Chapter 4

Point-Based Surface Representation

The surface alignment algorithm proposed in this thesis is based on representing the molecular surface and properties defined thereon by points. In this chapter we will explain how we distribute the points on the surface. Our method requires each property to be given by some scalar field defined on the molecular surface. From this scalar field we compute a point density according to which the points are distributed. The scalar field may be homogeneous, i.e. the scalar values are constant across the whole surface as, e.g. in the case of the *shape property*. For other properties, such as the *electrostatic potential*, the scalar field will be non-homogeneous, i.e. the scalar values vary across the surface. Note that a property does not necessarily have to be defined on the whole surface but it can also be defined in certain regions, only. This will be the case, e.g., for the *hydrogen bonding* property.

The aim of our approach is to distribute the points on the surface as regular as it is allowed by the given point density. Here, regularity means that the standard deviation of the distances of neighbored points should be as small as possible. This avoids oversampling of certain regions, while it guarantees that an equally weighted surface patch is assigned to each point. In order to achieve a regular point distribution, we employ a two-step algorithm. The first step generates an initial point distribution. In the second step, the point positions are then iteratively relaxed according to the desired point density.

There exist several ways to generate an initial point distribution. A first approach would be to distribute the points randomly on the surface. This work, in principle, but imposes extra work on the relaxation step, which will need more iterations to arrive at a satisfying distribution. We have therefore developed a second approach, which is based on *multilevel k-way graph-partitioning* [92]. In this approach, we partition the surface into equally weighted patches and place a point into the center of each patch.

For the relaxation step we employ a concept known as *centroidal Voronoi tesselation* (CVT) [51]. A centroidal Voronoi tesselation is a Voronoi tesselation with the property that its generators coincide with the mass centroids of the Voronoi regions. We have developed a novel approximation scheme for computing the centroidal Voronoi tessela-

tion of a 2-dimensional surface embedded in 3-dimensional space and represented by a 2-manifold triangular mesh. In this scheme we employ an iterative method known as *Lloyd's method* [113]. In each iteration step of Lloyd's method we compute the Voronoi tesselation of the surface w.r.t. the current point distribution and subsequently move the points to the centers of the new Voronoi regions. This method converges quickly and produces very good point distributions.

In order to obtain point distributions of good quality, however, the resolution of the triangular mesh needs to be sufficiently fine, i.e. the number of triangles in the mesh need be much larger than the number of points to be distributed on the mesh. This assumption should be kept in mind when proceeding with the subsequent sections.

We begin this chapter in Section 4.1 with basic notations on triangular meshes and graphs defined thereon. These notations will be referred to in subsequent sections. In Section 4.2 we propose two algorithms for initializing point distributions. Section 4.3 deals with the relaxation step. We present two algorithms for computing an approximation of the centroidal Voronoi tesselation of a 2-manifold triangular mesh based on the graphs defined in Section 4.1. Results will be given in Section 4.4, where we compare the algorithms proposed in Sections 4.2 and 4.3 w.r.t. quality and run times. The results show that the proposed algorithms are efficient and achieve point distributions of very good quality for both homogeneous and non-homogeneous point densities. We conclude this chapter in Section 4.5 with a short discussion including a comparison of our approach with previously published approaches to distribute points on molecular surfaces.

4.1 Basic Notations

4.1.1 Triangular Mesh

As mentioned above, we assume the molecular surface to be given as a 2-manifold triangular mesh.

Definition 4.1.1 (Triangular Mesh)**.** A triangular mesh $\mathcal{M} = (V, E, T)$ is a triple consisting of

- a *set of vertices* $V = \{v_1, \ldots, v_m\}$,
- a *set of edges* $E = \{e_1, \ldots, e_n\} \subset V \times V$, with $(v_i, v_j) \in E \Rightarrow i \neq j$, and
- a *set of triangles* $T = \{t_1, \ldots, t_s\} \subset V \times V \times V$, with $(v_i, v_j, v_k) \in T \Rightarrow i \neq j \wedge i \neq k \wedge j \neq j$,

for which the following two properties hold:

1. Each vertex belongs to at least one triangle.
2. Each edge belongs to at least one triangle.

The set of vertices of $\mathcal{M}$ will be denoted by $V(\mathcal{M})$. Likewise the set of edges and the set of triangles of $\mathcal{M}$ will be denoted by $E(\mathcal{M})$ and $T(\mathcal{M})$, respectively. If it is clear from the context, we will simply use V, E, and T.

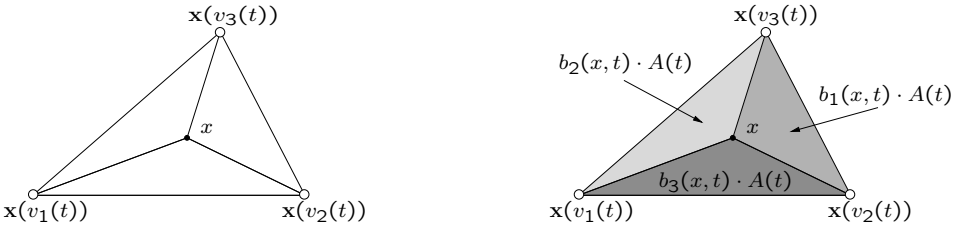


Figure 4.1: (a) Triangle t with vertices $v_1(t), v_2(t),$ and $v_3(t)$. A point x with barycentric coordinates $b_1(x,t)$, $b_2(x,t)$, and $b_3(x,t)$ w.r.t. triangle t resides in t. (b) The areas of the triangles spanned by x and the vertices of t are proportional to $b_1(x,t)$, $b_2(x,t)$, and $b_3(x,t)$.

Definition 4.1.2 (2-Manifold Triangular Mesh)**.** A *2-manifold triangular mesh*, $\mathcal{M}$, is a triangular mesh where every point on $\mathcal{M}$ is contained in some open set which is topologically equivalent to $\mathbb{R}^2$.

From this definition it immediately follows that at most two triangles can be adjacent to a single edge. In this thesis we will only consider 2-manifold triangular meshes. Hence, we will often simply use the term *triangular mesh* of even *mesh* instead of *2-manifold triangular mesh*.

For $v \in V$, we denote by $\mathbf{x}(v) \in \mathbb{R}^3$ the coordinates of vertex v. For $t \in T$, we denote by $v_i(t)$, $i = 1, 2, 3$, the i'th vertex of t. Using these notations we can write the *area* of t, denoted by $A(t)$, as

$$A(t) = \frac{1}{2} \left(\mathbf{x}(v_2(t)) - \mathbf{x}(v_1(t))\right) \times \left(\mathbf{x}(v_3(t)) - \mathbf{x}(v_1(t))\right) ,$$

where $\vec{x} \times \vec{y}$ is the cross-product of the vectors $\vec{x}$ and $\vec{y}$.

Let $x \in \mathbb{R}^3$ be a point residing on triangle t. Then, x can be written as

$$\begin{aligned} x = b_1(x,t)\,\mathbf{x}(v_1(t)) + b_2(x,t)\,\mathbf{x}(v_2(t)) + b_3(x,t)\,\mathbf{x}(v_3(t)) \ , \\ b_1(x,t), \ldots, b_3(x,t) \in [0,1],\ b_1(x,t) + b_2(x,t) + b_3(x,t) = 1 \ , \end{aligned}$$

where $b_1(x,t), \ldots, b_3(x,t)$ are the *barycentric coordinates* of x with respect to t (cf. Figure 4.1(a)). It can easily be seen that each $b_i(x,t)$, $i = 1, 2, 3$, is proportional to the area of the sub-triangle spanned by x and the edge opposite of $v_i(t)$ (cf. Figure 4.1(b)), i.e.,

$$A(\mathbf{x}(v_{i\oplus 1}), \mathbf{x}(v_{i\oplus 2}), x) = b_i(x,t)\, A(t) \ , \tag{4.1}$$

where $A(\mathbf{x}(v_{i\oplus 1}), \mathbf{x}(v_{i\oplus 2}), x)$ is the area spanned by $\mathbf{x}(v_{i\oplus 1}), \mathbf{x}(v_{i\oplus 2})$, and x, and $i \oplus j$ denotes $((i - 1 + j) \mod 3) + 1$, $i \in \{1, \ldots, 3\}$.

Given a scalar function $\rho : V \to \mathbb{R}$ defined on the vertices of $\mathcal{M}$, for each triangle $t \in T$ and each point x residing on t we can define a linear interpolant $\tilde{\rho}(x,t)$ by means of the barycentric coordinates of x as

$$\tilde{\rho}(x,t) := b_1(x,t) \cdot \rho(v_1(t)) + b_2(x,t) \cdot \rho(v_2(t)) + b_3(x,t) \cdot \rho(v_3(t)) \ . \tag{4.2}$$

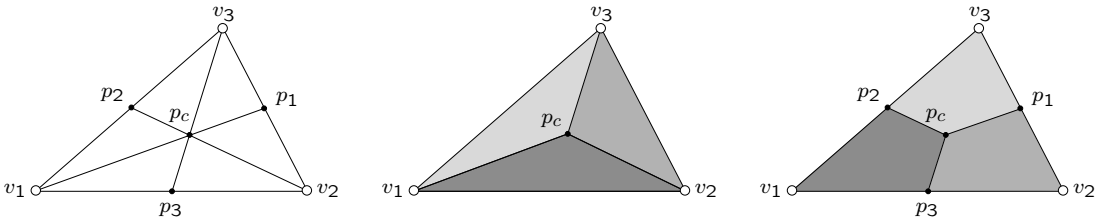


Figure 4.2: Triangle with vertices v_1, v_2, and v_3, and mass centroid p_c. The points p_1, p_2, and p_3 are the midpoints of the triangle's edges. (a) The line through vertex v_i and p_i is the bisecting line of the triangle's angle at vertex v_i. The intersection point p_c of these lines is the mass centroid of the triangle. (b) The mass centroid p_c subdivides the triangle into three equally sized triangles. (c) The mass centroid also subdivides the triangle into three equally sized quadrangles spanned by the triangle's vertices, the mass centroid, and the midpoints of the triangle's edges.

The coordinates of the *center of mass* of triangle t, denoted by $\mathbf{x}(t)$, are defined by

$$\mathbf{x}(t) := \frac{\mathbf{x}(v_1(t)) + \mathbf{x}(v_2(t)) + \mathbf{x}(v_3(t))}{3} ,$$

which means that the barycentric coordinates of the center of mass of t are equal to $1/3$. It also means that $\mathbf{x}(t)$ divides triangle t into three equally sized sub-triangles (cf. Equation (4.1) and Figure 4.2(b)). For the center of mass we will use the term *mass centroid* synonymously.

By $\mathbf{n}(t)$ we denote the *normal vector* of t. Assuming that the vertices of t are in *counter-clockwise order*, $\mathbf{n}(t)$ is given by

$$\mathbf{n}(t) = \frac{(\mathbf{x}(v_2(t)) - \mathbf{x}(v_1(t))) \times (\mathbf{x}(v_3(t)) - \mathbf{x}(v_1(t)))}{\|(\mathbf{x}(v_2(t)) - \mathbf{x}(v_1(t))) \times (\mathbf{x}(v_3(t)) - \mathbf{x}(v_1(t)))\|} ,$$

where $\|\vec{x}\|$ denotes the length of vector $\vec{x}$.

We define the *triangle neighborhood of some vertex* $v \in V$, denoted by $N(v)$, as the set of triangles incident to v (cf. Figure 4.3(a)), i.e.,

$$N(v) := \left\{ t \mid \exists i \in \{1,2,3\} : v = v_i(t) \right\} . \tag{4.3}$$

Likewise we can define the *triangle neighborhood of some edge* $e \in E$, denoted by $N(e)$, as the set of triangles that share edge e, i.e.,

$$N(e) := \left\{ t \mid e = (u,v) \wedge t \in N(u) \wedge t \in N(v) \right\} .$$

Then the *triangle neighborhood of some triangle* $t \in T$, denoted by $N(t)$, can be defined as the set of triangles that share a common edge with t (cf. Figure 4.3(c)), i.e.,

$$N(t) := \left\{ t' \mid t' \neq t \wedge \exists i,j \in \{1,2,3\}, i \neq j : t' \in N((v_i(t), v_j(t))) \right\} .$$

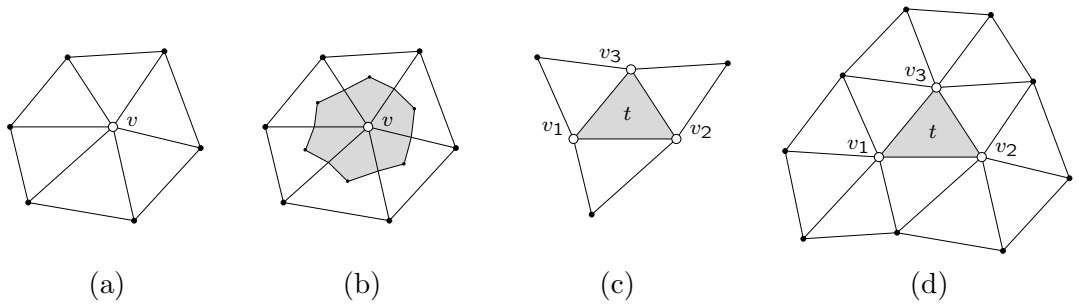


Figure 4.3: (a) Triangle neighborhood $N(v)$ of vertex v. (b) The gray polygon around vertex v depicts the surface area $A(v)$ assigned to vertex v. (c) Triangle neighborhood $N(t)$ of triangle t. (d) Extended triangle neighborhood $N^*(t)$ of triangle t. The gray triangle does neither belong to $N(t)$ nor to $N^*(t)$.

The *extended triangle neighborhood of some triangle* t, denoted by $N^*(t)$, consists of the set of triangles that share a common vertex with t (cf. Figure 4.3(d)), i.e.,

$$N^*(t) := \left\{ t' \mid t' \neq t \wedge \exists i \in \{1,2,3\} : t' \in N(v_i(t)) \right\} .$$

Let $\widetilde{T} \subseteq T$, then we can define the *area*, the *center of mass*, and the *normal vector* of $\widetilde{T}$, respectively, by

$$\begin{aligned} A(\widetilde{T}) &:= \sum_{t \in \widetilde{T}} A(t) \ , \\ \mathbf{x}(\widetilde{T}) &:= \frac{1}{A(\widetilde{T})} \sum_{t \in \widetilde{T}} A(t)\, \mathbf{x}(t) \ , \text{and} \\ \mathbf{n}(\widetilde{T}) &:= \frac{1}{A(\widetilde{T})} \sum_{t \in \widetilde{T}} A(t)\, \mathbf{n}(t) \ . \end{aligned}$$

Let $\rho : V \to \mathbb{R}^+$ be a weight function defined on the vertices V of $\mathcal{M}$. Then, the *weighted center of mass* and the *weighted normal vector*, respectively, of some $\widetilde{T} \subseteq T$ can be defined as

$$\mathbf{x}(\widetilde{T}, \rho) := \frac{1}{\sum_{t \in \widetilde{T}} \tilde{\rho}(\mathbf{x}(t), t) \cdot A(t)} \sum_{t \in \widetilde{T}} \tilde{\rho}(\mathbf{x}(t), t) \cdot A(t)\, \mathbf{x}(t) \ , \text{and} \tag{4.4}$$

$$\mathbf{n}(\widetilde{T}, \rho) := \frac{1}{\sum_{t \in \widetilde{T}} \tilde{\rho}(\mathbf{x}(t), t) \cdot A(t)} \sum_{t \in \widetilde{T}} \tilde{\rho}(\mathbf{x}(t), t) \cdot A(t)\, \mathbf{n}(t) \ , \tag{4.5}$$

where $\tilde{\rho}(\cdot, \cdot)$, defined in Equation (4.2), is the linear interpolant of $\rho(\cdot)$.

Given some $\widetilde{T} \subseteq T$, we define the *patch defined by* $\widetilde{T}$ as

$$\begin{aligned} P(\widetilde{T}) := \Big\{ x \in \mathbb{R}^3 \mid \exists t \in \widetilde{T} \wedge \exists b_i(x,t) \in [0,1],\ i = 1,2,3 : \\ b_1(x,t) + b_2(x,t) + b_3(x,t) = 1 \ \wedge \\ x = b_1(x,t)\, \mathbf{x}(v_1(t)) + b_2(x,t)\, \mathbf{x}(v_2(t)) + b_3(x,t)\, \mathbf{x}(v_3(t)) \Big\} \ . \end{aligned}$$

Occasionally, it is necessary to partition the surface such that each part of the surface is assigned to exactly one vertex. If we assigned to each vertex v its neighborhood $N(v)$ (cf. Equation (4.3) and Figure 4.3(a)), we would assign each triangle to its three vertices. But this neighborhood, also known as the 1-ring of v, is a good starting point. We therefore assign to each vertex that part of each neighbored triangle which is spanned by vertex v, the triangle's mass centroid, and the midpoints of the triangle's edges adjacent to v (cf. Figure 4.2(c)). Since each triangle mass centroid partitions the triangle into three equally sized sub-triangles, and the lines from the mass centroid to the midpoints of the triangle's edges subdivide these sub-triangles again into equally sized triangles, the area of each neighbored triangle t that will be assigned to vertex v is one third of the overall area of t. Thus, for the surface area $A(v)$ assigned to some vertex v (cf. Figure 4.3(b)) we get

$$A(v) = \frac{1}{3} \sum_{t \in N(v)} A(t) \ .$$

The *vertex normal vector* can now be defined as

$$\mathbf{n}(v) := \frac{\sum_{t \in N(v)} A(t)\, \mathbf{n}(t)}{\sum_{t \in N(v)} A(t)} \ .$$

Given some $\widetilde{V} \subseteq V$, we define the *area* of $\widetilde{V}$ and the *patch* defined by $\widetilde{V}$, respectively, as

$$A(\widetilde{V}) := \sum_{v \in \widetilde{V}} A(v) \ , \text{ and}$$

$$\begin{aligned} P(\widetilde{V}) := \Big\{ x \in \mathbb{R}^3 \mid \exists v \in \widetilde{V} \wedge \exists t \in N(v) \wedge \exists i \in \{1,2,3\} : v = v_i(t) \ \wedge \\ \exists b_j(x,t) \in [0,1],\ j = 1,2,3 :\ b_1(x,t) + b_2(x,t) + b_3(x,t) = 1 \ \wedge \\ x = b_1(x,t)\, \mathbf{x}(v_1(t)) + b_2(x,t)\, \mathbf{x}(v_2(t)) + b_3(x,t)\, \mathbf{x}(v_3(t)) \ \wedge \\ b_i(x,t) \geq b_j(x,t), j \neq i \Big\} \end{aligned}$$

Let $\rho : V \to \mathbb{R}^+$ be a weight function defined on the vertices V of $\mathcal{M}$. Then, similar to Equations (4.4) and (4.5), for some $\widetilde{V} \subseteq V$ we can define the *weighted center of mass* and the *weighted normal vector*, respectively, as

$$\mathbf{x}(\widetilde{V}, \rho) := \frac{1}{\sum_{v \in \widetilde{V}} \rho(v) \cdot A(v)} \sum_{v \in \widetilde{V}} \rho(v) \cdot A(v)\, \mathbf{x}(v) \ , \text{and}$$

$$\mathbf{n}(\widetilde{V}, \rho) := \frac{1}{\sum_{v \in \widetilde{V}} \rho(v) \cdot A(v)} \sum_{v \in \widetilde{V}} \rho(v) \cdot A(v)\, \mathbf{n}(v) \ .$$

Finally, we define the *geodesic distance* $d_{P(\cdot)}(x, y)$ between two points $x, y \in P(\cdot)$ as the length of the shortest path from x to y on $P(\cdot)$. In order for the shortest path to exist, $P(\cdot)$ needs to be connected.

4.1.2 Graphs Defined on the Triangular Mesh

In the following we will introduce some graphs that can be defined on a triangular mesh. These graphs play an important role in the point distribution process. The first graph, called *triangular mesh graph* or simply *mesh graph*, is denoted by $G(\mathcal{M})$ and defined directly by means of the sets $V(\mathcal{M})$ and $T(\mathcal{M})$.

Definition 4.1.3 (Triangular Mesh Graph)**.** Let $\mathcal{M} = (V, E, T)$ be a triangular mesh. Then the *triangular mesh graph* is defined by

$$G(\mathcal{M}) := (V, E).$$

The second graph is called the *dual mesh graph*, which, as the name suggests, is defined on the *dual mesh*. So let's first define the dual mesh of some triangular mesh $\mathcal{M}$.

Definition 4.1.4 (Dual Mesh)**.** Let $\mathcal{M} = (V, E, T)$ be a triangular mesh, then the *dual mesh* of $\mathcal{M}$ (cf. Figure 4.4(a) and (b)) is a triple $\widetilde{\mathcal{M}} = (\widetilde{V}, \widetilde{E}, \widetilde{F})$ consisting of three sets:

- A *set of vertices* $\widetilde{V} = \{\tilde{v}_1, \ldots, \tilde{v}_s\}$, where each triangle $t_i \in T$ is represented by vertex $\tilde{v}_i \in \widetilde{V}$, i.e. there is a one-to-one correspondence between the triangles of $\mathcal{M}$ and the vertices of $\widetilde{\mathcal{M}}$, given by the bijective function $m_T : T \to \widetilde{V}$ defined by

$$m_T(t_i) := \tilde{v}_i \ , \ \forall i \in \{1, \ldots, |T|\} \ .$$

- A *set of edges* $\widetilde{E} = \{\tilde{e}_1, \ldots, \tilde{e}_k\} \subset \widetilde{V} \times \widetilde{V}$, where each edge $\tilde{e} \in \widetilde{E}$ connects the vertices representing the triangles attached to some edge $e \in E$, i.e.,

$$\begin{aligned} \forall \tilde{e} = (\tilde{u}, \tilde{v}) \in \widetilde{E} \Rightarrow \exists t', t'' \in T \wedge \exists e \in E : \\ t', t'' \in N(e) \wedge m_T(t') = \tilde{u} \wedge m_T(t'') = \tilde{v} \ . \end{aligned}$$

- A *set of polygonal faces* $\widetilde{F} = \{\tilde{f}_1, \ldots, \tilde{f}_m\}$, where each vertex $v_i \in V$ is represented by a polygonal face $\tilde{f}_i \in \widetilde{F}$, i.e., there is a one-to-one correspondence between the vertices of $\mathcal{M}$ and the faces of $\widetilde{\mathcal{M}}$.

For some mesh $\mathcal{M}$, the set of vertices and the set of edges of $\mathcal{M}$'s dual mesh will also be denoted by $\widetilde{V}(\mathcal{M})$ and $\widetilde{E}(\mathcal{M})$, respectively.

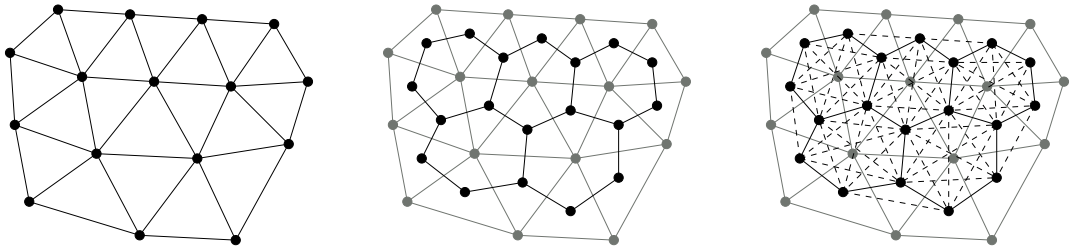
We can now give a definition of the dual mesh graph.

Definition 4.1.5 (Dual Mesh Graph)**.** Let $\widetilde{\mathcal{M}} = (\widetilde{V}, \widetilde{E}, \widetilde{F})$ be the dual mesh of some triangular mesh $\mathcal{M}$. Then we define the *dual mesh graph* as

$$\widetilde{G}(\mathcal{M}) := (\widetilde{V}, \widetilde{E}).$$

The dual mesh graph is used to obtain a good initial point distribution (cf. Section 4.2). An example of a dual mesh graph is given in Figure 4.4(b).

There will yet be another graph of interest in the subsequent sections, which we will call the *extended dual mesh graph*.



(a) (b) (c)

Figure 4.4: (a) Mesh graph $G(\mathcal{M})$. (b) Dual mesh graph, $\widetilde{G}(\mathcal{M})$, shown in black. (c) Extended dual mesh graph, $\widetilde{G}^*(\mathcal{M})$, shown as black solid and dashed lines.

Definition 4.1.6 (Extended Dual Mesh Graph)**.** Let $\widetilde{\mathcal{M}} = (\widetilde{V}, \widetilde{E}, \widetilde{F})$ be the dual mesh of some triangular mesh $\mathcal{M}$. Furthermore, let

$$\widetilde{E}^* := \left\{ (\tilde{v}_i, \tilde{v}_j) \mid i \neq j \wedge m_T^{-1}(\tilde{v}_i) \in N^*(m_T^{-1}(\tilde{v}_j)) \right\} \supset \widetilde{E} \ .$$

Then the *extended dual mesh graph* is defined by

$$\widetilde{G}^*(\mathcal{M}) := (\widetilde{V}, \widetilde{E}^*) \ .$$

It consists of the same set of vertices as $\widetilde{G}(\mathcal{M})$, but has an extended set of edges (cf. Figure 4.4(c)).

4.2 Initial Point Distribution

In this section we describe two approaches that generate an initial point distribution. The first one operates directly on the triangular mesh and uses a Monte Carlo method. The second approach is based on mesh partitioning using a graph partitioning scheme on the dual mesh graph.

4.2.1 Determination of Number of Points

Before we can start to distribute points on the molecular surface given by a triangular mesh $\mathcal{M} = \{V, E, T\}$, we need to determine the number of points to be distributed on $\mathcal{M}$. Let $\rho : V \to \mathbb{R}^+$ be the point density defined on the vertices of $\mathcal{M}$. The number of points $\pi(t)$ to be distributed on triangle t is proportional to the triangle area and is computed by

$$\pi(t) := \frac{\sum_{i=1}^{3} \rho(v_i(t))}{3} \cdot A(t) = \tilde{\rho}(\mathbf{x}(t), t) \cdot A(t) \ . \tag{4.6}$$

Since we are interested in distributing k points on $\mathcal{M}$, with $k \ll |T|$, $\pi(t)$ will be much smaller than 1. Hence, rather than giving us the actual number of points to be positioned into triangle t, $\pi(t)$ gives us the probability with which we position a single point into t.

The point densities we use are such that the distances between neighbored points in the final point distribution will be within a certain range, e.g. 0.5 to 3.0 Å. In Section 4.4 we give examples of point densities that are of interest to us. For a homogeneous point density, the distances between neighbored points should be as similar as possible. For non-homogeneous point densities, the point distances will vary according to the required point densities. Using Equation (4.6), the overall number of points to be distributed on $\mathcal{M}$ can be computed by accumulating the values $\pi(t)$ and rounding this sum to the nearest integer value, i.e.,

$$\pi(T) := \left\lceil \sum_{t \in T} \pi(t) \right\rceil .$$

4.2.2 Random Initialization Using a Monte Carlo Method

A simple way to distribute $\pi(T)$ points on $\mathcal{M}$, $\pi(T) \ll |T|$, according to a given point density $\rho(\cdot)$ is by using a *Metropolis-Monte Carlo* algorithm. A Monte Carlo algorithm generates an ensemble of states according to some given distribution by successively generating a Markov chain. In our case, the states of the Markov chain are the triangles, $\{t_k\}_{k=1}^{|T|}$. By generating a Markov chain, we want to determine the triangles into which we position the points. The Markov chain will only be used to determine $\pi(T)$ distinct triangles. Once $\pi(T)$ distinct triangles have been found, we stop.

In order to generate a Markov chain, we start with a randomly generated state, i.e. a randomly selected triangle t_{i_1}. From this initial state, the Monte Carlo algorithm generates a Markov chain

$$t_{i_1} \rightarrow t_{i_2} \rightarrow \ldots \rightarrow t_{i_k} \rightarrow \ldots \rightarrow t_{i_N} \ , \ k \in \{1, \ldots, |T|\} \ ,$$

such that the generated ensemble, i.e. the generated states of the Markov chain, has a distribution equal to $\pi(\cdot)$ (cf. Equation (4.6)). The generation of a new state is done in two steps. The first step proposes a new state, hence, it is called *proposal* step. The second step is called *acceptance step*, since it decides whether the proposed state is accepted or not. Let $\mathbf{P}(t \rightarrow \tilde{t})$ be the conditional probability to go from state t to state $\tilde{t}$. Then, according to the *detailed balance* condition (see, e.g., Section 2.3 in [170]), the conditional probability density function $\mathbf{P}(\cdot)$ must satisfy

$$\pi(t)\mathbf{P}(t \rightarrow \tilde{t}) = \pi(\tilde{t})\mathbf{P}(\tilde{t} \rightarrow t) \ . \tag{4.7}$$

If we use a *Metropolis-Hastings* algorithm, the conditional probability density function is split into two factors

$$\mathbf{P}(t \rightarrow \tilde{t}) = \mathbf{P}_{pr}(t \rightarrow \tilde{t})\mathbf{P}_{ac}(t \rightarrow \tilde{t}) \ ,$$

the proposal probability $\mathbf{P}_{pr}(\cdot)$, and the acceptance probability, $\mathbf{P}_{ac}(\cdot)$. Since we propose a new state by randomly selecting a triangle, the proposal probability is independent of the previous state and hence $\mathbf{P}_{pr}(\cdot) = \text{const}$. One possible choice for the acceptance probability satisfying Equation (4.7) is

$$\mathbf{P}_{ac}(t \rightarrow \tilde{t}) = \begin{cases} 1 & \text{, if } \pi(\tilde{t}) > \pi(t) \\ \frac{\pi(\tilde{t})}{\pi(t)} & \text{, otherwise} \end{cases} \quad \text{(see, e.g., [170]).}$$

Our subsequent relaxation step does not allow two points to be initially at the same position. In order to avoid this, we do not allow two points to be positioned into the same triangle. Hence, in order to generate $\pi(T)$ distinct triangles, we generate a Markov chain that contains $\pi(T)$ distinct triangles. The initial point positions are then generated by positioning a single point into the center of each of these triangles. The whole algorithm for generating an initial point distribution using a Metropolis-Hastings type algorithm is summarized in Algorithm 4.1.

Algorithm 4.1 Monte Carlo initialization of point distribution

Input: triangular mesh $\mathcal{M}$ and point density $\rho : V \to \mathbb{R}^+$
Output: point positions $\{x_i\}_{i=1}^{\pi(T)}$

```
1: Randomly select a triangle t.
2: T̃ ← {t}
3: while |T̃| < π(T) do
4:     Randomly select a triangle t̃.
5:     acc ← generateRandNumber(0,1)        ▷ Generate random number from [0, 1].
6:     if acc < P_ac(t → t̃) then
7:         if t̃ ∉ T̃ then
8:             T̃ ← T̃ ∪ {t̃}
9:         end if
10:        t ← t̃
11:    end if
12: end while
13: for i ← 1, π(T) do
14:    Select triangle t ∈ T̃.
15:    x_i ← x(t)
16:    T̃ ← T̃ \ {t}
17: end for
```

4.2.3 Graph Partitioning Based Initialization

While the previous approach to generate an initial point distribution on $\mathcal{M}$ does not take into account the triangle neighborhood of each point, in this section we will introduce a method based on mesh partitioning, that generates the triangle neighborhood prior to positioning a point. The basic idea is to partition mesh $\mathcal{M}$ into $\pi(T)$ equally weighted patches and then place a single point into the center of each patch.

Mesh Partitioning

Instead of partitioning $\mathcal{M}$ directly into $\pi(T)$ triangle patches, we work on $\mathcal{M}$'s dual mesh $\widetilde{\mathcal{M}}$, compute a *vertex partitioning* on $\widetilde{\mathcal{M}}$ and generate the dual of this vertex partitioning, giving a *triangle partitioning* on $\mathcal{M}$. To be more precise, we work on the dual mesh graph,

$\widetilde{G}(\mathcal{M}) = (\widetilde{V}, \widetilde{E})$, since we do not need the faces of $\widetilde{\mathcal{M}}$ (cf. Definition 4.1.5). Thus, we reduce the problem of partitioning the triangular mesh to the problem of *k-way graph partitioning*, which has been widely studied (see, e.g., [92] for references). The k-way graph partitioning problem can be defined as follows.

Definition 4.2.1 (k-way Graph Partitioning [92])**.** Given a graph $G = (V, E)$, partition V into k subsets $V_1, \ldots, V_k$, such that $V_i \cap V_j = \emptyset$, $i \neq j$, $|V_i| \approx |V_j|$, $\forall i, j \in \{1, \ldots, k\}$, $\bigcup_i V_i = V$, and the number of edges of E incident to vertices that belong to different V_i is minimized.

Here, the *minimal edge cut criterion* leads to the generation of rather compact subgraphs, which, when transformed back to the surface, represent circular-like surface patches. This is exactly what we are looking for.

Graph partitioning has applications in many different areas, such as parallel scientific computing, task scheduling, and VLSI design [92]. Even though the k-way graph partitioning problem is known to be NP-complete, there exist many algorithms that find good partitions in a reasonable time.

We apply an efficient method proposed by Karypis and Kumar [92], known as *multilevel k-way partitioning*. Their method is a three-stage process and works as follows. First, the original graph is successively coarsened down to some graph having a small number of vertices. Second, on the coarsest graph a k-way partitioning is computed, which, at this level, can be done easily and directly. Third, the k-way partitioning, which was computed on the coarsest graph, is projected back to the original graph. This is done by successively uncoarsening the graph and the partitioning thereon until the original graph is reached. At each uncoarsening step, a refinement of the k-way partitioning is performed to finally arrive at a good partitioning. This refinement step is the most crucial step in the method. In [92], Karypis and Kumar present two refinement strategies and compare both of these strategies with other existing graph partitioning schemes. Due to the multilevel approach, the run time of their approach is only $\mathcal{O}(|E|)$, where $|E|$ is the number of edges in the original graph.

The k-way graph partitioning problem can be reformulated for graphs with weights given on the vertices as follows:

Definition 4.2.2 (k-way Graph Partitioning with Vertex Weights)**.** Given a graph $G = (V, E)$ and a weight function $w : V \to \mathbb{R}^+$, partition V into k subsets $V_1, \ldots, V_k$, such that $V_i \cap V_j = \emptyset$, $i \neq j$, $\sum_{v' \in V_i} w(v') \approx \sum_{v'' \in V_j} w(v'')$, $\forall i, j \in \{1, \ldots, k\}$, $\bigcup_i V_i = V$, and the number of edges of E incident to vertices that belong to different V_i is minimized.

Fortunately, the multilevel k-way partitioning scheme proposed by Karypis and Kumar also works for *vertex-weighted* graphs. The algorithms proposed in [92] have been implemented in the graph-partitioning software library METIS [91] and can thus be used directly. Note that the algorithm solves the k-way graph partitioning problem only approximately, i.e., it is neither guaranteed that the edge cut is minimal, nor that the partitioning is optimal. In general, however, the edge cut will be small and the partitioning w.r.t. to the weights should be close to optimal.

Algorithm 4.2 Partition mesh into $k = \pi(T)$ equally weighted patches

Input: triangular mesh $\mathcal{M}$ and point density $\rho : V \to \mathbb{R}^+$
Output: triangle partitioning $\{T_i\}_{i=1}^{\pi(T)}$

1: Generate the dual mesh graph $\widetilde{G}(\mathcal{M}) = (\widetilde{V}, \widetilde{E})$ from mesh $\mathcal{M} = (V, E, T)$.
2: Define weight function $w : \widetilde{V} \to \mathbb{R}^+$ using Equation (4.6) by

$$w(\tilde{v}) := \pi(m_T^{-1}(\tilde{v})) \ , \ \forall \tilde{v} \in \widetilde{V}.$$

3: Apply the multilevel k-way partitioning algorithm to $\widetilde{G}(\mathcal{M})$ with weight function $w(\cdot)$, resulting in a partitioning, $\{\widetilde{V}_i\}_{i=1}^k$, of $\widetilde{V}$.
4: Generate a mesh partitioning, $\{T_i\}_{i=1}^k$, of T from the graph partitioning $\{\widetilde{V}_i\}_{i=1}^k$ as follows:

$$T_i := \left\{ t \mid \exists v \in \widetilde{V}_i : m(t) = v \right\} \ , \ \forall i \in \{1, \ldots, k\} \ . \tag{4.8}$$

With Algorithm 4.2 we propose a method to partition a triangular mesh $\mathcal{M}$ into $\pi(T)$ patches $T_1, \ldots, T_{\pi(T)}$. From Definition 4.2.2 and Equation (4.8) it directly follows that $T = \cup_i T_i$, and $T_i \cap T_j = \emptyset$, $i \neq j$. From the fact that the vertex partitions have almost equal weights, it further follows that

$$\sum_{t' \in T_i} \pi(t') \approx \sum_{t'' \in T_j} \pi(t'') \ , \forall i, j \in \{1, \ldots, \pi(T)\} \ ,$$

i.e. the triangle subsets T_i have almost equal weights, too.

Point Positioning

Definition of Patch Center. Let $\{T_i\}_{i=1}^k$ be a partitioning of T obtained by applying Algorithm 4.2 to mesh $\mathcal{M}$. We will now explain how this partitioning is used to position k points $p_1, \ldots, p_k$ on $\mathcal{M}$. As mentioned earlier, we position each point p_i into the center of its corresponding patch $P(T_i)$. The center of some patch $P(\widetilde{T})$ can be defined in several ways. For example, we could define the center of $P(\widetilde{T})$, denoted by $\mathbf{x}(P(\widetilde{T}))$, as the point minimizing the maximum geodesic distance to each point on the patch, i.e.

$$\mathbf{x}(P(\widetilde{T})) := \arg \min_{x \in P(\widetilde{T})} \max_{y \in P(\widetilde{T})} (d_{P(\widetilde{T})}(x, y)) \ .$$

We could also define $\mathbf{x}(P(\widetilde{T}))$ as the point that minimizes the average geodesic distance or the average quadratic geodesic distance, respectively, given by

$$\mathbf{x}(P(\widetilde{T})) := \arg \min_{x \in P(\widetilde{T})} \frac{\int_{P(\widetilde{T})} d_{P(\widetilde{T})}(x, y) \, \mathrm{d}y}{\int_{P(\widetilde{T})} \mathrm{d}y} \ , \text{ and}$$

$$\mathbf{x}(P(\widetilde{T})) := \arg \min_{x \in P(\widetilde{T})} \frac{\int_{P(\widetilde{T})} d^2_{P(\widetilde{T})}(x, y) \, \mathrm{d}y}{\int_{P(\widetilde{T})} \mathrm{d}y} \ . \tag{4.9}$$

If we replace the geodesic distance $d_{P(\widetilde{T})}(\cdot,\cdot)$ in Equation (4.9) by the Euclidean distance $\|\cdot\|$ and drop the constraint that the minimum has to be in $P(\widetilde{T})$, we obtain

$$\mathbf{x}(P(\widetilde{T})) := \arg\min_{x\in\mathbb{R}^3} \frac{\int_{P(\widetilde{T})} \|x-y\|^2 \,\mathrm{d}y}{\int_{P(\widetilde{T})} \mathrm{d}y}$$

which can be solved analytically and gives

$$\begin{aligned}\mathbf{x}(P(\widetilde{T})) &= \frac{\int_{P(\widetilde{T})} y \,\mathrm{d}y}{\int_{P(\widetilde{T})} \mathrm{d}y} \\ &= \frac{\sum_{t\in\widetilde{T}} A(t)\,\mathbf{x}(t)}{A(\widetilde{T})} \\ &= \mathbf{x}(\widetilde{T}) \ .\end{aligned}$$

In general, the center of mass of some patch $P(\widetilde{T})$ does not lie on $P(\widetilde{T})$, hence, we project it back onto $\mathcal{M}$ in the direction of $\mathbf{n}(\widetilde{T})$ or $-\mathbf{n}(\widetilde{T})$. Thus, we finally redefine $\mathbf{x}(P(\widetilde{T}))$ by

$$\mathbf{x}(P(\widetilde{T})) := \mathbf{x}(\widetilde{T}) + l\,\mathbf{n}(\widetilde{T}) \ , \tag{4.10}$$

where $l \in \mathbb{R}$ and $|l|$ is the shortest Euclidean distance to $P(T)$ along $\mathbf{n}(\widetilde{T})$, with $l > 0$, or $-\mathbf{n}(\widetilde{T})$, with $l < 0$.

If, in addition, a weight function $\rho : V \to \mathbb{R}^+$ is defined on the vertices V of $\mathcal{M}$, then Equation (4.10) can be generalized, giving the *weighted center* of $P(\widetilde{T})$, by

$$\mathbf{x}(P(\widetilde{T}),\rho) := \mathbf{x}(\widetilde{T},\rho) + l\,\mathbf{n}(\widetilde{T},\rho) \ . \tag{4.11}$$

Projection. The intersection of the rays starting from $\mathbf{x}(\widetilde{T},\rho)$ in either direction $\mathbf{n}(\widetilde{T},\rho)$ or $-\mathbf{n}(\widetilde{T},\rho)$ can be efficiently computed using a *triangle octree* (see, e.g., [152]). Octrees constitute a generalization of *quadtrees* [39] to three dimensions. Quadtrees and octrees represent hierarchical subdivisions of some subset of $\mathbb{R}^2$ and $\mathbb{R}^3$, respectively, and are used to efficiently store and retrieve data objects residing in this subset.

A quadtree (cf. Figure 4.5) is a rooted tree in which every internal node has four children. Each node of the quadtree corresponds to a square. The four children of an internal node $\mathfrak{n}$ correspond to the four quadrants of the square corresponding to node $\mathfrak{n}$. Since a square is always subdivided into four quadrants, the tree representing this subdivision is called quadtree. A square will be subdivided if its corresponding leaf contains more elements than a predefined threshold.

We use an octree to store all triangles of mesh $\mathcal{M}$. To generate the octree we compute the smallest cube enclosing the bounding box of the triangular mesh. The octree representing this cube is a single node, the root node. During insertion of all triangles of the triangular mesh, the cube is successively subdivided into sub-cubes such that no leaf representing a sub-cube contains more than a previously specified number of triangles. A triangle is added to a leaf if its corresponding sub-cube intersects the triangle. This data

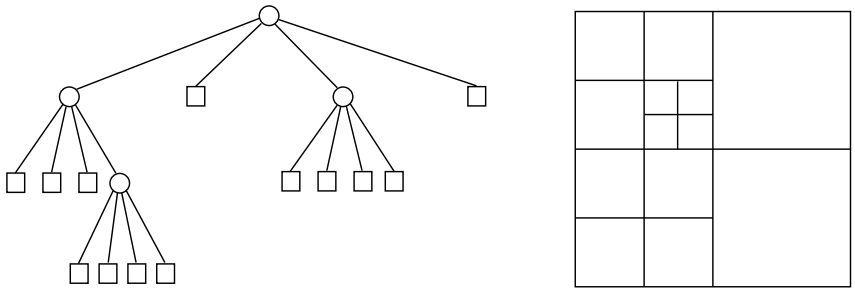


Figure 4.5: A quadtree and the corresponding subdivision of a square. The children of some node $\mathfrak{v}$ in the tree represent the quadrants of the square corresponding to $\mathfrak{v}$ in left to right and top to bottom order.

structure allows us to quickly identify those triangles that need to be checked for intersection. Instead of intersecting each ray from $\mathbf{x}(\widetilde{T}, \rho)$ directly with the triangles of $\mathcal{M}$, the ray is first intersected with the cubes represented by the octree. If an intersection with a cube occurs, the elements contained in this cube are potential candidates for intersection. The intersection with the cubes is done in a hierarchical fashion, i.e. we first intersect the ray with the cube represented by the root node of the octree. If this cube is intersected, its children will be tested and so forth. Finally, we will arrive at a cube that is not subdivided and hence, we need to check its elements, i.e. its triangles, for intersection with the ray.

For each $\mathbf{x}(T_i, \rho)$ we cast two rays, one into direction $\mathbf{n}(T_i, \rho)$, the other one into direction $-\mathbf{n}(T_i, \rho)$. For each ray we take the closest intersection point as feasible point for $\mathbf{x}(P(T_i), \rho)$. In case we obtain two feasible points, we take the one that is closer to $\mathbf{x}(T_i, \rho)$.

With the above considerations we can now formulate Algorithm 4.3, which generates an initial point distribution from a weighted mesh partitioning $\{T_i\}_{i=1}^k$.

Algorithm 4.3 Generate initial point positions from mesh partitioning

Input: triangle partitioning $\{T_i\}_{i=1}^k$ and weight function $\rho : V \to \mathbb{R}^+$
Output: $\mathbf{x}(p_1), \ldots, \mathbf{x}(p_k)$

1: **for** $i \in \{1, \ldots, k\}$ **do**
2: Compute $\mathbf{x}(T_i, \rho)$ and $\mathbf{n}(T_i, \rho)$.
3: Intersect rays from $\mathbf{x}(T_i, \rho)$ with directions $\mathbf{n}(T_i, \rho)$ and $-\mathbf{n}(T_i, \rho)$ with $P(T)$.
4: Set $\mathbf{x}(p_i) \leftarrow \mathbf{x}(P(T_i), \rho)$.
5: **end for**

4.3 Point Relaxation

In this section we present a new algorithm to improve the distribution of points, given on a 2-manifold triangular mesh, by relaxing the point positions w.r.t. to the neighborhood

of each point. The aim is to obtain a point distribution which should be as regular as possible w.r.t. a point density given on the triangular mesh.

Point relaxation is done using *centroidal Voronoi tesselation* (CVT), which we extend to 2-manifold triangular meshes. In Section 4.3.1 we give the definition of centroidal Voronoi tesselation and one general result, which explains, why CVT can be used for point relaxation. Finally, we give an algorithm for Lloyd's method. In Section 4.3.2 we then extend Lloyd's method to triangular meshes. Since the computation of a Voronoi tesselation requires the determination of geodesic distances on the surface, we first describe how the geodesic distances between points on the triangular mesh can be approximated using the graphs that were introduced in Section 4.1. This distance approximation then directly allows us to give an algorithm for the computation of an discrete approximate CVT on triangular meshes.

4.3.1 Centroidal Voronoi Tesselation

For the introduction of centroidal Voronoi tesselation (CVT) we largely follow the article of Du, Faber and Gunzburger [51]. For a detailed discussion on algorithms and application based on CVT, please see their article.

Definition 4.3.1 (Voronoi Tesselation [51])**.** Given some set $\Omega \subseteq \mathbb{R}^N$ and a finite set of points $X := \{x_1, \ldots, x_k\} \subseteq \Omega$, the set $\{\mathcal{V}_i\}_{i=1}^k$ is called a *Voronoi tesselation* of Ω, if $\mathcal{V}_i \cap \mathcal{V}_j = \emptyset$, $\forall i, j \in \{1, \ldots, k\}, i \neq j$, and $\bigcup_{i=1}^k \overline{\mathcal{V}}_i = \Omega$, where the *Voronoi region* $\mathcal{V}_i$ is defined as

$$\mathcal{V}_i := \left\{ x \in \Omega \mid d_\Omega(x, x_i) < d_\Omega(x, x_j), \forall j \in \{1, \ldots, k\}, j \neq i \right\} , \tag{4.12}$$

and where $d_\Omega(\cdot, \cdot)$ denotes some distance metric on Ω. The elements of X are called the *generators* of the Voronoi tesselation.

Definition 4.3.2 (Centroidal Voronoi Tesselation)**.** Given some set $\Omega \subseteq \mathbb{R}^N$, a finite set of points $X := \{x_1, \ldots, x_k\} \subseteq \Omega$, and a density function $\rho : \Omega \to \mathbb{R}^+$, the set $\{\mathcal{V}_i\}_{i=1}^k$ is called a *centroidal Voronoi tesselation* of Ω, if $\{\mathcal{V}_i\}_{i=1}^k$ is a Voronoi tesselation of Ω with the property that its generators coincide with the weighted centers of mass of the Voronoi regions, i.e.

$$x_i = \mathbf{x}(\mathcal{V}_i, \rho) := \frac{\int_{\mathcal{V}_i} x\rho(x)\,\mathrm{d}x}{\int_{\mathcal{V}_i} \rho(x)\,\mathrm{d}x} , \forall i \in \{1, \ldots, k\} . \tag{4.13}$$

If we consider a *discrete* set of points $W = \{y_i\}_{i=1}^n \subset \mathbb{R}^N$ instead of a continuous region Ω, we need to redefine Equation (4.12) slightly to

$$\begin{aligned} \mathcal{V}_i := \Big\{ x \in W \mid d_W(x, x_i) \leq d_W(x, x_j), \forall j \in \{1, \ldots, k\}, j \neq i, \\ \text{where equality is only allowed for } i < j \Big\} . \end{aligned} \tag{4.14}$$

Discrete centroidal Voronoi tesselations are related to optimal *k-means clusters*, which are used, e.g., in the discrete and vector quantization community [51]. There, Voronoi regions and their mass centroids are referred to as *clusters* and *cluster centers*, respectively.

In [51], Du et al. give properties and proofs concerning centroidal Voronoi tesselations. One important property is, that CVT can be used for minimization. This is formulated in the following theorem.

Theorem 4.3.3 (Minimization [51])**.** Given $\Omega \subset \mathbb{R}^N$, a positive integer k, and a density function $\rho(\cdot)$ defined on Ω, let $\{x_i\}_{i=1}^k$ denote any set of k points belonging to $\bar{\Omega}$ and let $\{\mathcal{V}_i\}_{i=1}^k$ denote any tesselation of Ω into k regions. Let

$$\mathcal{F}\Big((x_i, \mathcal{V}_i), i = 1, \ldots, k\Big) = \sum_{i=1}^{k} \int_{\mathcal{V}_i} \rho(x) \|x - x_i\|^2 \, \mathrm{d}x \ .$$

Then, a necessary condition for $\mathcal{F}$ to be minimized is that the $\mathcal{V}_i$'s are the Voronoi regions corresponding to the x_i's (in the sense of Equation (4.12)) and, simultaneously, the x_i's are the centroids of the corresponding $\mathcal{V}_i$'s (in the sense of Equation (4.13)).

Proof. See Du et al. [51], Section 3.1, page 650. □

In the discrete case, the functional being minimized is given by

$$\mathcal{F}\Big((x_i, \mathcal{V}_i), i = 1, \ldots, k\Big) = \sum_{i=1}^{k} \sum_{x \in \mathcal{V}_i} \rho(x) \|x - x_i\|^2 \ .$$

Lloyd's Method

Lloyd [113] proposed an iterative method to construct a centroidal Voronoi tesselation which he applied to vector quantization. Each iteration consists of two steps. In the first step, the Voronoi tesselation of a set of points is computed. In the second step, the positions of these points are moved to the centers of the Voronoi regions computed in the first step. These two steps are repeated until conversion is reached. It has been shown (see, e.g., [51]), that Lloyd's method locally converges, at least for the one-dimensional case. However, even though no global optimization is guaranteed, Lloyd's method has been successfully applied in many applications (see [51] for references), in particular for distributing points in $\mathbb{R}^2$ [46, 79, 121]. His method is given by Algorithm 4.4.

4.3.2 Centroidal Voronoi Tesselation on Triangular Meshes

We now want to apply Lloyd's method on triangular meshes to locally optimize the point positions given by an initial point distribution. In order to apply Lloyd's method on triangular meshes, essentially we need to solve the problem of computing the Voronoi tesselation of $\mathcal{M}$ for a given set of points on $\mathcal{M}$. For each Voronoi region we can then apply Algorithm 4.3 to obtain the new point positions.

Approximate Discrete Voronoi Tesselation

We have implemented three methods to compute an approximation of the Voronoi tesselation of the triangular mesh $\mathcal{M} = (V, E, T)$ with respect to a given set of points

Algorithm 4.4 Lloyd's method to distribute k points $p_1, \ldots, p_k$ on Ω

Input: some set $\Omega \subseteq \mathbb{R}^N$, number of points k, and point density $\rho : \Omega \to \mathbb{R}^+$
Output: relaxed point positions $\mathbf{x}(p_1), \ldots, \mathbf{x}(p_k)$

1: Generate initial positions $x_i, \ldots, x_k$.
2: **repeat**
3: Construct the Voronoi tesselation $\{\mathcal{V}_i\}_{i=1}^k$ of Ω w.r.t. $\{x_i\}_{i=1}^k$.
4: Compute the centers of mass $\mathbf{x}(\mathcal{V}_i, \rho)$ of all Voronoi regions $\mathcal{V}_i$.
5: $x_i \leftarrow \mathbf{x}(\mathcal{V}_i, \rho)$
6: **until** $\mathbf{x}(\mathcal{V}_i, \rho) \approx x_i$
7: $\mathbf{x}(p_i) \leftarrow \mathbf{x}(\mathcal{V}_i, \rho)$

$P = \{p_1, \ldots, p_k\}$ residing on $\mathcal{M}$. All three methods compute Voronoi tesselations of discrete subsets of $\mathcal{M}$. While the first method computes a Voronoi tesselation of the vertices V of $\mathcal{M}$, the other two methods compute a Voronoi tesselation of the triangles T of $\mathcal{M}$, where each triangle is represented by the triangle's mass centroid. Instead of computing a triangle tesselation directly, we work on $\mathcal{M}$'s dual mesh $\widetilde{\mathcal{M}}$, compute a vertex tesselation of $\widetilde{\mathcal{M}}$ and transform this tesselation back to $\mathcal{M}$ to obtain a triangle tesselation. To compute vertex tesselations of $\mathcal{M}$ and $\widetilde{\mathcal{M}}$, we use the mesh graphs defined in Section 4.1.2. In order to assign the vertices of the respective graphs to the points of P, we compute the shortest paths from each p_i to the vertices in the local neighborhood of p_i. We have implemented two versions to compute the shortest paths: a breadth first search algorithm and a Dijkstra multiple-source-multiple-destination algorithm.

Definition 4.3.4 (P-Extended Mesh Graph)**.** Let $G = (V, E)$ be the graph representing any of the mesh graphs. Let further denote by $t(p) \in T$ the triangle on which the point p resides. Since the point set P is not a subset of V, we need to extend G to the *P-extended mesh graph* $G' = (V', E')$ as follows:

1. $V' = V \cup P$
2. $E' = E \cup S$, where S depends on whether the used mesh graph is the *mesh graph* $G(\mathcal{M})$, the *dual mesh graph* $\widetilde{G}(\mathcal{M})$, or the *extended dual mesh graph* $\widetilde{G}^*(\mathcal{M})$, i.e.,

$$S := \begin{cases} \bigcup_{i=1}^k \big\{(p_i, v) \mid \exists j \in \{1,2,3\} : v = v_j(t(p_i))\big\} & \text{, if } G = G(\mathcal{M}) \\ \bigcup_{i=1}^k \big\{(p_i, v) \mid m_T^{-1}(v) \in N(t(p_i))\big\} & \text{, if } G = \widetilde{G}(\mathcal{M}) \\ \bigcup_{i=1}^k \big\{(p_i, v) \mid m_T^{-1}(v) \in N^*(t(p_i))\big\} & \text{, if } G = \widetilde{G}^*(\mathcal{M}) \end{cases} \tag{4.15}$$

Let $\mathbf{X}(P) = \{\mathbf{x}(p_1), \ldots, \mathbf{x}(p_k)\}$ be the coordinates of points $p_1, \ldots, p_k$. Then, the function $l : E' \to \mathbb{R}^+$, defined as

$$l\big((v', v'')\big) := \begin{cases} \|\mathbf{x}(v') - \mathbf{x}(v'')\| & \text{, if } v', v'' \in V(\mathcal{M}) \vee v' \in P \wedge v'' \in V(\mathcal{M}) \\ \|\mathbf{x}(v') - \mathbf{x}(t'')\| & \text{, if } v' \in P \wedge v'' \in V(\widetilde{\mathcal{M}}) : t'' = m_T^{-1}(v'') \\ \|\mathbf{x}(t') - \mathbf{x}(t'')\| & \text{, if } v', v'' \in V(\widetilde{\mathcal{M}}) : t' = m_T^{-1}(v'), t'' = m_T^{-1}(v'') \end{cases} \tag{4.16}$$

gives the weights of the edges of E' depending on the used mesh graph. That is to say, the edge weights are the Euclidean distances between the coordinates of points $p \in P$, vertices $v \in V(\mathcal{M})$ and the centers of mass of triangles $t \in T$, respectively. Only in the case of mesh graph $G(\mathcal{M})$ are the Euclidean distances between neighbored vertices equal to the geodesic distances.

Let $G' = (V' = V \cup P, E' = E \cup S)$, with S being defined as in Equation (4.15) and let $l : E' \to \mathbb{R}^+$ be the weight function defined by Equation (4.16). We can then define the discrete Voronoi tesselation, $\{\mathcal{V}_i\}_{i=1}^k$, of V' (cf. Equation (4.14)) for the set of generators $\{p_i\}_{i=1}^k$ as

$$\mathcal{V}_i := \Big\{ v \in V' \mid d_{G',l}(v, p_i) \leq d_{G',l}(v, p_j), \forall j \in \{1, \ldots, k\}, j \neq i, \text{ where equality holds only for } i < j \Big\}, \tag{4.17}$$

where $d_{G',l}(u, v)$, $u, v \in V'$, is the length of the shortest l-weighted path from u to v on G'.

Algorithm 4.5 Breadth first search (BFS) on G'

Input: graph $G' = (V' = V \cup P, E')$ and weight function $l : E' \to \mathbb{R}^+$
Output: Voronoi tesselation $\{\mathcal{V}_i\}_{i=1}^k$

```
1: Initialize queue Q ← P.
2: For each v ∈ V: dist(v) ← ∞, voronoi(v) ← −1
3: For each p_i ∈ P: dist(p_i) ← 0, voronoi(p_i) ← i
4: while Q ≠ ∅ do
5:     q ← Q.pop()
6:     for (q, v) ∈ E' do
7:         if dist(v) > dist(q) + l((q, v)) then      ▷ A shorter path to v has been found.
8:             dist(v) ← dist(q) + l((q, v))
9:             voronoi(v) ← voronoi(q)
10:            if v ∉ Q then
11:                Q.append(v)
12:            end if
13:        end if
14:    end for
15: end while
16: for v ∈ V' do
17:    V_voronoi(v) ← V_voronoi(v) ∪ {v}
18: end for
```

It is easy to see that Algorithms 4.5 and 4.6 both produce the desired Voronoi tesselation of Equation (4.17), since both algorithms compute the locally shortest paths.

If we implement Algorithm 4.6 using a *Fibonacci heap* as min-priority queue, its amortized run time is $\mathcal{O}(|V| \cdot \lg|V| + |E|)$ (see, e.g., [34]). The min-priority queue used in

Algorithm 4.6 Dijkstra search on G' (using min-priority queue)

Input: graph $G' = (V' = V \cup P, E')$ and weight function $l : E' \to \mathbb{R}^+$
Output: Voronoi tesselation $\{\mathcal{V}_i\}_{i=1}^k$

```
1: Initialize queue Q ← ∅.
2: For each v ∈ V: dist(v) ← ∞, voronoi(v) ← −1
3: For each p_i ∈ P: dist(p_i) ← 0, voronoi(p_i) ← i
4: for v ∈ V do
5:     Q.insert(v, dist(v))
6: end for
7: while Q ≠ ∅ do
8:     q ← Q.popMin()
9:     for (q, v) ∈ E' do
10:        if dist(v) > dist(q) + l((q, v)) then        ▷ Shorter path to v has been found.
11:            Q.relax(v, dist(q) + l((q, v)))
12:            voronoi(v) ← voronoi(q)
13:        end if
14:    end for
15: end while
16: for v ∈ V' do
17:    V_voronoi(v) ← V_voronoi(v) ∪ {v}
18: end for
```

Algorithm 4.6 guarantees that each vertex will be inserted into the queue only once, since, when removing the vertex with the minimal distance, we know that the shortest distance to that vertex has already been computed. This is due to the fact, that the edge weights of the graphs are strictly positive. Hence, if there existed a shorter path to the considered vertex, then there would be some vertex with shorter distance still in the priority queue, which contradicts our assumption that we are considering the vertex with shortest distance still in the queue.

In theory, Algorithm 4.5 performs worse than Algorithm 4.6, since vertices might be added to the queue more than once. The reason for this lies in the vertices being taken in breadth first search order and not according to their currently shortest distance.

In our application, however, we found that the BFS algorithm works better than Dijkstra's algorithm. This is due to the fact, that the edge weights do not differ greatly, and, hence, the order in which the vertices are taken from the *first-in-first-out* (FIFO) queue does not differ much from the order specified by a min-priority queue. Hence, the number of insertions is almost equal for both algorithms. The amortized cost of a single "getMin()" operation using a Fibonacci heap is $\mathcal{O}(\lg|V|)$ (cf. [34]) in contrast to the constant time needed for removing an element from a FIFO queue. Hence, in practice, this higher cost only equalizes if the number of elements to be considered using a FIFO queue is much larger then $|V|$.

From the Voronoi tesselation of Equation (4.17) computed with either Algorithm 4.5 or

4.6 we can directly compute a vertex partitioning $\{V_i\}_{i=1}^k$ or a triangle partitioning $\{T_i\}_{i=1}^k$ of $\mathcal{M}$, depending on the mesh graph used. This partitioning serves as approximate Voronoi tesselation of $\mathcal{M}$ for Lloyd's method.

Lloyd's Method

Let $\{W_i\}_{i=1}^k$ be a partitioning of $\mathcal{M} = (V, E, T)$ – vertex or triangle partitioning – transformed back from the Voronoi tesselation $\{\mathcal{V}_i\}_{i=1}^k$ generated by applying either Algorithm 4.5 or 4.6. Let $\rho : W \to \mathbb{R}^+$ be a weight function defined on W. Then we define the *weighted center* of $\mathcal{V}_i$ as

$$\mathbf{x}(\mathcal{V}_i, \rho) := \mathbf{x}(P(W_i), \rho) \ , \tag{4.18}$$

where $\mathbf{x}(P(\widetilde{V}), \rho)$, $\widetilde{V} \subseteq V$, is defined similarly to $\mathbf{x}(P(\widetilde{T}), \rho)$, $\widetilde{T} \subseteq T$ (cf. Equation (4.11)) as

$$\mathbf{x}(P(\widetilde{V}), \rho) := \mathbf{x}(\widetilde{V}, \rho) + l\, \mathbf{n}(\widetilde{V}, \rho) \ . \tag{4.19}$$

We can now specify the whole algorithm for distributing k points on the triangular mesh $\mathcal{M}$ according to some weight function $\rho(\cdot)$, which is given by Algorithm 4.7.

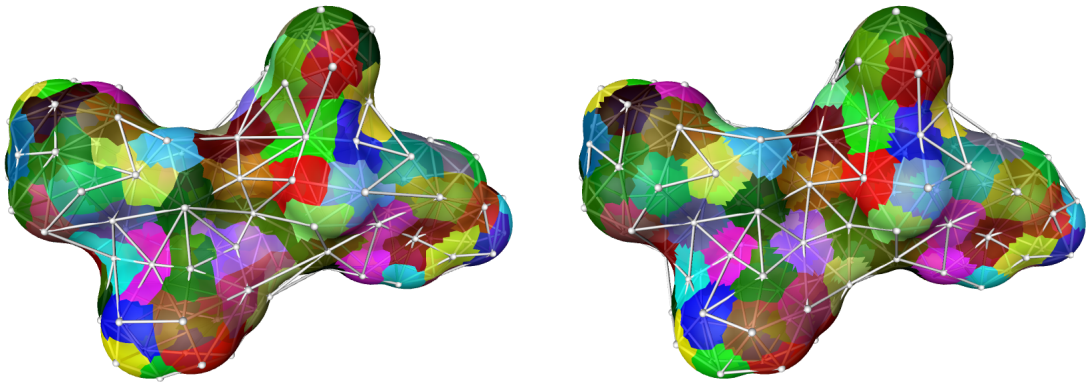
Algorithm 4.7 Lloyd's method to distribute points $\mathcal{M}$

Input: triangular mesh $\mathcal{M} = (V, E, T)$, and point density $\rho : V \to \mathbb{R}^+$
Output: relaxed positions $\mathbf{x}(p_1), \ldots, \mathbf{x}(p_k)$ of point set $P = \{p_1, \ldots, p_k\}$

1: Compute initial triangle partitioning, $\{T_i\}_{i=1}^k$. ▷ Algorithm 4.2
2: Generate initial positions x_i of points p_i from $\{T_i\}_{i=1}^k$. ▷ Algorithm 4.3
3: Construct P-extended mesh graph $G' = (V', E')$. ▷ Definition 4.3.4
4: **repeat**
5: Compute Voronoi tesselation, $\{\mathcal{V}_i\}_{i=1}^k$, of G' w.r.t. $\{x_i\}_{i=1}^k$. ▷ Algorithm 4.5
6: Compute the weighted centers, $\mathbf{x}(\mathcal{V}_i, \rho)$, of all Voronoi regions $\mathcal{V}_i$. ▷ Equation 4.18
7: **until** $\mathbf{x}(\mathcal{V}_i, \rho) \approx x_i$, $\forall i = 1, \ldots, k$
8: $\mathbf{x}(p_i) \leftarrow \mathbf{x}(\mathcal{V}_i, \rho)$

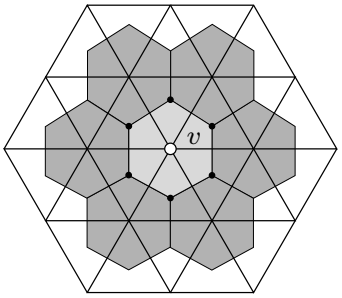
4.4 Experimental Results

We have investigated the quality of the proposed algorithms in several scenarios. For the investigation of specific parts of the overall algorithm, we fixed the other parts and varied only the respective sub-algorithm.



(a) (b)

Figure 4.6: Point neighborhood defined by the neighborhood of the triangle patches. The SES is shown semi-transparent so as to allow to better view the point neighborhood. Note that the point neighborhood partially changes during the relaxation process. (a) Point neighborhood due to initial mesh partitioning. (b) Point neighborhood due to centroidal Voronoi tesselation.



Since it is difficult to directly measure the local point density if the distance between neighbored points is rather large, we measure the local point density by measuring the distance between neighbored points. We assume a point distribution that positions the points according to a perfect equilateral triangulation in 2-dimensional space. This triangulation has the property, that six triangles are incident to each point (see image on the left). If we assign to each point v one third of the area of each triangle incident to v, then the overall area assigned to v is two times the size of one of the equilateral triangles. Let $A_\triangle$ denote the area of one such equilateral triangle. Then the point density ρ is given by

$$\rho = \frac{1}{2 \cdot A_\triangle} \ .$$

Given an equilateral triangulation, the distance D between neighbored points is equal to the length of the edges of the triangles. Since, in addition, the triangle area of an equilateral triangle is given by

$$A_\triangle = \frac{\sqrt{3}}{4} \cdot D^2 \ ,$$

the distance D between neighbored points can be computed by

$$D = \sqrt{\frac{2}{\sqrt{3} \cdot \rho}} \ . \tag{4.20}$$

So far, we have used the term *point neighborhood* without giving a clear definition for it. However, a definition of the point neighborhood is straight forward using the

neighborhood of the triangle mesh tesselation. So we define two points p and q, positioned on the triangular mesh, to be neighbored, if their corresponding triangle patches are neighbored (cf. Figure 4.6).

To illustrate the point distribution process and to analyze the quality of our algorithms, we carried out experiments using the HIV-1 protease inhibitor *amprenavir* (APV). The active conformer of APV was extracted from the protease-inhibitor complex (1HPV) from the Protein Data Bank (PDB) [1]. The molecule was parametrized with the Merck Molecular Force Field (MMFF) [73], using the program ZIBMOL [58] developed at the Zuse Institute Berlin. With the program ZIBMOL, we also generated 63 conformers of APV. These were used to analyze the point neighborhood generated during the point relaxation process. For each conformer we ran the point distribution algorithm, computed the mean distance of all neighbored points and the variance of the mean distance. These values, i.e. the mean distance and the variance, were then averaged over all 63 conformers.

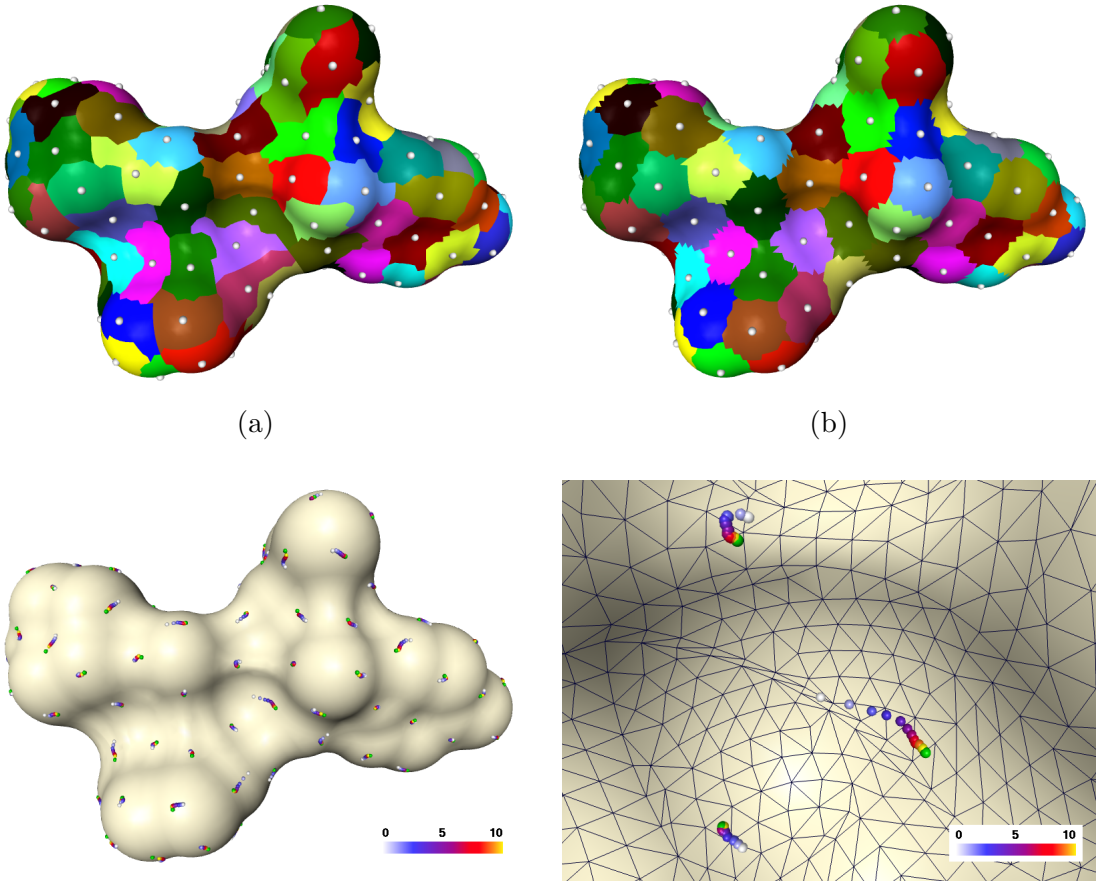
To analyze the distances between neighbored points, we computed the exact geodesic distances on the triangular mesh using an algorithm implemented in the visualization and analysis tool Amira [2]. This algorithm computes paths not along predefined edges but across triangles where at each triangle border the direction of the path might change. Computing the exact geodesic distances is much too expensive to be applied in the point relaxation step, but here we used it for analyzing the quality of the generated point distribution.

For the computation of the triangular meshes representing the solvent excluded surface (SES), an algorithm implemented in the molecular visualization and analysis software AmiraMol [3] was used (cf. Section 2.3.2). In all computations of the SES, the hydrogen atoms were omitted. The run times given in this section refer to a 3GHz Intel Xeon processor.

Even though our algorithms have been designed to handle non-homogeneous point distributions, in this section we have decided to investigate the results for homogeneous and non-homogeneous point distributions separately. We will start with the homogeneous case and compare the results using different sub-algorithms proposed in the previous section. For the non-homogeneous case, we will then only show that the quality of the gained distributions is comparable to the homogeneous case.

4.4.1 Homogeneous Point Distributions

Figure 4.7 shows a series of images illustrating the point distribution process for a homogeneous point density $\rho = 0.289\,\text{Å}^{-2}$ which corresponds to a distance between neighbored points of 2.0 Å (cf. Equation (4.20)). The point density of the triangular mesh was $20\,\text{Å}^{-2}$. For the initialization of the point distribution we used the graph partitioning approach (cf. Algorithm 4.3). The point relaxation was done using Algorithm 4.7 operating on the extended dual mesh graph (cf. Definition 4.1.6).



(a) (b)

(c) (d)

Figure 4.7: Uniform point distribution of 125 points on the SES of the active conformer of amprenavir with an approximate geodesic distance of 2.0 Å between neighbored points. (a) The initial point distribution was achieved using the graph partitioning approach. Shown is the initial triangle partitioning together with its resulting initial point positioning. (b) Final point positioning with corresponding centroidal Voronoi tesselation of the triangular mesh. (c) Point trajectories showing the intermediate positions up to iteration 10 for the point distribution shown in (b). The green spheres represent the final positions. (d) Close-up of the center of the image on the left side with the surface triangles displayed in outlined mode.

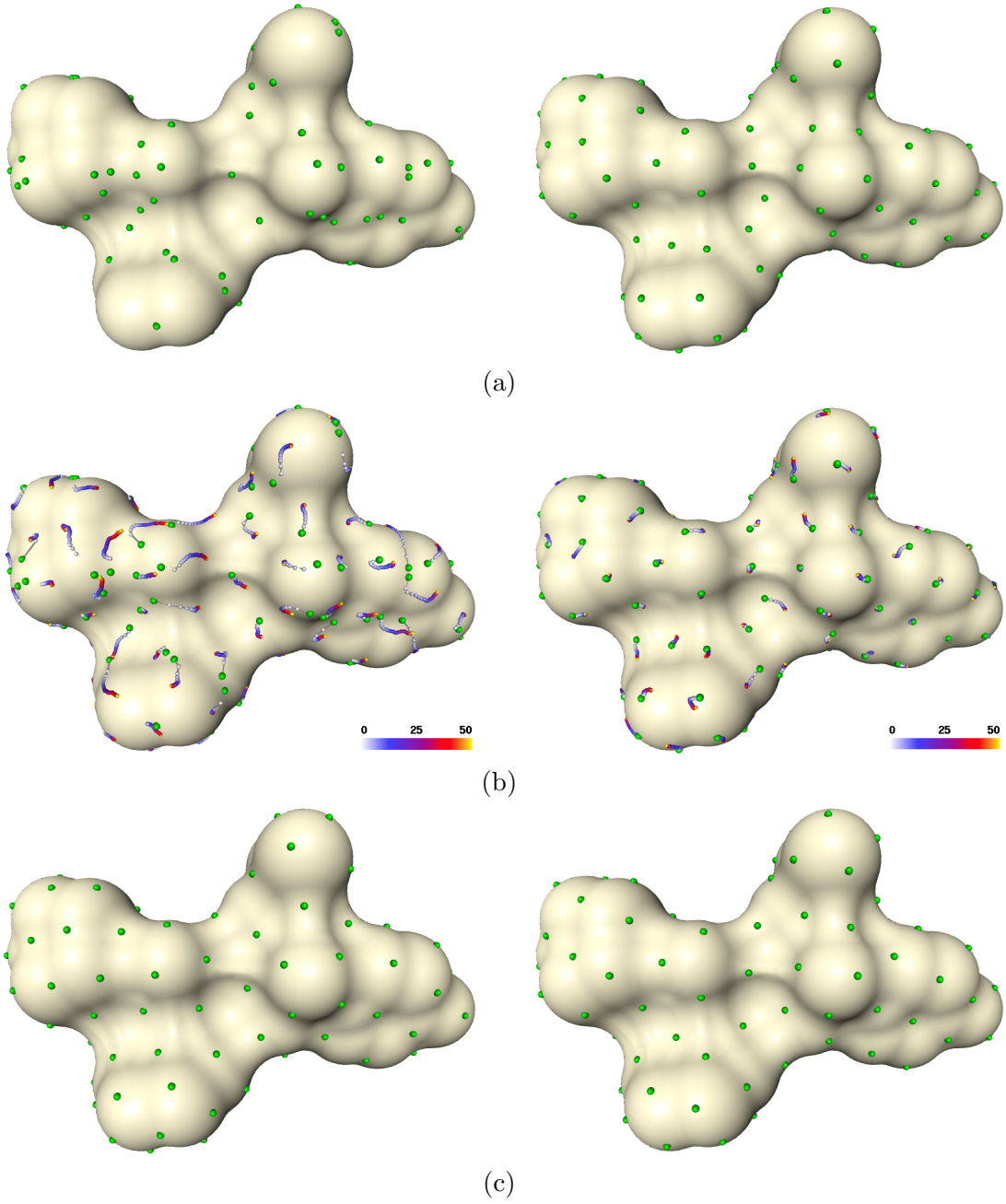


Figure 4.8: This series of images shows the iterative point positioning process on the SES of the active conformer of amprenavir. 125 points were uniformly distributed on the surface with an approximate geodesic distance of 2.0 Å between neighbored points. The images on the left hand side show results when starting with Monte Carlo initialization. The images on the right hand side represent the process when starting with graph partitioning based initialization. (a) Initial point positioning. (b) Trajectories of the points from initial to final positions after 50 iterations. The spheres are colored according to their corresponding iteration step. The color mapping is done using the colormap shown. The green spheres denote the initial positions as shown in (a). (c) Final point positions after 50 iterations.

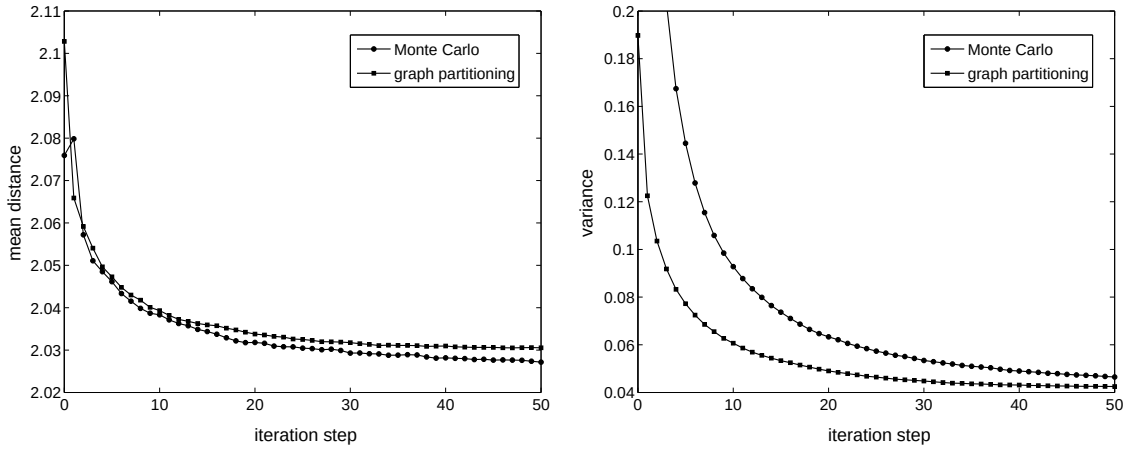


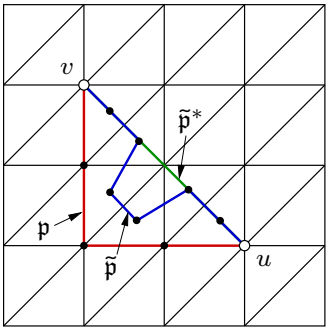
Figure 4.9: Comparison of point distribution process using Monte Carlo initialization and graph partitioning based initialization. The triangular mesh resolution was $20\,\text{Å}^{-2}$. The desired point distance was 2.0 Å. *Left:* Evolution of the mean distance. *Right:* Evolution of the variance of the point distance.

Initial Point Distributions

In Section 4.2 we proposed two methods to generate an initial point distribution. In the following, we want to investigate to what extend the choice of initialization method influences the final point distribution and the convergence behavior of the relaxation step. Figure 4.8 shows a comparison between the two methods. In both cases the same triangular mesh with a point density of $20\,\text{Å}^{-2}$ was used, and the same number of points was to be distributed on the surface according to a point density given by a desired neighbored point distance of 2.0 Å (cf. Equation 4.20). It can clearly be seen that the initial point distribution generated with the Monte Carlo approach is much worse than the initial point distribution generated by the graph partitioning approach (cf. Figure 4.8 (a)), which already looks rather homogeneous. Due to the unfavorable initial point distribution by the Monte Carlo approach, which, in the case of a homogeneous point distribution, is equal to a pure random sampling, the points need to cover a much larger distance during the relaxation step, apparent in the long point trajectories (cf. Figure 4.8 (b)). By visual inspection, the qualities of the point distributions after 50 iterations are similar for both initial distributions. However, using the graph partitioning approach, a satisfying point distribution is achieved much earlier. This is confirmed by Figure 4.9, which clearly shows that the variance of the mean distance of all neighbored points using graph partitioning is much smaller at the beginning and reaches convergence much earlier than using the Monte Carlo approach. Interestingly, the mean distance is slightly smaller using the Monte Carlo approach. One reason for this might be the different point neighborhoods that are generated by both approaches. If the point distribution is less regular, the number of neighbored points will, in general, be larger. Hence, we also have more shorter distances.

Point Relaxation Using Different Mesh Graphs

In this section we study the effect of the type of mesh graph used in the relaxation process. Let us recall that the mesh graphs are used to approximate the geodesic distances on the triangular mesh. Thus, the use of different mesh graphs influences the size and shape of the approximated Voronoi regions. This affects the quality of the final point distribution. To compare the point distributions gained by using different mesh graphs, we carried out two experiments. First, we generated point distributions with a desired point distance of 2.0 Å on triangular meshes with a point density of $20\,\text{Å}^{-2}$. Second, on triangular meshes with a point density of $40\,\text{Å}^{-2}$ we generated point distributions with a desired point distance of 1.0 Å.



The results of the two experiments are shown in Figures 4.10 and 4.11. For both experiments, it can clearly be observed that the extended dual mesh graph performs best, both in the length of the mean distance and in the variance. Second best performs the dual mesh graph and worst is the mesh graph. This effect is, at least partially, due to the triangulation. Consider a triangulation similar to the one seen in the left image and compare the shortest paths, $\mathfrak{p}$, $\tilde{\mathfrak{p}}$ and $\tilde{\mathfrak{p}}^*$, from u to v on the mesh graph $G(\mathcal{M})$, on the dual mesh graph $\widetilde{G}(\mathcal{M})$ and on the extended dual mesh graph $\widetilde{G}^*(\mathcal{M})$, respectively. In this case, $\mathfrak{p}$ is $\sqrt{2}$ times as long as $\tilde{\mathfrak{p}}^*$. Even though the presented scenario is the worst case, it reveals the problems with both the mesh and the dual mesh graph.

Figures 4.10 and 4.11 show that the point relaxation performs best using the extended dual mesh graph. However, we might ask the following question: Does point relaxation with, e.g., the dual mesh graph on a finer triangular mesh work better than point relaxation with the extended dual mesh graph on a coarser triangular mesh? We therefore carried out the following experiment. We compared the point relaxation process using the dual mesh graph with a mesh resolution of $40\,\text{Å}^{-2}$ with the point relaxation using the extended dual mesh graph with a mesh resolution of $20\,\text{Å}^{-2}$. The results are shown in Figure 4.12. The plots show, that even when using a finer resolution for the dual mesh graph approach, the extended dual mesh graph performs better both in convergence of the mean distance and the variance of the point distance. The section on run times will also show that, using these triangular mesh resolutions, point relaxation with the extended dual mesh graph is faster than with the dual mesh graph. Hence, the extended dual mesh graph should be preferred to both the dual mesh graph and the mesh graph, since the mesh graph performs even worse than the dual mesh graph.

Surface Resolution

A better resolution of the triangular mesh should yield a better approximation of the Voronoi regions on the triangular mesh, which should then lead to a better point distribution. This assumption could be confirmed with our experiments. Figure 4.13 shows that a point resolution of $40\,\text{Å}^{-2}$ leads to a better convergence towards the desired point

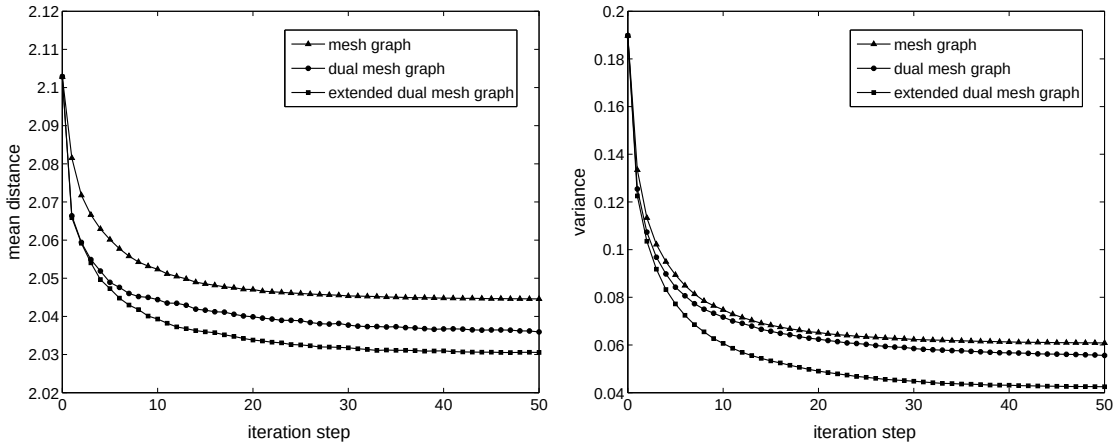


Figure 4.10: Comparison of point relaxation based on different mesh graphs. All three relaxation processes started from the same initial point distribution generated with the graph partitioning approach. The point density of the triangular mesh was $20\,\text{Å}^{-2}$. The desired point distance was 2.0 Å. *Left:* Evolution of the mean distance. *Right:* Evolution of the variance of the point distance.

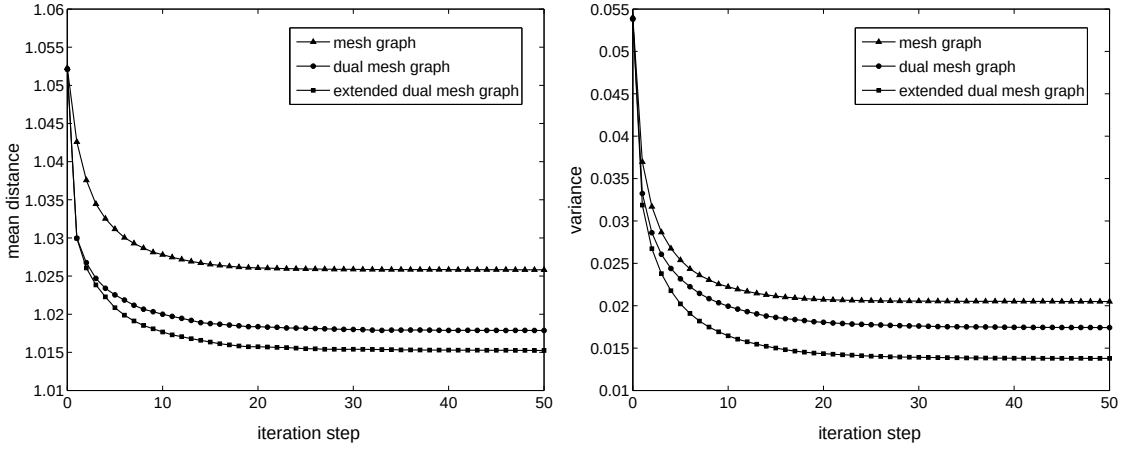


Figure 4.11: Same as Figure 4.10 with the modification that the triangular mesh had a point density of $40\,\text{Å}^{-2}$ and the desired point distance was 1.0 Å.

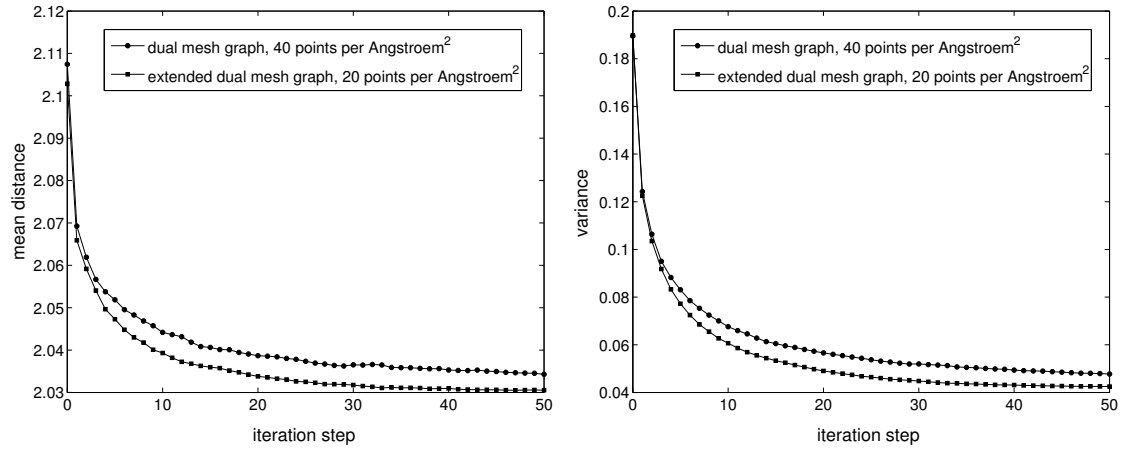


Figure 4.12: Comparison of point relaxation based on different mesh graphs and different mesh resolutions. For the point relaxation on the dual mesh graph we used a triangular mesh with a resolution of $40\,\text{Å}^{-2}$. For the point relaxation on the extended dual mesh graph we used a triangular mesh with a resolution of $20\,\text{Å}^{-2}$. The desired point distance was 2.0 Å in both cases. *Left:* Evolution of the mean distance. *Right:* Evolution of the variance of the point distance.

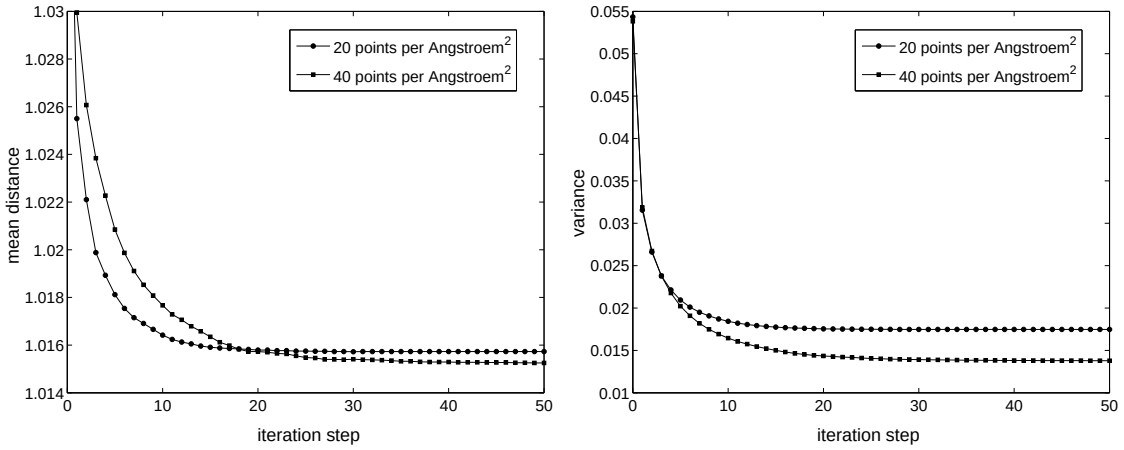


Figure 4.13: Comparison of point distribution process using different triangular mesh resolutions: 20 versus $40\,\text{Å}^{-2}$. The desired point distance was 1.0 Å. *Left:* Evolution of the mean distance. *Right:* Evolution of the variance of the point distance.

distance as well as to a smaller variance of the mean distance. However, the differences are small. Hence, we can reason, that a point density of $20\,\text{Å}^{-2}$ still constitutes a fine enough resolution for a desired point distance of 1.0 Å.

Run Times

Run times for the examples given in this section can be found in Table 4.1. Several observations concerning the run times can be made by inspection of this table.

1. The surface generation plays a minor role in the overall run time.
2. The preprocessing for the extended dual mesh graph takes significantly longer than the preprocessing for both mesh and dual mesh graph. The preprocessing includes the computation of the lengths of all edges in the graph. Since the extended dual mesh graph has significantly more edges than the other two graphs, the longer preprocessing is due to this larger number of edges.
3. The initialization also plays a minor role in the overall run time. Even though Monte Carlo initialization is faster than graph partitioning initialization, the longer run time is justified by the better quality of the initial point distribution generated.
4. The main fraction of the overall time is needed for the relaxation step. In all of the examples, 50 iterations were carried out. However, the plots show (cf. Figures 4.9, 4.10, 4.11, 4.12, 4.12, 4.13), that, in general, most of the relaxation is done after 20 or 30 iterations. Hence, if run time is a matter, we could stop after this number of iterations.

4.4.2 Non-Homogeneous Point Distributions

The results of Section 4.4.1 show that it is favorable to start the relaxation process with an initial point distribution generated using the graph partitioning approach. They also show that relaxation using the extended dual mesh graph produces the best results, both in terms of quality and convergence. Hence, in this section we will not deal with comparing different approaches, but we will show that the approaches that work best in the homogeneous case work well in the non-homogeneous case too.

Distance Field - An Artificial Point Density

We start with an artificial point density representing the scaled distance field on the surface from some arbitrarily selected point residing on the surface (cf. Figure 4.15(a)). In contrast to the electrostatic potential, which has negative and positive values that need to be handled separately, the distance field has only positive values, which vary continuously across the whole surface. This makes it well suited for illustrating the capabilities of our approach to non-homogeneous point distributions.

The computed distance field had values between 0.0 and about 18.0. In our experiments we mapped this range to a range between 0.0 and 4.0 in order that the point density does

Figure		Surface Gen.	Preproc.	Initial Point Positioning	Relaxation	Overall
4.9	MC	0.04	0.17	0.01	0.84	1.09
4.9	GP	0.04	0.16	0.05	0.80	1.07
4.10	$G(\mathcal{M})$	0.04	0.08	0.05	0.50	0.69
4.10	$\widetilde{G}(\mathcal{M})$	0.04	0.08	0.05	0.49	0.68
4.10	$\widetilde{G}^*(\mathcal{M})$	0.04	0.16	0.05	0.80	1.07
4.11	$G(\mathcal{M})$	0.07	0.17	0.18	1.73	2.18
4.11	$\widetilde{G}(\mathcal{M})$	0.07	0.15	0.18	2.10	2.54
4.11	$\widetilde{G}^*(\mathcal{M})$	0.07	0.30	0.18	2.30	2.91
4.12	$20\,\text{Å}^{-2}$,$\widetilde{G}^*(\mathcal{M})$	0.04	0.16	0.05	0.80	1.07
4.12	$40\,\text{Å}^{-2}$,$\widetilde{G}(\mathcal{M})$	0.07	0.19	0.08	1.15	1.56
4.13	$20\,\text{Å}^{-2}$,$\widetilde{G}^*(\mathcal{M})$	0.04	0.18	0.13	1.62	1.99
4.13	$40\,\text{Å}^{-2}$,$\widetilde{G}^*(\mathcal{M})$	0.07	0.30	0.18	2.30	2.91

Table 4.1: Run times (in seconds) for homogeneous point distributions given in this section. MC denotes Monte Carlo initialization, GP denotes graph partitioning. The notations of the graphs are as defined in Section 4.1.2.

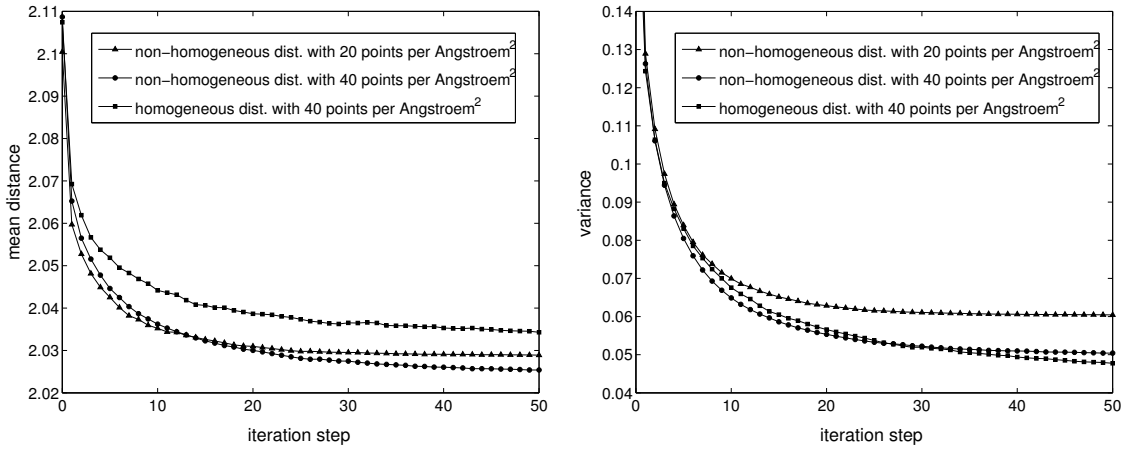


Figure 4.14: Comparison of point distribution process for non-homogeneous point distributions using different triangular mesh resolutions: 20 versus $40\,\text{Å}^{-2}$. The behavior of the homogeneous point distribution is given as reference. The point distances were scaled according to Equation (4.21) to allow comparison with the homogeneous case and to allow computation of mean distance and variance. The desired scaled point distance was 2.0 Å. *Left:* Evolution of the mean distance. *Right:* Evolution of the variance of the point distance.

not vary too greatly. We also scaled the point density by a factor $\tilde{\rho} = 0.289$ which corresponds to a point distance of 2.0 Å in the homogeneous case (cf. Equation (4.20)). This gave us a density comparable to those of electrostatic potentials. In order to be able to compare the point distances of the non-homogeneous point distribution with point distances in the homogeneous case, we had to scale the geodesic distances as follows. Let $\mathfrak{p}$ be a shortest path and let $d(\mathfrak{p})$ be its length. Let further denote by $\rho(\mathfrak{p})$ the averaged point density along $\mathfrak{p}$. Then $d(\mathfrak{p})$ is scaled to $d^*(\mathfrak{p})$ as follows

$$d^*(\mathfrak{p}) = \frac{1}{\sqrt{\rho(\mathfrak{p}) \cdot \tilde{\rho}}}\, d(\mathfrak{p}) \ . \tag{4.21}$$

The results of this experiment are shown in Figure 4.14. The plots confirm a similar behavior and quality as in the homogeneous case.

An example of a non-homogeneous point distribution according to a distance field is shown in Figure 4.15.

Electrostatic Potential

The electrostatic potential on the surface of some molecule can be easily computed from the atomic point charges of the molecule. As mentioned before, we used the Merck Molecular Force Field (MMFF) [73] to parametrize the molecules. This parametrization includes the computation of the atomic point charge of each atom. For each vertex on the triangular mesh, we computed the electrostatic potential by summing up the electrostatic potential created by all atomic point charges (cf. Equation 2.2.2). The electrostatic potential on the surface of the active conformer of amprenavir was computed using an artificial dielectric constant of 0.3, which gives values of the electrostatic potential on the surface roughly between -2.0 and 2.0. Note, that the electrostatic potential is reciprocally proportional to the dielectric constant. Hence, choosing a different value for the dielectric constant will only scale the potential by a different factor.

The electrostatic potential together with points distributed on the surface according to the potential are shown in Figure 4.16. Points representing the positive and negative parts of the electrostatic potential, respectively, were positioned separately. In order to do so, we first identified those triangles having the same sign of the electrostatic potential. For these triangles we identified the connected patches. On each of these connected triangle patches we then separately distributed points. The reason for handling each connected patch separately is obvious: the shortest path from each point to each other needs to exist in order for the algorithm to work.

4.5 Summary and Conclusion

In this chapter we presented a novel approach for distributing points regularly on a molecular surface given by a 2-manifold triangular mesh. This new approach is based on centroidal Voronoi tesselation. We applied centroidal Voronoi tesselation to triangular meshes in three dimensions. In order to do so, three subproblems had to be solved.

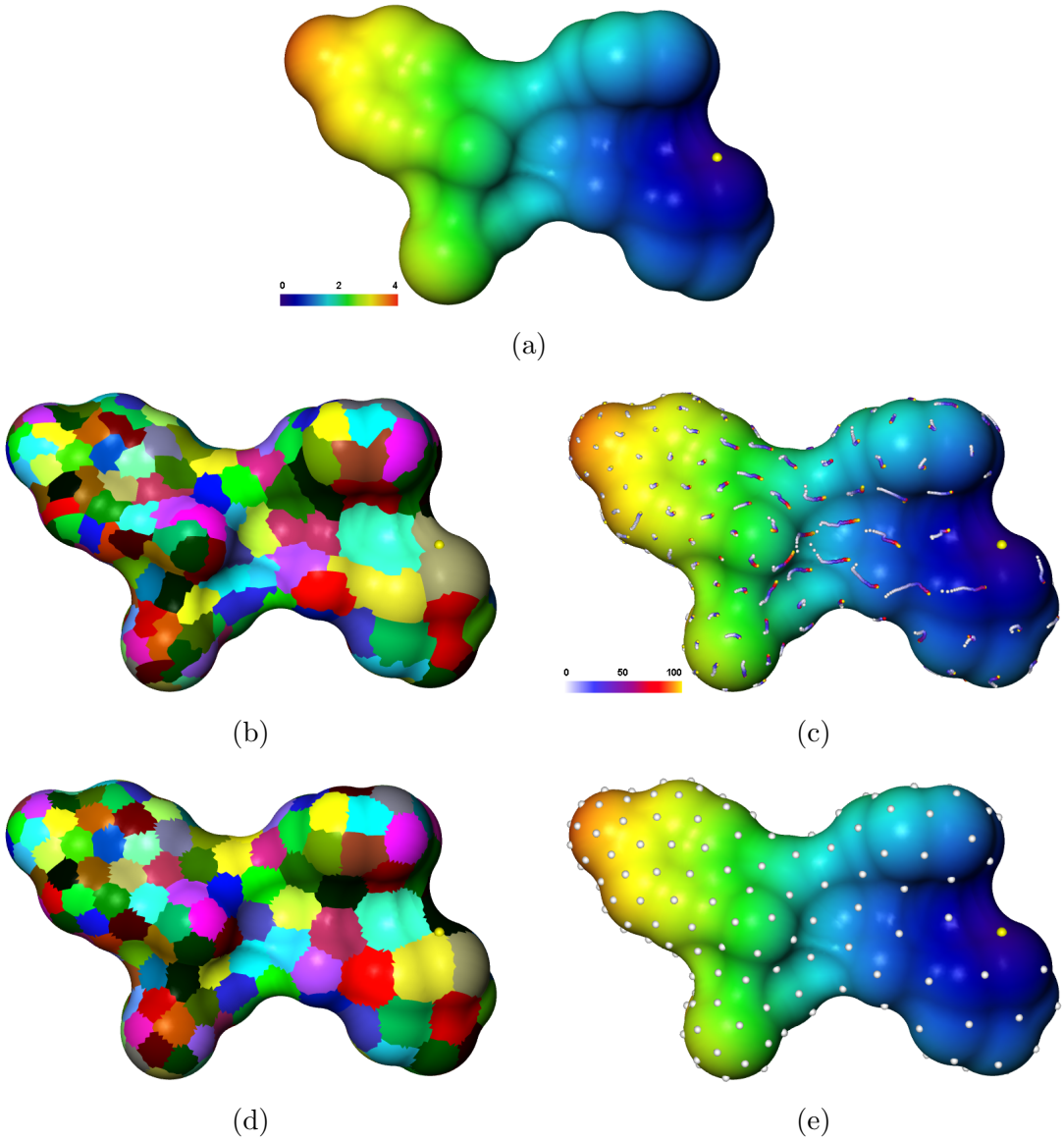


Figure 4.15: Non-homogeneous distribution of 247 points on the SES of amprenavir. For the triangulation of the SES, a point density of 40 Å^{-2} was used. (a) The surface is colored according to the *distance field* computed from a single point on the surface, represented by a yellow sphere at the right side of the image. The distance field was computed on the triangular mesh graph (cf. Definition 4.1.3) and scaled to a range of 0.0 to 4.0. The mapping of the scaled distances to the colors is given by the colormap shown in the image. (b) Initial weighted triangle partitioning. (c) Point trajectories generated during the point relaxation process. The colormap shows the color mapping of the trajectory points according to the iteration step. (d) Centroidal Voronoi tesselation of triangular surface. Note, that in the centroidal Voronoi tesselation the position of the yellow sphere, i.e., where the distance field is 0, is on the boundary of the Voronoi regions near the point. (e) Final point positions, which are given by the weighted centers of mass of the Voronoi regions seen in (d).

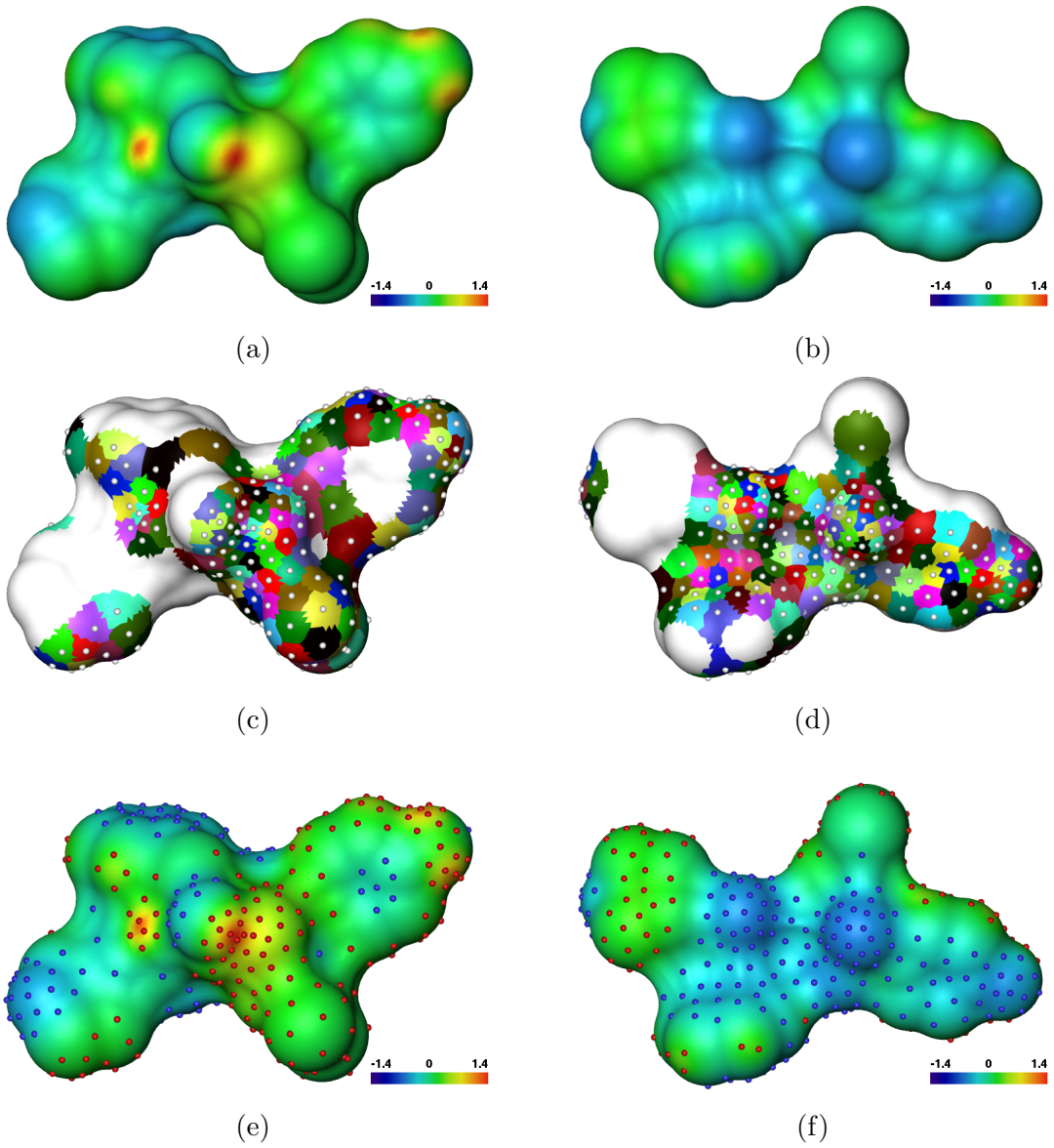


Figure 4.16: Non-homogeneous distribution of 397 points on the SES of amprenavir. For the triangulation of the SES, a point density of $20\,\text{Å}^{-2}$ was used. *First row:* The surface is colored according to the *electrostatic potential* on the SES. The color mapping is done according to the depicted colormap. One side of amprenavir, (a), has a more positive potential than the other side, (b), which is more negative. *Second row:* For those parts having positive or negative values, respectively, we separately computed point distributions. (c) This image shows the centroidal Voronoi tesselation (CVT) of the regions having positive values together with their generators. (d) Here, the CVT of the negative regions and their corresponding generators are shown. *Last row:* 216 points (red) representing the positive regions and 181 points (blue) representing the negative regions are shown on the surface.

1. *Initial point distribution.* The centroidal Voronoi tesselation needs an initial set of generators. In order for the algorithm to work efficiently, we developed an initial point distribution scheme based on graph partitioning. We showed that this scheme works better than initialization using a Monte Carlo method. We also showed that the scheme is efficient and that it can be used directly for non-homogeneous point densities.

2. *Approximation of the Voronoi diagram on a triangular mesh.* In order to approximate the Voronoi diagram on triangular meshes, we need to be able to efficiently approximate geodesic distances. We have proposed three methods based on graphs defined on the triangular mesh. We tested all of these three methods both in terms of quality and run time. The results show, that the extended dual mesh graph is superior in terms of quality over the other two graphs. Even though its run time is slightly worse than those of the other two methods, we suggest to favor this approach, since it yields the best quality.

3. *Computation of centers of Voronoi regions.* The centroidal Voronoi tesselation, as the name suggests, requires the computation of the center of each Voronoi region. We have defined the center of a Voronoi region as the center of mass of the Voronoi region projected back onto the triangular surface along the averaged surface normal of the Voronoi region. The results presented in Section 4.4 show that this definition is justified, at least for smooth surfaces, such as the solvent excluded surfaces used here. The definition also generalizes directly to non-homogeneous point densities.

There exist a couple of virtual screening applications that use points distributed on molecular surfaces. Stiefl et al. [159] and Bender et al. [23], e.g., use points distributed on molecular surfaces for computing transformation invariant molecular descriptors. These descriptors are then used to screen data bases of potential drugs. Bender et al. use points generated directly by the surface triangulation. These points are, in general, not homogeneously distributed, i.e. the distances between neighbored points may vary considerably. In contrast to this, Stiefl et al. approximate the molecular surface by points on a 3-dimensional uniform grid. While the approach by Bender et al. has the drawback that the points are not homogeneously distributed across the molecular surface, the approach by Stiefl et al. positions the points not directly onto the surface. Thus, both applications might benefit from the new point distribution method described in this chapter.

In addition to generating homogeneous point distributions, our method also works for distributing points according to non-homogeneous point densities. This allows us to generate point distributions that represent continuously varying properties on molecular surfaces such as the electrostatic potential. In the following chapter we show how the points distributed on solvent excluded surfaces can be used to compute partial matchings of molecular surfaces.