Algovision
Geometric Algorithms: Voronoi Diagram

Sites

The first scene of the tour shows an input for the problem we are going to study - a finite set of points in the plane. We will call the given points *sites* to distinguish them from the other points of the plane.

In the control bar, there is a menu that defines the way of working with the mouse

Move Site (a keyboard shorthand 's') - in this mode sites can be dragged by the mouse;

Move Screen (a keyboard shorthand 'x') the whole plane could be moved by the mouse;

Insert or Delete (a keyboard shorthand 'i') clicking into an empty space creates a new site, clicking a site destroys it.

Clicking the button $\boxed{\text{New Input}}$ creates a new set of sites that contains as many sites as it is given in the field $\boxed{\text{\# of Sites}}$.

The buttons $\boxed{\text{Zoom Out}}$ and $\boxed{\text{Zoom In}}$ have the obvious function.

Voronoi Diagram

The second scene shows a Voronoi Diagram of the given set of sites: each site s is surrounded by an area that consists of the point p of the plane such that their distance from s is smaller than their distance to any other site.

The boundaries of areas, where two areas meet, are composed of points that have two nearest sites in the same distance.

There are also points, where three areas meet - the points that have three nearest sites.

In general, there can exist points, where more than three areas meet (similarly as the Four Corners point that touches Colorado, New Mexico, Arizona and Utah), but it is unlikely that a randomly generated set of sites would generate such points.

In order to make the partition clearly visible, the areas have different colors, but colors can be switched off by the checkbox $\boxed{\text{Domain Colors}}$. Control elements of the preceding scene are also available. It is recommended to move certain site and to see how the Voronoi Diagram changes. You can also try inserting or deleting sites or to create Voronoi Diagram of a larger set of sites.

Animation

The scene shows the animation of an algorithm for finding Voronoi Diagram in the plane, which was discovered by Steve Fortune. The algorithm is so called sweep line algorithm: a horizontal line, which is given by the boundary between the blue area above and the green area below the line moves up. You can halt the animation by the button $\boxed{\text{Halt}}$ (a keyboard shorthand h), restart again in both directions by the buttons $\boxed{\text{Forward}}$ and $\boxed{\text{Backward}}$, finish by the button $\boxed{\text{Finish}}$ and reset by the button $\boxed{\text{Reset}}$.

The mouse function menu has now two choices:

Move Sweepline (a keyboard shorthand 'y') - when the mouse is down, the sweepline follows the cursor (even if the animation is halted);

Move Screen (a keyboard shorthand 'x') the whole plane could be moved by the mouse.

The lower boundary of the green domain below the sweepline is composed of arcs. In literature the curve representing the boundary is called a *beachline*, because it is said to remind a line separating a beach from the sea. Observe that the points where the arcs of the beachline meet draw boundary of Voronoi Diagram domains.

The beachline info in a white rectangle in the upper left corner of the window says how many arcs are on the beachline (some of them invisible, out of the window), the y-coordinate of the sweepline. The last part of the info will be explained later. The info can be switched off by the checkbox $\boxed{\text{Beachline Info}}$.

What is going on in the animation will be explained in the two following scenes.

Cones

The definition of Voronoi Diagram will now be explained in the three-dimension space.

Imagine that the plane containing sites is embedded into the 3D space as a horizontal plane. The plane is not visible in the scene, but its position can be deduced from the locations of sites that are visible.

From each site construct a cone that has its summit in the site, a vertical axis and the surface halflines starting in the summit (the site) have the slope 45 degrees. At the beginning, we observe the situation from above, looking vertically down to the cone surfaces. Only visible parts of the cones are shown, the other are hidden by other cones.

With the mouse button down, move the cursor in the window. This is the way to move the surface composed of the visible parts of the cones. Move first the cursor up - no more you see the surface vertically; try to make all sites appear in the same height, i.e., to look at the surface from side, horizontally. At this moment you feel where the plane containing sites is located.

It turned out that it is easier to understand what is shown if only a section of the surface is shown - the part that was visible in the initial vertical view. This is why, when observing the surface horizontally, from side, you can see also the reverse side of the surface, which is red. The section can be made wider or narrower by the buttons $\boxed{\text{Wider}}$ and $\boxed{\text{Narrower}}$. It can also be recommended to zoom the view in or out.

Now, click the button $\boxed{\text{Vertical}}$. We return back to the original vertical view of the configuration. And I hope you will see that the curves defined by intersection of cone surfaces, which in fact are hyperbolic arcs in 3D, now look like line segments and, what is important, they *delimit the Voronoi Diagram domains.* Keep clicking $\boxed{\text{New Input}}$ to see that the planar Voronoi Diagram is in fact a planar projection of a very natural 3D configuration.

The scene also brings a tool for a graphical proof of the statement of the previous paragraph. Move again the surface so that you see it in the direction somewhere between the vertical and the horizontal view. Select “3D Triangle” in the mouse function menu (or type ‘t’). Press the mouse button and move the cursor over the surface. A triangle appears that has one vertex in a site (the summit of the cone that is scanned by the cursor), and the other vertex is on the cone surface, determined by the cursor. Even though this is visible only in certain directions, the green segment is horizontal, the yellow segment is vertical, the angle between them is 90 degrees.

When the cursor is over the curve where two cones meet, two triangles appear, and you can even get three triangles after moving to a point where three cone surfaces meet. Prepare view so that 2 or 3 triangles are visible. Release the mouse button, switch back to the $\boxed{\text{Move Cones}}$ mouse function (or type ‘c’). If you have done everything carefully, the triangles are still visible. Now, move the surface to observe triangles from different angles.

We already know that the angle between the green segment and the yellow segment is 90 degrees. Since the slope of the red segment is 45 degrees, both the angle between the red and the green segment and the angle between the red and the yellow segment are 45 degrees. Moreover all triangles have the yellow segment in common. All this implies that the triangles are identical, in particular, the lengths of the green segments are the same.

Click again the button $\boxed{\text{Vertical}}$ and wait until you see the surface vertically. The planar distances from the sites (cone summits) to the point, where the green segments meet, are the same, and therefore a meeting point of the 3D cones appears now as a boundary point of the planar Voronoi Diagram.

Sweep Plane

The present scene uses the 3D embedding of the previous scene to explain the idea of the Fortune algorithm.

The mountain of cones is now swept by an inclined green plane (only a part of the plane is shown). The inclination of the plane is the same as the slope of the surface halflines of the cones. Under the selection $\boxed{\text{Move Cones}}$, arrange the 3D configuration so you can see what is going on. Start the animation by $\boxed{\text{Forward}}$ or $\boxed{\text{Backward}}$ (or switch to the $\boxed{\text{Move Sweepline}}$ mouse function and, having the mouse button down, move the mouse up and down). The mountain surface below the sweep plane is gray, the surface above the sweep plain is blue.

Having the animation running, click the button $\boxed{\text{Vertical}}$. When the surface is eventually observed vertically, you certainly realize that the animation in the third scene was a planar projection of the 3D configuration that you see right now. At this moment it is clear why intersections of arcs were drawing the boundaries of Voronoi Diagram in the third scene: the sweep plane just discloses more and more valleys, where the cone surfaces intersect, and we already know that the planar projections of valleys are Voronoi boundaries. The sweep line of the planar animation in the scene "Animation" is in fact (a projection of) the intersection of the horizontal plane containing sites and the sweep plane. The visualization does not show the whole sweep plane, but it rectangular part, and said intersection is a line is the extension of the upper side of the sweep plane rectangle, as it is shown in the window.

Now, it is clear that any arc of the beachline is a planar projection of the intersection of the sweep plane with certain cone. In this way, any arc is associated with the site which is the summit of the cone that determines the arc. (Note that one site can represent *more* beachline arcs.) Since the beachline (scanned left-to-right) is a sequence of arcs, it can be represented as a *sequence of sites* that represent the arcs. This is how the beachline is represented in the beachline info box in the upper left corner of the window, using site numbering that can be shown by checking the checkbox $\boxed{\text{Site Numbers}}$ [remark: the checkbox Site Numbers is not yet implemented].

Single Parabola

We are back from 3D; the cone visualization gives a very good and inspiring background intuition, but it is also possible to explain Fortune's algorithm using purely planar terms. We know that beachline arcs are parts of parabolae obtain as projection into the site plane of sections of the cones by the sweep plane. However, a parabola can also be defined as a collection of plane points, having the same distance from a given point (the focus of the parabola) and a given line (the directrix of the parabola). It is not surprising that in our case, the directrix is the sweep line, and foci are particular sites that have already been swept (i.e., shown below the sweep line).

Click to any point in the window: any beachline arc determines a vertical stripe within the window rectangle, and the click highlights the parabola of the arc, the stripe of which has been clicked. The focus of the parabola is highlighted as well [remark: this is not yet implemented].

The highlighted parabola is a collection of the points that have the same distance to the focal site and the directrix. The area under the curve (magenta color) consists of points that are closer to the focus than to the directrix, while the area above the curve (green or blue) contains points that are closer to the directrix than to the focus.

Let us consider a point p of the beachline, where two arcs with foci f_1 and f_2 meet. Denote by δ_i, $i = 1, 2$, the distance of p and the site f_i, and denote by δ the distance of p from the directrix. Since p belongs to the first parabola, $\delta_1 = \delta$. Since p belongs to the second parabola, $\delta_2 = \delta$. This together gives $\delta_1 = \delta_2$, in other words the distances of the point [to both sites are the same, p belongs to a boundary between Voronoi domains of sites f_1 and f_2. This is why meeting points of beachline arcs draw the Voronoi Diagram in the “Animation” scene.

All Parabolae

We assume that sites are “known”, if they have already been swept by the sweep line, and “unknown” if they still belong to the dark blue region above the sweep line. The beachline is composed of arcs that are parts of parabolae with know foci (sites). The present scene highlights automatically parabolae of all “known” sites.

The magenta domain is the collection of points of the plane that are closer to some known site than to the sweepline. This means that points in the magenta area are closer to some known site than to *any* unknown site, because unknown sites are located *above* the sweep plane. Therefore it is already possible to tell which is (are) the nearest site(s) of any point in the magenta area, and to determine the shape of the Voronoi domains in the magenta area.

The magenta area consists of points that are planar projections of cone surface points that are *above* the sweep plane in the scene “Sweep Plane”.

The domain above the beachline (green and blue) consists of points that are closer to the directrix than to any known site (and also points that correspond to cone surface points *below* the sweep plane in the scene “Sweep Plane”). Given a point p in the green area, there might exist a yet unknown site just above the sweep line that is closer to p than any already known site. Therefore it is not yet possible to tell anything about Voronoi domains and their boundaries in the green and the blue domains.

Events

In order to be able to determine Voronoi Diagram of a given set of sites, we do not need to know all points of Voronoi domain boundary segments; it is sufficient to know just their end-point that obviously give all information about the segments. Planar animation of Fortune's algorithm reveals that such segment end-points appear on the beachline as meeting points of two neighboring arcs at moments, when either a new arc appears on the beachline, or an existing arc disappears from the beachline. The moments during the sweeping action, when something like this happens, are called *events*.

Start animation of the algorithm again - in this scene, the animation halts every time an event occurs.

There are two kinds of events:

A site event occurs in a moment, when the sweep line meets an unknown site. A new "arc" is added to the beachline; we are using parenthesis, because if fact the new object is a vertical halfline, starting in the newly discovered site, which, however, can be regarded as a parabola of zero width (which, after an infinitesimally small moment becomes a true parabola of a very small "width"). The degenerated arc can be viewed as a parabola with the focus at the directrix.
You can see that the degenerated arc splits the arc of the beachline that is below the newly known site into two parts and a new Voronoi boundary segment appears (being drawn "from the middle").
In the 3D visualization, a site event occurs when the sweep plane touches a new cone; the degenerated arc is (a planar projection of) the halfline, where the cone and the sweep plane meet.

A circle event is how beachline arcs disappear. Some arcs have aggressive neighbors and are getting shorter as the beachline moves up, being eventually eliminated by the neighbors. At this moment, two Voronoi domain segments finishes by reaching their end points (the segments being drawn by the meeting point of the disappeared arc and its respective neighbors) and a new segments starts (being drawn by former neighbors of the disappeared arc that became neighbors at the moment of the circle event. (Why this type of event is called "circle" will be explained later). When a circle even occurs, the point when this happened is marked by a small red dot [remark: this feature not yet implemented].

In some rare occasions, events could look differently; this will be discussed in the scene "Special Cases".

Site Events

It is easy to determine when (i.e., at which position of the beachline) site events will occur: the position of the sweep line is determined by the y-coordinate of the location of the particular site. What a programmer has to implement in such a moment:

- The site that has been hit by the sweepline is put into the list of known sites;
- The beachline arc that is below the new known site is identified, split into two parts, and a new arc corresponding to the new site is inserted between them. Observe this activity in the beachline info box: as already said, the beachline is represented as a sequence of foci of the beachline arcs. If n is the new site, and o is the focus of the arc split in the above mentioned way, then $\ldots - o - \ldots$ in the beachline description is replaced by $\ldots - o - n - o - \ldots$.
- A new Voronoi domain boundary segment is created; we do not know any of its end-points, but we know one its point (the point where the vertical halfline from the new site intersect the beachline arc), and we know its direction (the axis of the segment connected the new site and the focus of the split arc).

In the next scene we will learn other measures that must be taken during a site event.

Circle Events

In this scene, animation of Fortune's algorithm halts automatically every time when a circle even occurs.

When Fortune's algorithm starts, we know for sure times and locations of all site events that are going to occur, because this is determined by the input site set in a very straightforward way.

On the other hand, predicting site events is very involved. When an event occurs, we might find that some circle event is likely to occur in the future. This is why a code implementing Fortune's algorithm advantageously maintain a *calendar of events*. The calendar is a priority queue, where events form a sequence ordered by the time when they occur (i.e., the y-coordinate of the sweep line). Let us remind that some circle events that have been put into the calendar would not occur if, during the computation, another event occurs making the intended circle event record invalid.

How to find that a circle event is likely to occur in the future:

- in the window, prepare a situation that certain arc is going to disappear soon because of a circle event. You can, e.g., let the animation run until it halts due to a circle event, and then push the beachline slightly back (either by clicking $\boxed{\text{Backward}}$ and then $\boxed{\text{Halt}}$ soon afterwards, or by selecting the mouse function $\boxed{\text{Move Sweepline}}$ and moving the sweepline slightly back manually).

- then choose the mouse function $\boxed{\text{Select Arc}}$ (shorthand 'a') in the control bar and click the arc that is going to disappear. A circle appears.

- Click the button $\boxed{\text{Forward}}$. The animation starts, and should halt at the moment when the selected cycle disappeared.

- Denote the arc that have just disappeared by a, its left arc neighbor by a_ℓ and the right neighbor by a_r, and denote their respective foci by f, f_ℓ and f_r. Moreover, denote the point, where the arc a has disappeared, by c. I assume you expect that the point c is the center of the circle you see, and this is correct:
  the point c belongs obviously to both neighboring arcs a_ℓ and a_r, but it is still a point of the middle arc a that has just degenerated to a single point equal to c. Denote by δ_ℓ, δ and δ_r the distances of c from the respective foci f_ℓ, f, and f_r, and by δ_d the distance of c from the sweep line (the directrix of all arcs a_ℓ, a, and a_r).
  Since c belongs to the left neighbor arc a_ℓ with the focus f_ℓ, we get $\delta_\ell = \delta_d$. Similarly, because c belongs to the right arc a_r, it is $\delta_r = \delta_d$. And finally, because c is the (now degenerated) middle arc a, we have $\delta = \delta_d$. This implies that $\delta_\ell = \delta = \delta_r = \delta_d$, in other words the circle with the center c and the radius δ passes through all f_ℓ, f, and f_r, and, moreover, the directrix touches the circle at its topmost point.

Therefore if we see a beachline arc a that is going to disappear, consider the circle that passes through the focus of the arc and the foci of its arc neighbors. The center of the circle tells us *where* the arc is going to disappear, and the y-coordinate of the top of the circle says *when* it happens. Using the latter value, we will insert the record of the future possible record to the calendar of events.

The other actions during a circle event are

- recording end-points of two Voronoi domain boundary segments that end in the center of the circle (drawn by the meeting points of the disappeared arc with its neighbors);

- creating a record of a new Voronoi domain boundary segment that starts in the circle center and will be drawn by the meeting point of the previous neighbors of the disappeared arc.

Sometimes it happens that a scheduled circle event would not occur. It may happen because the arc that was bound to be eliminated during the circle event is split before this by a degenerated arc created in the site event. In such a case the record for the future circle event must be removed from the calendar, but the code has to check whether the sections of the original arc (that have the new arc generated during the site event as one of the neighbors) are not subjected to future circle events, and possibly inserted into the calendar of events.

The role of a circle when analyzing an event destroying an arc explains the name that was given to this type of events.

Special Cases

This scene brings several special configurations of sites, some of them must be processed in a special way:

Example 1 There are two lowermost sites. In this case the first event is a double site event with two parabolae degenerated to halflines. When the beachline starts to move up, the boundary between Voronoi domains of the sites is drawn “from the infinity”. This case should be treated in a special way.

Example 2 Another variant of the previous case.

Example 3 This case seems to be another variant of the previous two, but it is more complicated. During the site event corresponding to the upper site, the halfline representing a degenerated arc of the new site does not split any arc of the beachline into two arcs, but the meeting point of two arcs. This is a special case of a site event that should be treated separately. In this case, only one boundary segment ends, but two new start exactly in the hit beachline point.

Example 4 The last event is a combination of a site event and a circle event that occur in the same time and act in the same location. Process the circle event first and then the site event (the special variant from the previous example).

Example 5 First, two site events occur in the same time, later two circle events occur in the same time and the same location. The simultaneous events can be processed in any order in the standard way.

Example 6 All sites are on one straight line. This is the only case when Voronoi Diagram is composed of two halfplanes and a collection of parallel infinite stripes between them. The boundary lines never reach any end and do not meet. Despite of the strange result, processing is a sequence of standard site events.

Example 7 Once more all sites in one line, but the line is horizontal. This is a generalization of Examples 1, 2, 3 that needs a special treatment.

Example 8 A complicated collection of sites, all belonging to the same circle. After complicated preparation, 9 circle events and one site events occur in the same time and in the same locations. As a consequence, “Twelve Corner” point is created.

Arc Search

It seems simple to find the arc of the beachline that is below the new site during the site event. However, the beachline can consist of very large number of arcs and simple methods, like scanning the beachline left-to-right could be time consuming. In such a case we can build a binary search tree above the beachline in such a way that leafs represent beachline arcs and internal nodes of the tree are advantageously above arc meeting points.

Of course, the tree must be dynamically maintained using standard techniques and advantageously balanced (e.g., an AVL-tree or a red-black tree) to allow efficient computing. Using such a tree, arc searching can be finished in the time proportional to the logarithm of the number of the beachline arcs.

For simplicity, implementation details are omitted.

Nearest Site

One of the main application of a Voronoi Diagram is finding the nearest site in the plane.

Suppose that a set S of sites is given and fixed, and then we have to process a long (possibly infinite) sequence of the following queries:
Given a point p of the plane, find the site $s \in S$ that minimizes the distance p and s.

A simple algorithm is to find distances of p to all sites in S and to determine the minimum. However, the time to answer one query is proportional to the size of the set S. It is possible to preprocess the set S so that the subsequent queries are answered in time proportional lo the logarithm of the size of S. The preprocessing uses Voronoi Diagram or the set in the following way:

Change the choice $\boxed{\text{No Trapezoids}}$ in the control bar to $\boxed{\text{Vertical Lines}}$ (it is better to uncheck $\boxed{\text{Domain Colors}}$). Vertical lines passing though all "Three Corner" points of Voronoi Diagram appear.

The lines partition the plane into parallel vertical stripes. Change the choice $\boxed{\text{Vertical Lines}}$ to $\boxed{\text{One Stripe}}$ an keep clicking to different points in the window. Each time the selected vertical stripe is highlighted. Moreover, you can see that Voronoi domain boundary segments partition the stripe into trapezoids and triangles that are distinguished by two colors (a triangle is in fact a trapezoid with one base of zero length). Finally, select $\boxed{\text{All Stripes}}$. All stripes are highlighted in the above way; different pairs of colors are used for neighboring stripes.

The key observation is that each trapezoid or triangle is contained in a Voronoi domain of one site. Since the set of sites is fixed, the preprocessing associates each trapezoid or triangle with the (uniquely determined) nearest site. Thus, in order to process a nearest site query, it is sufficient to determine the trapezoid that contains the query point and then read the result.

First, the stripe is located that contain the query point. Since the set sites as well as the set of stripes are fixed, the stripes can be arranged in an array ordered by x-coordinates. First, look at the stripe in the middle of the array. If the query point is in the middle stripe, we are done. If not, it is easy to determine if the point is left or right to the middle stripe, thus halving the number of stripes that can contain the query point. Repeating the halving procedure in the standard way, the stripe we are looking for can be found in the time proportional to the logarithm of the number of stripes.

Then we proceed in the same way, using halving, in the stripe to find the trapezoid containing the query point. This procedure is a bit more complicated, but it also finishes in the logarithmic time.

Delaunay

Since the definition of Delaunay Triangulation for a given finite set S of points (sites) in the plane is a bit complicated and out of the scope of the present algorithm tour, the reader is asked to find it in external sources like Wikipedia, where the importance and applications are explained as well.

Given a Voronoi diagram of S, it is known that it is easy to get Delaunay Triangulation as follows: connect two sites if and only if their domains (that are polygons) have a common side.

Check $\boxed{\text{Delaunay}}$ to see Delaunay Triangulation and check its connection to Voronoi Diagram. Uncheck $\boxed{\text{Voronoi}}$ to see just the triangulation.

Let us mention that a triangulation is obtained only if Voronoi Diagram has no "Four Corner" or higher corner points. If there is a Four Corner point, then the sites that own domains that meet in the corner point are on the same circle (a very unlikely situation for a random set of sites) and form a quadrangle instead of a triangle. In this case Delaunay Triangulation is not unique, and the quadrangle in the "triangulation" obtained from Voronoi Diagram can be transformed into two triangles in two ways by connecting one pair of its opposite vertices. In the presence of higher corner points the procedure is similar.

Original Fortune

The original Fortune's algorithm was different from the way it is usually explained now, but it is easy to explain their relation, because they are obtained by viewing the 3D embedding (as shown in scenes "Cones" and "Sweep Plane") in different directions.

The scene is initialized in the same way as in the scene 'Sweep Plane". Using the mouse, cone mountain can be rotated and inclined. Go back to the vertical view by clicking the button $\boxed{\text{Vertical}}$. Now, click $\boxed{\text{Fortune}}$. We still observe the same 3D scene, but in a different direction: now, the observation angle is 45 degrees, and we are looking in the direction that is parallel with the sweep plane.

It follows that we observe the sweep plane not as a plane that covers all observation scope, but as a simple line. As a consequence, this angle of view allow us to formulate the algorithm as a simple sweep line algorithm and there is no need of a beachline, because the beachline is also a part of the 3D sweep plane, and hence under the new view direction it coincides with the sweep line.

Another advantage of this view is that any site seems to sit at the bottom of its (deformed) domain, and therefore when we start to construct the domain, the site is the first point that is hit by the sweep line.

On the other hand, visible parts of cones are not any more polygons, and Voronoi boundary segments are not any more line segments but complicated curves. However, this is not a problem, because we just need to determine their end points and after transforming the end points to the vertical projections, we simply connect them in a linear way to get Voronoi boundary segments.