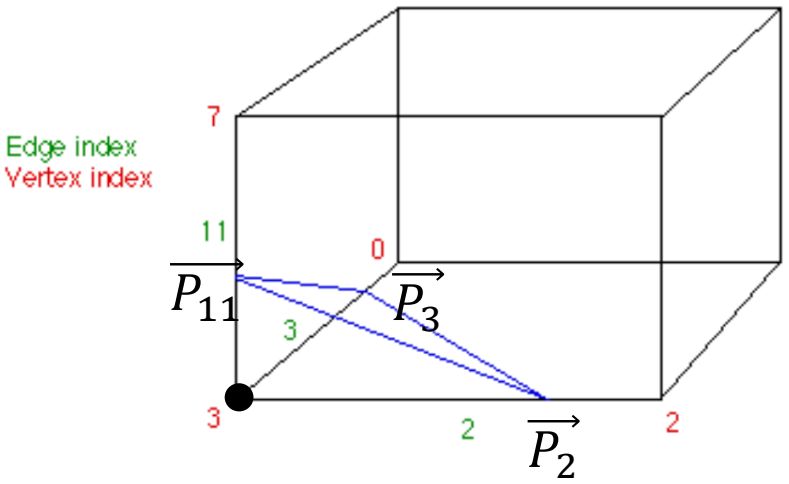
$$\vec{P} = \overrightarrow{P_1} + \frac{V_{iso} - V_1}{V_2 - V_1}\overrightarrow{P_2}$$

“The” Implementation: Step 3 List Triangles

by Paul Bourke

Ref: “Polygonising a scalar field”, http://paulbourke.net/geometry/polygonise/





Vertex 3 inside (or outside) the volume

Isosurface facet

Triangle list = [(3,11,2)]

$\overrightarrow{P_3}$ = [intersect at edge 3]

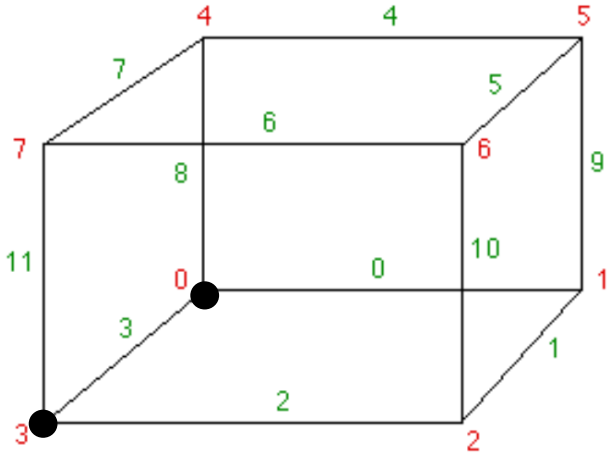
$\overrightarrow{P_{11}}$ = [intersect at edge 11]

$\overrightarrow{P_2}$ = [intersect at edge 2]

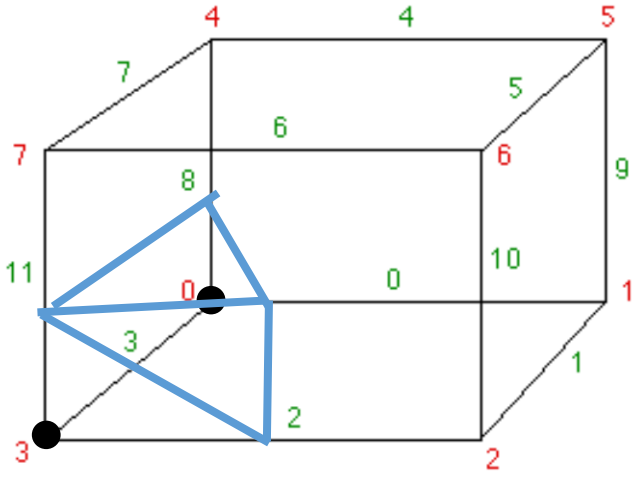
Exercise

by Paul Bourke

Ref: “Polygonising a scalar field”, http://paulbourke.net/geometry/polygonise/



Edge index
Vertex index



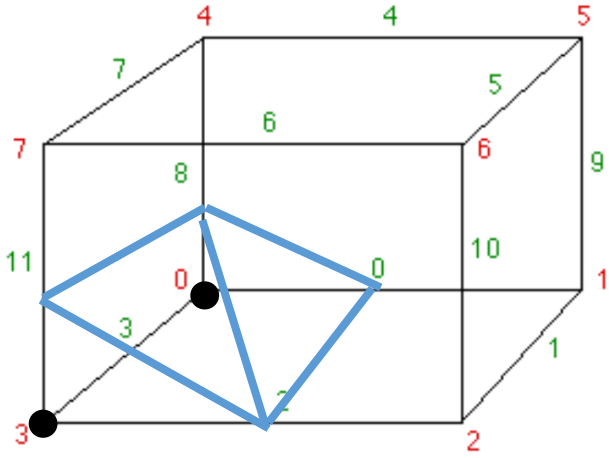
Edge index
Vertex index

vertex	< iso.
7	0
6	0
5	0
4	0
3	1
2	0
1	0
0	1

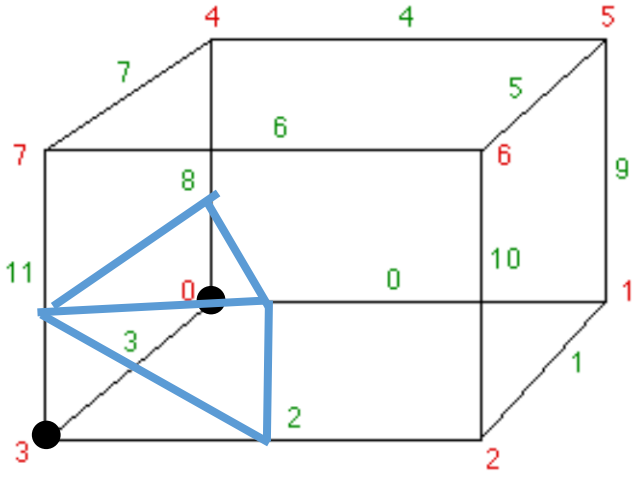
→ Edge table →

intersect	edge
1	11
0	10
0	9
1	8
0	7
0	6
0	5
0	4
0	3
1	2
0	1
1	0

Possible Ambiguity?



Edge index
Vertex index



Edge index
Vertex index

Constructing the Edge Table

1. Realize that there are $2^8 = 256$ cases to consider.

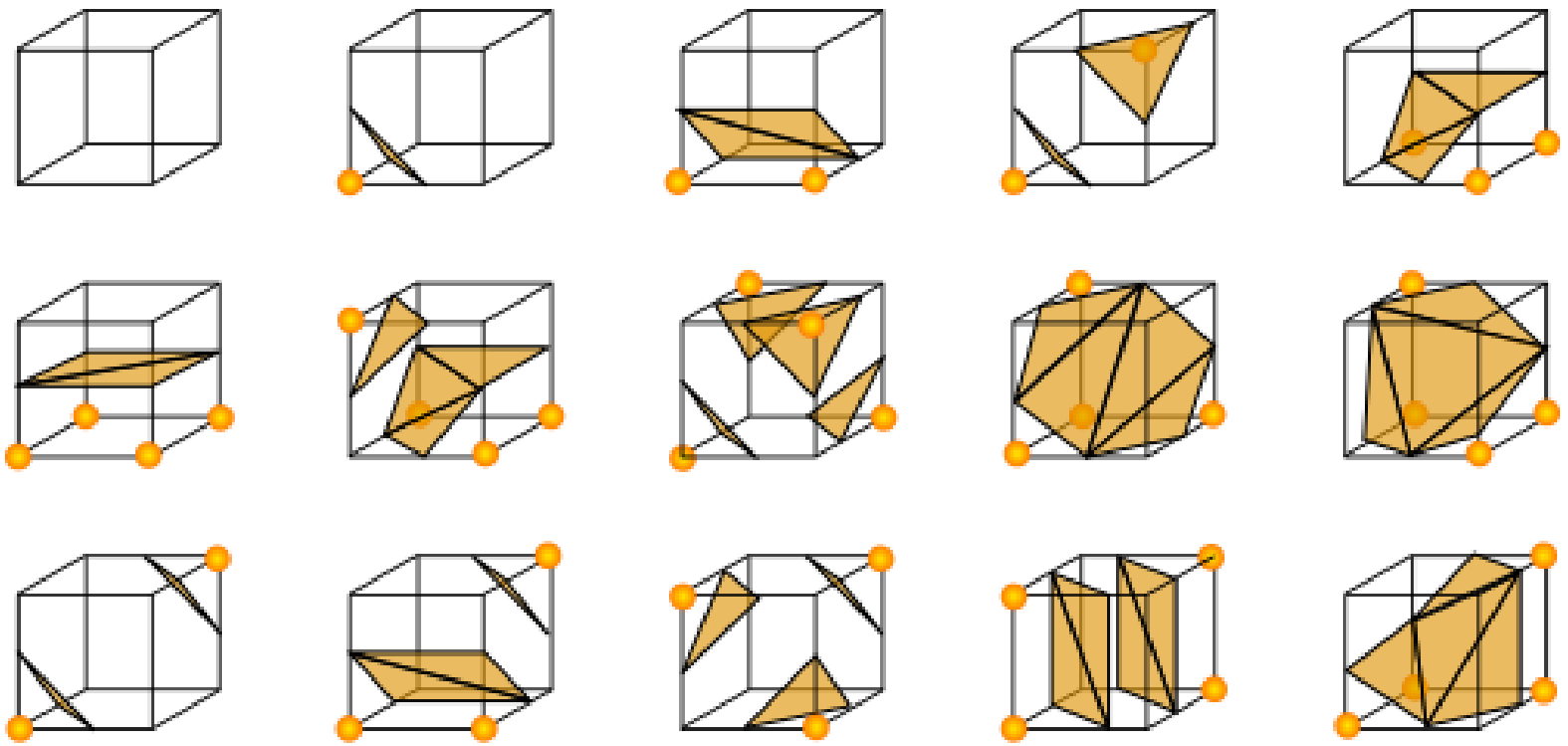
2. Use symmetries to reduce to 15 unique cases

3. Go through each case and resolve ambiguity

vertex	< iso.
7	0
6	0
5	0
4	0
3	1
2	0
1	0
0	0

→ Edge table →

intersect	edge
1	11
0	10
0	9
0	8
0	7
0	6
0	5
0	4
1	3
1	2
0	1
0	0

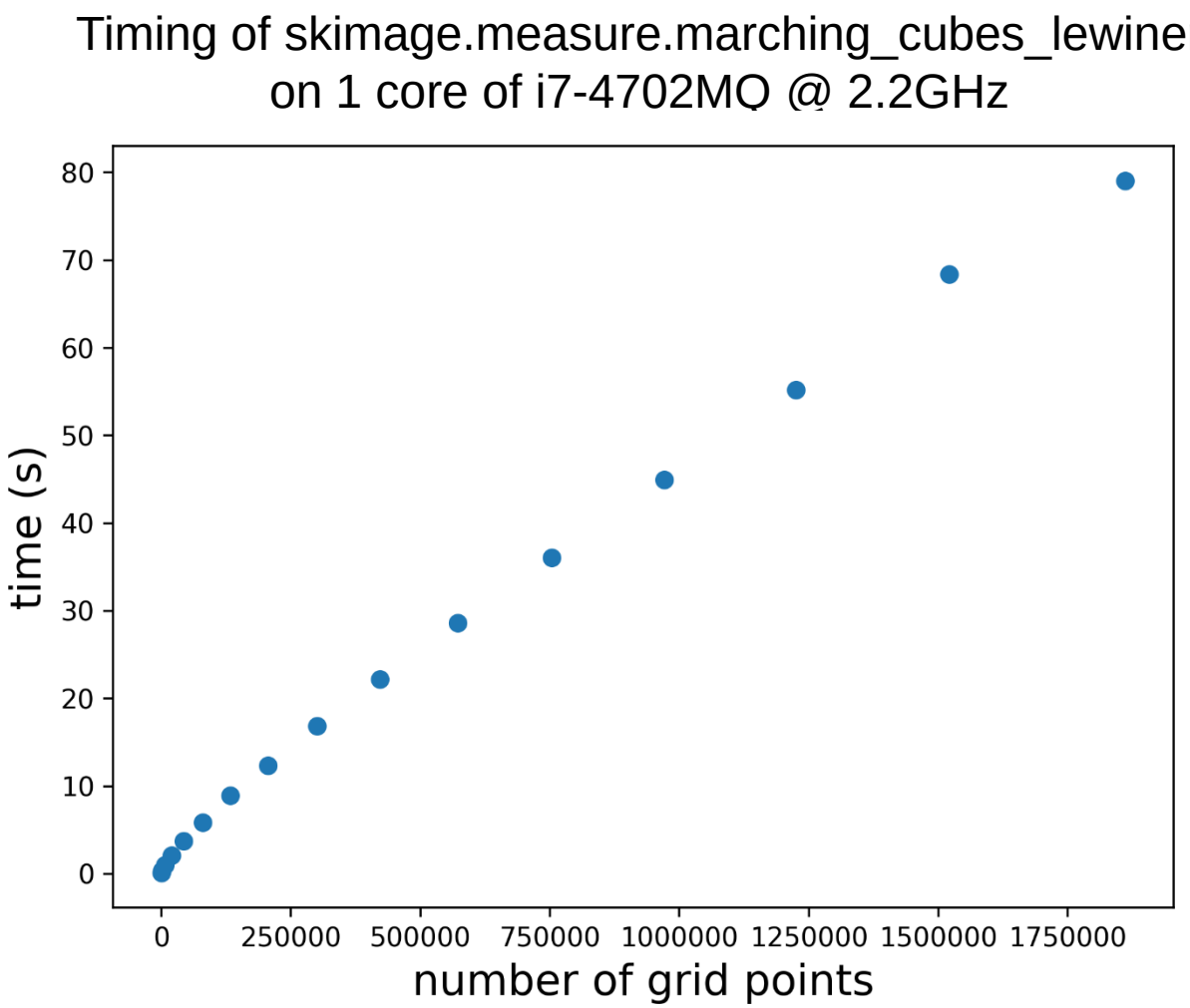


Wikipedia

Timing

The marching cubes algorithm is almost entirely table look-up

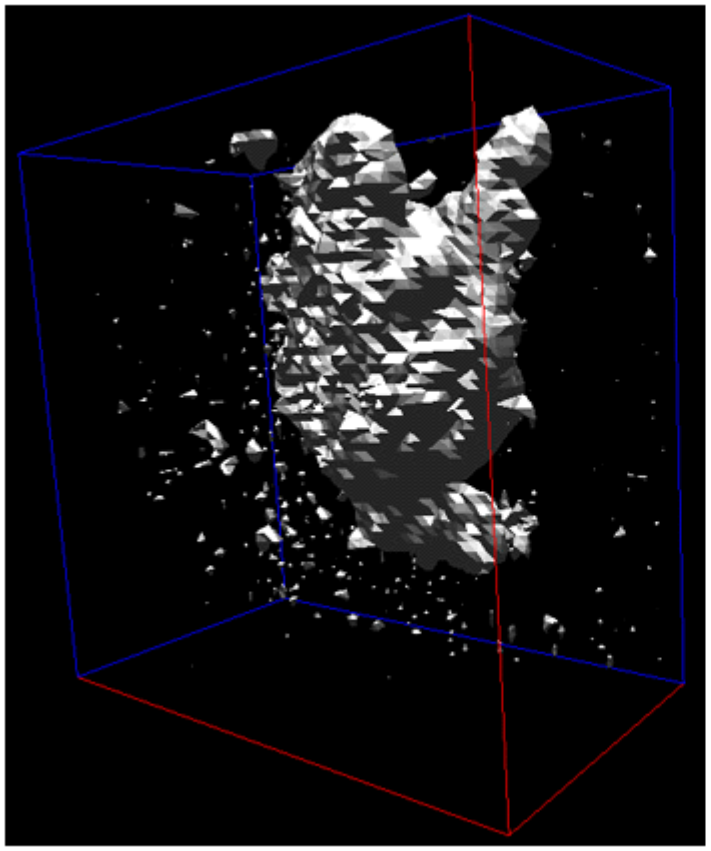
Slowness in matplotlib is likely due to 2D projection overhead (matplotlib does not do actual 3D rendering)



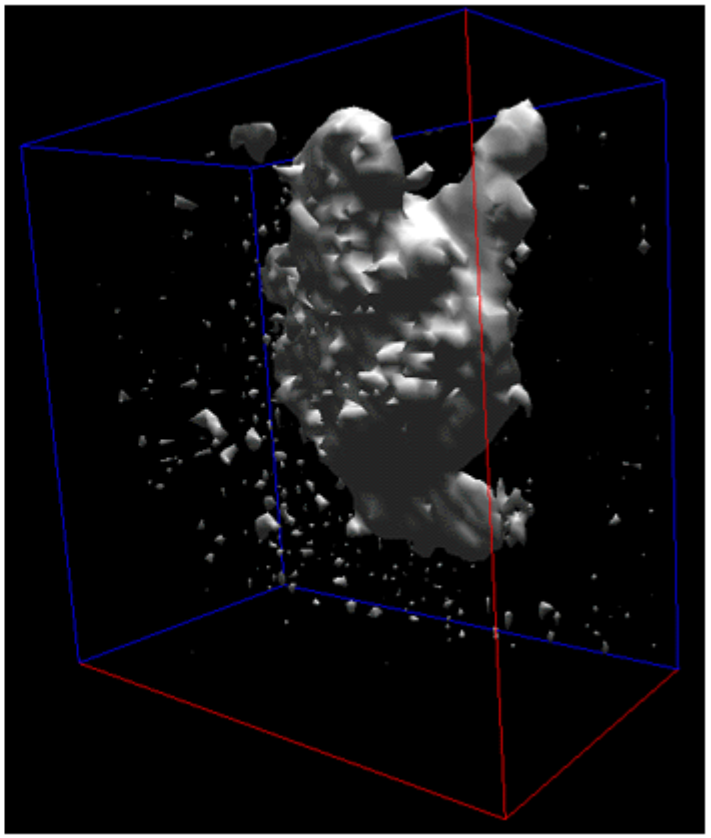
Normal Mapping

Dot **averaged** face normal vectors with light rays to determine luminescence

No vertex normals (OpenGL)



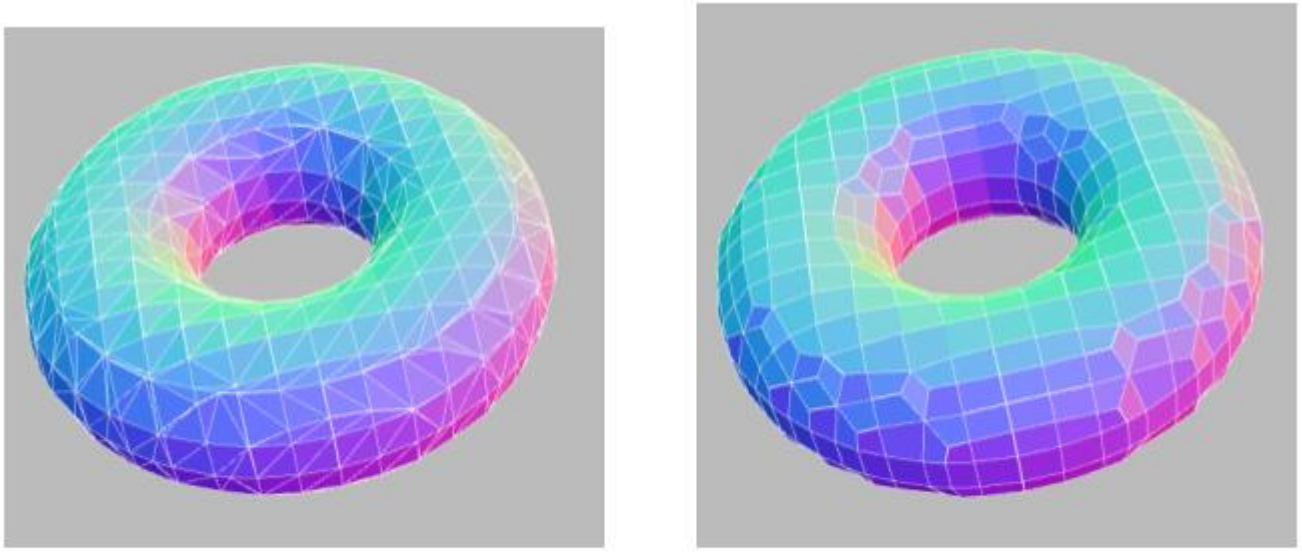
Vertex Normals (OpenGL)



Modern Replacement: Surface Nets

Compute the edge crossings (like we did in marching cubes) and then take their center of mass as the vertex for each cube.

Fewer vertices than marching cubes



Left: Marching cubes. Right: Naive surface nets.

Mikola Lysenko, “Smooth Voxel Terrain” (2012)
S.F. Gibson, (1999) “Constrained Elastic Surface Nets” Mitsubishi Electric Research Labs, Technical Report.

References:

[1] Paul Bourke, “Polygonising a scalar field” (1994)

[2] Mikola Lysenko, “Smooth Voxel Terrain” (2012)

[3] Sarah F. Gibson, “Constrained Elastic Surface Nets” (1999)

Utilities:

a click bate you will NOT regret falling for!

plot basis from PySCF

plot orbitals from PySCF (call the show_moR function)

plot spin density from PySCF