Hexagonal Delaunay Triangulation

Gerd Sußner[1] and Günther Greiner[2]

[1] RTT AG, Munich, Germany
Gerd.Sussner@rtt.ag
[2] Computer Graphics Group, University of Erlangen-Nuremberg, Germany
guenther.greiner@cs.fau.de

Abstract. We present a novel and robust algorithm for triangulating point clouds in $\mathbb{R}^2$. It is based on a highly adaptive hexagonal subdivision scheme of the input domain. That hexagon mesh has a dual triangular mesh with the following properties:

- any angle of any triangle lies in the range between 43.9° and 90°,
- the aspect ratio of triangles is bound to 1.20787,
- the triangulation has the Delaunay property,
- the minimum triangle size is bounded by the minimum distance between input points.

The iterative character of the hexagon subdivision allows incremental addition of further input points for selectively refining certain regions. Finally we extend the algorithm to handle planar straight-line graphs (PSLG). Meshes produced by this method are suitable for all kinds of algorithms where numerical stability is affected by triangles with skinny or obtuse angles.

Keywords: Unstructured Mesh Generation, Delaunay Triangulation, Guaranteed Angle Bounds, Hexagon Subdivision.

1 Introduction

Mesh generation is the subject of many articles due to its importance to computational geometry, computer graphics, numerical simulation and various other areas. Many problems are based on input data lying in a 2-dimensional domain. This data is often triangulated to allow computations on the mesh, e.g. for finite element methods. The result usually strongly depends on the quality of the triangular mesh, e.g. small or obtuse angles reduce the numerical stability for a high number of elements. The number of triangles also determines the runtime for solving the problem. Therefore it is desirable to have a well-shaped triangular mesh with as few triangles as possible.

There are several kinds of unstructured mesh generators. Some are based on grid-techniques, e.g. quad-trees, while others try to improve an existing

triangulation iteratively. These and other approaches like advancing front techniques are surveyed by Owen [7].

In this paper we use a different approach. Instead of a quad-tree a hexagonal highly adaptive subdivision scheme is used for the input domain. When the domain is finally subdivided a dual mesh is extracted from the centers of the full hexagons. It turns out that this mesh has the Delaunay property, i.e. maximizing the minimum angle of any triangle, which is guaranteed to be 43.9° – **independent** of the input data. The method is also aware of different regions of interest, meaning that areas with less input points have larger triangles. The size of the triangles quickly decreases in regions of high interest. It is limited by the minimum distance among the input points which also ensures termination of the algorithm. Both properties lead to triangular meshes with a reasonable number of triangles.

After describing previous work, we give a short introduction into the hexagon subdivision scheme. Then refinement rules are defined to subdivide the input domain adaptively according to the location of the input points. From the hexagonal mesh a dual triangular mesh is extracted where any angle is between 30° and 120°. In order to achieve much tighter bounds the hexagons are classified according to their adjacent neighbors. Additional refinement rules and shifting the centers of the classified hexagons results in a triangular mesh with angles between 43.9° and 90°.

As an additional result the algorithm is extended to handle also PSLGs. This may be important to limit the domain only to valid input values. However at these boundaries the proven angle bounds are lost.

2 Previous Work

Baker et al. [1] introduced guaranteed shape properties for the resulting meshes so that all angles of the triangles lie within 13° and 90°, yielding an aspect ratio of at most 4.6. This is achieved by placing a square-grid over the polygons to include Steiner points. The size of the grid is determined by the smallest distance among the input points and edges. However the resulting meshes may be very large.

Bern et al. [2] use a quad-tree instead of a uniform grid. This method gives bounds to shape property and the number of triangles. Subdividing the domain by a quad-tree allows a local refinement with Steiner points where necessary while regions of low interest remain coarse. For point sets the angle bounds are 36° and 80° and for a planar straight-line graph (PSLG) the range is 18.4° and 153.2°. To achieve this, the key-idea is to move the corners of the quad-tree according to some patterns. The method of Neugebauer and Diekmann [6] uses rhombi instead of squares when subdividing the domain, yielding angle bounds of 30° and 90° for polygonal input.

A different approach is refining an already obtained Delaunay-triangulation of a point set by inserting Steiner points at certain positions iteratively which is called *Delaunay refinement*. The original approach of Chew [3] and Ruppert

[8] uses the circumcenter of badly shaped triangles while latest methods of Üngör [10] and Erten [4] use different locations . The guaranteed angle bounds for point sets are 30° and 120°. For PSLGs they are set to 20° and 160°. In practice often higher angles up to 34° are achieved, depending on the input data which was examined by Shewchuk [9]. The Off-Center approach even raises this limit up to 42°. However it is always possible to get input data, where the Delaunay refinement fails for high angles. Both methods are implemented in `Triangle` which is a robust software for creating Delaunay-refined meshes available at `http://www.cs.cmu.edu/~quake/triangle.html`.

3 Hexagonal Subdivision

Sußner et al. [5] use a bidirectional subdivision of a hexagonal mesh to adaptively refine the domain of huge height-fields for interactive purposes. The subdivision method follows simple and easy-to-implement rules and is therefore an efficient way to adaptively subdivide planar regions.

3.1 Subdividing a Hexagon

A hexagon is subdivided by scaling it to half size and filling the remaining space with semi-hexagons. The hexagons are organized in levels: Full hexagons are located in even levels while semi-hexagons only occur in odd levels. When subdividing a full hexagon, the scaled full hexagon is moved into the next even level while the semi-hexagons are put in the odd level in-between. After the operation each of the semi-hexagons is checked for a fitting adjacent semi-hexagon (see Fig. 1). If there is one both semi-hexagons are joined to form a new full hexagon one level above.

3.2 Reverse Operation

The above operation is reversible. In a first step adjacent full hexagons may be separated into two semi-hexagons. In order to break the right hexagons,

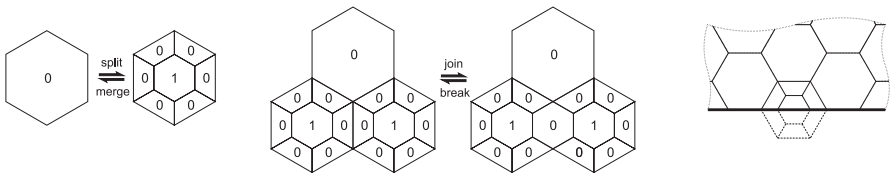

Fig. 1. The numbers denote the split-counter of each hexagon. On the left the split-/merge-operator is shown which increases/decreases the split-counter by one. The join-/break-operator in the middle creates a new full hexagon with a split-counter value of zero. As shown on the right side, a boundary semi-hexagon is treated like a virtual full hexagon.

each hexagon gets a subdivision counter which is set to zero when a new hexagon was created by joining two semi-hexagons. Every time a hexagon is subdivided the counter is increased by one and decreased if it was scaled up to a hexagon of lower level. Finally a hexagon may only be broken up into two semi-hexagons if the subdivision counter is zero.

3.3 Adaptive Refinement

For the sake of a smooth increase of level-of-detail, a simple balancing rule is applied. After each subdivision step the level difference of adjacent (semi-) hexagons must not be larger than one. If the difference is larger the hexagon of lower level is subdivided as well. If this is a semi-hexagon, the opposite neighbor, which must be a full hexagon, is subdivided. The balancing rule is illustrated in Fig. 2. Note that a single subdivision step may trigger the refinement of large parts of the hexagon mesh.

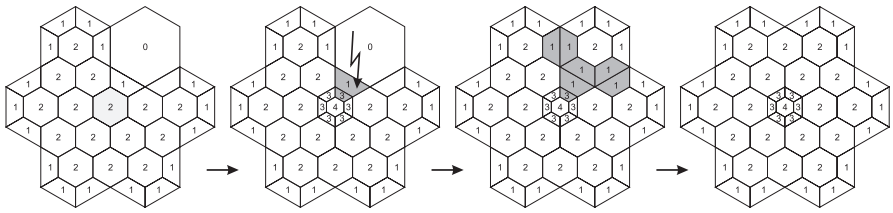

Fig. 2. The numbers represent the level counter of each hexagon. If the level difference is greater than one further splits and joins of semi-hexagons are forced.

4 Simple Refinement

In this section we present a simple version of the triangulation algorithm, with angle range of 30° to 120°. Beyond these angles triangles are usually considered as *bad*.

4.1 Refinement Rules

As a first step, a bounding hexagon is created around all input points. In order to keep the number of triangles as low as possible, a good choice is the minimum bounding sphere as the inner circle of the hexagon as shown in Fig. 3. This sphere may be efficiently computed by the method presented by Welzl [11]. In addition to the balancing rules of section 3.3 the following rules must be obeyed in order. A hexagon containing an input point is marked *occupied*. Given a set of hexagons $\mathcal{H}$, the rules are:

1. subdivide any hexagon $h_i \in \mathcal{H}$ occupied by several input points,
2. if $h_i \in \mathcal{H}$ is an occupied semi-hexagon, subdivide the full hexagon adjacent to the long edge of h_i,
3. subdivide an occupied hexagon $h_i \in \mathcal{H}$ if any adjacent neighbor is occupied as well,
4. if any of the adjacent neighbors is of lower level, subdivide the (full) hexagons adjacent to the long edge of the neighboring semi-hexagon,
5. subdivide an occupied hexagon $h_i \in \mathcal{H}$ if any adjacent neighbor is of higher level.

At each single subdivision step, i.e. splitting full hexagons and joining semi-hexagons, input points were re-distributed among the affected hexagons. In order to test the whole hexagon-mesh after each operation, a list of affected hexagons is maintained. Refinement stops if this list is empty. Termination is guaranteed since an input point is only associated to one hexagon. Fig. 3 shows examples of these rules. In the final hexagon mesh all occupied hexagons are full hexagons surrounded by a ring of full non-occupied hexagons.

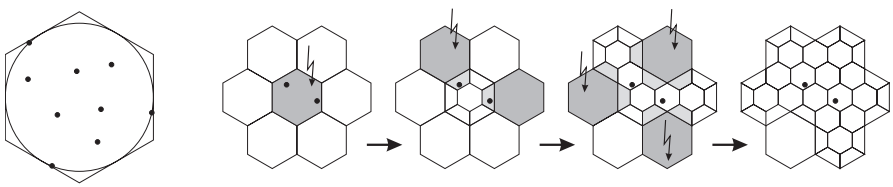

Fig. 3. Initial setup of input points on the left side, followed by a sequence of applied refinement rules: several input points, input point in semi-hexagon, adjacent lower level hexagons.

4.2 Extracting the Dual Mesh

For getting the dual triangulation from the hexagon mesh just add an edge from the center of each full hexagon to the centers of its adjacent hexagons. If the neighbor is a semi-hexagon connect the edge to the center of the opposite full hexagon of the neighbor (see Fig. 4). In a final step the center positions of the *occupied* hexagons are replaced by the location of corresponding assigned input points.

4.3 Proving Angles

The proof for a minimum angle of 30° is split into two parts. First the angles of the plain dual mesh, i.e. without relocated centers, are examined. In the second part we take a look at the situation of a relocated center.

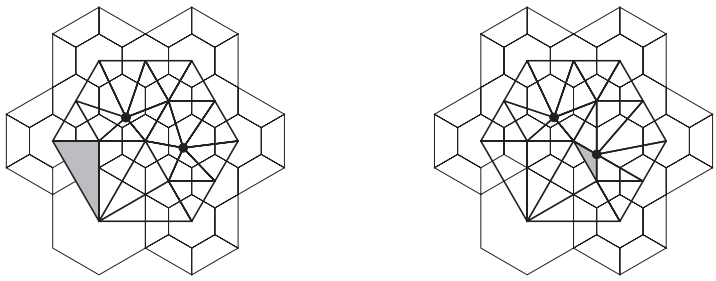

Fig. 4. The dual mesh consists of the edges among adjacent full hexagons. For semi-hexagons the opposite neighbor is taken. Note that the minimum angle of 30° is always in the triangulation due to the adaptive subdivision (grey triangle on the left). On the right side there is a triangle consisting of both angle bounds, 30° and 120°.

Angles of the Plain Dual Mesh

As shown in Fig. 4 the centers of three adjacent full hexagons form a equilateral triangle with three angles of 60°. If one of them is subdivided, the center of scaled hexagon still remains at the same position. If two of them are subdivided, they form a new full hexagon located right in the middle of them. Therefore the equilateral triangle is split into two with half angle at the center of the remaining un-subdivided hexagon. Since the center of the new hexagon lies on the middle of the two existing centers both triangles have an angle of 90° at the new center.

Angles at Relocated Centers

A hexagon containing a relocated center is surrounded by full hexagons as shown on the right side in Fig. 4. The input point may be located anywhere within the hexagon. In extremum the center is moved to a hexagon's vertex. In this case one triangle has two angles of 30°, but not less. This also means that the third angle must be 120°.

5 Extended Refinement

For simple refinement the dual mesh is extracted by connecting the centers among all full hexagons. But there is plenty of freedom to shift the centers' position to achieve tighter angle bounds. To reach this goal, two enhancements are necessary.

First the full hexagons are classified according to their adjacent (semi-) hexagons. Based on that classification the hexagon mesh is adaptively refined. In a second step, when the dual mesh is extracted, the centers of some classified hexagons are shifted to yield higher angle bounds of 43.9° and 90° respectively.

5.1 Classification of Hexagons

The classification of the full hexagons is based on the number of fins. A fin is a semi-hexagon which joins at its long edge to the full hexagon (see top row of Fig. 5). If there is any fin the hexagon $h_i \in \mathcal{H}$ is marked as follows:

MOVED_CENTER_1	- h_i has only one fin,
MOVED_CENTER_2	- h_i has only two consecutive fins,
MOVED_CENTER_3	- h_i has only three consecutive fins,
SUBDIVIDE	- else.

Full hexagons adjacent to the edge before the fin strip are called *left neighbor* h_{i_l}. Those adjacent to the edge after the fins are called *right neighbor* h_{i_r}. In case that h_i is of type MOVED_CENTER_3 the remaining adjacent full hexagon is called *top neighbor* h_{i_t}. Full hexagons $h_i \in \mathcal{H}$ without fins are marked as well (see bottom row of Fig. 5):

OCCUPIED_1	- input point lies within the half-sized hexagon of h_i,
OCCUPIED_2	- input point lies outside of half-sized hexagon of h_i,
FIXED_CENTER_IN_RING	- h_i is in 1-ring of a hexagon of type OCCUPIED_1 or in 2-ring of a hexagon of type OCCUPIED_2,
MOVED_CENTER_IN_RING	- h_i is in 1-ring of a hexagon of type OCCUPIED_2,
MOVED_CENTER_4	- h_i is top neighbor of a hexagon of type MOVED_CENTER_3,
FIXED_CENTER_REGULAR	- else.

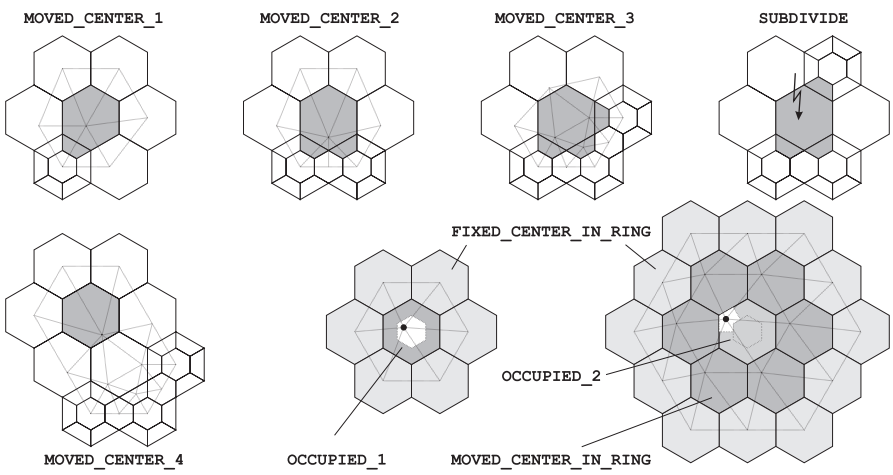

Fig. 5. Hexagons are classified to their number of fins. Hexagons containing an input point marked as OCCUPIED_[12] depending on the location of the input point within the hexagon (white areas). OCCUPIED_2-hexagons are surrounded by ones of type MOVED_CENTER_IN_RING. Input points are also surrounded by FIXED_CENTER_IN_RING-hexagons either in the 1-ring or 2-ring. The 1-ring of a hexagon h_i is defined by the full hexagons surrounding h_i. The 2-ring of h_i then consists of all full hexagons surrounding the hexagons of the 1-ring of h_i.

5.2 Extended Refinement Rules

According to the classified hexagons further refinement rules are defined:

- subdivide any hexagon $h_i \in \mathcal{H}$ of type `SUBDIVIDE`,
- subdivide any hexagon $h_i \in \mathcal{H}$ of type `MOVED_CENTER_x` lying in the 1-ring of a hexagon of type `OCCUPIED_1`,
- subdivide any hexagon $h_i \in \mathcal{H}$ of type `MOVED_CENTER_x` lying in the 1- or 2-ring of a hexagon of type `OCCUPIED_2`,
- subdivide any hexagon $h_i \in \mathcal{H}$ of type `MOVED_CENTER_[123]` if any full hexagon adjacent to its fins is not of type `FIXED_CENTER_x`,
- subdivide a hexagon h_i of type `MOVED_CENTER_4` if it is the top neighbor of more than one non-consecutive hexagons of type `MOVED_CENTER_3`.
- Given a hexagon h_i of type `MOVED_CENTER_3` with a left neighbor of the same type. If its right neighbor $h_j := h_{i_r}$ is of type `MOVED_CENTER_2`, subdivide right neighbor h_{j_r} of h_j. (Applied symmetrically with left neighbor!)

The last rule is necessary, since we have a special treatment for two or more consecutive hexagons of type `MOVED_CENTER_3` when extracting the dual mesh. The consequences of this rather complicated rule are illustrated in Fig. 6. Note that six hexagons of type `MOVED_CENTER_3` may form a ring - called *blossom*. This is exploited later in order to reduce the number of hexagons. Also note that each sequence of `MOVED_CENTER_2`/`MOVED_CENTER_3`- hexagons, except for a blossom, is enclosed by hexagons of type `MOVED_CENTER_1`. Fig. 7 shows an example of a valid subdivision.

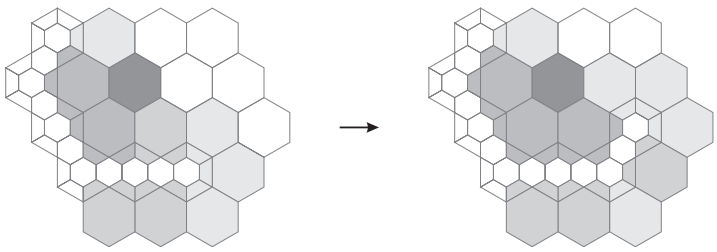

Fig. 6. Angle bounds do not hold if a `MOVED_CENTER_2`-hexagon is adjacent to a pair of `MOVED_CENTER_3`-hexagons. Subdividing its right/left neighbor creates another hexagon with three fins.

5.3 Local Coarsening

Applying above rules may lead to a subdivision with greater areas of hexagons of same size. In order to avoid an unnecessary large number of triangles, a local coarsening step is performed according to *as fine as necessary but as coarse as possible* (see Fig. 7). For this the neighborhood of each hexagon h_i with a split-counter of 0 is examined whether it is possible to execute all necessary break-/merge-operations to form a blossom. That is all of the

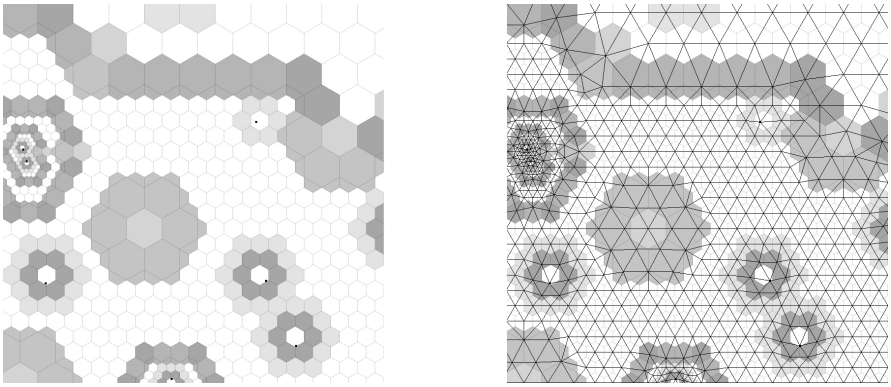

Fig. 7. This subdivision shows almost all possible combinations among the different hexagon types. Blossoms are created at a final local coarsening step in order to avoid large regions of small sized hexagons. Note that two input points may share hexagons of type `FIXED_CENTER_IN_RING`, e.g. in the lower right quarter. The corresponding dual mesh is drawn as an overlay on the right side.

hexagons in the 4-ring of h_i must be of type `FIXED_CENTER_REGULAR`. By this operation 37 hexagons are replaced by 7. In the dual mesh a blossom is represented by 60 triangles instead of 96 for the plain region.

5.4 Modifications to the Dual Mesh

In this section we show how to adapt the center positions for the dual mesh. For hexagons of type `MOVED_CENTER_3` it is additionally necessary to insert moved centers of their fins.

Lower Angle Bound

First the lower angle bound is determined since this angle is used to compute some of the relocated centers. Have a look at the left side of Fig. 8 where a hexagon of type `OCCUPIED_1` with radius r is shown. The input vertex is located at a vertex of the half-sized hexagon, causing extremal angles. The smallest angle in this configuration is α and is determined analytically:

$$\tan\alpha = \frac{y}{x} = \frac{\frac{5}{4}r}{\frac{3}{2}r\frac{\sqrt{3}}{2}} = \frac{5}{3\sqrt{3}} \quad \Rightarrow \alpha \approx 43.8979$$

Centers of `MOVED_CENTER_IN_RING`-hexagons

In case that the input point does not lie in the half-sized hexagon of h_i, the closest vertex v_{c_i} to the input point of h_i is determined. As illustrated at the right side in Fig. 8 the input point may be located anywhere in the grey

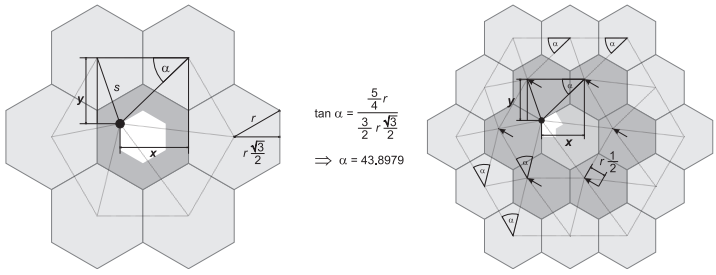

Fig. 8. On the left side the input point is located at a vertex of the half-sized hexagon (white area). The smallest angle α is determined by x and y. On the right side the input point is located outside of the half-sized hexagon h_i. The centers of the surrounding hexagons are shifted by vector of $\frac{1}{2}r$ from the center of h_i towards the nearest corner of the input point. For that corner the input point may lie anywhere in the white area. Note that in either configuration all triangles are acute.

region. In order to maintain angle bounds, the half-sized vector between v_{c_i} and the center C_{h_i} is added to each of the surrounding hexagons which are of type `MOVED_CENTER_IN_RING`.

Centers of `MOVED_CENTER_1`-hexagons

Given a hexagon h_i of type `MOVED_CENTER_1` and a hexagon $h^{f_i^0}$ of type `FIXED_CENTER_x` adjacent to the only fin f_i^0 of h_i. The new center position $\widehat{C}_{h_i}$ of h_i lies on the axis between the original center position C_{h_i} and the center $C_{h^{f_i^0}}$ of $h^{f_i^0}$:

$$\widehat{C}_{h_i} := \frac{5}{6}C_{h_i} + \frac{1}{6}C_{h^{f_i^0}}.$$

The ratios are shown on the right side of Fig. 9 together with special configuration of `MOVED_CENTER_3`-hexagons.

Centers of `MOVED_CENTER_2`-hexagons

Given a hexagon h_j of type `MOVED_CENTER_2` and three hexagons $h^{f_j^0}$, $h^{f_j^{01}}$ and $h^{f_j^1}$ of type `FIXED_CENTER_x` adjacent to the fins f_j^0 and f_j^1 of h_j. The new center position $\widehat{C}_{h_j}$ of h_j lies on the axis between the original center position C_{h_j} and the center $C_{h^{f_j^{01}}}$ of $h^{f_j^{01}}$. Looking at Fig. 9 $x := \frac{\sqrt{3}}{2}r$ is the edge between $C_h^{f_j^0}$ and $C_h^{f_j^{01}}$, i.e.

$$\tan\alpha = \frac{x}{y} \quad \Rightarrow y = \frac{\frac{\sqrt{3}}{2}r}{\frac{5}{3\sqrt{3}}} = \frac{9}{10}r.$$

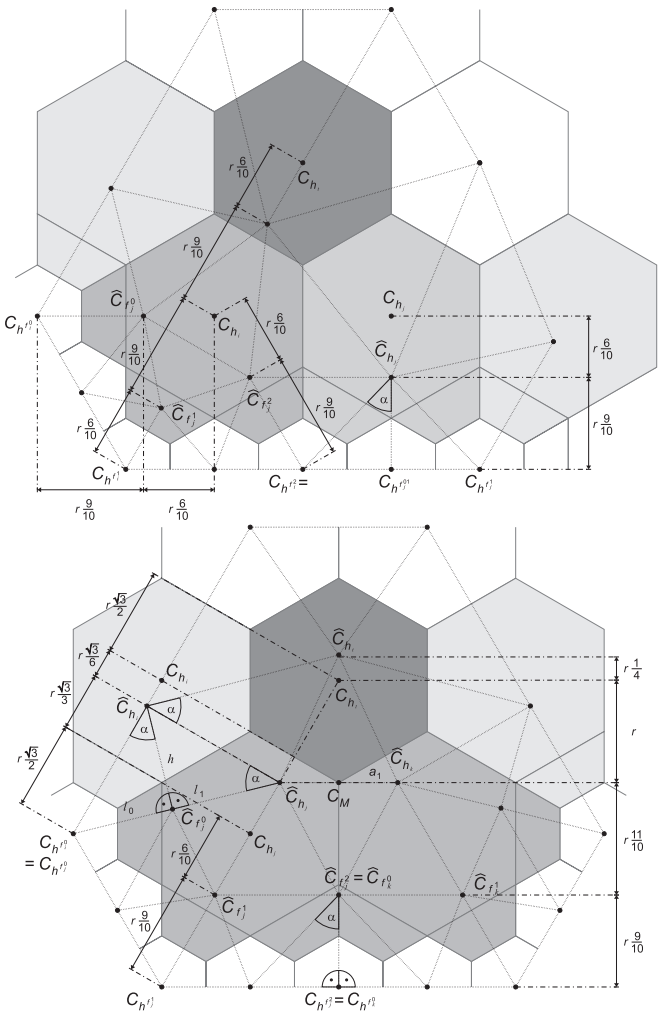

Fig. 9. In this picture the ratios for computing the new location of the centers are illustrated. On the left side the configuration with a single MOVED_CENTER_3-hexagon and on the right side a pair of them. Note that the ratio for a 1-fin-hexagon is derived from the right side.

The distance between the centers of h_j and $h^{f_j^{01}}$ is $\frac{3}{2}r$. Hence the hexagon's center is relocated to

$$\widehat{C}_{h_j} := \frac{3}{5}C_{h_j} + \frac{2}{5}C_{h^{f_j^{01}}}$$

Centers of single MOVED_CENTER_3-hexagons

Given a hexagon h_i of type MOVED_CENTER_3 and five hexagons $h^{f_i^0}$, $h^{f_i^{01}}$, $h^{f_i^1}$, $h^{f_i^{12}}$ and $h^{f_i^2}$ of type FIXED_CENTER_x adjacent to the fins f_i^0, f_i^1 and f_i^2 of

h_i. If it is a single MOVED_CENTER_3-hexagon, i.e. neither the left or nor the right neighbor is of same type, the original center is neglected. Instead the three centers of the fins are shifted and included in the triangulation. For computing the new positions of three centers the same ratios are used as for MOVED_CENTER_2-hexagons. However for the middle one the ratio is swapped.

$$\begin{aligned}\widehat{C}_{f_i^0} &:= \tfrac{3}{5}C_{h_i} + \tfrac{2}{5}C_{h^{f_i^0}}\\ \widehat{C}_{f_i^1} &:= \tfrac{2}{5}C_{h_i} + \tfrac{3}{5}C_{h^{f_i^1}}\\ \widehat{C}_{f_i^2} &:= \tfrac{3}{5}C_{h_i} + \tfrac{2}{5}C_{h^{f_i^2}}\end{aligned}$$

Centers of consecutive MOVED_CENTER_3-hexagons

As mentioned in Sec. 5.2 we have different topologies for two or more consecutive hexagons of type MOVED_CENTER_3. In addition to the previous case, the center of h_j is included in the triangulation. As shown in Fig. 9, it lies on the axis between the original center position C_{h_j} and the center of its top neighbor $C_{h_j^t}$, i.e.

$$\widehat{C}_{h_j} := \frac{2}{3}C_{h_j} + \frac{1}{3}C_{h_j^t}.$$

Suppose the left neighbor is of type MOVED_CENTER_1, then the right neighbor must be of MOVED_CENTER_3 as shown on the right side of Fig. 9. The center $C_{f_j^1}$ lies on the line through $\widehat{C}_{h_j}$ and $C_{h_j^{f^0}}$. Since we want an upper angle bound of 90°, it must form two right triangles together with center of the left neighbor. Both triangles have the same height and one angle of α, then let l_0, l_1 and l be defined as:

$$l_0 := \frac{5}{3\sqrt{3}}h \qquad l_1 := \frac{3\sqrt{3}}{5}h \qquad l := \left\|\widehat{C}_{h_j} - C_{h^{f_j^0}}\right\|.$$

Solving this for l_0 and l_1 leads to a ratio of 25 : 27, i.e. the new center is

$$\widehat{C}_{f_j^0} := \frac{25}{52}\widehat{C}_{h_j} + \frac{27}{52}C_{h^{f_j^0}}.$$

Note that this configuration also determines the ratio for hexagons of type MOVED_CENTER_1. The center of the second fin is set to

$$\widehat{C}_{f_j^1} := \frac{3}{5}C_{h_j} + \frac{2}{5}C_{h^{f_j^1}},$$

in contrast to the single configuration where the ratio was swapped. At the common edge of the two MOVED_CENTER_3-hexagons the corresponding neighbors coincide and are set to

$$\widehat{C}_{f_j^2} := \frac{9}{20}C_M + \frac{11}{20}C_{h^{f_j^2}},$$

with C_M as the average of the two hexagons' new centers.

Centers of MOVED_CENTER_4-hexagons

The relocated position depends on the number n of adjacent hexagons of type MOVED_CENTER_3. If it is forming a blossom, the center is not relocated. In any other case the most left hexagon h_{k0} of the consecutive MOVED_CENTER_3-hexagons is determined:

$$\begin{aligned}
\widehat{C}_{h_i} &:= \tfrac{3}{5}C_{h_i} + \tfrac{2}{5}C_{h_{k0}} && && (n=1),\\
\widehat{C}_{h_i} &:= \tfrac{5}{4}C_{h_i} - \tfrac{1}{4}C_M, && C_M := \tfrac{1}{2}(C_{h_{k0}} + C_{h_{k1}}) && (n=2),\\
\widehat{C}_{h_i} &:= \tfrac{7}{6}C_{h_i} - \tfrac{1}{6}C_{h_{k1}} && && (n=3),\\
\widehat{C}_{h_i} &:= \tfrac{4}{3}C_{h_i} - \tfrac{1}{3}C_M, && C_M := \tfrac{1}{2}(C_{h_{k1}} + C_{h_{k2}}) && (n=4).
\end{aligned}$$

6 Properties

6.1 Angle Bounds

The minimum angle bound was already explained in the previous section where it was used to relocate the hexagon centers so that each triangle has no angle below that value. The center relocation also took care of right angles, i.e. no triangle is created with an angle greater than 90°. Yet there is still missing an analytic proof of all possible combinations which is omitted due to space constraints. We rather show in Fig. 10 a sample of almost all possible configurations and and angles. It is left to the reader to verify them by computing the single angles of each triangle.

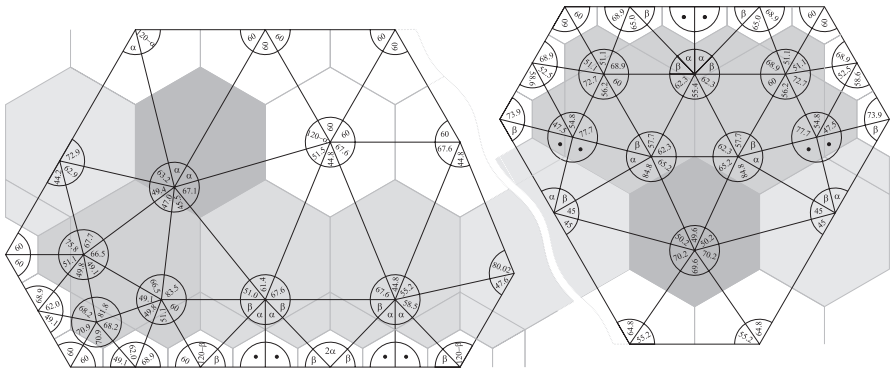

Fig. 10. This figure shows almost all possible hexagon-configurations, missing only MOVED_CENTER_3-strips of size 3 and 4 and 6 (blossoms). α is the minimum angle ($\tan\alpha = \frac{5}{3\sqrt{3}} \quad \Rightarrow \alpha = 43.9$) and β is its counterpart in a right triangle ($\tan\beta = \frac{3\sqrt{3}}{5} \quad \Rightarrow \beta = 46.1$).

6.2 Aspect Ratio

Both angle bounds also limit the aspect ratio of a triangle, i.e. the ratio between the diameter of incircle and radius of the circumcircle. Given a triangle T having both bounds as angles as shown in Fig. 11 and the shortest edge has length s. The radii of the circumcircle and the incircle of a right triangle are defined as

$$\varphi := \frac{c}{2} = \frac{1}{2}\, s \sqrt{1^2 + \left(\frac{3\sqrt{3}}{5}\right)^2} = \frac{1}{2} s \sqrt{\frac{52}{25}}.$$

$$\rho := \frac{a+b-c}{2} = \frac{s\left(1 + \frac{3\sqrt{3}}{5} - \sqrt{\frac{52}{25}}\right)}{2}$$

Thus the aspect ratio of T is $\frac{2\rho}{\varphi} \approx 1.20786$.

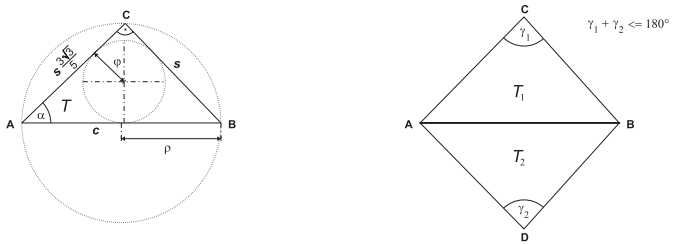

Fig. 11. On the left side the circumcircle reaches its maximum extent at the angle bounds of 43.9° and 90°. This also defines the maximum aspect ratio of the triangle T. The Delaunay property is achieved if for every inner edge (AB) the sum of γ_1 and γ_2 is not greater than 180° as shown on the right side.

6.3 Minimum Triangle Size

The minimum triangle size is determined by the minimum distance among the input points. As shown in Fig. 7 the minimum distance of two hexagons of type `OCCUPIED_2` may share some hexagons of type `FIXED_CENTER_IN_RING`, i.e. they have at least a distance of three full hexagons. Since the point-in-hexagon-test may produce undefined results if an input point lies exactly on an edge between two hexagons or on a vertex among three adjacent hexagons, the vertex is assigned to the first hexagon to be tested. Therefore an additional distance of three full hexagons is added. The shortest edge in the triangular mesh is the one marked in Fig. 8 in the 1-ring of a hexagon h with radius $r_{\min}$ of type `OCCUPIED_1` and has a length of

$$l_{\min} := r_{\min} \sqrt{\left(\frac{3}{2}\frac{\sqrt{3}}{2}\right)^2 + \left(\frac{1}{4}\right)^2} = \frac{r_{\min}}{2}\sqrt{7}.$$

The radius $r_{\min}$ is defined by the minimum distance $d_{\min}$ among the input points and a number of at most six full hexagons in between, i.e.

$$r_{\min} := \frac{d_{\min}}{6 \cdot 2\frac{\sqrt{3}}{2}} \quad \Rightarrow \quad l_{\min} := \frac{d_{\min}\sqrt{7}}{12\sqrt{3}}.$$

6.4 Delaunay Property

The upper angle bound of 90° automatically implies the Delaunay property. Note that a triangular mesh has the Delaunay property if no other vertex lies within the circumcircle of any triangle. This is equivalent that for each inner edge the sum of the apex angles of the adjacent triangles is not greater than 180° (see Fig. 11). Since there is no larger angle than 90° this is obviously satisfied.

7 Triangulating Planar Straight-Line Graphs

Domains are often bounded by a sequence of straight line segments, e.g. to restrict the computation to valid input values or reduce the number of computed elements. For this purpose we present a method to integrate a PSLG in the hexagon subdivision assuming that line segments do not intersect each other. However guaranteed angle bounds are lost at the line segments - but are guaranteed in the interior of the bound region.

7.1 Input Line Segment

Additionally to the input points one or several line segments are associated to the hexagons as shown in Fig. 12. During splitting-/join-operations line segments are re-assigned to the children, according to a hexagon-intersects-line test. Note that a line segment will never be split into several pieces while refining the hexagon mesh. When extracting the dual mesh, edges of the triangles are moved onto the line segment to ensure that all input lines are represented in the final triangular mesh.

7.2 Extended Refinement Rules

All hexagons intersecting a line segment must not be of type `MOVED_CENTER_[1234]`. These kinds of hexagons are neither allowed in the 1-ring of the intersected hexagons. Additionally a hexagon may be intersected at most by two line segments, but only if they are consecutive (see Fig. 12). Therefore the following rules for an intersected hexagon h_i are added to the refinement rules of Sec. 5.2:

- if any neighbor h_j of h_i is of higher level subdivide h_i,
- if any neighbor h_j of h_i is of lower level subdivide h_j,
- if h_i is intersected by two or more non-consecutive line segments subdivide h_i.

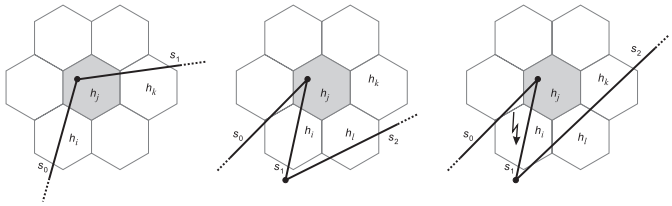

Fig. 12. Similar to input points line segments are assigned to hexagons. On the left side s_0 is assigned to h_i and h_j. s_1 is assigned to h_j and h_k. A hexagon is allowed to have at most two assigned input segments if they are consecutive. On the right side h_i gets all three line segments assigned and must be subdivided.

7.3 Line intersections

The basic idea is to relocate the centers of intersected hexagons. For this consider the three main axis of a hexagon which are orthogonal to the corresponding edges of the hexagon as shown in Fig. 13. Pick the axis forming the largest angle with respect to the line segment and compute the intersection of this axis and the line segment. If the intersection is still within the hexagon the new center location and the corresponding distance to the start point of the segment is kept in a list for that segment.

If the hexagon does not contain an input point but is intersected by two line segments, e.g. at acute input segments, a link to its neighbor intersection is stored in the record list as well. Finally for each line segment the associated list records are sorted with respect to the distance to the start point s_0 of the line segment. Afterwards for each line segment the affected hexagons are traversed according to the sorted intersection entries and their centers are shifted accordingly. For hexagons with two intersections a new point and new triangles have to be inserted into the triangulation as explained in the following subsection.

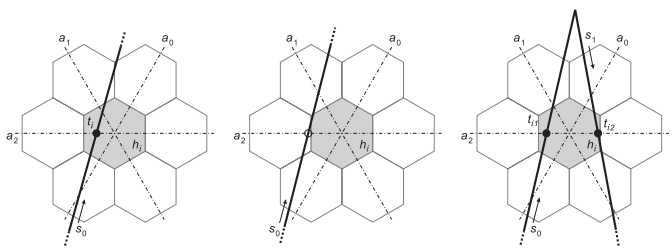

Fig. 13. The axis a_2 of h_i forms the largest angle with line segment s_0. On the left side the intersection lies within h_i and therefore an intersection record with t_i is created. In contrast to the situation in the middle. Here the intersection lies in the neighbor hexagon and the center of h_i will not be relocated. On the right side a second intersection record is created for s_1. For these kinds of hexagons additional triangles have to be added to the dual mesh.

7.4 Adding Triangles

Hexagons intersected only by one line segment are treated as non-intersected ones. However hexagons with two projected centers require additional triangles. The original single center is split into two, creating an inner region in between the two line segments. This *inner region* is filled with two new triangles as seen in Fig. 14. The triangles outside of the *inner region* are built in a regular way by connecting the two centers with the centers of the adjacent hexagons.

Special care is also needed for consecutive projected centers. This happens for example at input points lying outside of the half-sized hexagon in Fig. 14. In this case degenerated triangles would be created. The situation is solved by omitting the projected center of the concerned hexagon if the distance between the moved center and the projected center is greater than the radius r of the hexagon.

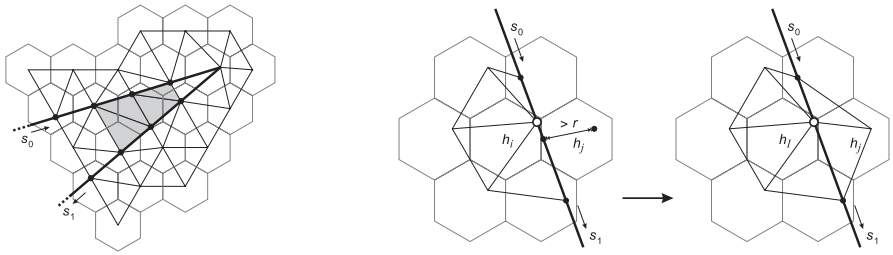

Fig. 14. The original center is split into two, opening a gap between the two line segments. This gap is filled by four new triangles. The edges outside the gap are connected as usual. On the other side in case of almost co-linear segments, degenerated triangles may occur. The projected center is not used if the distance between the moved center of h_j and the projected center is greater than the radius r of h_j.

8 Results

The method was implemented and tested on a regular PC with an Intel Core2 Duo processor at 2.6 GHz using only a single thread. The implementation is a *proof-of-concept*, i.e. it uses a conservative approach for the refinement loop with a runtime complexity of $O(n^2)$ which makes the current implementation unsuitable for larger data sets.

In this section we compare the results of our method with the current version of `Triangle` (v1.6). We have chosen 6 data sets. The first data set – a popular benchmark – consists of two single points with a close distance to each other, 0.02 units in this case. It is located within a hexagon with unit radius. The second data set consists of 100 random points lying on a straight line. In the third data set the 100 points are distributed randomly

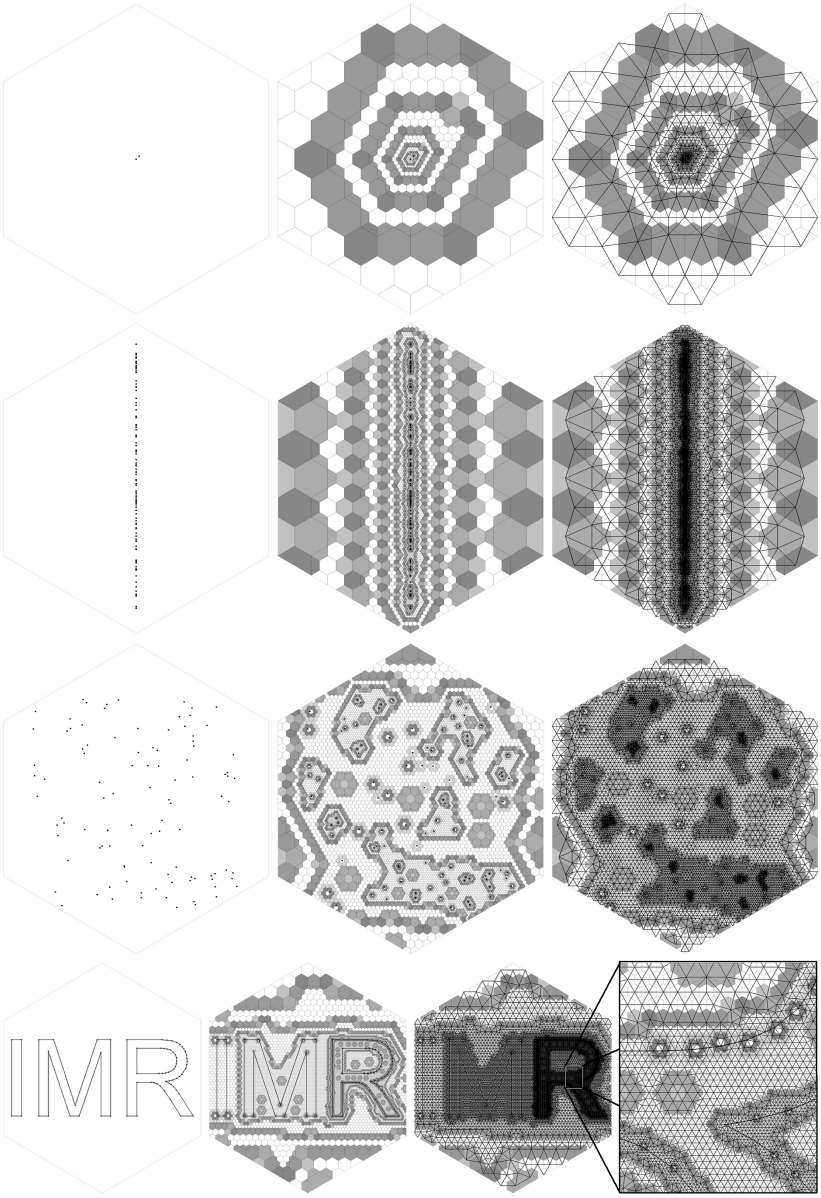

Fig. 15. Four data sets triangulated with our method. In the first row two close points were triangulated. In this case the bounding hexagon was not adapted to the bounding sphere of input points. In the second row a random set of 100 points lying on a straight line is shown, while in the next row the 100 points were randomly distributed. The last row shows the inclusion of line segments. The density of the triangles increases with level-of-detail of the line segments. The right picture shows a close-up of the triangulation.

Table 1. `Triangle` was executed in single precision as well as in double precision while our method uses just single precision. Per trial-and-error the angle was increased by 0.25 degrees until the `Triangle` fails. When adding a PSLG the angle bounds of the hexagon method are lost. For the IMR-letters the minimum angle is 13.7° and the maximum angle is 133.4° near the line segments. In the interior regions the mininum and the maximum angle are 43.9° and 90°. Timings for `Triangle` are below 0.03 seconds for all data sets. The timings of our method are listed in parenthesis behind the angle numbers.

# triangles of	`Triangle` v1.6 (single)	`Triangle` v1.6 (double)	Hexagon Delaunay
2 close pts	122 at 36.75°	131 at 37.00°	684 at 43.9° (0.005 s)
100 pts on line	4890 at 35.00°	5289 at 35.00°	18667 at 43.9° (0.13 s)
100 pts on plane	1149 at 34.50°	1133 at 34.50°	12748 at 43.9° (0.21 s)
`IMR`-letters	2569 at 34.50°	1770 at 34.50°	30551 (1.03 s)
1k pts on plane	10423 at 34.5°	14414 at 34.75°	127729 at 43.9° (30.4 s)
2k pts on plane	19377 at 34.25	19244 at 34.25	268138 at 43.9° (123 s)

on the plane. The fourth data set consists of the sampled letters `IMR` where the level-of-detail increases from letter `I` to letter `R`. Finally we tested the algorithm with slightly larger data sets consisting of 1k and 2k resp. random points. The timings clearly show the quadratic runtime complexity. However it should be possible to implement a more sophisticated refinement loop to reach a runtime complexity of $O(n \log n)$ or slightly above.

The resulting triangulations of the first four data sets are shown in Fig. 15. In Table 1 we have listed the number of triangles for each data set according to the maximum reachable angle bounds. For `Triangle` we increase the angle by 0.25 degrees step by step until the program fails.

9 Conclusion and Future Work

We have presented a new approach of generating meshes of high quality by triangulating point clouds. All angles of the triangles lie within the range of 43.9° and 90°. We have also shown that the triangular meshes have the Delaunay property.

The triangulation is locally adaptive, i.e. regions of interest have more triangles than regions with less input points. By balancing the underlying hexagon-subdivision, the triangles grow quickly from high to low detailed regions. Furthermore the minimum distance among the input points determines the minimum size of the triangles, which may be pre-computed in advance.

In order to limit the domain to certain regions we provided a simple method to include a planar straight-line graph in the triangulation. Although angle bounds may fail directly at these line segments, they are still valid in the interior regions, i.e. one hexagon away from the lines.

Future work will be concentrated on finding a refinement loop implementation with a better runtime complexity than $O(n^2)$ and an integration of a PSLG with guaranteed angle bounds, preferably in the range of 30° and 90°.

An open problem is the extension to 3D space. It is not clear which 3D-polytope will substitute the 2D-hexagon. A promising candidate seems to be the octahedron.

References

1. Baker, B.S., Grosse, E., Rafferty, C.S.: Nonobtuse triangulation of polygons. Discrete Comput. Geom. 3(2), 147–168 (1988)
2. Bern, M., Eppstein, D., Gilbert, J.: Provably good mesh generation. J. Comput. Syst. Sci. 48(3), 384–409 (1994)
3. Chew, L.: Guaranteed-quality triangular meshes. Tech. Rep. TR 89-983. Cornell University (1989)
4. Erten, H., Üngör, A.: Triangulations with locally optimal steiner points. In: SGP 2007: Proceedings of the fifth Eurographics symposium on Geometry processing, Aire-la-Ville, Switzerland, pp. 143–152. Eurographics Association (2007)
5. Sußner, G., Dachsbacher, C., Greiner, G.: Hexagonal LOD for Interactive Terrain Rendering. In: Vision Modeling and Visualization, pp. 437–444 (2005)
6. Neugebauer, F., Diekmann, R.: Improved mesh generation: Not simple but good. In: 5th Int. Meshing roundtable, pp. 257–270. Sandia National Laboratories (1996)
7. Owen, S.J.: A survey of unstructured mesh generation technology. In: IMR, pp. 239–267 (1998)
8. Ruppert, J.: A new and simple algorithm for quality 2-dimensional mesh generation. In: SODA 1993: Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms, Philadelphia, PA, USA, pp. 83–92. Society for Industrial and Applied Mathematics (1993)
9. Shewchuk, J.R.: Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In: FCRC 1996/WACG 1996: Selected papers from the Workshop on Applied Computational Geormetry, Towards Geometric Engineering, London, UK, pp. 203–222. Springer, Heidelberg (1996)
10. Üngör, A.: Off-centers: A new type of steiner points for computing size-optimal quality-guaranteed delaunay triangulations. In: Farach-Colton, M. (ed.) LATIN 2004. LNCS, vol. 2976, pp. 152–161. Springer, Heidelberg (2004)
11. Welzl, E.: Smallest enclosing disks (balls and ellipsoids). In: Maurer, H.A. (ed.) New Results and New Trends in Computer Science. LNCS, vol. 555. Springer, Heidelberg (1991)