3. Map overlay

Introduction. The purpose of this chapter is to show how to store data on a computer that allows us to draw maps and work with them, and how to create new data describing the overlapping of two maps from the data describing these two maps. What is a *map overlay* is shown in the following figure, where we have two maps, red and blue, and their overlay, which is black.

FIGURE 3.1 The overlay of the red map and the blue map creates the black map.

Description of maps - planar subdivisions. The map will be a *planar subdivision* consisting of a finite number of distinguished points, line segments which connect these points and do not intersect in the inner points, and connected areas that are bounded by these segments. The plane subdivisions will be described by means of the so-called *double-connected edge lists.*

Such a list consists of three tables that describe the vertices, edges, and faces. Given a planar subdivision *vertices* are its significant points. *Edges* are line segments of the planar subdivision together with an orientation. The vertex from which the edge emerges is called the *origin.* To every edge there is an edge defined by the same line segment but with the opposite orientation. We call it the *twin* of a given edge. The third term in the description of planar subdivisions is the *face.* It is a connected area bounded by line segments. The face lying to the left of a given edge (what is left and what is right is determined by the edge orientation) is called *adjacent to the edge.* This term allows us to define for a given edge e its successor $\text{next}(e)$ and its predecessor $\text{prev}(e)$. The next edge comes from the end of the edge e, has the same adjacent face as e, and there are no other edges with these properties between it and the edge e. The previous edge is the edge that ends at the origin of the edge e, it also has the same adjacent region, and there are no other edges with the previous two properties between it and the edge e. All these concepts are demonstrated in the following pictures.

FIGURE 3.2 The edge e_1 emerges from the vertex v_1, the edge e_2 is the twin of the edge e_1, the face f_1 is adjacent to the edges e_1, e_3 and e_6. The next edge to the edge e_1 is e_3, not e_6. The previous edge to e_2 is e_7.

Another term we will use is the notion of a cycle. It is the sequence of edges $e_1, e_2, \dots, e_n, e_{n+1}$ such that e_{i+1} is the successor of e_i and $e_{n+1} = e_1$. Each limited face is bounded by an outer boundary, which may be described as a cycle of edges that have a given face as adjacent. We talk about the *outer cycle.* Some faces are also bounded by one or more cycles from inside. We take the orientation of the edges so that the face is adjacent to them. We call these cycles as *inner.*

FIGURE 3.3 The sequence of the edges $e_1, e_2, e_3, e_4, e_5, e_6, e_7$ is an outer cycle of the face f, the sequence e_8, e_9, e_{10} and $e_{11}, e_{12}, e_{13}, e_{14}$ are inner cycles of f.

Now we can describe the double-connected edge list tables in more detail.

Vertices. In the table there is one row for each vertex. This row contains the name of the vertex, its coordinates, and the pointer to an edge coming from the vertex.
Edges. The row in the edge table shows the name of the edge, a pointer to its origin (the point at which it begins), a pointer to its twin, a pointer to the adjacent area, a pointer to the next edge, and a pointer to the previous edge.
Faces. The row in the face table contains the name of the face, a pointer to one edge from the outer boundary cycle (if an outer boundary exists), a pointer to one edge of each inner boundary cycle (if an inner boundary exists).

An example of a simple planar subdivision and its description using a double-connected edge list can be found in Figure 3.4 and in Table 3.1.

FIGURE 3.4. An example of a simple (although a bit atypical) planar subdivision.

TABLE 3.1 In the description replace Czech terms by the English ones:
Vertex, Coordinates, Edge coming from the vertex.
Edge, Origin, Twin, Next, Prev, Adjacent face.
Face, Outer cycle, Inner cycles.

The description using a double-connected edge list can be understood as a minimal description of a planar subdivision, from which we can algorithmically obtain all other data in time $O(n)$, where n is the amount of data required. For example, if we want to find all the edges of the outer cycle of a face f, we take the edge of the outer cycle given in the row for f in the face table, find the successor, find the successor to the successor, and do so until we reach the original edge. Because we use pointers, the successor search can be considered as a step of running time $O(1)$, so the running time for finding all the edges of the outer cycle equals to $O(n)$, where n is the number of edges in this cycle . In a similar way one can solve the following tasks as simple exercises:

- Find all the edges coming out of a given vertex v and list them in the anticlockwise direction.
- Find all the faces that are adjacent to a given vertex v and list them in the anticlockwise direction. (Beware, some faces may repeat.)

Algorithm for map overlay. We are given two planar subdivisions $\mathcal{S}_1$ and $\mathcal{S}_2$, described by the double-connected edge lists $\mathcal{D}(\mathcal{S}_1)$ and $\mathcal{D}(\mathcal{S}_2)$. The goal of our algorithm is to create a double-connected edge list $\mathcal{D}$ for the overlay $\mathcal{O}(\mathcal{S}_1, \mathcal{S}_2)$ of maps $\mathcal{S}_1$ and $\mathcal{S}_2$. In addition, for each new face f in the map overlay, we want to find original faces in $\mathcal{S}_1$ and $\mathcal{S}_2$ in which f lies.

For simplicity, we will talk about the planar subdivision $\mathcal{S}_1$ as about a red map and about the subdivision $\mathcal{S}_2$ as about a blue map. This will correspond to our pictures. The algorithm starts by creating a new list $\mathcal{D}$ by joining the tables for vertices and edges of double-connected edge lists of both maps. Editing the list $\mathcal{D}$ into a double-connected edge list of the resulting overlay has two steps. In the first one, we add and modify records in $\mathcal{D}$ for vertices and edges to match the overlay. In the second part, we create a table for faces.

Vertices and edges. The procedure for creating tables of vertices and edges of the overlay is based on the sweep algorithm which finds the intersections of line segments and was described in detail in the previous chapter. However, this is a relatively laborious modification, so in this chapter we will only describe the algorithm verbally and demonstrate it in the examples. Pseudocode description would be too long. Readers are referred to pseudocodes 4 - 10 in the diploma thesis of Dominik Janků (https://is.muni.cz/auth/th/359588/fi_m/diplomka.pdf).

We start by creating a list of line segments specified by the red and blue edges and the list of their endpoints. For the sake of simplicity, let's assume that the intersection of a red segment and a blue segment is either a point or an empty set or the two lines are identical. In the latter case, we consider both segment to be the only segment that is the wearer of both colours. We assign $L(p)$ and $U(p)$ to each vertex p with the same meaning as in the previous chapter, while we remember the colours of segments. We activate the event queue $\mathcal{Q}$ to include all vertices, and a binary balanced tree that will be blank at the beginning. Just as in the previous chapter, we start the sweep line method.

We describe the algorithm when passing an event p. If the set $C(p)$ of segments, for which p is an inner point, is empty, and the union of $L(p) \cup U(p)$ contains only segments of one colour, we do not add any new rows into the list $\mathcal{D}$ but we can change records concerning the previous and next edges. The way how to do it is shown farther. If $C(p)$ is not empty (i.e. it contains at least one red and one blue segment), each segment of $C(p)$ is divided into two segments with the end point p and these new segments are put into $L(p)$ or $U(p)$. We assign edges to these segments - to those that end in p we assign the original edges, to those starting at p we assign a new edge, see the following picture.

FIGURE 3.5 Original and new notation of segments and edges.

We now make these changes in the list $\mathcal{D}$. In the table for vertices we add a row for the vertex p, in the table for edges we modify the rows for the original edges corresponding to the segments containing the vertex p, and we add rows for the new edges. To determine the new successors and predecessors of these edges, we first arrange the segments from $L(p)$ (using the binary tree for the position of the sweep line over the event p) and then we arrange the segments from $U(p)$ (using the binary tree in the sweep line position under the event p). This will give the segments of $L(p)$ and $U(p)$ around the vertex p an arrangement in the clockwise direction. Concrete implementation of this idea is shown in the example from Figure 3.5

The order of segments from $L(p)$ is $s_u < s'_u$. The order of segments from $U(p)$ is $s'_l < s_l$. This gives the order of the segments around the vertex p in the clockwise direction: s_u, s'_u, s_l, s'_l. From this order we deduce successors and predecessors, for example the successor of the edge e_1 is e_8 and the predecessor of e_5 is e_3. The new row for the vertex p, the modified row for the edge e_1, and the new row for e_5 will look like this:

TABLE 3.2 English description is:
Vertex, Coordinates, Edge coming from the vertex.

TABLE 3.3 English description is:
Edge, Origin, Twin, Next, Prev, Adjacent face
e_1, v_1, e_5, e_8, we leave the original, we leave the original.
e_5, p, e_1, the same as for e_2, e_3, the same as for e_2.

In the table for edges the item regarding the adjacent face is either left unchanged (see the row for e_1 in the previous table) or refers to the original map according to the colour of the edge (see the row for e_5). This procedure gives us complete information about the relationship between vertices and edges of the overlay in the list $\mathcal{D}$.

Faces. Each limited face of the overlay is determined by its outer boundary represented by an outer cycle. Therefore, to identify the overlay faces, we need to find all the cycles and to identify those that are outer ones.

To determine cycles, information about successors that are already captured in the list $\mathcal{D}$ is sufficient. We take an edge e, find its successor, another successor, and so on until we get back to the edge e. Then we take an edge that is not run in this cycle. By the same procedure, we will get another cycle. This is how we get all the cycles.

Now we will show how to distinguish outer and inner cycles. In a given cycle, we take the vertex v, which is leftmost (the smallest in the lexicographic arrangement from the previous chapter). Let the edge e_1 come into the vertex v and let the edge e_2 come out of v. If the angle from e_1 to e_2 measured over the adjacent area is less than 180^o, it is an outer cycle. If this angle is greater than 180^o, the cycle is inner.

FIGURE 3.6 The cycle c_1 with the angle $\alpha < 180^o$ is outer, the cycle c_2 with the angle $\beta > 180^o$ is inner.

Computationally, we decide on this by the sign of the determinant

$$\det \begin{pmatrix} e_{1x} & e_{1y} \\ e_{2x} & e_{2y} \end{pmatrix},$$

where we take e_1 and e_2 as vectors with coordinates (e_{ix}, e_{iy}). If the sign of the determinant is positive, the cycle is outer, if negative, the cycle is inner.

So we know all outer and inner cycles for the overlap. For an unrestricted face, let's introduce a fictional cycle c_∞ as the outer cycle of this face. Now every outer cycle defines just one face. To assign inner cycles to faces, we construct the following graph $\mathcal{G}$. It is a graph in the sense of the theory of graphs, whose vertexes are all cycles. We create the edges of this graph as follows:

We take an inner cycle c_1. Let p be its most left vertex. Let s be the segment closest to the left of p (The information on the nearest left segment is found out in the first step of the algorithm for each event when using the sweep line). The segment s determines an edge e with the same adjacent face as the inner cycle c_1 has. This edge further determines a cycle c_2, which can be inner or outer. If there is no segment to the left of p, we take the fictional outer cycle c_∞ as the cycle c_2. We join the cycles c_1 and c_2 by an edge in the graph $\mathcal{G}$.

FIGURE 3.7 In the graph $\mathcal{G}$ the inner cycle c_1 is connected by an edge with the inner cycle c_2 and this is connected by an edge with the outer cycle c_3.

In the graph $\mathcal{G}$ we determine the components of connectivity. There is just one outer cycle in each component, which determines one face of the overlay. Therefore, there is a clear correspondence between the components of the graph and the faces. The inner cycles in a given component of the graph are the inner cycles of the corresponding face. With knowledge of the outer cycle and all inner cycles of each region, we can now fill in the table for faces in the list $\mathcal{D}$. Further, to each edge we find the cycle in which it lies, and to this cycle we look out its face. This is the adjacent face to this edge. This completes the last item in the edge table.

FIGURE 3.8 The cycles in a planar subdivision and the graph $\mathcal{G}$ which they determine. The components of connectivity correspond to the faces f_1, f_2, f_4, f_5, f_∞, respectively.

Identification of original faces. We still have to find for each face f of the overlay a face f_1 of the red map and a face f_2 of the blue map in which f lies. There are two possibilities. If there are edges of both colours in the boundary cycles of the face f, then the adjacent face to a red edge is f_1, and the adjacent face to a blue edge is f_2.

FIGURE 3.9 Specification of the original faces f_1 and f_2 for the face f of the overlay.

If all boundary cycles of the face f are single colour, let us say red, we determine f_1 as in the previous case. If f is unlimited, f_2 is the unlimited face in the blue map. For the bounded face f, we specify the face f_2 as follows: Take the most left vertex of the outer cycle of the face f. We find the closest segment to the left of this vertex. This segment determines the inner or outer cycle of a face f' which is adjacent to f. If at least one edge of the cycles of the face f' is blue, we specify f_2 as the face adjacent to this edge in the blue map. If f' is unlimited, the face f_2 is the unlimited face in the blue map. If f' is bounded and if all edges of its cycles are red, let us proceed with the same procedure, only f is now replaced f'. The appropriate blue face for f' is f_2 for the original face f.

FIGURE 3.10 Specification of the original faces f_1 and f_2 for the face f of the overlay.

Intersections and unions of polygons. The previous map overlay algorithm can be used to find the intersection, the union, or the difference of two generally non-convex polygons. A simply connected polygon is such a polygon that is connected and does not have "holes"in its interior, i.e. each closed curve can be shrunk continuously without breaking into a point inside the polygon. A simply connected polygon thus determines a planar subdivision with two faces, inner and external. The intersection of two polygons consists of those faces of the overlay of their planar subdivisions, which lie in the bounded faces of both subdivisions. Likewise, the union of polygons is made up of all faces of the overlay which are bounded. See the following pictures.

FIGURE 3.11 The intersection of the red and blue polygons is the union of the faces f_2 a f_4. The union of polygons is described as the union of the faces f_0 to f_{10}.

Pseudocodes of the algorithm, its implementation and running time. The algorithm can be briefly summarized as follows:

ALGORITHM 6 from pseudo.pdf
Corrections:

(1) Instead of half-edge write only edge.
(2) Replace Section 2.1 by Chapter 2.
(3) Remove the text in brackets: (This was explained for the case where an edge of ... passes through a vertex of ...)
(4) Replace OuterComponent(f) by Outer cycle, InnerComponents(f) by Inner cycle, InnerFace by Adjacent face.

Detailed pseudocodes can be found in the diploma thesis Map Overlay written by Dominik Janků (in Czech)

https://is.muni.cz/auth/th/359588/fi_m/diplomka.pdf

Lines 1 a 2 of the given pseudocode are realized by Pseudocodes 4 to 10 of the diploma thesis. The search of cycles in line 4 is described by Pseudocode 13, inner and outer cycles are found using Pseudocodes 11 and 12. Components of connectivity of the graph $\mathcal{G}$ in line 5 are constructed using Pseudocode 14, lines 6 and 7 of our pseudocode are in detail described by Pseudocode 16 and line 8 by Pseudocode 15.

A part of the diploma thesis is also an implementation of the algorithm. Its description is in the last chapter of the thesis, the necessary files are compressed in the cdrom.zip file.

https://is.muni.cz/auth/th/359588/fi_m/fakulta=1431

The complexity of the algorithm is characterized by the following theorem, which we give without proof.

Theorem 3.1. *Let $\mathcal{S}_1$ be a planar subdivision of complexity n_1 and let $\mathcal{S}_2$ be a planar subdivision of complexity n_2. Let $n = n_1 + n_2$. Then the running time of the algorithm that builds a double-connected edge list for the overlay of these subdivisions is*

$$O((n+k)\log n)$$

where k is the complexity of the overlay.