Supporting Information

Voronoi Polyhedra Probing of Hydrated OH Radical

Lukasz Kazmierczak and Dorota Swiatla-Wojcik*

Institute of Applied Radiation Chemistry, the Faculty of Chemistry,

Lodz University of Technology, Zeromskiego 116, 90-924 Lodz, Poland

Mathematical details on construction of VP. Let $C(x_0,y_0,z_0)$ and O_i (x_i,y_i,z_i) $(i = 1, 2,..., N)$ denote the central point (the radical oxygen atom) and the oxygen atom of the i-th molecule in the spherical neighbourhood of C. The equation of a plain Π_i bisecting the segment CO_i is given by:

$$\underset{(x,y,z)\in\Pi_i}{\forall} A_i x + B_i y + C_i z + D_i = 0 \qquad \text{(S1)}$$

where the coefficients A_i B_i, C_i, D_i are defined as:

$$A_i = x_i - x_0,\ B_i = y_i - y_0, C_i = z_i - z_0, D_i = \frac{1}{2}(x_0^2 + y_0^2 + z_0^2 - x_i^2 - y_i^2 - z_i^2) \qquad \text{(S2)}$$

The initial condition for a point $V_n(x_n,y_n,z_n)$ be a vertex of VP is that V_n must be the intersection of three perpendicular bisector planes. Considering intersection of any three bisector planes, Π_i, Π_j and Π_k, one can determine set of points $\{V_n(x_n,y_n,z_n)\}$, where coordinates x_n, y_n, and z_n satisfy the following set of equations:

$$\underset{\substack{(i,j,k=1,2,\ldots,N,\\ i\neq j, i\neq k, j\neq k)}}{\exists} \begin{cases} A_i x_n + B_i y_n + C_i z_n + D_i = 0 \\ A_j x_n + B_j y_n + C_j z_n + D_j = 0 \\ A_k x_n + B_k y_n + C_k z_n + D_k = 0 \end{cases} \qquad \text{(S3)}$$

Solution of set (S3) exists if the determinant $W = \begin{vmatrix} A_i & B_i & C_i \\ A_j & B_j & C_j \\ A_k & B_k & C_k \end{vmatrix} \neq 0$. Then

$$x_n = \frac{\begin{vmatrix} -D_i & B_i & C_i \\ -D_j & B_j & C_j \\ -D_k & B_k & C_k \end{vmatrix}}{W}, y_n = \frac{\begin{vmatrix} A_i & -D_i & C_i \\ A_j & -D_j & C_j \\ A_k & -D_k & C_k \end{vmatrix}}{W}, z_n = \frac{\begin{vmatrix} A_i & B_i & -D_i \\ A_j & B_j & -D_j \\ A_k & B_k & -D_k \end{vmatrix}}{W} \qquad \text{(S4)}$$

If $V_n(x_n,y_n,z_n)$ is the intersection point of planes Π_i, Π_j and Π_k, **the second condition** for $V_n(x_n,y_n,z_n)$ to be classified as a vertex of VP constructed about the central point $C(x_0,y_0,z_0)$ is expressed by Eq. (S5):

$$\underset{\substack{l=1,2,\ldots,N \\ l\neq i, l\neq j, l\neq k}}{\forall} \operatorname{sgn}(A_l x_n + B_l y_n + C_l z_n + D_l) = \operatorname{sgn}(A_l x_0 + B_l y_0 + C_l z_0 + D_l) \qquad \text{(S5)}$$

The condition (S5) means that for all other bisector plains Π_l $(l=1,2,\ldots,N,\ l\neq i,j,k)$ the point $V_n(x_n,y_n,z_n)$ must be located on the same side of Π_l as the point $C(x_0,y_0,z_0)$.

Sorting of vertices belonging to each bisector plane is required to determine VP edges and then to calculate properties of the constructed VP. We assign number 1 to an arbitrarily chosen vertex belonging to the i-th bisector plane and construct a reference vector $\overrightarrow{M_i V_{1i}}$, where M_i is the intersection point of CO_i line and the i-th bisector plane. Further numbering depends on the angles between the reference vector and vectors connecting M_i with the other vertices. Illustration of the sorting method is shown in Figure S1.

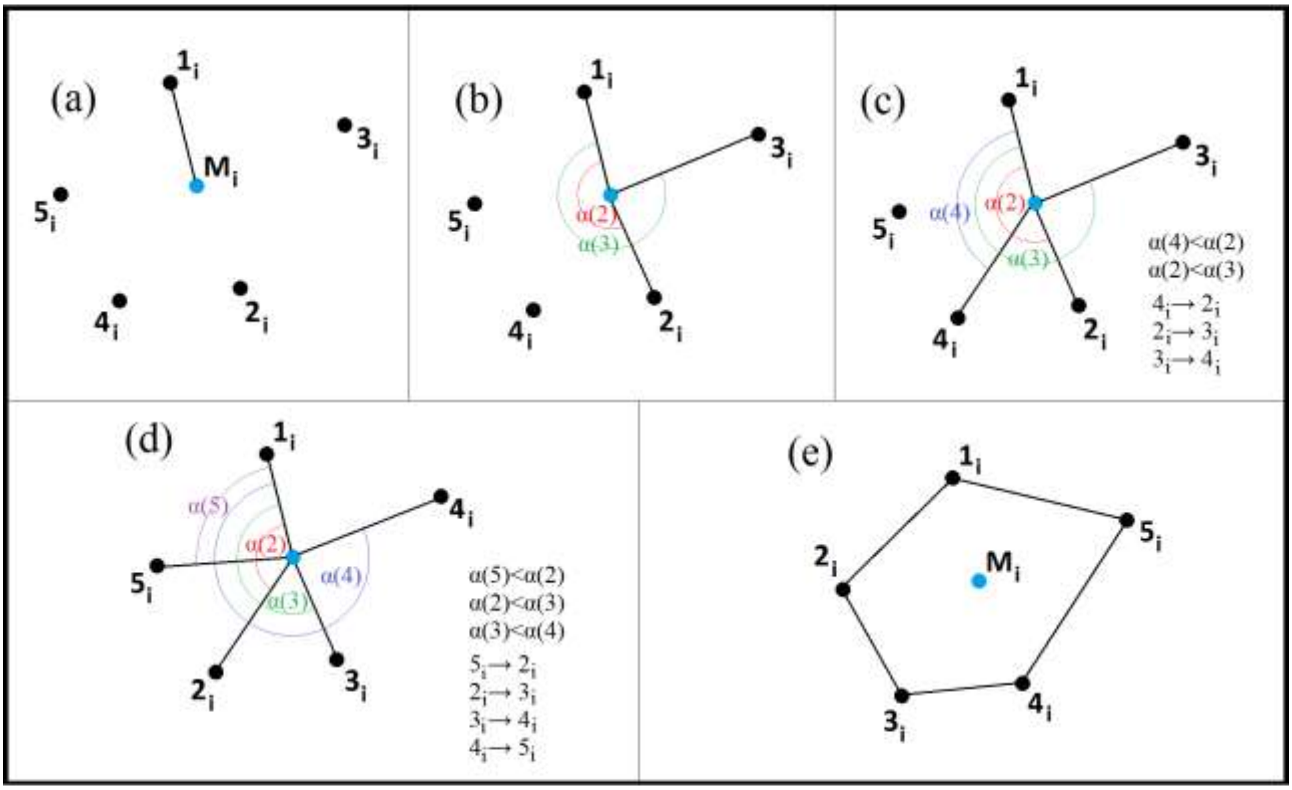

Fig. S1. Sorting of vertices belonging to the *i*-th bisector plane: (a) unsorted vertices and the reference vector $\overrightarrow{M_{\mathrm{i}}V_{1\mathrm{i}}}$; (b)-(d) sorting of vertices (e) anti-clockwise numbered vertices connected by edges.

Visualization methods. We have tested 3D-visualization of the VP by three methods using graphical facilities provided by *Microsoft Excel* spreadsheet application, *Maple* computer algebra system, and *Persistence of Vision Raytracer* **(***POV-Ray*) program. All the methods require a set of coordinates of sorted vertices belonging to the individual faces of the constructed *VP*.

Microsoft Excel spreadsheet application is suitable for simple 3D-presentationof VP. To use this program we follow the Gram-Schmidt process (see Ref. 18), which is a method for orthonormalising a set of vectors in an inner product space. We set the origin of a coordinate system in the VP centre $C(x_0,y_0,z_0)$ and accordingly recalculate coordinates of all vertices. Then a point of view $P(\cos\alpha\cos\beta\,,\sin\alpha\cos\beta\,,\sin\beta)$ is selected on the sphere of unit radius, where angles α and β are defined in Fig. S2.

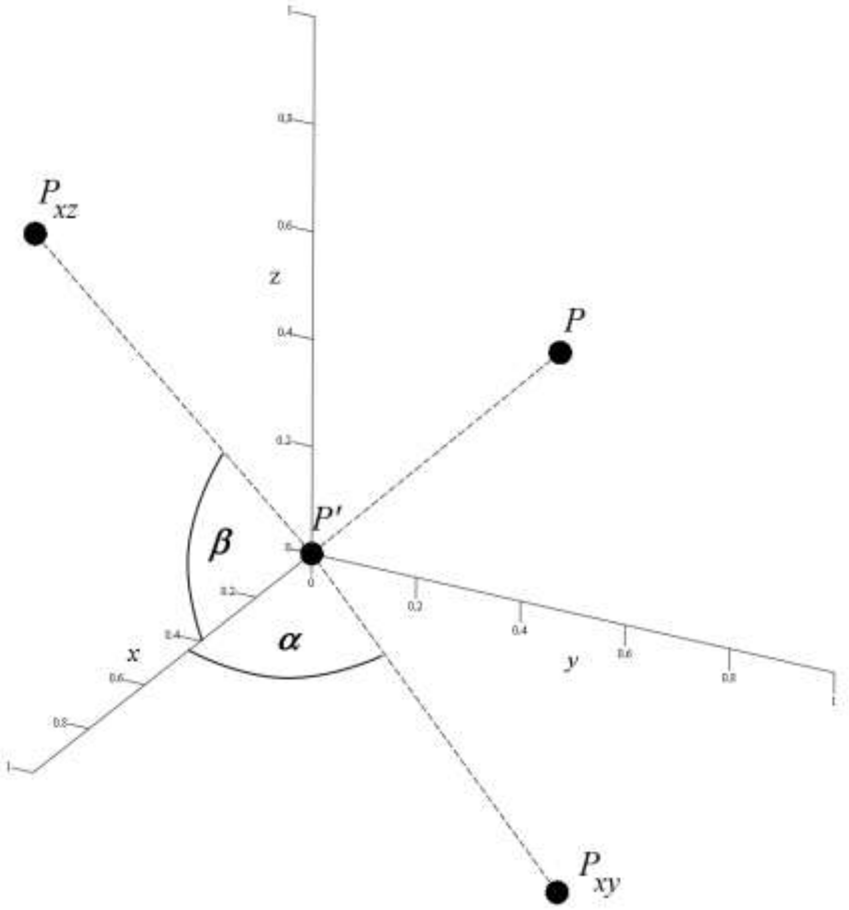

Fig. S2. Definition of angles α and β used in the Gram-Schmidt process: P'(0,0,0) is the orthogonal projection of the point of view $P(\cos\alpha\cos\beta, \sin\alpha\cos\beta, \sin\beta)$, P_{xy} is the orthogonal projection of P on the xy-plane, P_{xz} is the orthogonal projection of P on the xz-plane, α is the angle between $P'P_{xy}$ and the x-axis, β is the angle between $P'P_{xz}$ and the x-axis.

Our aim is to project vertices on the plane, which contains $\overrightarrow{P'P}$ vector. After the Gramm-Schmidt process the recalculated coordinates of the n-th vertex, ($a = x_n - x_0, b = y_n - y_0, c = z_n - z_0$), are given by Eq. (S6) :

$$a' = a + \frac{\cos\alpha\cos\beta}{r};\ b' = b + \frac{\sin\alpha\cos\beta}{r};\ c' = c + \frac{\sin\beta}{r} \qquad \text{(S6)}$$

where

$$r = \frac{-1}{a \cdot \cos\alpha \cos\beta + b \cdot \sin\alpha \cos\beta + c \cdot \sin\beta} \quad \text{(S7)}$$

Rotating $\overrightarrow{P'V_n}$ vector, where $V_n'(a', b', c')$, by the angle $(-\alpha)$ and next by $(-\beta)$, we obtain $V_n^1(a_1, b_1, c_1)$ and $V_n^2(a_2, b_2, c_2)$, expressed by Eqs. (S8) and (S9), respectively.

$$a_1 = a' \cos\alpha + b' \sin\alpha; \;\; b_1 = -a' \sin\alpha + b' \cos\alpha; \; c_1 = c' \quad \text{(S8)}$$

$$a_2 = a_1 \cos\beta + c_1 \sin\beta; \;\; b_2 = b_1; \; c_2 = -a_1 \sin\beta + c_1 \cos\beta \quad \text{(S9)}$$

Selecting the VP centre as a single point and using *MS Excel*Chart: XY(Scatter)-Straight-Lines option for vertices belonging to each of the *VP* faces (treated as data series) we obtain a graphical presentation of a solvation cage.

***Maple* computer algebra system.** The *Maple* program offers a command-line utility and ready-to-use macros accepting basic graphical options (colour, transparency, line-style, *etc*.). It makes 3D-visualisation of a solvation cage intuitive and easy.

<u>We start with the following calling sequences:</u>

> with(plots); with(plottools);

<u>Then, we define every face by using the sequence:</u>

>*name_of_face*:=display(polygon([[*coordinates_of_1st_vertex*],

[*coordinates_of_2nd_vertex*],…),options)

For example:

>f91:=display(polygon([[15.4,12.3,3.9],[14.4,12.7,2.1],[14.2,12.6,2.0],[14.1,8.3,3.1]]),colour=COLOUR(RGB,32/255,178/255,170/250),linestyle=solid,thickness=2,transparency=0.0);

<u>Optionally, the VP centre can be defined as a single point by using the sequence:</u>

>*name_of_centre*:=point([*coordinates*],options);

For example:

C9:=point([13.5,11.7,3.9],symbol=solidcircle,symbolsize=50,colour=navy);

Finally, we visualize the defined objects by the calling the macro:

>display(*name_of_face_1*,*name_of_face_2,* …,*name_of_centre*);

POV-Ray program (The Persistence of Vision Raytracer program, http://www.povray.org/)

POV-Ray program creates photo-realistic images using an advanced rendering technique, called ray-tracing. It produces very high quality images with realistic reflections, shading and perspective. The *POV-Ray* code is written in object-oriented C++. Fragments of the code used to visualize a solvation cage are given below.

The VP centre is defined by:

```
sphere
   {
<Center> Radius
     [OBJECT_MODIFIERS...]
   }
```

For example :

```
sphere {<13.5,11.7,3.9>0.5 texture{pigment{color rgbt<0,0,0.4>}} finish{reflection 0.1 phong 0.1}}
```

To visualize VP-faces we divided a given face (a convex polygon) into triangles and used the following sequence:

```
merge
   {
     triangle
   {
<Corner_1><Corner_2><Corner_3>
```

```
    [OBJECT_MODIFIER...]
}
        …
}
```

To visualize edges we used the sequence:

```
cylinder
  {
<Base_Point><Cap_Point> Radius
    [open][OBJECT_MODIFIERS...]
  }
```

Finally, all the defined objects were combined:

```
union
  {
sphere
  {….}                          The centre of a Voronoi-cell
merge
  {                             Faces
    triangle
  {…}
        …
}
merge
  {                             Edges
cylinder
  {…)
        …
    }
```