Laboratorio 3- El TAD Pila. Algoritmos de relleno de polígonos

GUIÓN DEL LABORATORIO

1.- Objetivos del laboratorio

- Diseño de clases en C++
- Comprensión y uso del TAD Pila
- Razonar sobre el comportamiento y limitación de los algoritmos presentados

2.- Antes de asistir al laboratorio

Antes de asistir al laboratorio debes realizar las siguientes tareas:

1. Leer los apuntes de clase sobre el TAD Pila.
2. Revisar cuidadosamente el guión del laboratorio (este documento).
3. Contestar a la cuestiones 1 a 5 de la Actividad 0 del Prerrequisito. Recuerda que es preciso entregar las respuestas a esta actividad antes de la fecha prevista. En caso contrario, no podrás acceder al laboratorio ni entregar de ninguna manera la solución a esta práctica.

3.- Enunciado

Introducción

La recursividad es una técnica usada en ciertos métodos de resolución de problemas como la estrategia de "divide y vencerás" o los métodos de "vuelta atrás" (backtracking). Aunque el uso de funciones recursivas permite expresar la solución del problema de una manera breve, elegante y comprensible, no es eficiente en problemas donde se hace un uso intensivo de ella, como es el caso del problema que nos proponemos abordar en esta práctica.

El problema del empleo de la recursividad reside en que cada llamada a la función recursiva obliga al sistema a instanciar recursos (variables locales, información del estado de ejecución del programa,...) y a alojarlos en la memoria principal con el consecuente empleo de tiempo en estas tareas. Además no es posible controlar a nivel de aplicación la pila del programa por lo que si las llamadas recursivas son masivas, corremos el riesgo de desbordar la pila del programa sin posibilidad de controlar esta situación.

Por todo ello, en problemas donde es esperable que la función recursiva sea llamada muchas veces, se opta por elaborar un código iterativo (bucles) equivalente al recursivo pero que usa una **pila** para ejecutar en forma adecuada el código de la función recursiva. De esta manera se elimina la recursividad a costa de usar una estructura de datos de tipo pila que la sustituye.

Vamos a utilizar el TAD Pila para este propósito sobre un problema interesante relacionado con los gráficos por ordenador, rellenar de color una figura poligonal cerrada.

Para ello, suponemos que nuestra superficie de dibujado está formada por puntos de color (llamados pixeles), a la manera de esos juegos infantiles consistentes en una superficie con muchos agujeros en los que se insertan chinchetas de colores.

Dentro del problema de rellenado, existen diferentes aproximaciones. En esta práctica nos centraremos en el llamado algoritmo de la semilla. El algoritmo de la semilla necesita un punto interior del polígono a partir del cual comienza a colorear (semilla). Como requisito, esta versión necesita que la frontera del polígono esté coloreada con un único color (color de frontera) que ningún otro pixel interior al polígono debe de tener.

El método de la semilla para rellenar figuras poligonales cerradas

Dada una línea poligonal cerrada que tiene pintada su frontera de un color determinado, el método recursivo de relleno por la semilla tiene el siguiente esquema:

```
rellena(píxel p)
      entrada: píxel p //son las coordenadas de pantalla del píxel candidato a ser coloreado
      salida: nada
      variables auxiliares: pixel aux;

      Si el pixel p es del color de fondo
            pinta(p) //pinta (x,y) con el color de relleno
            aux.x = p.x-1;
            aux.y = p.y;
            rellena(aux); //pixel de la izquierda
            aux.x = p.x+1;
            aux.y = p.y;
            rellena(aux); //píxel de la derecha
            aux.x = p.x;
            aux.y = p.y-1;
            rellena(aux); //píxel inferior
            aux.x = p.x;
            aux.y = p.y+1;
            rellena(x, y+1); //píxel superior
      fSi
frellena
```

Como se ve, el procedimiento tratará de colorear los cuatro píxeles que rodean al que es pasado como argumento. Esto genera una llamada a la función por cada pixel interior del polígono.
Este algoritmo es válido para la mayoría de las formas geométricas poligonales, incluso con polígonos "con agujeros" en su interior, aunque puede fallar en algunas formas concretas.

El problema con este algoritmo es que es posible que se mantengan abiertas un gran número de llamadas en cada momento, lo cual puede provocar un desbordamiento de pila si el área del polígono es muy grande.

El método de la semilla con pila

Este método es una versión mejorada del anterior algoritmo en dos aspectos:

1. Elimina la necesidad de recursión usando una pila.
2. No todos los píxeles del interior de la figura son apilados, ya que se usa la vecindad de pixeles en una línea para colorear por tramos.

Una versión limitada de este algoritmo (la versión general debe considerar algunos problemas que no vienen al caso) es el siguiente:

```
rellena_mejor(pixel p)
       entrada: pixel interior al polígono
       salida: nada
       variables locales: semilla, aux, iar, iab de tipo pixel

       apila(p);

       Mientras (pila < > vacia) hacer
              semilla  = desapila()

              Si (semilla < > color de pintado)
                           aux = semilla
                           Hacer
                              pinta(aux)
                              aux.x-- //supone un origen de coordenadas en una esquina izquierda de la ventana
```

```
                        Hasta ((aux) = = color de frontera)
                        fHacer

                  iar = pixel de la línea superior inmediatamente a la derecha del pixel frontera
                  iab = pixel de la línea inferior inmediatamente a la derecha del pixel frontera
                        aux = semilla
                        Hacer
                            pinta(aux)
                            x++
                        Hasta (aux = = color de frontera)
                        fHacer

                Si (iar < > color de relleno AND iar esta dentro del poligono)
                             apila(iar)
                         fSi

                Si (iab < > color de relleno AND iab esta dentro del poligono)
                             apila(iab)
                         fSi

            fSi
        fMientras
frellena_mejor
```

La idea de este algoritmo es pintar todos los pixeles de una línea y luego apilar los pixeles del extremo izquierdo de la línea superior e inferior a la que se está pintando (si procede). Estos nuevos puntos serán las semillas para pintar nuevas líneas. El programa acaba cuando se agotan las semillas apiladas.

Material suministrado

Se suministran los siguientes ficheros en código C++:

`lienzo.h`

En el que se encuentran:

TAMX , TAMY que configuran la resolución del lienzo (por defecto 640x480)

Se define el tipo de datos `color`

Se define la estructura de datos

```
struct pixel {
  int x;
  int y;
}
```

que indica la posición del punto en la superficie de dibujado (lienzo).

Se define la clase

```
class lienzo
{
  public:
    void pinta(pixel p);
    color indica_color(pixel p);

    (...)//algun metodo  mas que no importa a tu trabajo
  private:

    (...) algunos métodos mas que no importan a tu trabajo
    color matriz[TAMX][TAMY];
}
```

Esta clase representa a una superficie de dibujado de TAMX x TAMY pixeles y contiene el registro del color de cada pixel así como algunas herramientas para dibujar rectas. Es en ella donde se cambia el color de un pixel (a través del método `pinta`) o se consulta el color (a través del método `indica_color`).

Detalles técnicos de la clase `lienzo`

La clase `lienzo`:

1. Usa una estructura de datos de tipo "pixel" que es la que sirve para identificar los puntos de color.
2. Usa los tres colores que hemos definido en ese mismo fichero donde se encuentra la clase y que llamaremos
   1. FONDO (color por defecto del lienzo)
   2. FRONTERA (color de las aristas del poligono)
   3. RELLENO (color con el que rellenamos el poligono)

Estos colores son los que devuelve el metodo `indica__color()` de la clase `lienzo`

`lienzo.cpp`
En él se encuentra el código de los métodos de la clase "`lienzo`"

`alg_relleno.h y alg_relleno.cpp`
Estos son tus ficheros de trabajo. En el fichero `alg_relleno.h` se define la clase

```
class alg_relleno{
  public:
    lienzo papel; //objeto de la clase anterior

    alg_relleno();

    void rellena(pixel p);//metodo de la actividad 1
    void rellena_mejor(pixel p); //método de la actividad 2
};
```

en el fichero `alg_relleno.cpp` hay una implementación del método

```
void rellena_mejor(pixel p);
```

ya que no te vamos a pedir que lo implementes porque tiene algunos detalles que no son interesantes para esta practica pero que su elaboración podría resultar costosa.

`grafic.h y grafic.cpp`
Estos ficheros contienen la estructura del programa que usa la librería gráfica *opengl* para visualizar el polígono y maneja la interacción con el ratón.
También instancia un objeto de tipo `alg_relleno` que contiene los algoritmos de relleno y que, a su vez, instancia un objeto de tipo `lienzo`.

4.- Actividades.

Actividad 1

Implementación de la versión recursiva del algoritmo.
Se pide implementar el método `void rellena(pixel p)` de la clase `alg_relleno` que rellena de color un polígono usando el algoritmo recursivo de la semilla.
Este método lo debéis codificar como método de esta clase incluyéndolo en el fichero `alg_relleno.cpp`

Nota: No toques ninguna otra parte del código de este fichero. Solo incluye tu código en él.

Actividad 2

Implementación de la versión con pila del algoritmo.
Como la implementación iterativa del algoritmo de la semilla tiene ciertos detalles que no son al caso en este curso, se os proporciona ya implementado este método llamado `rellena_mejor(...)`
Dicha implementación usa una pila de tipo **pila estática** con sus métodos, organizada del modo en que ya conoces, a saber, definidas en dos ficheros `Pila.h` y `Pila.cpp`.
Así el trabajo consiste en implementar completamente una clase `Pila`
Tras esto se debe de incluir en el proyecto de compilación los dos ficheros. `pila.h` y `pila.cpp`

Nota: No hace falta poner los `#include Pila.h` **porque ya están puestos donde hace falta.**

5.- Ficheros a entregar.

Correspondiente a la **Actividad 1**: Hay que entregar el fichero `alg_relleno.cpp`
Correspondiente a la **Actividad 2: Hay que entregar los ficheros** `pila.h y pila.cpp`

Los archivos con los programas correspondientes a las actividades 1 y 2 se deberán entregar, a través del aula virtual, al finalizar la sesión de prácticas, junto con la Hoja de Trabajo del estudiante.

6.- Compilación

Para generar un proyecto compatible con la librería gráfica opengl hacer lo siguiente:
Nuevo->proyecto->(pestaña "Multimedia"->Elegir OpenGLUT

Incluir a ese proyecto a través de la opción del menu **Proyecto ->Añadir a proyecto** los ficheros que se os suministran (Incluir para la segunda parte también los ficheros Pila.h y Pila.cpp)

Eliminar del proyecto el fichero main.cpp **ESTO ES IMPORTANTE** ya que nuestra función principal está en grafics.cpp. Para ello ir a **Proyecto->Eliminar Fichero**

ESTO ES IMPORTANTE. Para visualizar la ventana de entrada salida por la que aparecerán las instrucciones y los datos estadísticos, ir a
Proyecto->opciones de proyecto y elegir la opción **"Consola Win32 (modo texto)"**
Y recompilar todo el proyecto.

7.- Interacción con el programa.

Al ejecutar el programa te saldrán dos pantallas, una es la consola de entrada /salida por donde te saldrá información, otra es la ventana de dibujo. Activa esta última pinchando en ella.
Lo primero que debes hacer es definir los vértices del polígono. Para ello, pincha con el botón izquierdo del ratón e irás visualizando los puntos que serán los vértices del polígono. Para terminar de introducir vértices, pulsa con el botón derecho para introducir ese punto como último vértice del polígono. Automáticamente se visualizarán las aristas del polígono.
Para rellenar el polígono, SITUA EL PUNTERO DEL RATON EN EL INTERIOR DEL POLIGONO y pulsa la tecla 'r' o 'R'. El punto donde se encuentre el ratón constituirá la semilla inicial del algoritmo. CUIDADO si el puntero está fuera del polígono, se generara un error de ejecución porque los índices de recorrido de la matriz se saldrán del rango de ésta.
Luego, siguiendo el menú que te sale por pantalla, pulsa 'a' o 'b' para elegir el algoritmo de rellenado o pulsa 'c' para borrar el polígono y empezar de nuevo.