Triangulación de Polígonos

comp-420

Triangulación de polígonos

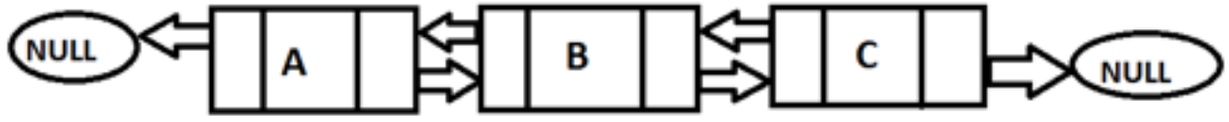
- Triangulación: descomposición de un polígono simple en triángulos.
- Cualquier triangulación de un polígono simple con n vertices tiene $n-2$ triángulos.
- El algoritmo más simple, llamado de corte de orejas (ear clipping) es de orden $O(n^2)$.
- El método antes visto para particionar polígonos en partes monotonas es útil porque podemos triangularlas con complejidad $O(n)$.

Ear clipping triangulation

- Una oreja de un polígono es un triángulo formado por tres vertices consecutivos dentro del cual no hay otro vértice del polígono.
- El segmento de recta entre v_{i-1} e v_{i+1} se llama diagonal.
- El vértice v_i se llama punta de la oreja.
- Un triángulo consiste en una sola oreja aunque se puede poner como punta de la oreja cualquiera de los tres vértices.
- Un polígono de cuatro o más lados siempre tiene al menos dos orejas que no traslapan (probado por *G.H. Meisters, Polygons have ears, Amer. Math. Monthly*). Esto sugiere un algoritmo recursivo para la triangulación.
- Si podemos encontrar una oreja en un polígono con $n \geq 4$ vertices y removerla, tendremos un polígono de n-1 vertices y podemos repetir el proceso.
- Una implementacion inmediata tomaría $O(n^3)$.

D. Eberly. Geometric Tools. (www.geometrictools.com)

Ear clipping triangulation



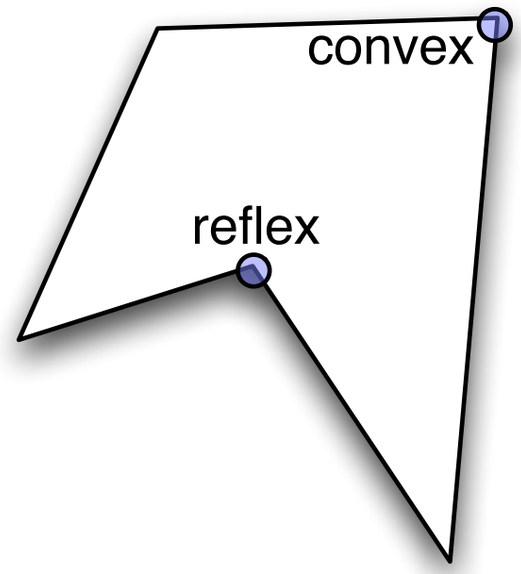
Doubly Linked List (Idlebrains.org)

- Almacenar el polígono como una lista doblemente ligada de manera a poder eliminar rapidamente las puntas de las orejas:
  - construir la lista toma O(n).
- Iterar sobre los vertices para encontrar orejas.
  - para cada vértice v_i y su triángulo correspondiente (v_{i-1}, v_i, v_{i+1}) (el indexado es módulo n por lo que $v_n=v_0$ y $v_{-1} = v_{n-1}$) probar todos los otros vértices para ver si hay alguno dentro del triángulo.
  - Si no hay ninguno dentro del triángulo es una oreja, si hay al menos uno no lo es.

D. Eberly. Geometric Tools. (www.geometrictools.com)

Ear clipping triangulation

- Es suficiente considerar solamente los vertices reflex para ver si estan contenidos en el triangulo.
  - vertice reflex - el ángulo interior formado entre las dos aristas que lo comparten es mayor a 180 grados.
  - vertex convex - el ángulo interior formado entre las dos aristas que lo comparten es menor a 180 grados.
- La estructura de datos de polígono mantiene 4 listas doblemente ligadas simultáneamente:
  - los vertices del polígono en una lista ligada cíclica,
  - los vertices convex en una lista lineal,
  - los vertices reflex en una lista lineal,
  - las puntas de orejas en una lista ligada cíclica.



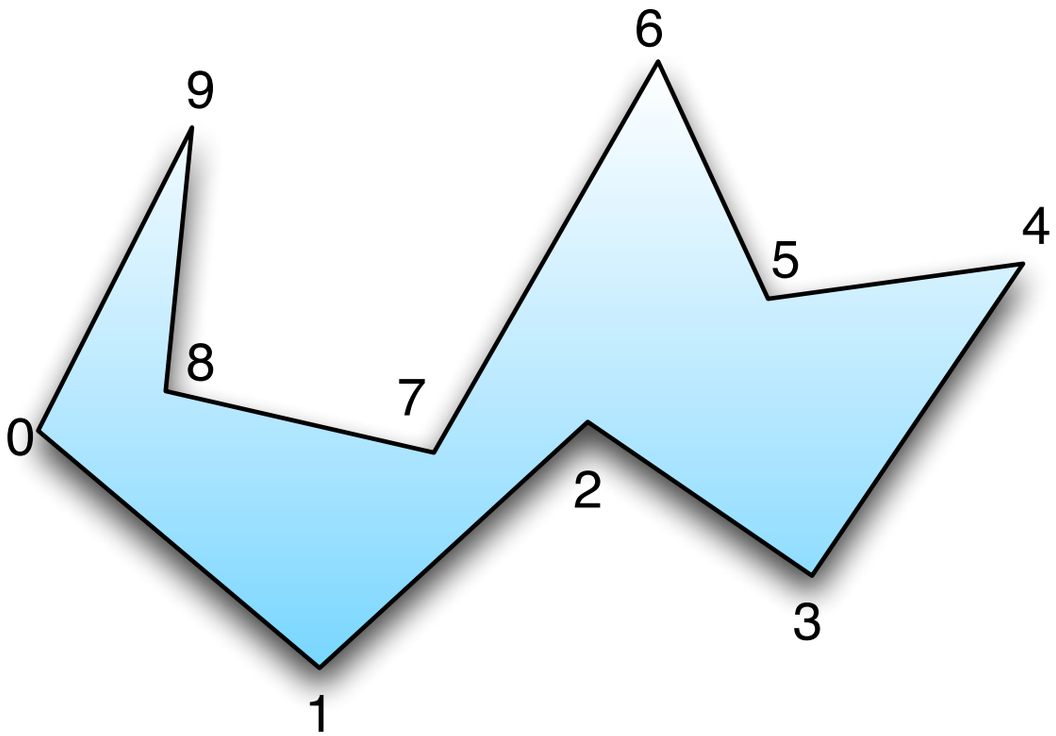
D. Eberly. Geometric Tools. (www.geometrictools.com)

Ear clipping triangulation

- Una vez construida la lista inicial de vertices reflex y orejas, se eliminan las orejas una a una.
- si v_i es una oreja que se elimina, la configuracion de la arista en los vertices adyacentes puede cambiar. Si el vertice adyacente es...
  - convexo es facil convencerse que va a seguir siendo convexo.
  - oreja, no se queda oreja necesariamente despues de eliminar a v_i.
  - reflex, es posible que se vuelva convex o posiblemente oreja después de eliminar a v_i.
- Despues de eliminar a vi, si un vertice adyacente es convex, probar si es una oreja iterando sobre los vertices reflex y probando si estan contenidos en el triangulo de ese vertex. Hay O(n) orejas, hay que verificar O(n) veces por oreja. El proceso total de eliminación de orejas toma $O(n^2)$.

D. Eberly. Geometric Tools. (www.geometrictools.com)

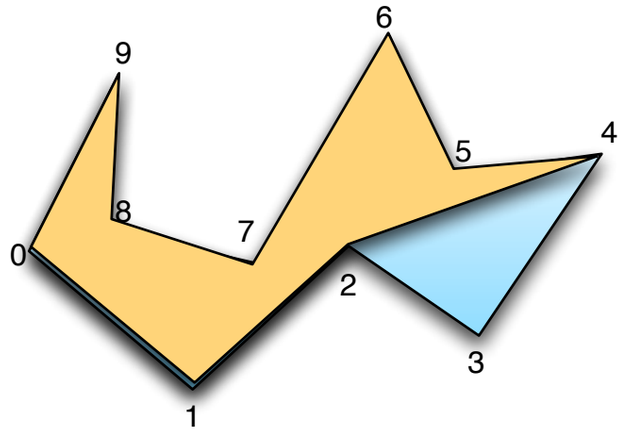
Ear clipping triangulation



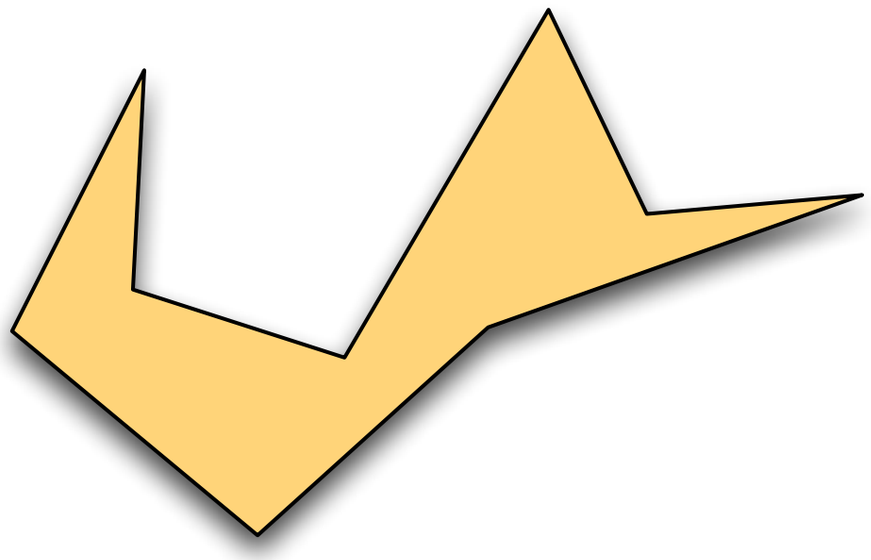
- Vertices convex C = { 0, 1, 3, 4, 6, 9 }
- Vertices reflex R = { 2, 5, 7, 8 }
- Vertices oreja E = { 3, 4, 6, 9 }

D. Eberly. Geometric Tools. (www.geometrictools.com)

Ear clipping triangulation



- Vertices convex C = { 0, 1, 3, 4, 6, 9 }
- Vertices reflex R = { 2, 5, 7, 8 }
- Vertices oreja E = { 3, 4, 6, 9 }



- Eliminar oreja 3 :
- Primer triangulo en la triangulación T0 = ⟨ 2, 3, 4⟩.

D. Eberly. Geometric Tools. (www.geometrictools.com)

Triangulación de un polígono monónoto

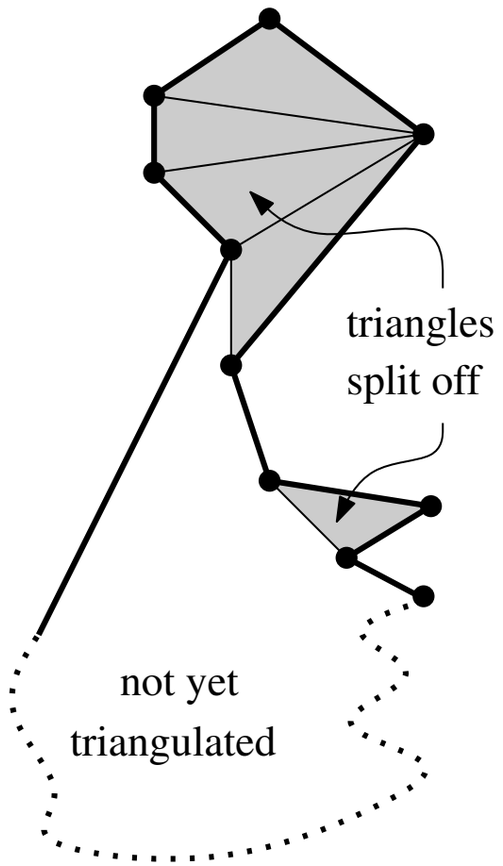
- Sea $\mathcal{P}$ un polígono monótono respecto a y con n vértices.
- Suponemos que $\mathcal{P}$ es estrictamente monótono (que no contiene aristas horizontales)
- Siempre podemos ir hacia abajo al "caminar" en las fronteras izquierda o derecha del polígono a partir del vértice más alto de $\mathcal{P}$ hasta el más bajo.
- Podemos seguir a $\mathcal{P}$ desde arriba hacia abajo en ambas cadenas en su frontera agregando diagonales cuando sea posible.
- Este es un algoritmo glotón: conectamos la mayor parte de vértices posible con diagonales en cada iteración.

Triangulación de un polígono monónoto

- El algoritmo maneja los vértices en orden decreciente de su coordenada y.
- Se requiere una estructura auxiliar: una pila (stack)
- Inicialmente, esta pila está vacía, luego contiene los vértices de $\mathcal{P}$ que se han encontrado (al paso de la línea de barrido) pero que necesitan más diagonales.
- Al manejar un vértice se le añaden tantas diagonales con los vértices de la pila como sea posible.
- Estas diagonales dividen al polígono en triángulos.
- El vértice más bajo será el que esté arriba en la pila.

Triangulación de un polígono monónoto

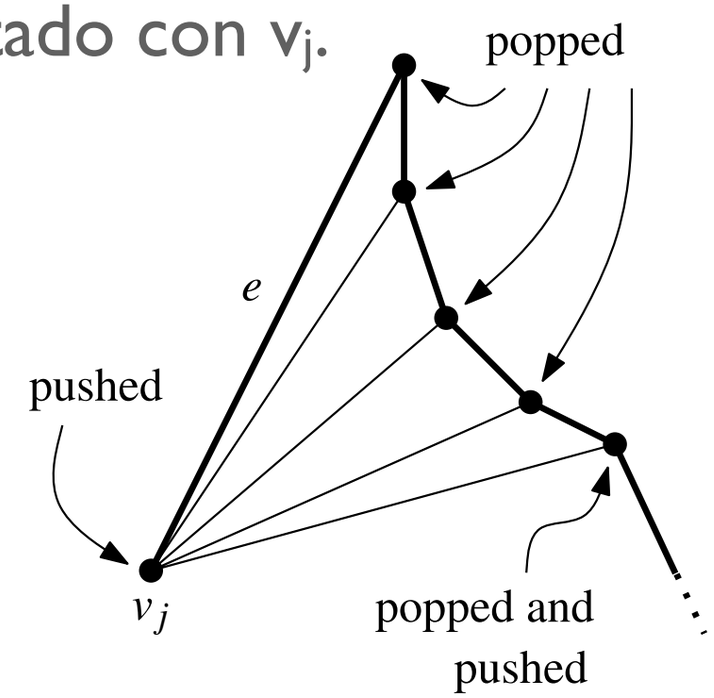
- La porción de $\mathcal{P}$ que no ha sido triangulada pero que se encuentra arriba del último vértice visitado tiene forma de embudo.

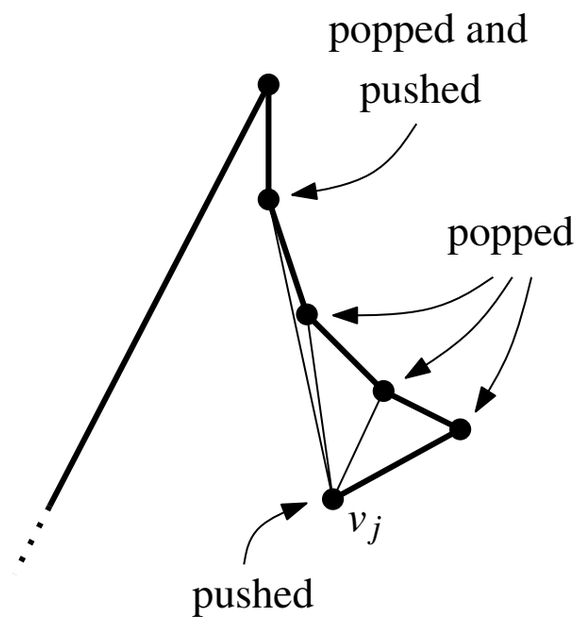


- Una frontera del embudo consta de parte de una sola arista de $\mathcal{P}$.
- La otra frontera es una cadena que consta de los vértices reflejo (*reflex*), es decir, aquellos cuyo ángulo interior es de al menos 180°.
- Solo el vértice más alto (el más profundo en la pila) es convexo.
- Esta propiedad se mantiene durante toda la ejecución del algoritmo (invariante).

¿Dónde agregar diagonales?

- Al manejar un vértice v_j distinguimos entre dos casos.
  - v_j se encuentra en la *misma cadena* que los vértices reflejo (dentro de la pila). Se puede triangular a los vértices inmediatamente superiores sobre la misma cadena.
  - v_j se encuentra en la *cadena opuesta* a los vértices reflejo debe ser el vértice inferior de la arista frontera del embudo. Se pueden agregar diagonales a todos vértices de la pila excepto al último que ya estará conectado con v_j.





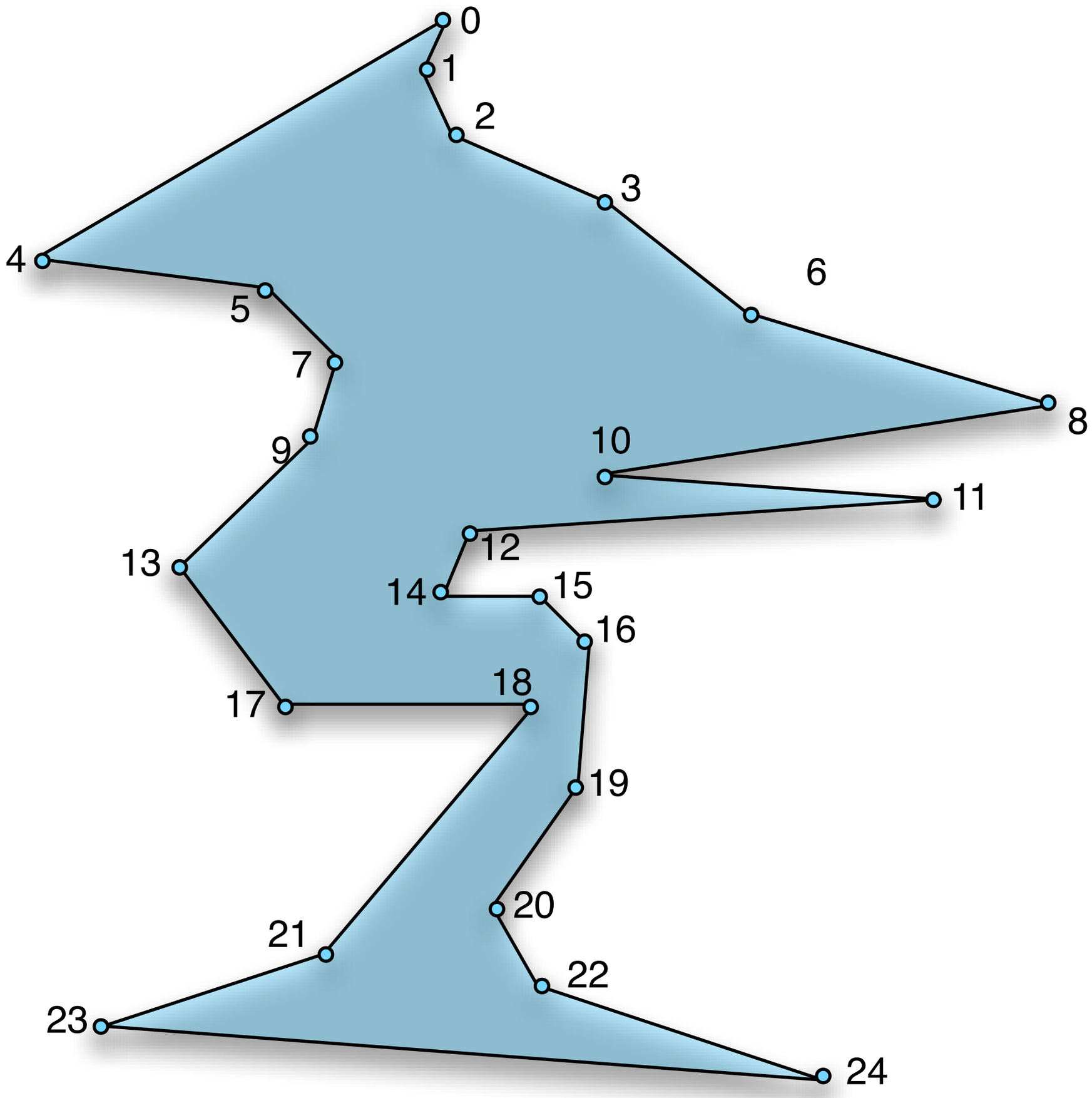
Triangulación de un polígono monónoto

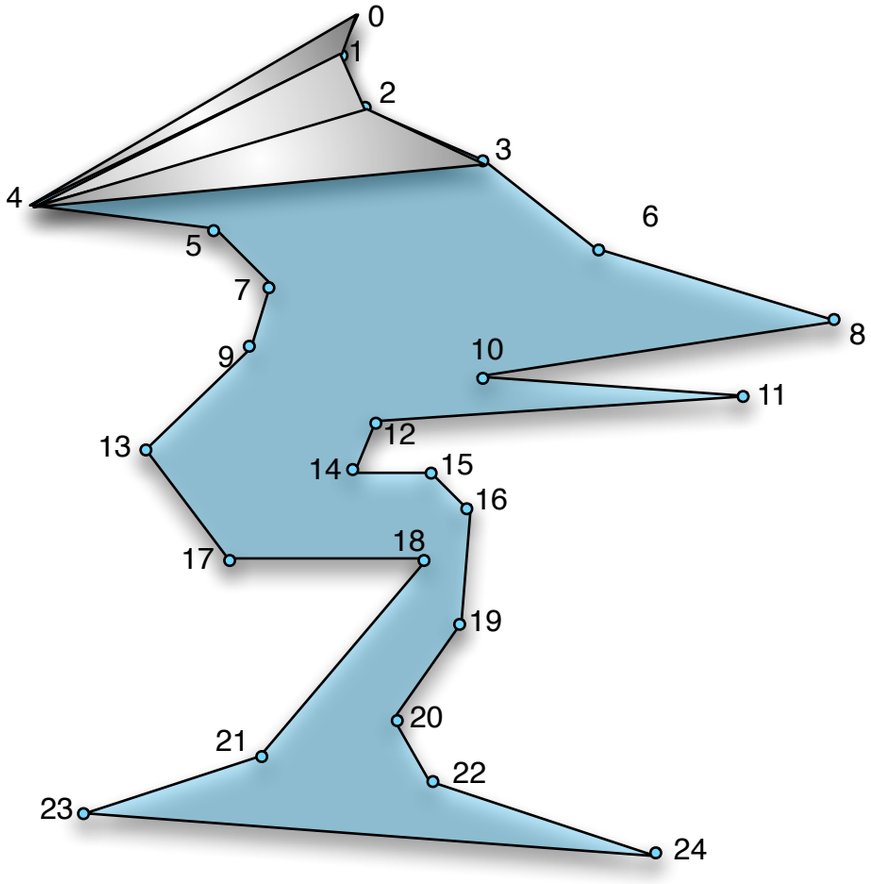
Algorithm TRIANGULATEMONOTONEPOLYGON($\mathcal{P}$)

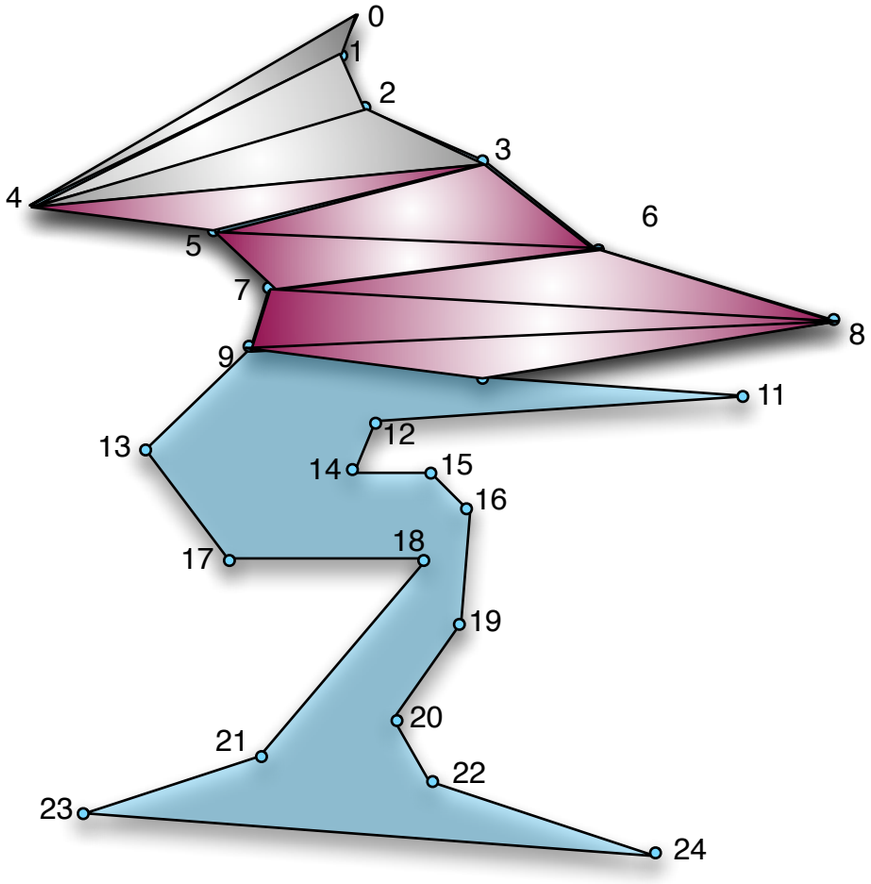
Input. A strictly y-monotone polygon $\mathcal{P}$ stored in a doubly-connected edge list $\mathcal{D}$.

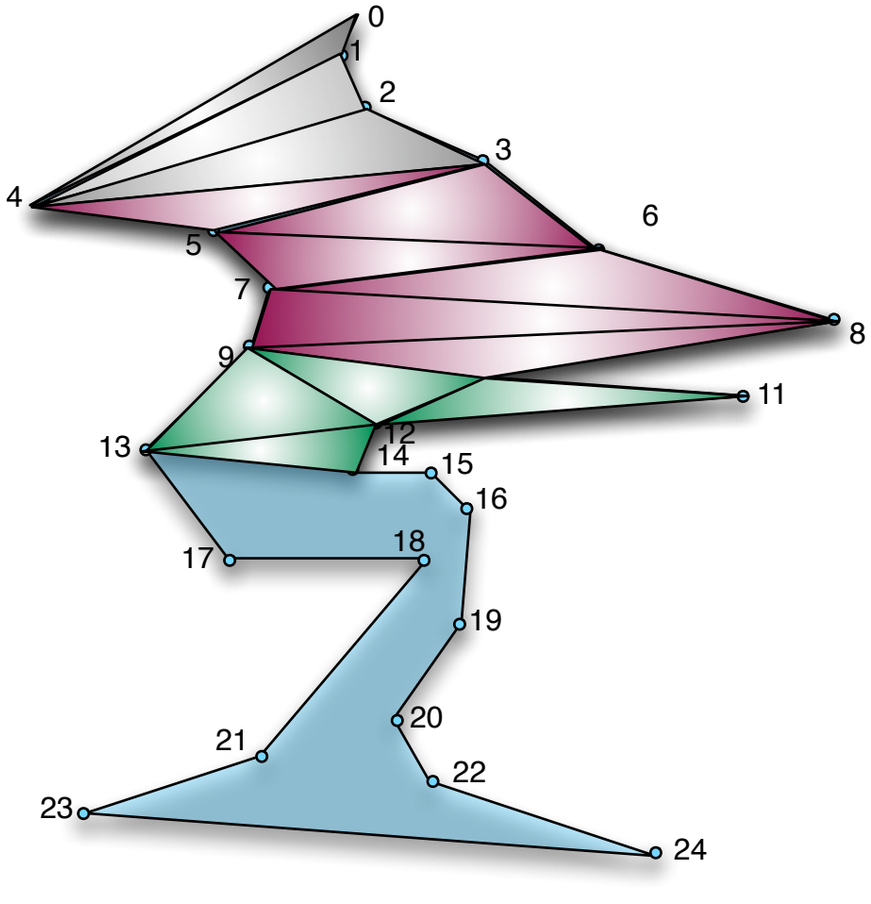
Output. A triangulation of $\mathcal{P}$ stored in the doubly-connected edge list $\mathcal{D}$.

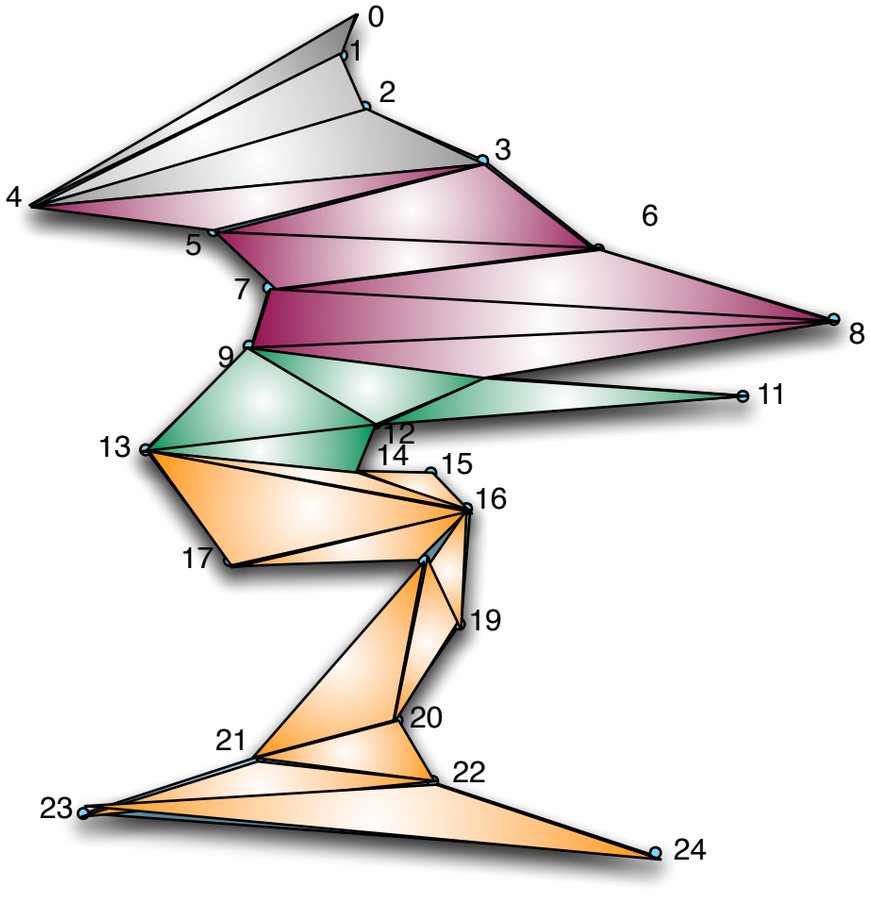
1. Merge the vertices on the left chain and the vertices on the right chain of $\mathcal{P}$ into one sequence, sorted on decreasing y-coordinate. If two vertices have the same y-coordinate, then the leftmost one comes first. Let $u_1, \ldots, u_n$ denote the sorted sequence.
2. Initialize an empty stack $\mathcal{S}$, and push u_1 and u_2 onto it.
3. **for** $j \leftarrow 3$ **to** $n-1$
4. **do if** u_j and the vertex on top of $\mathcal{S}$ are on different chains
5. **then** Pop all vertices from $\mathcal{S}$.
6. Insert into $\mathcal{D}$ a diagonal from u_j to each popped vertex, except the last one.
7. Push u_{j-1} and u_j onto $\mathcal{S}$.
8. **else** Pop one vertex from $\mathcal{S}$.
9. Pop the other vertices from $\mathcal{S}$ as long as the diagonals from u_j to them are inside $\mathcal{P}$. Insert these diagonals into $\mathcal{D}$. Push the last vertex that has been popped back onto $\mathcal{S}$.
10. Push u_j onto $\mathcal{S}$.
11. Add diagonals from u_n to all stack vertices except the first and the last one.


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24


0
1
2
3
4
5
6
7
8
9
11
12
13
14
15
16
17
18
19
20
21
22
23
24


0
1
2
3
4
5
6
7
8
9
11
12
13
14
15
16
17
18
19
20
21
22
23
24


0
1
2
3
4
5
6
7
8
9
11
12
13
14
15
16
17
18
19
20
21
22
23
24

Triangulación de un polígono monónoto

- El paso 1 toma tiempo lineal
- El paso 2 toma tiempo constante
- El ciclo for se ejecuta n-3 veces, una ejecución puede tomar tiempo lineal.
- En cada ejecución del ciclo for se insertan a lo más dos vértices en el stack, entonces el número total de inserciones en el algoritmo está acotado por 2n-4.
- Como el número de operaciones pop no puede exceder el número de operaciones push, el tiempo total de ejecuciones para el ciclo for es O(n).
- El último paso del algoritmo toma a lo más tiempo lineal.
- El algoritmo se ejecuta en O(n).