Geometría Computacional
Práctica 03

Profesor:
Adriana Ramírez Vigueras

Ayudante:
José Emiliano Cabrera Blancas

27 de marzo de 2014

1 Objetivos:

- Que el alumno logre programar el algoritmo de barrido de línea para partircionar un polígono simple en piezas monótonas.

2 Descripción

En la práctica anterior ustedes tuvieron la tarea de implementar el algoritmo para encontrar las intersecciones de n segmentos usando un barrido de línea, en esta ocasión deberán reutilizar su *priority queue* que programaron previamente para implementar el algoritmo que se les pide usando las estructuras que se les proporciona, siendo éstas: el árbol de búsqueda que mantiene el estado del barrido de línea y la *DCEL*.
La entrada del algoritmo que se les pide es una *DCEL* que represente un polígono simple, y como salida deberán regresar la misma *DCEL* pero particionada en piezas monótonas. Como es de sospecharse en esta ocasión el directorio fuente es una combinación de la primera y segunda parte de la práctica anterior donde agregué los archivos *make_monotone.c* y *make_monotonce.h*.
Ya vienen implementadas algunas funciones triviales que aunque son sencillas de pensar son un poco molestas para escribirse en código, dichas funciones serán descritas más adelante
Como recordarán la *DCEL* es una estructura estática y hacer pruebas significa construir una "a mano", para evitarnos ese problema, al igual que la práctica anterior, les proporcionaré una *DCEL* que representa un polígono simple que se describe en la siguiente sección.

3 *DCEL* ejemplo

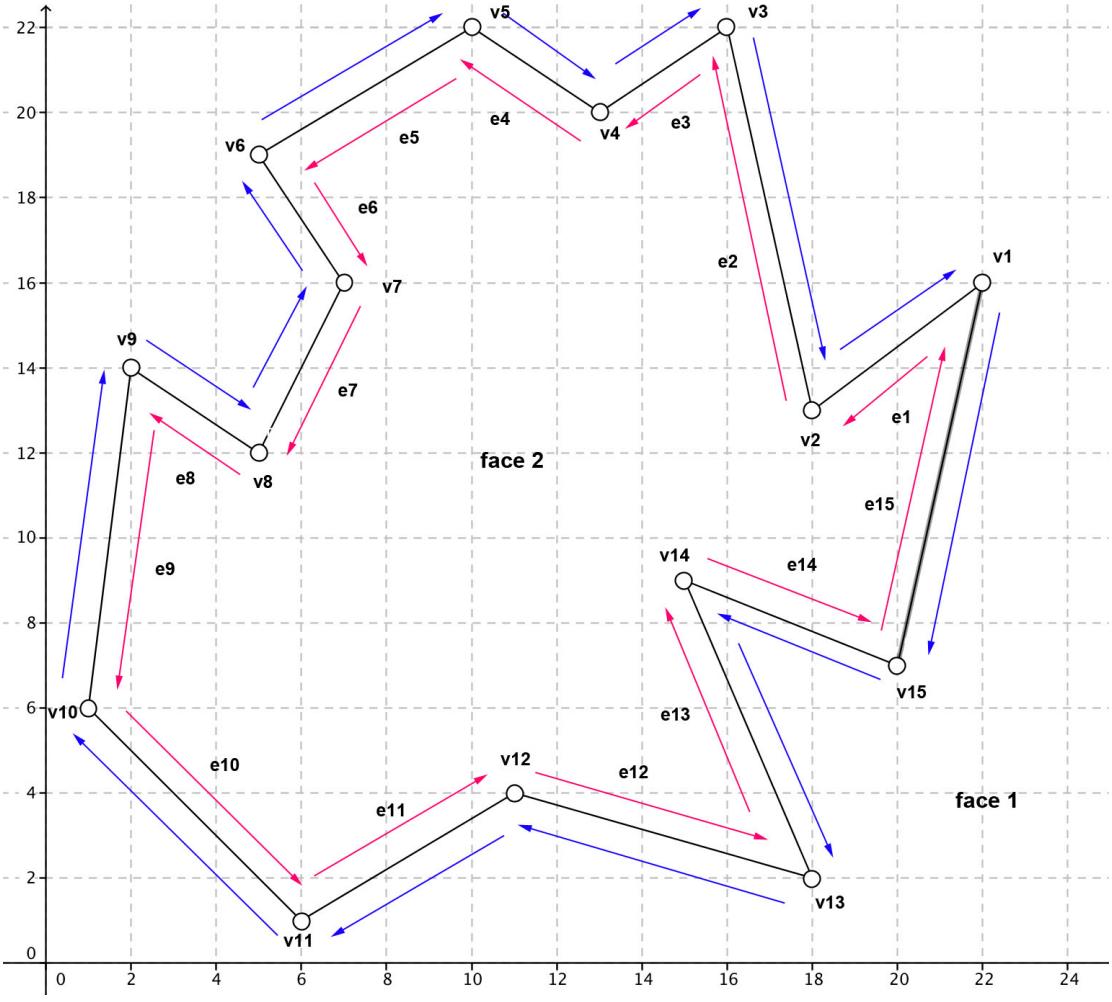

Figura 1

En la figura 1, podemos observar gráficamente la *DCEL* que contruye el archivo *example.c*, y en las tablas siguientes se detalla cada campo que tiene la *DCEL* de ejemplo.

Vértices

name	x	y	inc. edge
v_1	22	16	e_1
v_2	18	13	e_2
v_3	16	22	e_3
v_4	13	20	e_4
v_5	10	22	e_5
v_6	5	19	e_6
v_7	7	16	e_7
v_8	5	12	e_8
v_9	2	14	e_9
v_{10}	1	6	e_{10}
v_{11}	6	1	e_{11}
v_{12}	11	4	e_{12}
v_{13}	18	2	e_{13}
v_{14}	15	9	e_{14}
v_{15}	20	7	e_{15}

Aristas

name	first	inc. face	twin	next	prev
e_1	v_1	$face\ 2$	e'_1	e_2	e_{15}
e'_1	v_2	$face\ 1$	e_1	e'_{15}	e'_2
e_2	v_2	$face\ 2$	e'_2	e_3	e_1
e'_2	v_3	$face\ 1$	e_2	e'_1	e'_e
e_3	v_3	$face\ 2$	e'_3	e_4	e_2
e'_3	v_4	$face\ 1$	e_3	e'_2	e'_4
e_4	v_4	$face\ 2$	e'_4	e_5	e_3
e'_4	v_5	$face\ 1$	e_4	e'_3	e'_5
e_5	v_5	$face\ 2$	e'_5	e_6	e_4
e'_5	v_6	$face\ 1$	e_5	e'_4	e'_6
e_6	v_6	$face\ 2$	e'_6	e_7	e_5
e'_6	v_7	$face\ 1$	e_6	e'_5	e'_7
e_7	v_7	$face\ 2$	e'_7	e_8	e_6
e'_7	v_8	$face\ 1$	e_7	e'_6	e'_8
e_8	v_8	$face\ 2$	e'_8	e_9	e_7
e'_8	v_9	$face\ 1$	e_8	e'_7	e'_9
e_9	v_9	$face\ 2$	e'_9	e_{10}	e_8
e'_9	v_{10}	$face\ 1$	e_9	e'_8	e'_{10}
e_{10}	v_{10}	$face\ 2$	e'_{10}	e_{11}	e_9
e'_{10}	v_{11}	$face\ 1$	e_{10}	e'_9	e'_{11}
e_{11}	v_{11}	$face\ 2$	e'_{11}	e_{12}	e_{10}
e'_{11}	v_{12}	$face\ 1$	e_{11}	e'_{10}	e'_{12}
e_{12}	v_{12}	$face\ 2$	e'_{12}	e_{13}	e_{11}
e'_{12}	v_{13}	$face\ 1$	e_{12}	e'_{11}	e'_{13}
e_{13}	v_{13}	$face\ 2$	e'_{13}	e_{14}	e_{12}
e'_{13}	v_{14}	$face\ 1$	e_{13}	e'_{12}	e'_{14}
e_{14}	v_{14}	$face\ 2$	e'_{14}	e_{15}	e_{13}
e'_{14}	v_{15}	$face\ 1$	e_{14}	e'_{13}	e'_{15}
e_{15}	v_{15}	$face\ 2$	e'_{15}	e_1	e_{14}
e'_{15}	v_1	$face\ 1$	e_{15}	e'_{14}	e'_1

Caras

name	outer_component	inner_components
$face\ 1$	$NULL$	e'_1
$face\ 2$	e_1	$NULL$

4 *Tests*

El único *test* que debe pasar su implementación es el siguiente:

Tests

```
make_monotone(): Verifica que tu algoritmo hace ....          [PASS]
```

5 ¿Qué debes de programar y dónde comentar tus cambios?

En esta práctica tienes que implementar las siguientes funciones que se encuentran en el archivo *algorithms/make_monotone.c*:

- *void make_monotone*(*dcel∗ dcel*): Inicializa las estructuras pertinentes para el barrido de línea y procesa los eventos.
- *void handle_start_vertex*(*vertex∗ vi*, *rb_tree∗ tree*, *dcel∗ dcel*): Procesa el caso donde el vértice sea de tipo $START$.
- *void handle_merge_vertex*(*vertex∗ vi*, *rb_tree∗ tree*, *dcel∗ dcel*): Procesa el caso donde el vértice sea de tipo $MERGE$.
- *void handle_regular_vertex*(*vertex∗ vi*, *rb_tree∗ tree*, *dcel∗ dcel*): Procesa el caso donde el vértice sea de tipo $REGULAR$.
- *void handle_split_vertex*(*vertex∗ vi*, *rb_tree∗ tree*, *dcel∗ dcel*): Procesa el caso donde el vértice sea de tipo $SPLIT$.
- *void handle_end_vertex*(*vertex∗ vi*, *rb_tree∗ tree*, *dcel∗ dcel*): Procesa el caso donde el vértice sea de tipo END.

Recuerden que la función *make_monotone* depende de una *priority queue* que ustedes deben proporcionar, el árbol de búsqueda ya viene implementado. A la estructura $half\_edge$ se le agrego el campo *vertex∗ helper* necesario para su algoritmo.
Por último las funciones que ya vienen implementadas son las siguientes:

- *vertex_type calculate_vertex_type*(*vertex∗ p*): Calcula el tipo de vértice que es p ($MERGE$, $SPLIT$, $REGULAR$, END o $START$).
- *void connect_diagonal*(*vertex∗ first*, *vertex∗ last*, *dcel∗ dcel*, *rb_tree∗ tree*): Crea una diagonal entre los dos vértices que se le pasan a la función.

Como podrán darse cuenta, no se les pide que implementen gráficar el polígono monótono, eso se debe a que en la práctica anterior ustedes ya programaron el método para dibujar una *DCEL* en *Ruby-Processing*.
En la misma carpeta que contiene el directorio *src* deben incluir un archivo *README.txt* donde indiquen en que carpetas estan implementadas la *priority queue* que necesita el barrido de línea. Recuerden que es importante que sigan las convenciones de código de C que se les pasó al principio del curso. También deben recordar documentar sus estructuras de forma minusiosa y conciza.

Entrega 9 de abril de 2014