# Geometría Computacional
# Práctica 02: primera parte

Profesor:
Adriana Ramírez Vigueras

Ayudante:
José Emiliano Cabrera Blancas

04 de marzo de 2014

## 1. Objetivo:

- Que el alumno implemente el algoritmo de barrido de línea para encontrar la intersección de segmentos.
- Que el alumno logre dibujar la intersección de segmentos previamente calculados usando *Ruby-Processing*.

## 2. Descripción

### 2.1. Compilación

Para compilar ésta práctica seguiremos usando el mismo *script Makefile.rb*, por lo que las acciones que pueden realizar son las mismas descritas en la práctica pasada. Para agregar los nuevos archivos de C y estos se compilen, en el *script* existen ejemplos ilustrativos de como agregarlos.

## 3. Descripción del directorio fuente

1: Archivos

```
.
|--- Makefile.rb
|--- algorithms
|    |--- intersection.c
|    |--- intersection.h
|--- double_linked_list
|    |--- double_linked_list.c
|    |--- double_linked_list.h
```

```
|--- gui
|    |--- intersections.rb
|--- half_edge
|    |--- half_edge.c
|    |--- half_edge.h
|--- lib
|--- main.c
|--- points
|    |--- 2d_points.c
|    |--- 2d_points.h
|--- red_black_tree
|    |--- rb_tree.c
|    |--- rb_tree.h
|--- shared_libraries
|    |--- double_linked_list.rb
|    |--- half_edge.rb
|    |--- intersection.rb
|    |--- points.rb
|    |--- rb_trees.rb
|--- tests
|    |--- half_edge_tests.rb
|    |--- intersection_tests.rb
|    |--- list_tests.rb
|    |--- points_tests.rb
|    |--- rb_tree_tests.rb
|    |--- tests.rb
|--- types
     |--- types.c
     |--- types.h

10 directories, 26 files
```

Los directorios estan organizados de la siguiente forma:

- **algorithms:** En este directorio están los archivos que contienen la información para implementar el algoritmo de barrido de linea para detectar intersección de segmentos.
- **double_linked_list:** La lista doblemente ligada en esta ocasión fue modificada para que puedan utilizarla para crear listas de puntos y listas de segmentos. En el archivo *main.c* vienen ejemplos de como crear y destruir listas de ambos tipos.
- **gui:** En ésta práctica deberán de implementar el código de *Ruby* para dibujar los puntos que calcula su algoritmo de intersección de puntos, al igual que la práctica anterior ya vienen implementados los métodos para mapear las funciones de C a *Ruby-Processing*, y solo deben de enfocarse en completar el método para dibujar los puntos de intersección.

- **half_edge:** Dado que en la siguiente parte de ésta práctica usaremos una estructura especial para representar aristas dirigidas, desde esta primera parte vamos a usar una implementación parcial de las aristas dirigidas de tal forma que funcionen como segmentos dirigidos.
  Ya viene implementada la función para calcular la intersección de dos segmentos, a menos de que quieran modificar esta estructura, los archivos de esta carpeta no deberían verse modificados.

- **lib:** Aquí se guardan las bibliotecas dinámicas que compila *GCC* para *Ruby*.

- **main.c**: Aquí coloqué ejemplos sobre como crear y destruir listas y árboles rojo-negro. También esta ideado para que aquí ustedes puedan probar las funciones que ustedes implementen.

- **points:** En ésta carpeta se encuentra la cabecera y la implementación que abstrae puntos en el plano.

- **red_black_tree:** Como ustedes bien saben, la implementación del barrido de línea implica crear dos árboles especiales para procesar los eventos y detectar de forma efícaz que segmentos se intersectan. El árbol rojo negro que aquí esta implementado, sirve como una guía para que ustedes implementen esas dos estructuras, pueden reutilizar el código del árbol rojo-negro o ustedes crear sus propias estructuras (les recomiendo la segunda opción).
  Este árbol rojo negro ordena puntos por el eje de coordenada $X$ o $Y$. La recomendación de siempre, es que antes de escribir código, primero piensen y razónen los campos y las funciones que esas estructuras deben de tener.

- **shared_libraries:** Contiene el código de *Ruby* que pone a nuestra disposición las estructuras y funciones de *C*.

- **tests:** Como su nombre lo indica, aquí están implementados los *tests* de *Ruby* que comprueban las funciones principales de *C*.

- **types:** Aquí se encuentran funciones que utliza *Ruby-Processing* y también se encuentran los tipos *enum* que describen de que tipo es la lista que se crea.

## 4. *Tests*

Para verificar que el algoritmo que implementen funciona en casos particulares deben de pasar los siguientes *tests*:

2: *Tests* del cierre convexo.

```
intersection.h: find_intersections recibe nulo como entrada    [PASS]
intersection.h: find_intersections recibe una lista vacia      [FAIL]
intersection.h: find_intersections recibe una lista no vacia   [FAIL]
```

Estos tres *tests* están implementados en el archivo **tests/intersection_tests.rb** y no deben de ser modificados.
Los *tests* del árbol rojo-negro están implementados para que ustedes puedan verificar que la estructura de ejemplo que se les brinda funciona correctamente. Si para el final de su práctica éstas pruebas no sirven, no tendrá ninguna repercusión en la calificación final de su práctica.

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

En esta práctica tienes que implementar la función *find_intersections()* que esta declarada en **algorithms/intersection.h**, usando el algoritmo de barrido de línea visto en clase, una muy buena descripción del algoritmo y de las estructuras que necesita el algoritmo se encuentran en el *Overmars*, por lo que es muy recomendado primero leer y entender el texto del libro y una vez hecho esto, implementar las estructuras y el algoritmo de barrido de línea.
Debes verificar que tu implementación pase los tres *tests* que se te piden y por último terminar de implementar el método *draw_intersections* declarado en **gui/intersections.rb**.

En la misma carpeta que contiene el directorio *src* deben incluir un archivo *README.txt* donde indiquen en que carpetas estan implementadas las dos estructuras auxiliares 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.

**Fecha de entrega: 17 de marzo de 2014**